Державний вищий навчальний заклад
“Прикарпатський національний університет імені Василя Стефаника”
Кафедра інформаційних технологій

УДК 004

ДИПЛОМНИЙ ПРОЕКТ

Тема Розробка бекенд частини сайту для внутрішньої роботи IT компанії

Спеціальність 121 Інженерія програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДП.ІПЗ-13.ПЗ
(позначення)

Рецензент
доц. Козленко М. І.
(посада) (підпис) (дата) (розшифровка підпису)

Студент
ІПЗ-41 Матчак А. Л.
(шифр групи) (підпис) (дата) (розшифровка підпису)

Нормоконтролер
доц. Козленко М. І.
(посада) (підпис) (дата) (розшифровка підпису)

Керівник дипломного проекту
доц. Ткачук В. М.
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Завідувач кафедри
доц. Козленко М. І.
(посада) (підпис) (дата) (розшифровка підпису)

2020
(рік)

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Факультет математики та інформатики Кафедра інформаційних технологій
Спеціальність 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри Козленко М. І.
студенту Матчаку Андрію Любомировичу
„_____”_____ 20___р.

ЗАВДАННЯ
НА ВИКОНАННЯ ДИПЛОМНОГО ПРОЕКТУ

Студенту Матчаку Андрію Любомировичу
(прізвище, ім’я, по батькові студента)

1. Тема проекту Розробка бекенд частини сайту для внутрішньої роботи IT компанії затверджена розпорядженням по факультету математики та інформатики від „25” жовтня 2019р. №7.
2. Термін здачі студентом закінченого проекту 22 травня 2020 р.
3. Вихідні дані до дипломного проекту технології розробки - .NET, Entity Framework, Sql Server, Swager API
4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати)
1. Аналіз предметної області;
2. Проектування та архітектура проекту;
3. Практична реалізація проекту;
4. Економіка проекту.
5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень) 1. Вступ; 2. Аналіз Корпоративного порталу; 3. Архітектура програми; 4. Проектування функціоналу; 5. Проектування бази даних; 6. Google OAuth; 7. Економіка проекту; 8. Висновки.

6. Дата видачі завдання 11.09.2019 р.

Керівник _____ Ткачук В. М.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Матчак А. Л.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
1. Аналіз предметної області	02.12.2019	виконав
2. Проектування та архітектура проекту	15.02.2020	виконав
3. Практична реалізація проекту	17.04.2020	виконав
4. Економіка проекту	11.05.2020	виконав
5. Оформлювання пояснювальної записки	18.05.2020	виконав

Студент _____ Матчак А. Л.
(підпис) (розшифровка підпису)

Керівник проекту _____ Ткачук В. М.
(підпис) (розшифровка підпису)

РЕФЕРАТ

Пояснювальна записка: 94 сторінок (без додатків), 25 рисунків, 6 таблиць, 19 джерел, 2 додатки на 49 сторінках.

Ключові слова: СУБД, ORM, GOOGLE, РЕПОРТ, АКТИВНОСТІ, ПОРТАЛ, API, КОРИСТУВАЧ.

Об`єкт дослідження: Методи розробки програмних засобів.

Мета роботи: Вивчення складових частин внутрішнього порталу, основних принципів їх побудови і функціонування. Практичне освоєння методів розробки внутрішнього порталу.

Стислий опис тексту пояснювальної записки:

Даний дипломний проект присвячений розробці внутрішнього порталу для організації роботи всередині компанії, на основі платформи .NET. Під час виконання здійснено аналіз інструментів, що зараз існують, та обрано ті, які найкраще підходять для виконання поставленої задачі. У проекті розроблено алгоритми, що дозволяють максимально швидко вирішити потрібного роду завдання. Проведено детальний опис складових частин програмного застосунку, а саме архітектури та зв’язків в базі даних, логування часу, контроль відпусток, внутрішніх та зовнішніх активностей компанії.

ABSTRACT

Explanatory note: 94 pages (without appendix), 25 figures, 6 tables, 19 references, 2 appendix on 49 pages.

Key words: DBMS, ORM, GOOGLE, REPORT, ACTIVITIES, PORTAL, API, USER.

Object of study: Methods of software development.

Purpose: To study the components of the internal portal, the basic principles of their construction and operation. Practical development of methods of internal portal development.

Brief description of the text of the explanatory note:

This diploma project is devoted to the development of an internal portal for the organization of work within the company, based on the .NET platform. During the implementation, the existing tools were analyzed and those that are best suited for the task were selected. The project has developed algorithms that allow you to quickly solve the desired type of problem. A detailed description of the components of the software application, namely the architecture and connections in the database, time logging, vacation control, internal and external activities of the company.

ПЕРЕЛІК СКОРОЧЕНЬ

PM - менеджер проекту.

AM - менеджер по роботі з клієнтами.

Timelog - залоговані робочі години.

CRM - система управління відносинами з клієнтами.

Google - американська публічна транснаціональна корпорація.

API - програмний інтерфейс програми.

REST - підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів.

ORM - технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування.

.NET - програмна технологія, запропонована фірмою Microsoft як платформа для створення як звичайних програм, так і веб-застосунків.

Авторизація - керування рівнями та засобами доступу до певного захищеного ресурсу, як у фізичному розумінні, так і в галузі цифрових технологій та ресурсів системи.

Автентифікація - процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора.

СУБД - набір взаємопов'язаних даних і програм для доступу до цих даних.

Логування – занесення робочих годин в базу даних.

ASP.NET - технологія створення веб-застосунків і веб-сервісів від компанії Майкрософт.

HTML - це мова тегів, якою пишуться гіпертекстові документи для мережі Інтернет Веб-браузери отримують HTML-документи з веб-сервера або з локальної пам'яті і передають документи в мультимедійні веб-сторінки.

OLAP - це інтерактивна система що дозволяє переглядати різні підсумки по багатовимірних даних.

URL - розташування веб-сторінки або файлу в Інтернеті.

SSL - криптографічний протокол, який забезпечує встановлення безпечного з'єднання між клієнтом і сервером.

Token - використовується для ідентифікації його власника, безпечного віддаленого доступу до інформаційних ресурсів і т. д.

ЗМІСТ

ВСТУП

Основною метою створення різного роду програмного забезпечення є автоматизація процесів, заміна роботи людини та пришвидшення рішень багатьох проблем. Так і розробка порталу підприємства, допоможе контролювати внутрішні процеси, скоротити кількість робочих місць і цим самим стати джерелом непрямого прибутку. В наш час, кожна велика компанія має свій внутрішній портал.

Особливо актуальними портали стали в IT сфері. В кожної компанії, постає необхідність в веденні проектів та внутрішніх відділів, бухгалтерії та контролю роботи працівників. В менеджменті, для керівника важливо контролювати якнайбільше процесів, витрачаючи якнайменше свого часу. Такі провідні компанії як “SoftServe”, “Epam”, “Eleks” та інші, використовують портали для залучення нових працівників, контролю навчання та інших процесів.

Об’єкт дослідження: функціонування внутрішніх процесів IT компаній.

Предмет дослідження: автоматизація котролю за внутрішніми процесами IT компанії.

Методи дослідження: основні результати і висновки зроблено на основі теорії інформації, методів кваліметрії, методів алгоритмічно-програмних засобів організації внутрішніх WEB-сторінок.

Мета дипломної роботи: аналіз і розробка нового програмного забезпечення для внутрішньої роботи IT компанії.

Для досягнення мети дипломної роботи поставлено такі завдання:

- розглянути основні засоби для розробки програмного забезпечення;
- проаналізувати основні внутрішні процеси компанії;
- на основі даних аналізу, розробити програмне забезпечення, яке автоматизує основні внутрішні процеси, пришвидшить їх виконання, та покращить їх якість.

Завдання роботи: Розробити програмне забезпечення, яке автоматизує основні внутрішні процеси, пришвидшить їх виконання, та покращить їх якість.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис основного призначення порталу

Корпоративні портали забезпечують різноманітним специфічним функціоналом під потреби бізнесу. Деякі компанії самостійно розробляють індивідуальний і унікальний Корпоративний портал для роботи (але це м'яко кажучи не дешево), а можна взяти готове рішення і налаштувати його під свою специфіку.

При використанні корпоративного порталу ми отримуємо систему зберігання і систематизації всієї важливої інформації про клієнтів. Контролюємо вхідні заявки клієнтів. До працівників надходять листи, дзвінки, заявки з сайту, повідомлення в соціальних мережах і месенджерах. Це можуть бути заявки від нових або існуючих клієнтів, пропозиції від постачальників і партнерів. (Все це має назву Ліди). Для початку, всю цю інформацію потрібно зберегти і бажано класифікувати. Потім проаналізувати і використати, наприклад для виявлення найбільш ефективних рекламних площадок. Бітрікс24 автоматично збирає всі Ліди з різних джерел в єдину базу: збираються Ліди з корпоративних пошт, системи «відкриті лінії», з телефонних дзвінків, чатів і веб форм сайтів та багато іншого [14].

Співробітники мають доступ до цієї інформації, з будь-якого місця де є інтернет, вони можуть увійти в портал, навіть з телефону. На комп'ютер не потрібно встановлювати ніякого додаткового ПЗ, у співробітника робочий день почнеться просто - прийшов, сів за стіл, підключився до корпоративного порталу та все, він готовий працювати. Всі співробітники (навіть якщо вони працюють в різних офісах) можуть отримати оперативний доступ до інформації і спілкуватися між собою.

Важливим для відділу продажів є система управління взаємовідносинами з клієнтами - CRM. Тобто після того як прийшла заявка менеджеру, необхідно виконати ряд дій, перш ніж клієнт заплатить.

Такого роду послідовність дій в компанії називається «Бізнес-процес». Основних бізнес-процесів насправді дуже багато і добре б, щоб частина з них була автоматизованою.

CRM необхідна для того, щоб не втратити жодного клієнта. Для того, щоб не вчити нового менеджера як працювати, а щоб він чітко розумів, що йому робити. Для того, щоб була повна інформація про клієнта і змога на її підставі щось допродати. Щоб коли звільняється співробітник, була змога легко передати його клієнтів іншому менеджеру, так як зберігатися вся історія роботи з цими клієнтами. Якщо багато рутинних дії виконуються автоматично, то співробітники не витрачають на це час, працювати виходить швидше і вони можуть обробити більше клієнтів і оперативніше відповідати їм (в порівнянні, наприклад, з вашими конкурентами). А це вигідно і з точки зору продажів, і з точки зору загального іміджу компанії [13].

Також, важливим аспектом Корпоративного порталу є систем управління завданнями і проектами. Будь-яка, навіть скромна справа складається з завдань і підзадач, які потрібно доручити певним особам і завершити до певного часу. Часто повторювані послідовності, такі як виставлення рахунків на оплату або послідовне затвердження договорів, узгодження з усіма керівниками заяви на відпустку, можуть бути автоматизовані. Від корпоративного порталу при цьому потрібно вимагати зручного і інтуїтивно зрозумілого (наочного) інтерфейсу, для управління і контролю всіх завдань, розподіл ролей в задачах. Простий контроль термінів завдань, з нагадуваннями, угруповання завдань по проектам, наявність регулярних завдань, які ставляться, наприклад, раз на місяць або в квартал.

При роботі з чистими продажами, дана система, може прискорити внутрішні процеси фірми (наприклад, завдання до відділу забезпечення на закупівлю канцелярського приладдя або завдання в IT відділ на установку монітора новому співробітнику і т.д.). Якщо це фірма-виробник, тобто випускає якийсь продукт (матеріальний або програмний), то така система допомагає вирішити більшість робочих процесів. Від того, на скільки співробітникам

зручно користуватися системою управління проектами, безпосередньо залежить дохід компанії.

Чим більше стає компанія, тим складніше контролювати співробітників. Постає необхідність мати можливість отримати статистику по ефективності кожного співробітника. Керівники відділів повинні мати можливість аналізувати працездатність відділу. Корпоративний портал в свою чергу повинен дозволяти будувати звіти по найрізноманітнішим параметрам, робити це просто і наочно.

Портал забезпечує поліпшення трудової дисципліни та виявлення слабких місць компанії. Допомагає зрозуміти, на якому етапі гальмуватися реалізація проектів і продажів. Також для визначення стратегії розвитку компанії. До недоліків Корпоративного порталу відноситься те, що вся аналітика актуальна тільки тоді, коли весь необхідний персонал працює через нього [15].

1.1 Проблеми, які вирішує портал

Практично будь-яка сфера діяльності, так чи інакше створює навколо себе інформаційну систему. Наприклад, всі менеджери відділу продажів отримують заявки по телефону і поштою. Між собою обмінюються інформацією через Скайп. Затверджують договори і заяви з начальством в паперовому вигляді. У виробничий відділ потрапляють наряди, теж в паперовому вигляді. Нагадування і списки завдань кожен хороший співробітник носить у себе в телефоні. А база клієнтів в кращому випадку в 1С, з якою не кожен може працювати, в гіршому випадку, часто база взагалі в табличках Excel. А начальники відділів носять звіти з непровіреними даними [15].

Почати потрібно з виявлення, фіксації та стандартизації найважливіших для компанії бізнес процесів. Швидше за все, без порталу, кожен виконує роботу не з якоїсь загальної інструкції, а так, як він вміє і як у нього виходить, тому потрібно написати, а бажано намалювати блок-схеми всіх важливих бізнес процесів компанії. Без аналізу основних бізнес процесів компанії, не можливо

створити справді хороший Корпоративний портал. Тому стандартизація бізнес процесів в першу чергу, потім інструкції по роботі з порталом, і тільки тоді можна встановлювати корпоративний портал [15].

Корпоративний портал, як правило, включає в себе такий функціонал як:

- стрічка новин компанії;
- календар подій співробітників і всієї компанії;
- іструменти для роботи з завданнями;
- чат і дзвінки, соціальна мережа для спілкування співробітників;
- звіти, облік робочого часу, інструменти для документообігу служби персоналу;
- документи і сховище даних;
- навчальний портал;
- інтеграція з поштою і телефоном;
- інтеграція з соціальними мережами, мессенджерами;
- управління часом і завданнями, планування роботи у відділі продажів.

Календар в корпоративному порталі допомагає ефективно організувати робочий день і налагодити роботу в команді. Він дозволяє співробітникам заздалегідь розпланувати час, відсіяти менш важливі справи і сконцентруватися на основних. Календар сумісний з поштою і планувальником завдань. Календар можна синхронізувати з особистим календарем співробітника, в тому числі на смартфоні. Нагадування про майбутню подію не дозволить співробітнику його пропустити, а коментар допоможе підготуватися до нього якнайкраще.

Корпоративний портал допомагає керівникам контролювати виконання завдань, а підлеглим - не допускати порушень. Він дозволяє підключати до задачі колег, оцінювати роботу, враховувати витрачений час, планувати терміни.

Для роботи у відділі продажів, ці можливості є суттєвими. Швидкість і зручність при постановці завдань, а також можливість відстежити історію роботи, проконтролювати результат, важливі для керівника відділу [14].

У Корпоративному порталі, передбачена можливість зберігання всіх файлів, з якими йде робота у співробітників і клієнтів. За рахунок єдиного простору, колеги можуть працювати з одним і тим же файлом, редагувати його, оперативно обговорювати зміни.

Співробітники зберігають власні робочі файли на диску Корпоративного порталу, а також мають доступ до загальних файлів компанії будь-яким способом: в поїздці зі смартфона, або на домашньому комп'ютері, з ноутбука на зустрічі з клієнтом. Де б не знаходився співробітник або клієнт, він завжди може провести необхідні операції з робочими документами.

Сховище даних використовують і як внутрішню бібліотеку компанії, де зібрані всі стандарти, інструкції, маркетингові матеріали, зразки документів, навчальні матеріали та інші спеціалізовані документи. Всі файли синхронізуються. Співробітники і клієнти знають, що працюють з актуальною версією файлу. Також всередині самого порталу можна обмінюватися файлами, здійснювати пошук потрібної інформації.

Важливо, що документи, збережені в сховище Корпоративного порталу, знаходяться під захистом, і не будуть загублені, що б не трапилося з комп'ютером клієнта або співробітника.

Давши клієнту права на роботу з його документами в Корпоративному порталі, менеджер виключить зайві дії по відправці документів, і сам буде мати до них постійний доступ, що також збільшує швидкість комунікації з клієнтом.

Однією, з найбільш необхідних функцій корпоративного порталу є функція навчання і розвитку співробітників. Адже для досягнення цілей компанії, дуже важливо швидко доносити необхідну інформацію про продукти, сервіси та умови компанії, а також швидко навчати нових співробітників необхідним навичкам, проводити навчання з продажу та роботи з клієнтами [9].

На корпоративному порталі можуть зберігатися дані з Корпоративної книги продажів, навчальні відео та онлайн-курси. Також за допомогою корпоративного порталу можна проводити атестацію співробітників.

За рахунок інструментів Корпоративного порталу, спрощується процес створення робочих і проектних груп. Якщо раніше необхідно було вручну розсилати запрошення співробітникам, в листах фіксувати всіх учасників, їх ролі і завдання, дати зустрічей і відповідальних по кожному питанню, то тепер співробітники об'єднуються в групи, в залежності від проектів або завдань, створених на порталі.

Всередині груп ведеться листування з обговоренням проекту або завдання, призначаються зустрічі, відбувається обмін файлами, фіксується результат роботи кожного з учасників. Робота групи стає прозорим процесом, з фіксацією кожного кроку учасників і наочним відображенням стадії проекту.

Збори також можуть бути проведені в корпоративному порталі, не обов'язково збирати всіх в переговорній кімнаті. Де б не знаходилися учасники проектної або робочої групи, вони можуть підключитися до зборів по відеотрансляції, а також ознайомитися з порядком, і в онлайн-режимі задавати питання, демонструвати документи, ставити завдання колегам. Вся інформація буде зафіксована в системі.

У календарі подій співробітників і всієї компанії, кожен з учасників бачить завантаження колег, найкращим чином планує спільні активності за завданнями групи, будь то зустріч з клієнтом, збори за підсумками місяця, або запуск нового продукту в продаж.

Швидку комунікацію, як всередині компанії, так і з клієнтами і партнерами, забезпечують такі інструменти корпоративного порталу, як Корпоративний месенджер і бізнес-чат.

Вже не потрібно усно просити і нагадувати щось колезі. Можна не переживати за те, що хтось не прочитає лист або пожаліється на наявність завдань в потрібний час. Функціонал корпоративного порталу дозволяє швидко і максимально конфіденційно донести завдання до виконавця. У календарі співробітника можна побачити, коли він зможе це завдання виконати, а в чаті вже уточнити, чи прийняв він завдання до виконання [9].

Всі співробітники мають можливість спілкуватися один з одним і з клієнтами, де б вони не знаходилися.

Важливо, що всі комунікації фіксуються, а також «підтягуються» до картки ліда, угоди або клієнта в разі, якщо завдання або коментар з ними пов'язаний. Історія листування поштою, дзвінки, вчинені за допомогою корпоративного порталу зберігаються для подальшої аналітики.

Всі корпоративні портали володіють базовим набором функцій.

Управління завданнями і проектами. Інструменти для постановки завдань, управління часом, чек-листи, фільтри, шаблони і конструктори документів.

Спільна робота з документами. Сервіси для створення, редагування та зберігання документів, налаштування спільного доступу. Легко і зручно ділитися документами з колегами і клієнтами, працюючи в онлайн-режимі без скачування.

Планування і облік робочого часу. Одночасно полегшує роботу і створення звітів і підвищує робочу дисципліну. Облік витраченого часу дозволяє оцінити витрати на певні типи завдань.

CRM для продажів і роботи з клієнтами. Частина порталів інтегруються з CRM, а також пропонує їх функції. Це дозволяє створювати бази клієнтів і партнерів. Фіксація та облік контактів дозволять підвищити ефективність роботи відділу продажів.

HR - управління персоналом. Платформи дозволяють легко уявити структуру компанії. Вся необхідна інформація про співробітників завжди під рукою.

Автоматизація бізнес-процесів. Виконання рутинних операцій можна віддати роботам, що знизить навантаження на персонал і підвищить ефективність роботи.

Серед основних завдань порталу, можна виділити наступні:

- підвищити якість управління і ефективність бізнесу. Корпоративні портали дозволяють оцінювати завантаженість співробітників і легко формувати звіти, щодо виконання завдань;
- поліпшити керованість і контроль. Щоб приймати обдумані рішення, підвищити якість управління і конкурентоспроможність.
- зекономити робочий час. Платформа дозволить швидко знайти і обробити необхідну інформацію, ефективно спілкуватися з колегами і швидше виконувати завдання;
- підвищити якість роботи співробітників. Робочі групи всередині порталів дозволяють співробітникам бути в курсі робочих процесів і термінів. Також це дає можливість зберігати накопичений досвід.
- оптимізувати процеси бізнесу. Швидкість передачі інформації та документообігу значно підвищиться, що спростить роботу.

1.2 Аналіз ефективності порталу

Запуск корпоративного порталу - тільки початок великої роботи. Щоб зрозуміти, наскільки ефективно працює портал, необхідно відстежувати реакції користувачів. Для кожної функції, є власні метрики.

Для аналізу зручно використовувати вбудовані або додаткові системи аналітики, наприклад платформи Business Intelligence, і інші системи, які можна використовувати для створення дашборда.

В Порталу є 2 великих категорії завдань: комунікації та сервіси.

Ефективність комунікацій можна відстежити, проаналізувавши дані елементи:

Відвідуваність. Кількість візитів і унікальних користувачів, відсоток активних користувачів.

Поведінка. Глибину перегляду і час на порталі, середня кількість з основних розділів, середній коефіцієнт корисності матеріалів, частку візитів і переглядів з мобільних пристроїв.

Створення контенту. Кількість створеного і відредагованого матеріалу, нові робочі групи та активність в них.

Залученість. Додавання до вибраного, лайки і коментарі, відсоток минулих опитування, участь в мотиваційних програмах, заповненість профілів співробітників, повідомлення з форми зворотного зв'язку[13].

Роботу сервісів і бізнес-процесів можна оцінити за такими чинниками:

- часу завантаження сторінок і пошуку відповіді на питання;
- коефіцієнту корисності матеріалу.
- часу, який необхідний для вирішення завдання виключно засобами порталу;
- відсотку форм, які не заповнюються на папері;
- скороченню персоналу, який підтримує процеси;
- активності в робочих чатах;
- відсотку проектів, які реалізуються онлайн;
- кількість відправлених та отриманих e-mail;
- кількість e-mail, дзвінків або квитків, пов'язаних з підтримкою персоналу (звернення до IT, HR, офіс-менеджеру);
- кількість особистих звернень, пов'язаних з підтримкою персоналу.
- час, необхідний для вирішення часто повторюваних завдань поза порталу.

В аналітиці, важливо виділяти сегменти. Можна розділити користувачів по географії, віком, відділу, посади або по розділах порталу: проекти, сервіси, база знань, медіа та ін [3].

Крім технічних характеристик, можна виміряти задоволеність користувачів від порталу, розділів і функцій. Можна опитати співробітників,

наскільки портал допомагає в роботі, наскільки зручно користуватися мобільною версією, які функції найбільше подобаються, а які не використовуються.

1.4 Користь від використання порталу

При використанні корпоративного порталу, виросте швидкість пошуку інформації, за рахунок того, що вона буде акумульована на одному ресурсі. Знизиться ризик витоку конфіденційної інформації. Прискориться адаптація нових співробітників. У співробітників з'явиться можливість одночасно і спільно працювати над документами, що знизить тривалість погоджень. Посилиться корпоративна культура - за рахунок зростання горизонтальних зв'язків в колективі. Результати роботи стануть прозорішими для всього колективу, за рахунок загальних обговорень. Можна накопичувати досвід роботи і вдосконалювати робочі процеси. Типові завдання можна вирішувати, використовуючи накопичену базу знань. Внутрішнє листування, зайнятість і присутність співробітників буде під контролем. Виросте якість і швидкість зворотного зв'язку з співробітниками. Підвищиться якість обслуговування клієнтів і ефективність роботи компанії [12].

Керівникам підрозділів і топ-менеджерам стане доступною можливість отримувати зворотний зв'язок від співробітників з будь-якого питання. Використовувати інструменти бізнес-аналітики - збір, попередню обробку та візуалізацію комерційних показників, фінансового стану, плинності кадрів, виконання планів і т.д.

ІТ-фахівцям:

- виводити зображення з камер спостереження;
- надавати єдиний доступ до всіх інформаційних систем компанії;
- використовувати адресну книгу з швидким пошуком по ПІБ, посади, телефону;

- відображати і аналізувати статистику звернень в технічну підтримку;
- використовувати зручний механізм для обробки заявок - принцип одного вікна, інтеграцію з helpdesk-системами, статистичну обробку результатів та інформування користувачів про зміну статусу заявки.

Менеджерам з комунікацій:

- інформувати співробітників про заходи, в яких бере участь компанія, про нові послуги та продукти, орієнтації всіх працівників на кінцевий результат;
- оцінювати настрої всередині компанії, збирати і обробляти думки співробітників про умови, кадрові перестановки і т.п;
- проводити внутрішні маркетингові кампанії - опитування працівників про нові товари і послуги, які виходять на ринок;
- збирати інформацію про захоплення співробітників, мотивації і внутрішніх цінностях.

HR-фахівцям:

- проводити опитування та анкетування співробітників, робити автоматичну розсилку, обраній групі користувачів;
- створювати архів організаційно-правових документів - посадових інструкцій, положень про відділи, організаційних діаграм;
- інформувати і адаптувати нових співробітників - знайомити з компанією, зразками заяв, до кого звертатися та інше;
- відслідковувати динаміку чисельності персоналу - візуалізувати прийнятих і звільнених співробітників в розрізі професії, віку, відділу;
- основна вигода від впровадження інтранет-порталу - організаційна. Люди будуть працювати швидше і продуктивніше, знизиться кількість паперової тяганини. У компаніях, де працюють тисячі і

десятки тисяч співробітників, навіть кілька відсотків економії часу значно знизять витрати.

Однак, слід врахувати, що корпоративні портали і інтернет - це всього лише інструменти [15]. І щоб отримувати від них ефект, ними потрібно грамотно користуватися: впроваджувати, налаштовувати, відстежувати ефективність і допрацьовувати. Системний підхід і автоматизація бізнесу дозволять з окупити інвестиції.

2 ПРОЕКТУВАННЯ ТА АРХІТЕКТУРА ПРОЕКТУ

2.1 Основна будова програми

2.1.1 Розробка архітектури

Для розробки програми, вибрано класичну трирівнева архітектуру.

Існує безліч різних видів і типів архітектур, які успішно застосовуються. Однією з найбільш використовуваних є класична трирівнева система, яка передбачає поділ додатка на три рівні.

Тут відразу треба сказати, що для багаторівневої архітектури часто позначають два, не зовсім пов'язаних поняття: n-layer і n-tier. I layer, і tier, як правило, позначаються словом "рівень", іноді по відношенню до "layer" ще вживається слово "шар". Однак в обох випадках рівні будуть різного порядку [1].

Tier, представляє фізичний рівень. Тобто, якщо ми говоримо про трирівневу архітектуру, то n-tier додаток міг бути розділений на такі рівні: сервер бази даних, веб-додаток на веб-сервері і браузер користувача. Тобто кожен рівень представляв би особливий окремий фізичний процес, навіть, якщо б і сервер баз даних, і веб-сервер, і браузер користувача знаходилися б на одному комп'ютері. Якби в якості клієнта альтернативно використовувався мобільний додаток, то це був би ще один фізичний рівень.

Layer представляє логічний рівень. Тобто, у нас може бути рівень доступу до даних, рівень бізнес-логіки, рівень уявлення, рівень сервісів і так далі. При цьому, логічні рівні не збігаються з фізичними. Так, звичайно зовнішній рівень в додатку ASP.NET містить і контролери, які обробляють введення, і уявлення, які відображаються в веб-браузері, тобто розділяється на два фізичних рівня.

В даному випадку ми будемо говорити саме про логічних рівнях, тобто про n-layer архітектурі [2].

Класична трирівнева система складається з наступних рівнів:

Presentation layer (рівень представлення): це той рівень, з яким безпосередньо взаємодіє користувач. Цей рівень включає компоненти для користувача інтерфейсу, механізм отримання введення від користувача. Стосовно ASP.NET MVC, на даному рівні розташовані уявлення, і всі ті компоненти, який складають призначений для користувача інтерфейс (стилі, статичні сторінки html, javascript), а також моделі уявлень, контролери, об'єкти контексту запиту.

Business layer (рівень бізнес-логіки): містить набір компонентів, які відповідають за обробку отриманих від рівня уявлень даних, реалізує всю необхідну логіку додатка, все обчислення, взаємодіє з базою даних і передає рівнем подання результат обробки.

Data Access layer (рівень доступу до даних): зберігає моделі, що описують використовувані суті, також тут розміщуються специфічні класи для роботи з різними технологіями доступу до даних, наприклад, клас контексту даних Entity Framework. Тут також зберігаються репозиторії, через які рівень бізнес-логіки взаємодіє з базою даних.

При цьому треба зазначити, що крайні рівні не можуть взаємодіяти між собою, тобто рівень представлення (стосовно ASP.NET MVC, контролери) не можуть безпосередньо звертатися до бази даних і навіть до рівня доступу до даних, а тільки через рівень бізнес-логіки .

Рівень доступу до даних не залежить від інших рівнів, рівень бізнес-логіки залежить від рівня доступу до даних, а рівень представлення - від рівня бізнес-логіки [1].

Компоненти, як правило, повинні бути слабо зв'язані, тому невід'ємною ланкою багаторівневих додатків є впровадження залежностей. У той же час, невірно думати, що кожному рівню, обов'язково повинен відповідати окремий проект. При необхідності, можна роздрібнити один рівень на кілька проектів, головне, щоб його функціонал представляв єдину логічну ланку (рис. 2.1).

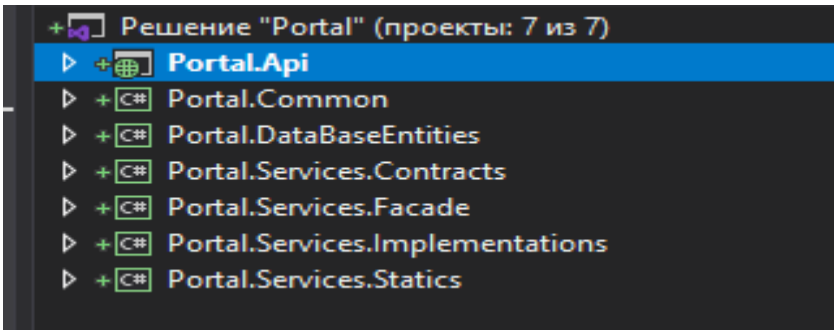


Рисунок 2.1 – Будова збірки

2.1.2 Вибір СУБД

Для розробки програми, даного типу, найбільше підходить MS SQL Server. MS SQL Server. Microsoft SQL Server представляє собою інтегрований пакет програм, призначений для:

- створення, розгортання та управління промисловими програмами, які є безпечними, масштабованими та надійними;
- збільшення продуктивності інформаційних технологій за рахунок зменшення складності побудови, розгортання та управління програмами по роботі з базами даних;
- розділення даних між платформами, програмами та пристроями для полегшення поєднання внутрішніх та зовнішніх систем.

Платформа даних SQL Server включає наступні інструменти:

Реляційна база даних: безпечне, надійне, масштабоване, легкодоступне ядро з покращеною продуктивністю і підтримкою структурованих та неструктурованих даних.

Replication Services: реплікація даних для розподілених та мобільних програм обробки даних, висока доступність систем, масштабований паралелізм із вторинними сховищами даних, та інтеграція з різнорідними системами, включаючи існуючі бази даних Oracle.

Notification Services: розвинуті можливості повідомлень, для розробки і впровадження масштабованих програм, що можуть доставляти персоналізовані та своєчасні обновлення інформації в великій кількості, як підключених, так і мобільних пристроїв.

Integration Services: засоби отримання, перетворення та завантаження інформації для сховищ даних та інтеграції даних в масштабі підприємства.

Analysis Services: аналітична обробка інформації в реальному часі та технології інтелектуального аналізу даних (Data Mining), які використовуються для швидкого та складного аналізу великих і змішаних наборів даних.

Reporting Services: засіб для створення, управління і доставки як традиційних паперових звітів, так і інтерактивних, що базуються на Web-технології [10].

Інструменти управління: SQL Server включає засоби управління та настроювання баз даних. Стандартні протоколи доступу до даних суттєво зменшують час, необхідний для інтеграції даних SQL Server з існуючими системами. Крім того є вбудована підтримка Web-служб для забезпечення взаємодії з іншими системами та платформами.

Інструменти розробки: SQL Server надає інтегровані інструменти розробки для ядра бази даних, отримання, трансформації та завантаження даних, OLAP та звітності, які тісно інтегровані з MS Visual Studio для надання наскрізних можливостей розробки інформаційних систем. Кожна головна підсистема SQL Server поставляється із своєю власною об’єктною моделлю та набором API для розширення системи в довільному напрямку, який є унікальним для бізнесу компанії.

Платформа даних SQL Server надає наступні переваги :

Використання активів даних. SQL Server дозволяє користувачам отримати більше вигоди від їх даних завдяки використанню таких вбудованих функцій як звітність, аналіз та отримання інформації.

Збільшення продуктивності. Завдяки широким можливостям інтелектуальних ресурсів підприємства та інтеграції з популярними програмами, такими як MS Office, SQL Server надає працівникам інформаційної сфери підприємства, важливу та своєчасну інформацію, пристосовану для їх конкретних потреб.

Зменшення складності інформаційної технології. SQL Server спрощує розробку, впровадження та управління галузями промисловості та аналітичними програмами, надаючи програмістам гнучке середовище розробки та інтегровані, автоматизовані інструменти управління адміністраторам баз даних.

Зниження загальної вартості володіння. Інтегрований підхід та фокусування на простоті використання і впровадження приводить до зменшення витрат на реалізацію і підтримку, що сприяє швидкому поверненню інвестицій в бази даних [10].

2.1.3 Вибір ORM системи

Для розробки програми, обрано ORM систему Entity Framework. При її використанні, код стає більш лаконічнішим і полегшується подальша підтримка при розростанні коду. Також вибрано Code first підхід для роботи з базою даних.

ORM або Object-relational mapping - це технологія програмування, яка дозволяє перетворювати несумісні типи моделей в ООП, зокрема, між сховищем даних і об'єктами програмування. ORM використовується для спрощення процесу збереження об'єктів в реляційну базу даних і їх вилучення, при цьому ORM, сама піклується про перетворення даних, між двома несумісними станами. Більшість ORM-інструментів, значною мірою покладаються на метадані бази даних і об'єктів, так що об'єктам нічого не потрібно знати про структуру бази даних, а базі даних - нічого про те, як дані організовані в додатку. ORM забезпечує повне розділення завдань в добре спроектованих додатках, при якому

і база даних, і додаток можуть працювати з даними кожен у своїй вихідній формі [6].

Entity Framework, являє собою спеціальну об'єктно-орієнтовану технологію, на базі фреймворка .NET для роботи з даними. Якщо традиційні засоби ADO.NET дозволяють створювати підключення, команди та інші об'єкти для взаємодії з базами даних, то Entity Framework, являє собою більш високий рівень абстракції, який дозволяє абстрагуватися від самої бази даних і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні, оперується таблицями, індексами, первинними і зовнішніми ключами, то на концептуальному рівні, який нам пропонує Entity Framework, виконується робота з об'єктами.

Центральної концепцією Entity Framework, є поняття сутності, або entity. Сутність представляє набір даних, асоційованих з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх наборами.

Будь-яка сутність, як і будь-який об'єкт з реального світу, має низку властивостей. Наприклад, якщо сутність описує людини, то ми можемо виділити такі властивості, як ім'я, прізвище, зріст, вік, вага. Властивості необов'язково представляють прості дані типу int, а й можуть представляти більш комплексні структури даних. І у кожної сутності може бути одна або кілька властивостей, які будуть відрізняти цю сутність від інших і будуть унікально визначати цю сутність. Подібні властивості називають ключами [19].

При цьому суті можуть бути пов'язані асоціативної зв'язком один-до-багатьох, один-до-одного і багато-до-багатьох, подібно до того, як в реальній базі даних відбувається зв'язок через зовнішні ключі.

Відмінною рисою Entity Framework, є використання запитів LINQ для вибірки даних з БД. За допомогою LINQ ми можемо не тільки отримувати певні рядки, що зберігають об'єкти, з бд, а й отримувати об'єкти, пов'язані різними асоціативними зв'язками.

Іншим ключовим поняттям, є Entity Data Model. Ця модель зіставляє класи сутностей з реальними таблицями в БД. Entity Data Model складається з трьох рівнів: концептуального, рівень сховища і рівень зіставлення.

На концептуальному рівні відбувається визначення класів сутностей, які використовуються в додатку.

Рівень сховища визначає таблиці, стовпці, відносини між таблицями і типи даних, з якими порівнюється використовувана база даних.

Рівень зіставлення служить посередником між попередніми двома, визначаючи зіставлення між властивостями класу суті і стовпцями таблиць.

Таким чином, ми можемо через класи, визначені у додатку, взаємодіяти з таблицями з бази даних.

Entity Framework передбачає три можливі способи взаємодії з базою даних:

- database first: Entity Framework створює набір класів, які відображають модель конкретної бази даних;
- model first: спочатку розробник створює модель бази даних, по якій потім Entity Framework створює реальну базу даних на сервері;
- code first: розробник створює клас моделі даних, які будуть зберігатися в бд, а потім Entity Framework за цією моделлю генерує базу даних і її таблиці.

Entity Framework має контекст, який управляє взаємодією між цими класами та вашою базою даних. Контекст не специфічний для Code First, це особливість Entity Framework.

Code first додає конструктор моделі, який перевіряє класи, якими керує контекст, а потім використовує набір правил або конвенцій, щоб визначити, як ці класи та взаємозв'язки описують модель, і як ця модель повинна відображатися у базі даних.

Code First також має можливість використовувати модель для створення бази даних, якщо це необхідно.

Він також може оновити базу даних, якщо модель зміниться, використовуючи функцію під назвою Code First Migrations.

2.1.4 Проектування класів та інтерфейсів

Для побудови залежностей між класами, буде використано IoC-контейнер Autofac. Він допомагає побудувати взаємодію з класом через інтерфейси, які він реалізує. Він розширює базові можливості ASP.NET, допомагаючи задавати час життя компонентів, керувати класами, в момент виконання програми, вирішуючи, яку саме реалізацію потрібно підставити [18]. Для цього, в ньому побудована внутрішня реалізація патерну Фабричный метод (Factory Method).

Фабричний метод (Factory Method) - це патерн, який визначає інтерфейс для створення об'єктів деякого класу, але безпосереднє рішення про те, об'єкт якого класу створювати, відбувається в підкласах. Тобто, патерн передбачає, що базовий клас делегує створення об'єктів класам-спадкоємцям [5].

Даний патерн необхідно застосовувати в наступних випадках:

- коли заздалегідь невідомо, об'єкти яких типів необхідно створювати;
- коли система повинна бути незалежною від процесу створення нових об'єктів і розширюється, тобто в неї можна легко вводити нові класи, об'єкти яких система повинна створювати;
- коли створення нових об'єктів необхідно делегувати з базового класу, класам спадкоємцям.

На мові UML патерн можна описати таким чином (рис. 2.2):

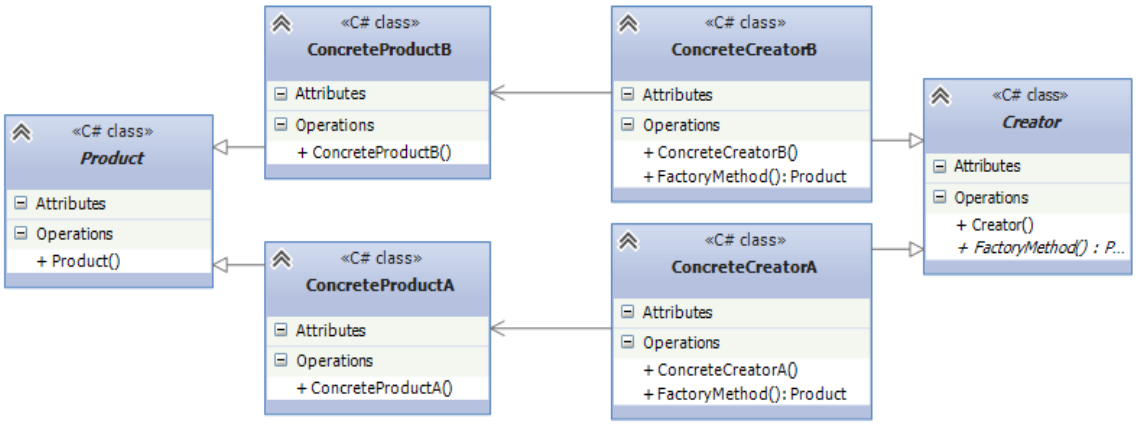


Рисунок 2.2 – UML схема роботи патерну “Фабричний метод”

Запуск всіх залежностей відбуватиметься при старті програми, і для них, буде створено окрему бібліотеку, в якій буде застосовано патерн Фасад, для доступу до всіх класів, через нього. Так, в подальшому програма зможе сильно розростатись, тому можна буде зручно керувати всіма взаємодіяти, та конфігурувати їх в потрібний вигляд.

Фасад (Facade) являє шаблон проектування, який дозволяє приховати складність системи за допомогою надання спрощеного інтерфейсу для взаємодії з нею.

Даний патерн необхідно застосовувати в наступних випадках [5]:

- коли є складна система, і необхідно спростити з нею роботу. Фасад дозволить визначити одну точку взаємодії між клієнтом і системою;
- коли треба зменшити кількість залежностей між клієнтом і складною системою. Фасадні об'єкти дозволяють відокремити, ізолювати компоненти системи від клієнта, розвивати і працювати з ними незалежно;
- коли потрібно визначити підсистеми компонентів в складній системі. Створення фасадів для компонентів кожної окремої підсистеми, дозволить спростити взаємодію між ними і підвищити їх незалежність один від одного.

В UML загальну схему фасаду можна представити таким чином (рис. 2.3):

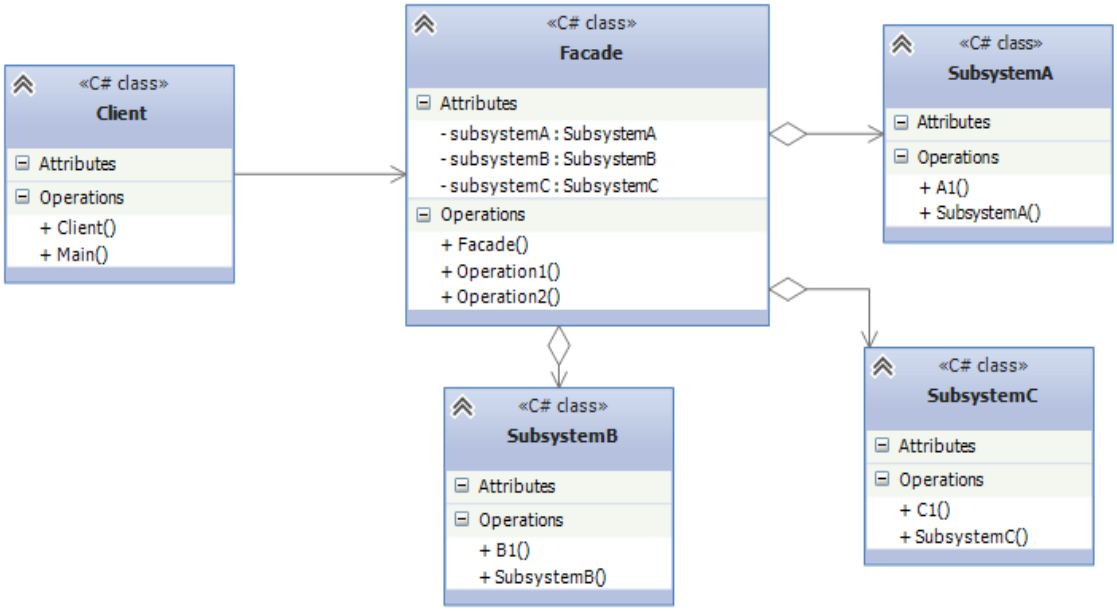


Рисунок 2.3 – UML схема роботи патерну “Фасад”

Основний функціонал програми міститиме необхідні CRUD операції з наступними сутностями (таб. 2.1):

Таблиця 2.1 – Опис основного функціоналу

Calendar	View Calendar, Holidays, Events
	Add / Edit / Remove Events and Holidays
People	View / Filter & Search People / open User profiles
	Edit / disable user profiles
Projects	View / Filter & Search / Open Projects
	Create / Edit / Remove Projects
Departments	View / Open Departments
	Create / Edit / Remove Departments
Timelog	View Timelog and My report. Log time on various projects and other activities.
	View/Manage Timelog requests View Project report View User report Log for user
Vacations	View My requests, Vacation Information, and Add / Edit / Cancel Request
	View For Approval, Manage employees' requests.

2.2 Timelog

2.2.1 Загальний опис

Логування часу, є основним функціоналом для контролю робочого процесу. Після додавання користувача до певної активності, в репорті, буде можливість логування часу на нього (таб. 2.2).

Таблиця 2.2 – Можливості використання timelog відносно ролей

Timelog	View Timelog. Log time on various projects, pre-sales, departments and other activities.	All users
	View/Manage Timelog requests View Reports View User report View General Report Log time for user	PM, AM

Для логування будуть доступні 3 типи логованих годин (звичайні, понаднормові, години очікування). При логуванні часу, потрібно основну увагу приділити понаднормовим годинам. Для цього використовуватимуться квитки підтвердження. При логуванні понаднормових годин, для одної людини, яка є адміністратором, буде створюватись квиток підтвердження, і залоговані години не будуть враховуватись, до моменту підтвердження (рис. 2.4). Для звичайних користувачів, логування буде доступне за сьогоднішній та вчорашній дні, для AM-ів дозволяється логування в будь який час а для PM-ів, лише за теперішній місяць. Так як репорт корстувачу приходитиме за ввесь місяць, потрібно врахувати можливі хибні варіанти логування часу:

- логування людиною, яка вже не знаходиться в команді;
- логування на людину, якой в те число, ще не було в команді, або була вже видалена з неї;
- дана активність, ще не почалась або вже завершилась.

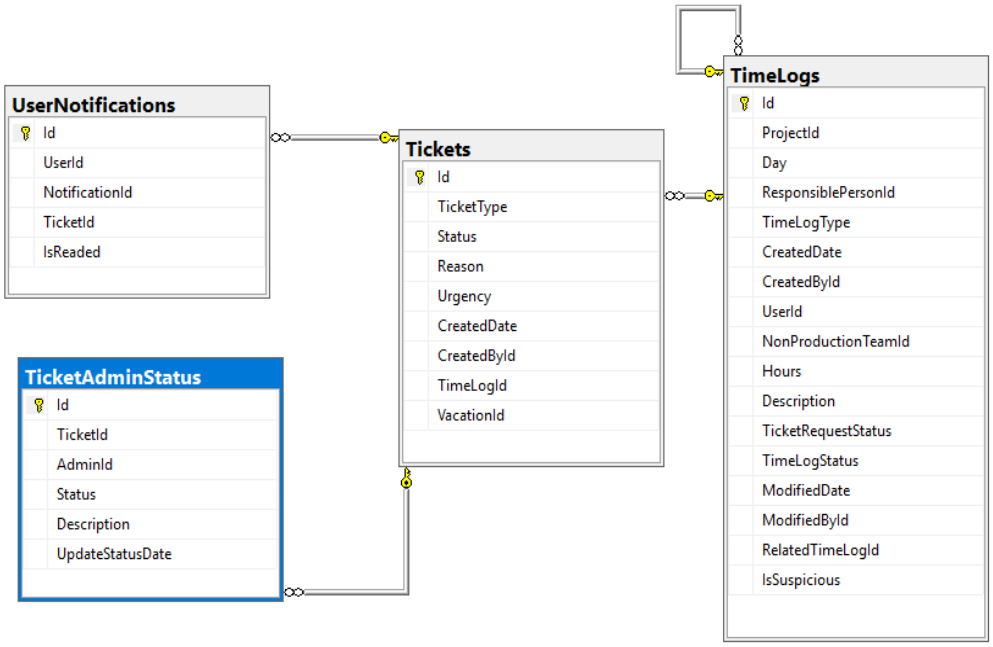


Рисунок 2.4 – Схема основних зв’язків з timelog

Для логування відпрацьованих годин, репорти будуть розділені на два типи. Перший репорт, буде по користувачу, а другий, по проектах. Для розробки вихідної моделі, спочатку, потрібно вирішити, якою формою його планується відображати. Одже, для того щоб звичайні користувачі могли логувати години, PM-и могли керувати проектом та внутрішньою роботою команди, а бухгалтери вели звітність, планується розподіл на 5 типів репортів. Орієнтуюсь розділити їх на 5 вкладок,

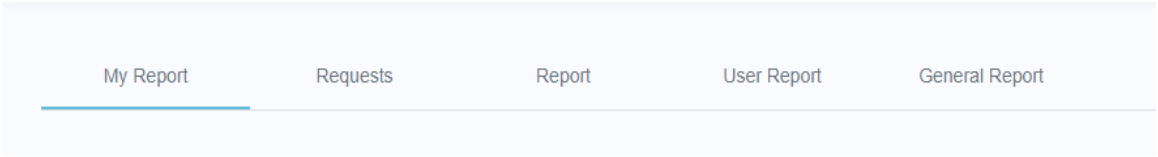


Рисунок 2.5 – Приклад поділу функціоналу

для кожної, надавати доступ, в залежності від того, чи є в користувача потрібні клейми (рис. 2.5).

Для доступу до всіх вкладок, небхідною буде клейма «Timelog management». Для доступу до “Requests”, також, необхідно мати клейму “Tickets

management”, а для “General Report” необхідно мати “Company Staff Report”. Для репроту по всіх проектах, необхідною буде “Company Staff Report”.

Логування часу буде доступне для наступних категорій:

- Projects: Технічні персонал. Люди, які беруть участь у проектах: PM, QA, Developers, Designers тощо;
- Pre-Sales: Технічний персонал. Люди, які беруть безпосередню участь у передпродажній діяльності: PM, дизайнер та розробник;
- Departments: Оперативний персонал. Люди, які в основному виконують непроектну діяльність: HR, Рекрутери, AM, Engagers, Marketers тощо. Технічний персонал також може зареєструвати час у відділах, якщо вони задіяні у заходах, пов'язаних із їх відділом;
- Other: Технічний та експлуатаційний персонал. Усі працівники компанії, можуть зареєструвати час для цієї категорії часових журналів.

Для ексорту в Excel документ всіх репортів, буде дотримано наступні вимоги:

- застосування формул;
- шрифт Verdana;
- колір для вихідних, повинен бути “#EDEDED”;
- колір для оплачуваних відпусток, повинен бути “#D3D4EB”, а для неоплачуваних, повинен бути “# E7DDEC”;
- колір для свята, повинен бути “# F9F5E2”;
- нульові години, зареєстровані - нічого не відображається, не 0;
- якщо під час відпустки, були залоговані години - відображати в клітинку лише ці години;
- усього (по горизонталі) - сума відображення стовпця (план включений);
- понаднормові години, стосуються лише конкретного проекту;

- в стовпці відпусток, відображати лише оплачувані відпустки та години. Якщо користувач, має неоплачувані відпустки, не потрібно змінювати стовпчик плану;
- назва файлу має бути: "Тип_Репорту _ Місяць";
- назва аркуша повинна бути Місяць;
- excel має бути адаптований до аркушів Google.

2.2.2 Репорт по користувачу

Вкладка “My Report”, буде доступна для всіх користувачів, і використовуватиметься для отримання репорту по користувачу, який на даний момент знаходиться в системі. Також, потрібно щоб користувач міг вибрати місяць, чи декілька, за які він хоче отримати репорт. Тому, в кожній частині потрібно розмістити календар такого типу (рис. 2.6):

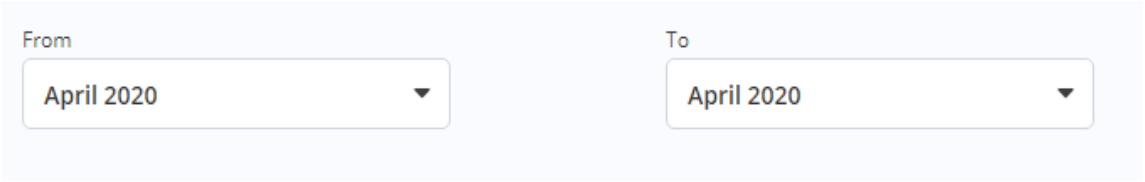


Рисунок 2.6 – Приклад поля вибору дат

Також, для кожного репорту, потрібно розробити експорт в Excel. Це допоможе бухгалтерії вести звітність, а користувачу зберігати репорти з годинами, та розпоряджатись ними по потребі (рис. 2.7).

Рисунок 2.7 – Приклад кнопки експорту репорту

Репорт по користувачу призначений, зокрема, для внутрішньої роботи в компанії, а не для роботи з клієнтами. В ньому повинні бути всі активності, на яких є користувач. Вони розділені по типу, до кожного типу вказуються підсумки. Також в репорті містяться план, позначки про відпустки користувача та свята (рис. 2.8).

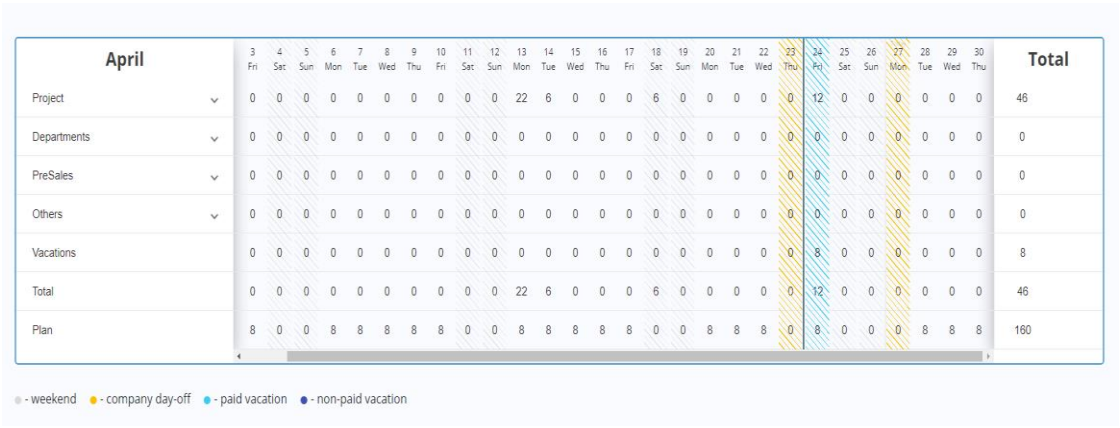


Рисунок 2.8 – Приклад репорту

Для кожного проекту, чи інших активностей надсилається звіт по робочих годинах, розділених по їх типу (рис. 2.9).

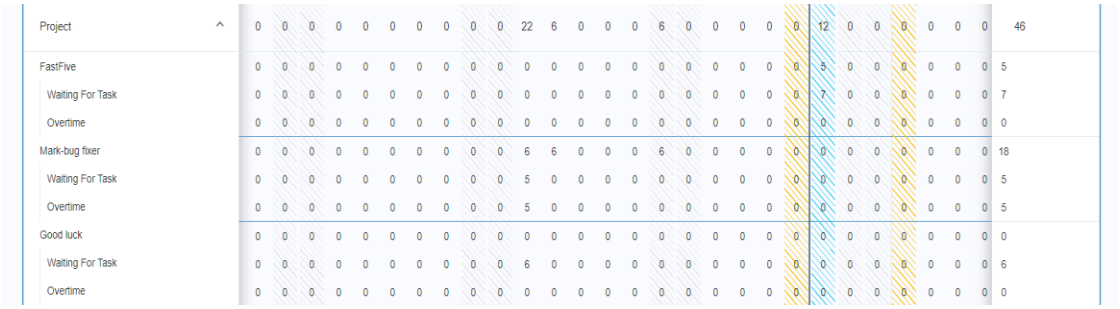


Рисунок 2.9 – Приклад поділу на робочі години

При нажиманні на певну годину в дні, виводиться форма створення чи оновлення існуючих годин. Форма повинна валідуватись по символах, та по максимальній кількості годин. Створення понаднормових годин, повинно бути недоступне, якщо не створювались звичайні години. Форма буде однакова для всіх репортів, важливою буде лише привязка до дня та проекту (рис. 2.10).

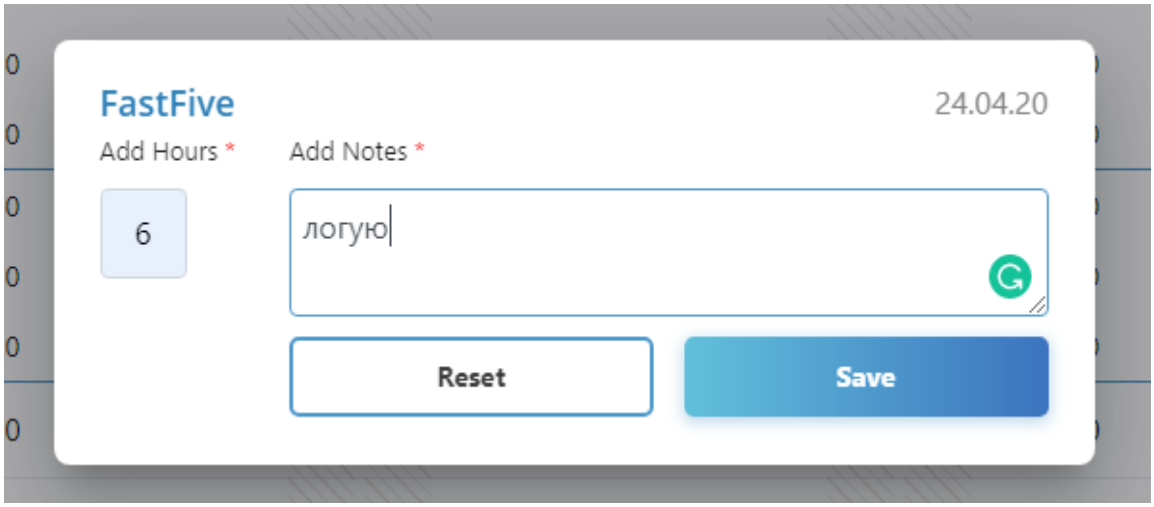


Рисунок 2.10 – Приклад форми логування часу

2.2.3 Репорт по проектах

Репорт по проеках, потрібен як для внутрішньої роботи, так і для роботи з клієнтими. Він допоможе відслідковувати за робочими годинами працівників та описами їхньої роботи на кожен день. До проектів буде прикліплений наступний функціонал (рис. 2.11):

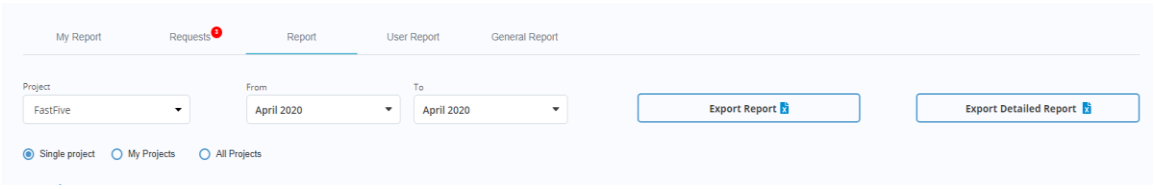


Рисунок 2.11 – Приклад спільного функціоналу в репортах

При зміні радіокнопки, буде доступний інший репорт, а поле вибору проекту, буде доступне лише для радіонопки “single project”. В полі вибірки проекту, знаходитимуться лише ті проекти, на яких даний користувач,

знаходиться на керуючій посаді. В репорті будуть користувачі, які були в команді, в період вибірки, навіть якщо вони зараз видалені. На всіх вкладках, буде мінятись сама структура вивидення, але форма репорту для всіх вкладок буде одинакова (рис. 2.12).

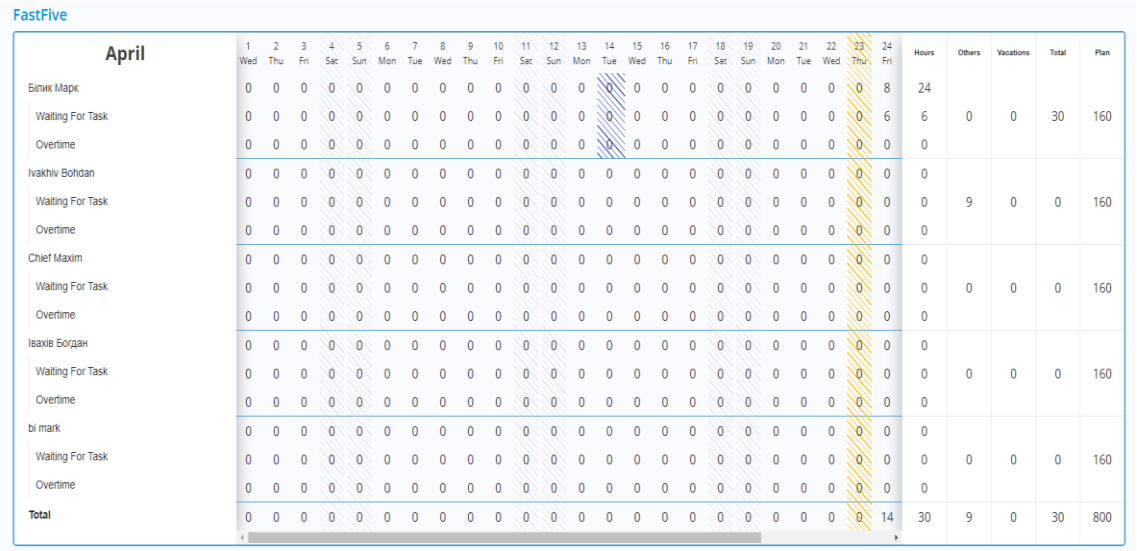


Рисунок 2.12 – Приклад репорту по проекту

В “My Projects” виводитимуться всі проекти, на яких даний користувач є на керівній посаді.

В “All Projects” виводитимуться всі проекти, поділені по всім AM-ах.

Для надсилання звітності клієнтам, планується розробка окремого репорту, який буде доступний лише в вигляді Excel документу. В ньому відображатиметься детальна звітність по кожному користувачеві, з відображенням опису роботи до залогованих годин. Листи в документі, будуть розділені по місяцях, а крім детального опису кожного користувача, в кінці документу міститиметься підсумок по всіх.

2.2.4 Загальний репорт

В Excel репорті загального проекту, повинні бути дотримані наступні вимоги:

- шрифт Verdana;
- стовпці повинні бути в наступній послідовності: Total – загальний підсумок по інших стовпцях, Except, Name, % from Plan, Comments;
- % from Plan – повинно містити відношення між загальним часом та планом;
- Vacation – відображення лише оплачуваних, підтверджених годин.
- % from plan = total/plan*100%;
- Difference = total-plan, якщо значення позитивне, то виділити зеленим кольором, в іншому випадку, червоним;
- Comments – поле необхідно заповнити, лише в випадку, якщо є коментар;
- документ повинен бути адаптованим до Google sheets;
- назва файлу, повинна бути наступного вигляду: ‘All Employees Report’_Month_’Portal’;
- назва листа повинна бути місяцем репорту.

Таблиця 2.3 – Заголовок загального репорту

Name (A-Z)	Project Hours	Project Overtime	Pre-Sales	Departments

Продовження таблиці 2.3

Others				
Interview (HR&Recruitment	Mentorship	Lectures	Waiting for task	Interview (Sales)

Кінець таблиці 2.3

Vacation	Total	Plan	% from Plan	Difference	Comments

Для клієнтської частини, буде розроблено такий самий репорт по будові, що і в Excel репорті (таб. 2.3).

2.3 Відпустки

Планується розробка функціоналу для керування відпустками користувача. Відпустки необхідно поділити на 3 типи: оплачувана, не оплачувана, та за додаткові дні. Кожному користувачеві будуть нараховуватись дні відпусток, по певній формулі нарахування. Для того, щоб зробити формулу більш гнучкою, в базі будуть зберігатись дані, які застосовуватимуться в формулі нарахування. Для нарахування, буде застосовано наступний алгоритм (рис. 2.13):

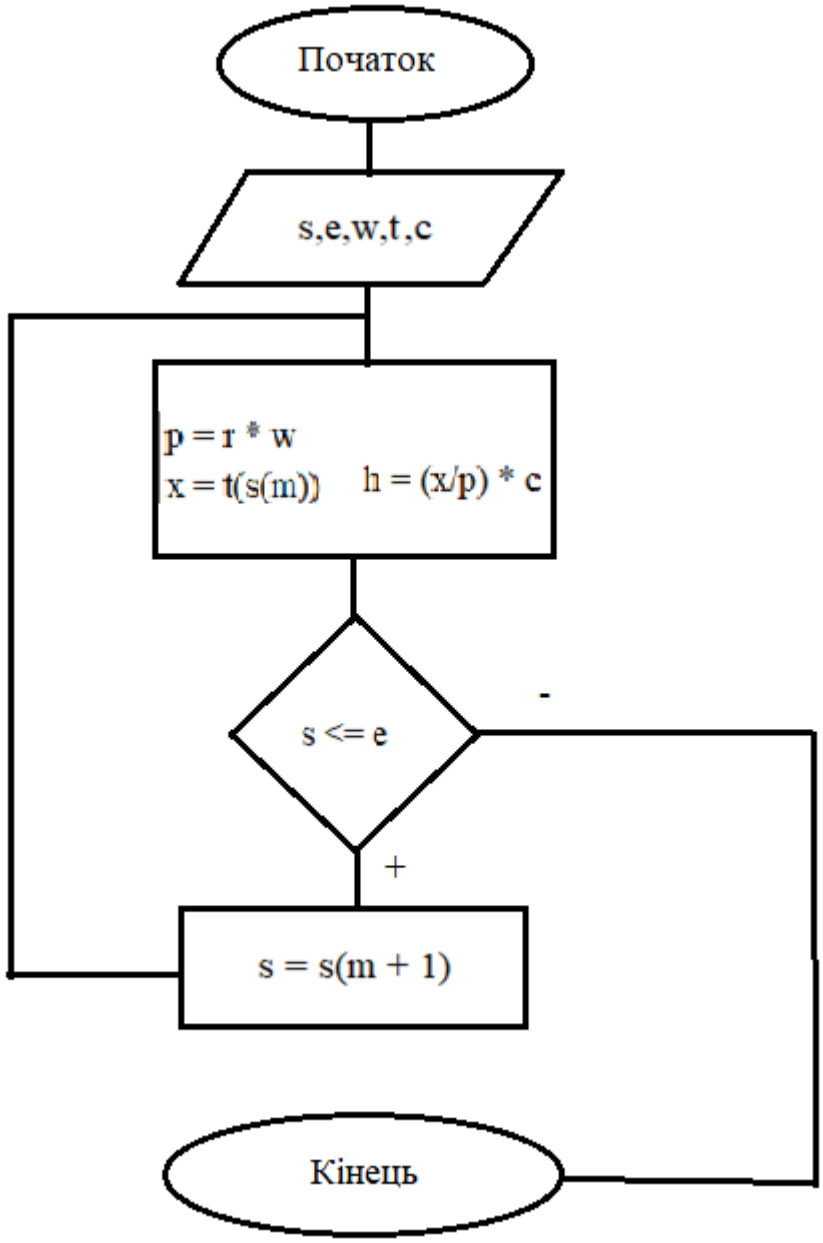


Рисунок 2.13 – Блок-схема алгоритму нарахування відпусток

s – стартова дата нарахування;

e – кінцева дата нарахування;

w – кількість годин в робочому дні;

t – залоговані робочі години;

c – коефіцієнт нарахування за 100% виконаного плану.

Це будуть стартові дані, необхідні для вираховування днів відпустки в місяць. Стартову дату, кількість робочих годин в день та коефіцієнт нарахування, будуть надходити з таблиці бази даних. Це дозволить зробити нарахування більш

гнучким, та помінявши значення в базі, змінити формулу. Кількість залогованих користувачем годин, будуть надходити з бази даних, і будуть обиратись лише підтверджені. Так як робочий план буде різним для кожного місяця, то вираховувати потрібно помісячно, так як відсоток виконання для кожного місяця, буде мати різну вагу. Одже, потрібно знайти план (p), помноживши кількість робочих днів (r), на кількість робочих годин в день (w). Далі знайти суму всіх залогованих користувачем годин ха місяця (x), роблячи вибірку з усіх годин (t), лише ті, які були залоговані на місяць стартової дати нарахування (s(m)). Після чого, вирахувати, яку частину від плану, виконав користувач, і в залежності від цього, надати певний відсоток днів відпустки (h), відносно коефіцієнту (c). Далі збільшити стартову дату нарахування (s) на 1 місяць, і повторювати нарахування формули до того часу, допоки стартова дата, не стане більшою за кінцеву дату (e). Також, в користувача буде можливість, щоб дні відпусток, якщо проходить робочий рік, переносились в попередній. Тобто, якщо користувач, має певний борг або не використані дні, то вони будуть занесені в дні за попередній рік і будуть впливати на теперішній рік. Для того, щоб зробити даний функціонал, в таблиці користувача, буде константа за теперішній і попередній робочий рік, кількість використаних днів в даному році, та поля з кількість днів відпусток за теперішній та попередній роки. Одже, з тієї формули, будуть нараховуватись робочі дні та відніматись використані, і в залежності від результатів, заносити дані в відповідні поля. Даний алгоритма занесення даних працюватиме наступним чином (рис. 2.14):

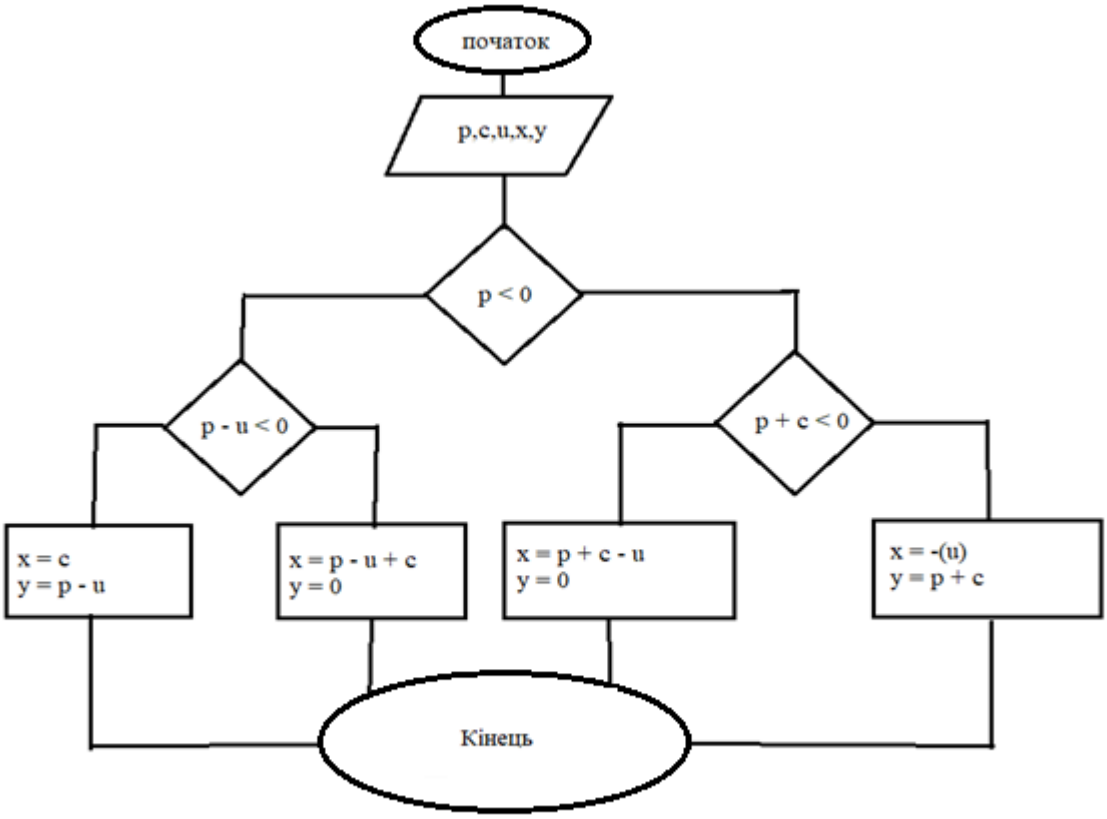


Рисунок 2.14 – Блок-схема алгоритму розподілу відпусток

p – константа днів з попереднього року;

c – кількість днів для нарахування;

u – кількість використаних днів відпустки в цьому році;

x – кількість днів в цьому році;

y – кількість днів в попердньому році.

З попередньо описаної формули нарахування, отримано вхідне значення днів, для нарахування (c). Для того щоб, якщо в користувача є борг з попернього року, значення спочатку вносились в попередній рік, знімаючи заборгованість, а лише потім нараховувалась у цей, є необхідним алгоритм занесення по умовах. В ньому, якщо константа днів з попереднього року (p), менша за нуль, це означає, що необхідно розділити нараховані дні (c), для погашення боргу та нарахування в теперішній рік. Для цього, необхідно здійснити ще одну перевірку, а саме, якщо до боргу додати нараховані дні, і в підсумку вийде менше нуля, значить в

теперішніьому році (х), будуть використані дні, а за минулий рік (у), буде підсумок. В іншому випадку, в теперішній рік, внесеться сума константи за поперній рік та нараховані дні, мінус використані дні. Якщо ж в користувача додатня кількість днів за попередній рік, поревіряю чи використаних днів відпутки більше, ніж в попередньому, і якщо це так, в попередній вношу 0 днів, а в теперішній, кількість днів за попередній рік, плюс нараховані дні, та мінус використані користувачем. В іншому випадку, в теперішній рік вношу нараховані дні, а за минулий рік, константу, мінус використані користувачем дні.

Для контролю відпусток користувачів адміністраторами, будуть створюватись квитки підтвердження та сповіщення для адміністраторів. Для цього, при створенні відпустки користувачем, будуть вибиратись користувачі, які є на керуючій посаді, для користувача, який запросив відпустку. Якщо такі користувачі є, буде створений основний квиток підтвердження, а для кожного користувача, який відповідальний за підтвердження, окремий квиток. Також, для всіх них, буде створене сповіщення, з текстом, в залежності від операції над відпусткою. Кожен користувач може міняти статус свого квитка, а при зміні його, будуть перевірятися статуси квитків інших адміністраторів, і в залежності від них, змінювати статус основного. В залежності від основного квитка, проводитимуться висновки, щодо відпустки (підтверджено, заборонено або ще розглядається) (рис. 2.15).

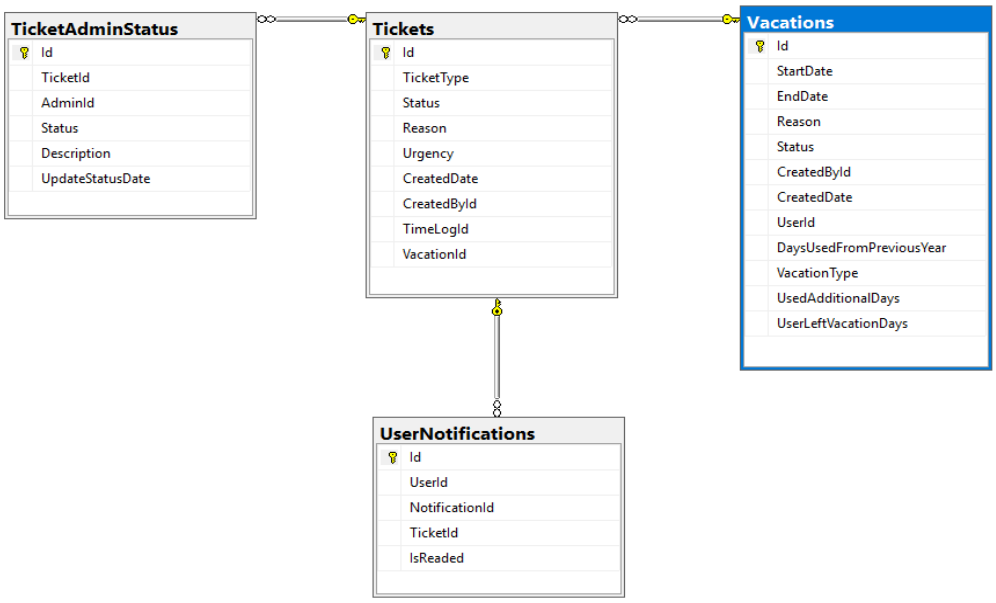


Рисунок 2.15 – Схема основних зв’язків з відпустками

Також формула нарахування буде викликатись кожного дня, для всіх користувачів, за допомогою організації фонових завдань.

2.4 Користувачі

Звичайно, в центрі всіх звязків знаходиться користувач. Для керування користувачами, буде використано платформу Identity. ASP.NET Identity, представляє вбудовану в ASP.NET, систему аутентифікації і авторизації. Дана система, дозволяє користувачам створювати облікові записи, аутентифікуватись, курувати обліковими записами або використовувати для входу на сайт облікові записи зовнішніх провайдерів, таких як Facebook, Google, Microsoft, Twitter та інших. Так як вхід в програму здійснюватиметься через провайдер Google, ця платформа допоможе створити користувача, з даними, наданими провайдером. При першому вході користувача, він буде вводити основні дані про себе. На вибір йому будуть доступні занесені в базу спеціальності, навички, та інше. В

залежності від вибраної коритсувачем спеціальності, йому добавлятимуться права доступу до відповідної сфери (таб. 2.4).

Таблиця 2.4 – Розподіл ролей

Role	Position
User	UI/UX Designer Front-end Developer .NET Developer iOS Developer Android Developer QA Automation QA Manual Lead Generation Manager System Administrator etc.
Admin	HR - Human Resources Manager
	PM - Project Manager
	AM - Account Manager
	EM - Engagement Manager
	TL - Team leader of a department, and/or Technical lead on the project
	FM - Financial Manager, Accountant
	CL - C-level representative

Також, для отримання користувачів, буде доступна вибірка по фільтрах з пагінацією. Для кожного користувача, буде можливість оновлення даних профілю. В залежності від прав користувача, який його оновлює, буде доступна певна части даних. В залежності, від наданих користувачеві прав доступу, він матиме наступні права (таб. 2.5):

Таблиця 2.5 – Розподіл прав доступу

№	Claims	Place / Descrtiption	Specialities / Positions							
			User	HR	PM	AM	EM	TL	FM	C-Level
1	**Company Staff Report**	Timelog page - Report & General Report Ability to 1. View all the projects report divided by AMs 2. Pivot table of all the users in the company	-	-	-	✓	-	✓ (Bench Lead only)	✓	✓
2	**Departments management**	*Departments page - Create department* Ability to create a team and add / remove members from it	-	✓	✓	✓	-	✓	-	✓
4	**Events Management**	*Calendar page - Holidays tab - Add Event* Ability to create / edit / remove events.	-	✓	-	-	-	-	-	✓
5	**Financial Report Management**	Timelog page - Report & User Report Ability to 1. View all the projects report 2. View all the users in the company	-	-	-	-	-	-	✓	✓
6	**Holidays management**	*Calendar page - Holidays tab - Add Holiday* Ability to create / edit /remove holidays.	-	✓	-	-	-	-	-	-
8	**Non-working days management**	Responsibility for the non working days (days off, vacations) of the employees. Vacation notification to HR	-	✓	-	-	-	-	-	-
9	**Partial profile management (contact information)**	*People page - User profile - Edit profile* Ability to edit the contact information of the user.	-	✓	-	-	-	-	-	✓
10	**Partial profile management (personal information)**	People page - User profile - Edit profile Ability to edit the personal information of the user.	-	✓	-	-	-	-	-	✓
11	**Partial profile management (work information)**	People page - User profile - Edit profile Ability to edit the work information of the user.	-	✓	✓	✓	-	-	-	✓
12	**Projects management**	*Projects page - Create project* Ability to create / edit / remove projects.	-	-	✓	✓	✓	-	-	✓
13	**Tickets management**	*Dropdown menu on the top right - Vacations* Ability to manage vacation requests of the employees.	-	-	✓	✓	-	✓	-	-
14	**Timelog management**	*Timelog page* Ability to 1. Manage timelog requests 2. View the project report 3. View the user report 4. View General Report 5. View and Export My projects	-	-	✓	✓	-	✓	✓	✓

За допомогою міграцій, в базу даних будуть заноситись стартові данні, такі як: права доступу, спеціальності, ролі, системні параметри, вміння та інше. Далі, користувачі в таблиці користувачів, будуть поєднуватись з відповідними таблицями, через проміжні, це необхідно для звязків багато-до-багатьох. Основними таблицями стануть відпустки, команди, та залоговані години (додаток А). Користувач не може бути повністю видаленим з системи, для цього,

в таблиці користувачів, буде створене поле, яку відповідатиме за те, видалений він, чи ні.

2.5 Управління активностями

Для керування робочими процесами, буде доступною можливість створення проектів та іншої активності. Для цього, в базі створена таблиця з проектами, з якою повязана команда проекту. Тобто, за дані про проект буде відповідати таблиця проекту, а за команду, табляця команди. До таблиці команди, для кожного користувача, міститимуться дані в таблиці користувачав команди. Для кожного користувача будуть заноститись особисті дані, стосовно роботи в команді, посади та інші. Також, буде не обовязковим створення проекту, доступне створення команди з працівниками, для внутрішніх департаментів, бухгалтерії, тощо. При додаванні користувача до певної команди, він буде відображатись в всіх репортах, навіть після його видалення, буде відображатись до кінця місяця. Через створені звязки, буде проводитись пошук керуючих працівників для відпусток, та підтвердження залогованих годин (рис. 2.16).

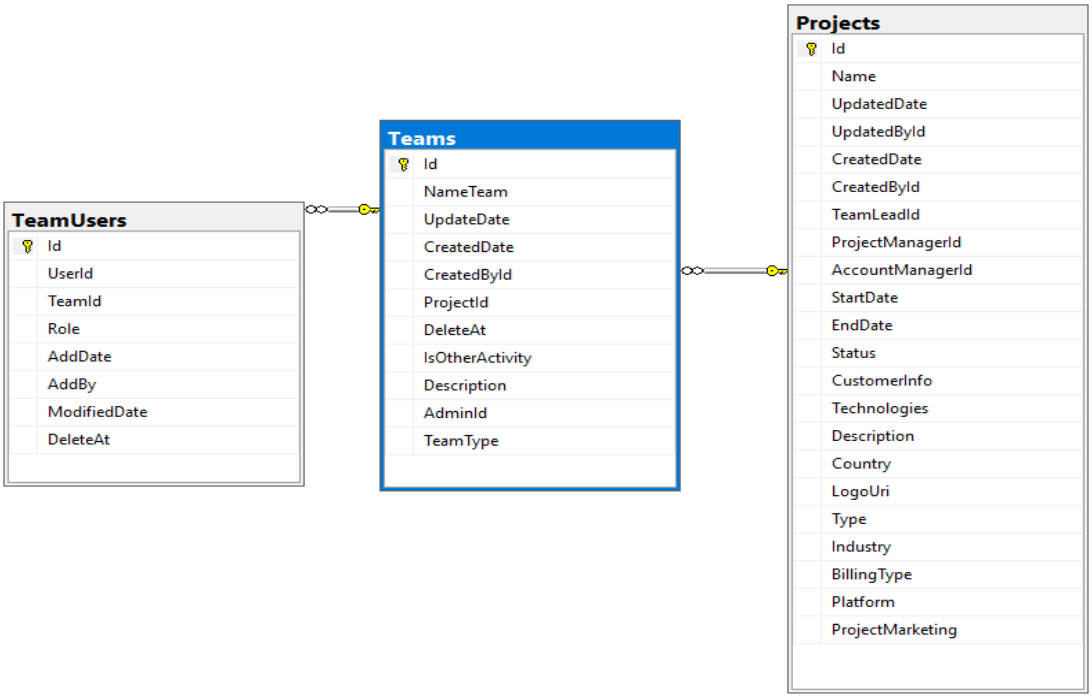


Рисунок 2.16 – Схема основних зв’язків активностей

2.6 Google Console

2.6.1 Створення та налаштування проекту

Для того, щоб використовувати сервіси Google, та авторизацію через нього, необхідно створити та налаштувати проект в Google Console. Для цього, необхідно увійти в Google Console, під акаунтом Google, та створити новий проект. Після створення проету, буде надано “Google credentials” в форматі Json файлу. За допомогою них, сервіси Google зможуть аутенифікувати розроблений проект, робити аналіз та відображати статистику роботи (рис. 2.17). Для забезпечення безпеки, потрібно налаштувати дозвола доступу, обмежити IP адреса, з яких можуть надходити запити. Для керування доступами, створено сервісний акаунт Google, який не потрібно буде кожного разу аутентифікувати, і який буде основним для зв’язків з іншими сервісами. За допомогою налаштувань

доступу OAuth, вхід в програму буде доступним лише для працівників певних компаній.

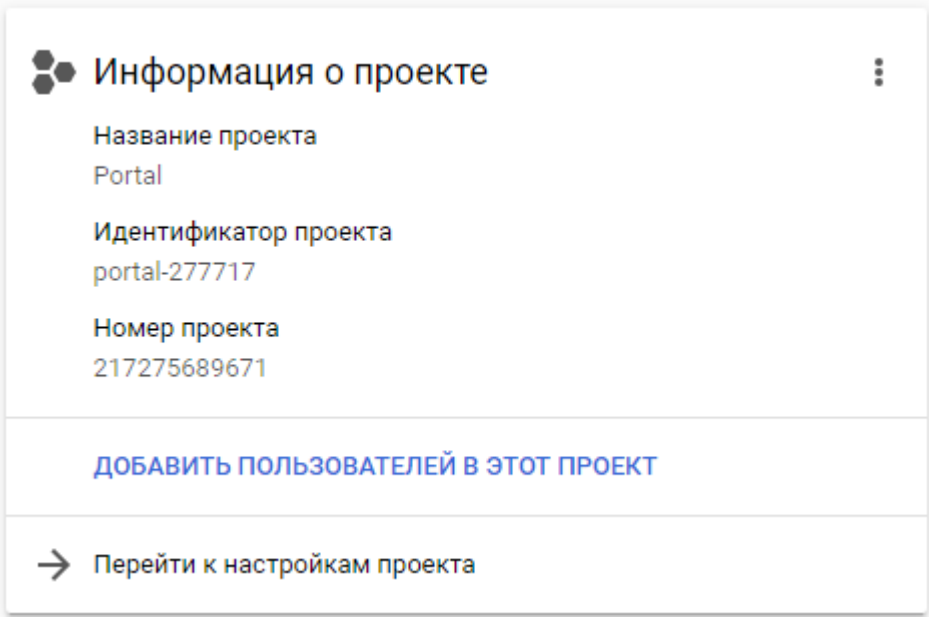


Рисунок 2.17 – Схема основних звязків активностей

2.6.2 Google Calendar API

Для відображення подій, свят та днів народжень працівників компанії, буде доступною взаємодія з Google Calendar API. Calendar API дозволяє відображати, створювати та змінювати події календаря, а також працювати з багатьма іншими об'єктами, пов'язаними з календарем, такими як події або елементи управління доступом. Його підключення та налаштування, здійснюються також через Googe Console. При підключенні Calendar API, необхідно провести налаштування доступу для створеного раніше, сервісного акаунту. Для платформи .NET, існує розроблена компанією Google бібліотека, для роботи з Calendar API. Після успішного налаштування в Googe Console, стане доступною можливість моніторингу роботи Google Calendar (рис. 2.18).

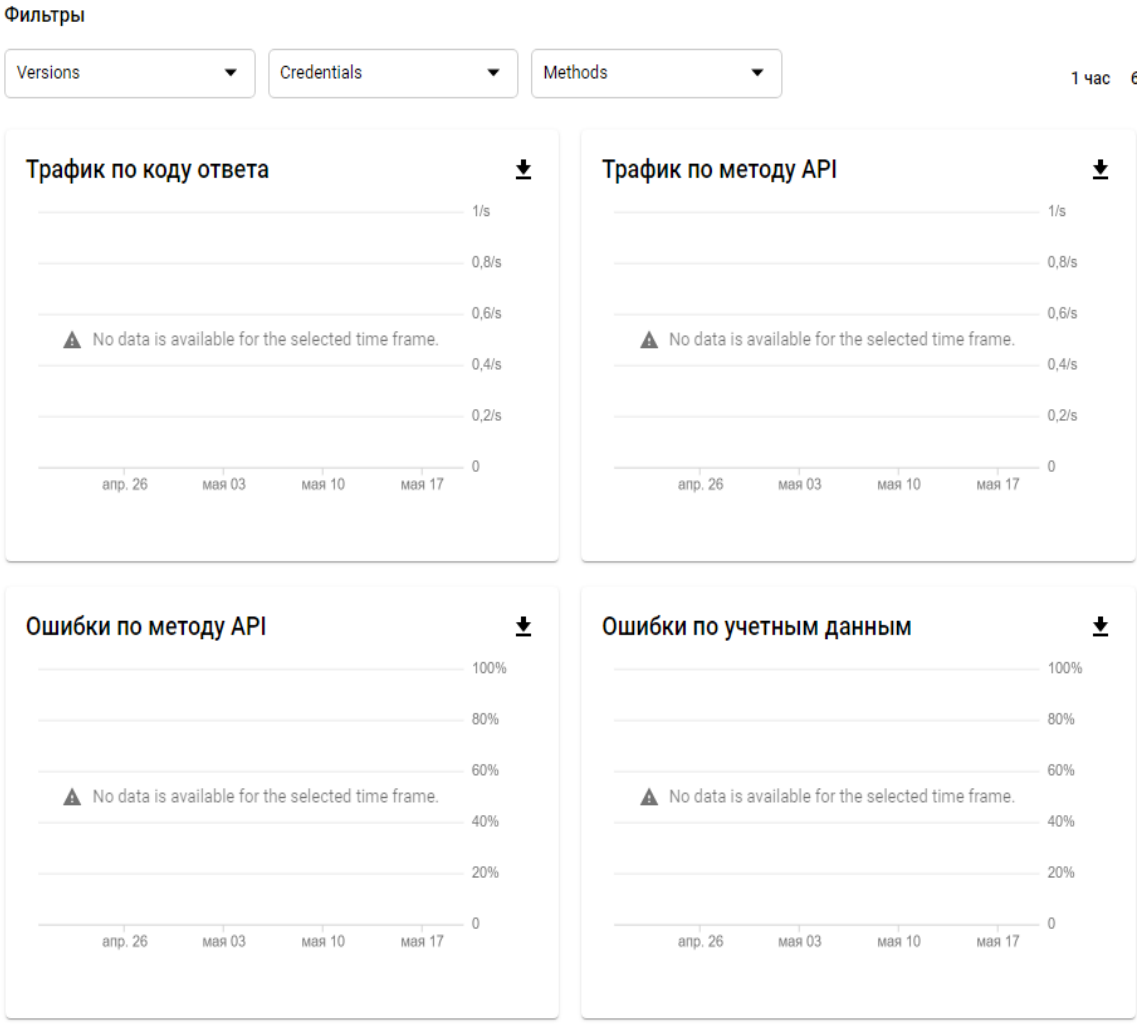


Рисунок 2.18 – Схема основних звязків активностей

2.7 Авторизація та Автентифікація

Для авторизації та аутентифікації користувача, використано можливості платформи Identity та спроектованих допоміжних Owin Middleware. Для входу використано “Google OAuth 2.0”, для того, щоб інтегрувати програму з безплатними сервісами Google.

Кінцева точка Google OAuth 2.0 підтримує програми веб-сервера, які використовують мови та платформи, такі як PHP, Java, Python, Ruby та ASP.NET.

Послідовність авторизації починається, коли програма перенаправляє браузер на URL-адресу Google;

URL-адреса включає параметри запиту, які вказують тип запиту. Google обробляє автентифікацію користувача, вибір сеансу та згоду користувача. Результат - код авторизації, який програма може обміняти на token доступу та token оновлення [11].

Додаток має зберігати token оновлення для подальшого використання та використовувати token доступу для доступу до API Google. Як тільки token доступу закінчується, програма використовує token оновлення для отримання нового (рис. 2.19).

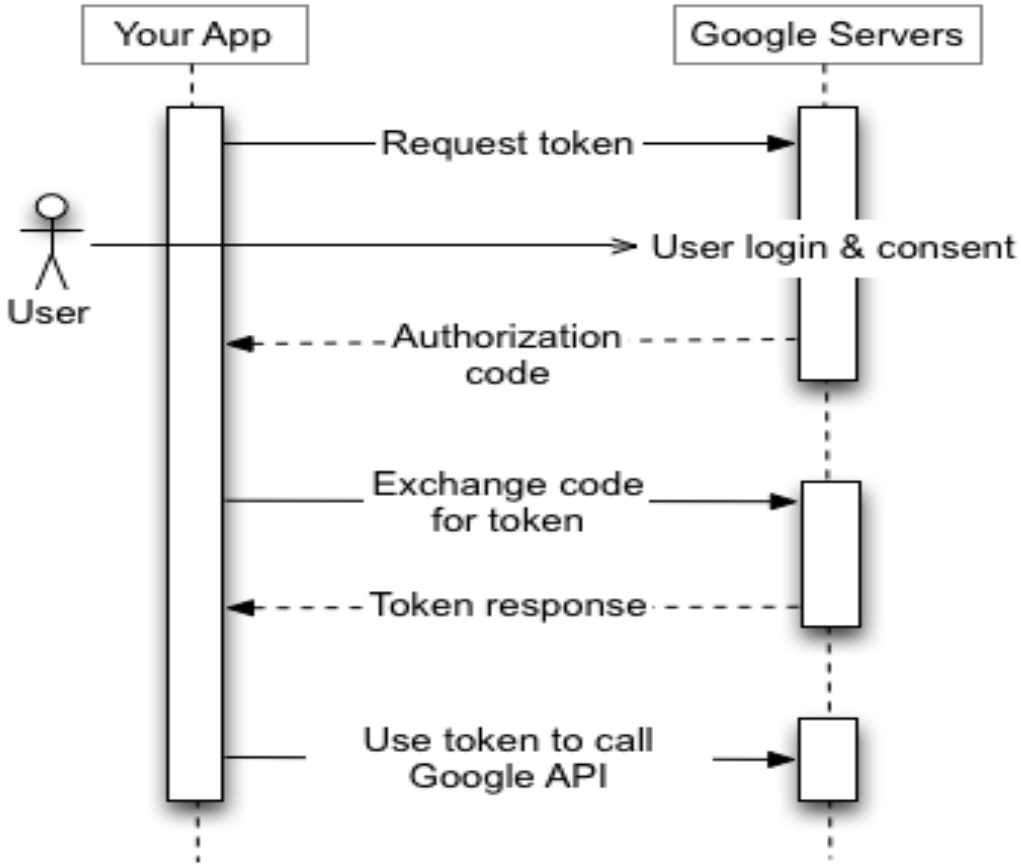


Рисунок 2.19 – Схема взаємодії з Google Servers

Google APIs, такі як Prediction API і Google Cloud Storage, можуть діяти від імені програми, без доступу до інформації про користувачів. У цих ситуаціях програмі потрібно довести власну ідентичність API, але згода користувача не потрібна. Аналогічно, у корпоративних сценаріях програма може вимагати делегованого доступу до деяких ресурсів.

Для цих типів взаємодії сервер-сервер, потрібен обліковий запис служби - це обліковий запис, який належить програмі, а не окремому кінцевому користувачеві. Програма звертається до API Google від імені облікового запису послуги, і згода користувача не потрібна. (У сценаріях, що не стосуються облікових записів, програма звертається до API Google від імені кінцевих користувачів, і іноді потрібна згода користувача.)

Вхідні дані облікового запису служби, які отримано від консолі API Google, містять унікальну створену електронну адресу, ідентифікатор клієнта та принаймні одну пару відкритих / приватних ключів. Можна використовувати ідентифікатор клієнта та один приватний ключ, для створення підписаного JWT та побудови запиту доступу до token-а, у відповідному форматі. Потім програма надсилає запит token-а на сервер авторизації Google OAuth 2.0, який повертає token доступу. Додаток використовує token для доступу до API Google. Коли термін закінчується, програма повторює процес (рис. 2.20).

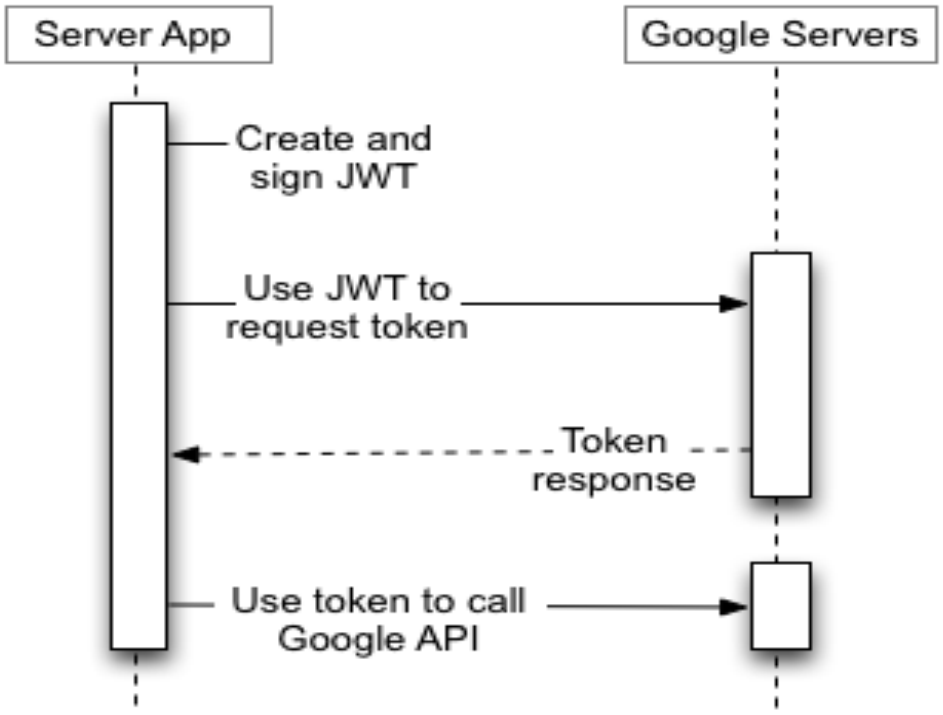


Рисунок 2.20 – Схема взаємодії з посторонніми Google APIs

Потрібно написати свій код так, щоб передбачити можливість того, що наданий token оновлення може більше не працювати. Token оновлення, може припинити роботу з однієї з таких причин:

- користувач відкликав доступ до вашої програми.
- Token оновлення не використовується вже півроку.
- користувач змінив паролі, а token оновлення містить області застосування Gmail.
- обліковий запис користувача перевищив максимальну кількість наданих (в прямому ефірі) ознак оновлення.

Наразі обмежено 50 token-ів оновлення, для кожного облікового запису користувача. Якщо ліміт досягнуто, створення нового token-а оновлення автоматично виключає найдавніший token оновлення без попередження. Цей ліміт не поширюється на облікові записи послуг.

Існує також більший ліміт на загальну кількість token-ів оновлення, які може мати обліковий запис або обліковий запис служби для всіх клієнтів. Більшість звичайних користувачів не перевищують цю межу, але тестовий рахунок розробника може бути.

Якщо потрібно авторизувати кілька програм, машин чи пристроїв, одним вирішенням, є обмеження кількості клієнтів, які авторизовані для облікового запису користувача, до 15 або 20. Якщо ви адміністратор G Suite, то є можливість створити додаткових користувачів-адміністраторів та використовувати їх для того, щоб авторизувати деяких клієнтів [17].

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Розробка користувачів

Для керування профілем користувача, в контролері “UserController”, було створено наступні методи:

OnBoarding() - метод, який використовуватиметься при першому вході користувача і повідомлятиме користувачів, з спеціальністю "Human Resources", про нового користувача в системі. Для цього, викликається метод InformHRsAboutNewUser() сервісу “UserService”, в якому, з бази даних отримуються id та пошта користувачів, які мали спеціальність "Human Resources". Після цього, за допомогою розроблених раніше методів, відбувається створення сповіщень та надсилання поштових повідомлень.

Update() – метод, який приймає розроблену раніше модель, для оновлення даних певного користувача, звичайним користувачем, “UpdateUsersInformationByUserIncomeModel”. Після перевірки моделі, викликається метод UpdateUsersInformationByUser(), в якому з бази отримується потрібний користувач, оновлюються відповідні дані з моделі та фіксуються зміни.

Update() – метод, який приймає розроблену раніше модель для оновлення даних певного користувача адміністратором “UpdateUsersInformationByAdminIncomeModel”. Після перевірки моделі викликається метод UpdateUsersInformationByAdmin(), в якому з бази отримується потрібний користувач, і перевіряється, чи запит на оновлення даних не надаіслав користувач, дані якого необхідно оновити. Якщо дані оновлює інший користувач, проводимо перевірку на права доступу до кожної категорії даних. При наявності клейми "Partial profile management (contact information)", відбувається оновлення скайпу, номера телефону, офісу та робочих годин користувача. Далі, такими ж блоками умови, оновлюються інші дані

користувача. Якщо, дані користувача оновлюються тим самим користувачем, оновлюються лише дозволені при проектуванні дані.

GetUser() – метод, в якому отримується Id користувача, і повертається модель “UserViewModel”, з основними даними користувача. Для цього викликається метод GetUserById(), в якому з бази отримується користувач, проводиться перевірка його наявність в системі та заповнення даних моделі. Додатково створено метод GetAllSkillsByUserId(), в якому отримуються навички користувача, і GetUserSpecialityName(), для отримання спеціальності.

GetList() – метод, в якому приходить вхідна модель “GetPaginatedListUserIncomeModel”, для отримання всіх користувачів, з відповідними параметрами та використовується пагінація. Після перевірки вхідної моделі, викликається метод GetListByFilter(), сервісу “UserService”. Спочатку, в методі створено змінну “itemsToSkipCount”, в яку занесено кількість моделей даних, які потрібно пропустити. Далі, створено колекцію, типізовану командами, та розміщено блок умови, в якому, якщо в вхідній моделі є Id проекту, в колекцію добавляється команда проекту. З отриманих команд, проводиться вибірка користувачів, які ще не видалені, та за допомогою методу Select(), отримуємо лише колекцію Id користувачів. Далі, проводиться вибірка всіх користувачів з бази даних, окрім адміністратора, і з них, за допомогою методу Where(), проводиться фільтрація по кожному з властивостей моделі. Для цього, в умові робиться перевірка по властивості, лише в випадку, якщо властивість моделі не null. Після вибірки проводиться сортування користувачів по імені, пропускається потрібна кількість, та формується для кожного, вихідна модель, додаючсь дані, необхідні для пагінації.

GetUserStatus() – метод, для отримання статусу користувача, який приймає id користувача. В методі за допомогою блоків умов, перевіряється тип дня, і в залежності від цього, додається відповідний статус. Якщо ж це звичайний робочий день, за допомогою методу IsUserAvailable(), перевіряється чи користувач в момент запиту знаходиться на роботі. Для цього, необхідно

отримати теперішню годину та хвилину за допомогою структури “DateTime”, та прирівняти до вказаного користувачем часу.

GetUserProjects() – метод для отримання всіх проектів користувача, в якому, за допомогою методу GetProjectsByUserId(), отримуються всі проекти даного користувача. Вибірка проводиться через команду, адже зв'язок починається з моделі “TeamUser”. Після отримання проектів, для кожного формується модель “ProjectViewModel”, відсортована по статусу проекту та його назві.

GetAllSpecialities() – метод, для отримання занесених в базу спеціальностей користувачів.

GetAllUsersSkills() – метод, для отримання занесених в базу навичок користувачів.

DisableUser() – метод, для видалення користувача, в якому, з бази отримується потрібний користувач та для властивості “IsDisabled”, вказується значення true, а даті видалення, вказується теперішній час. Далі, відбувається видалення користувача з усіх команд, та вказується всім прикріпленим до нього квикам підтведження відпусток, статус “Approved”, після чого, робиться те саме для квитків підтвердження понаднормових годин. Також, потрібно видалити всі відпустки та квитки підтвердження відпусток, які до них прикріплені, даного користувача. Всі вказані вище дії, виконуються за допомогою вибірки даних з необхіднох таблиці та перебір їх за допомогою методу foreach(), після чого фіксуються зміни.

AllBirthdays() – метод, для отримання всіх дат, днів народжень користувачів, разом з фото та іменем користувача.

GetYearStatistics() – метод, для отримання кількості нових та видалених користувачів за рік. В ньому проводиться звичайна вибірка з бази даних, з вказаним параметром та функції Count().

GetHrStatisticExelReport() – метод, для отримання статистики по користувачам за певний період, в форматі excel, захищений атрибутом

ClaimsAuthorization(), який забороняє доступ, якщо в заголовку запиту немає прав “HolidaysManagement”. В ньому викликається метод GetUserListForStatistic(), для формування моделі, яку буде занесено в excel документ. В ньому отримуються користувачі, в яких перший робочий день, входить в вибірку та проходить сортування по ньому, для кожного формується модель, яка містить дату та імя користувача. Так само проводиться вибірка для користувачів, в яких дата видалення входить в вибірку. Після формування моделі, вона передається в метод GetHrStatisticExelReport(), сервісу “UserExcelExport”. В методі ствророється документ з робичим листом, під назвою "UserStatistic", після чого, викликається метод AddHeaderToTable(), в якому добавляється відповідний заголовок та його стилізація. Для добавлення самих користувачів, створено метод AddUsersToTable(), який буде спільним для видалних та доданих користувачів. Він приймає колекцію користувачів та координати початкової клітинки, починає перебір коритувачів та занесення в таблицю. Принцип роботи з excel документом описаний в розділі про логування часу (рис. 3.1).

Name	Added User		Name	Removed User
Андрей Щадило	12.05.2020			
Oleksiy	07.05.2020			
Olena Smol	30.04.2020			
Ihor Bordun	30.04.2020			
One Love	28.04.2020			
Olena Max	27.04.2020			
Olena Developer	24.04.2020			
mark PM	24.04.2020			
BIBIbi PM	24.04.2020			
Ihor Chipak	24.04.2020			
Tatiana Husnay	23.04.2020			
Maxim Chief	23.04.2020			
Andryi Matchak	23.04.2020			
Микола Куціль	23.04.2020			
Марк Білик	22.04.2020			
Lorem Ipsum	22.04.2020			
Mike Shevchenko	14.04.2020			
Olena PM	14.04.2020			
Богдан Івахів	13.04.2020			
Olena HR	13.04.2020			
Тарас PM	13.04.2020			
Taras Pylypchuk	13.04.2020			
Тарас Пилипчук	13.04.2020			
Наталя HR	13.04.2020			
Ihor Bordun	12.04.2020			
M Bi	10.04.2020			
Marian - Makar Bilyk	10.04.2020			
Bohdan Ivakhiv	10.04.2020			
Andryi Matchak	10.04.2020			
Olena AM	10.04.2020			

Рисунок 3.1 – HR Excel таблиця по користувачах

3.2 Авторизація та Автентифікація

Для авторизації та аутентифікації користувача, використовуються можливості платформи Identity, спроектованих допоміжних Owin Middleware та створених в контролері “AccountController” методів.

Клас “ApplicationUserManager” – Owin Middleware, який запускається при старті програми і базується на базавому класі платформи Identity. Він відповідає за регулювання додаткових даних про користувача.

Клас “GoogleAuthenticationConfig” – клас, де знаходиться конфігурація GoogleOAuth2, за допомогою методу GenerateConfig(). В ньому, спочатку з конфігурації отримуються “ClientId” та “ClientSecret”. Далі за допомогою “Microsoft.Owin.Security.Google” namespace створюється екземпляр класу “GoogleOAuth2AuthenticationProvider”, який і буде нашим провайдером аутентифікації. В провайдері вказується URl для редіректу, та в разі успішного входу додаються email, ім’я користувача, token та token овлення.

Для аутентифікації користувача, використовуються базові класи “OAuthAuthorizationServerOptions” та “OAuthBearerAuthenticationOptions”. Також в класі Startup, розміщено параметри OAuth, які є екземпляром класу “OAuthAuthorizationServerOptions”. В них задано час життя token-у, який дорівнює сорока хвилинам, кінцеву точку вторизації, шлях для отримання нового token-у доступу, провайдер Google та клас “ApplicationRefreshTokenProvider”, який відповідатиме за обробку token-у оновлення.

Клас “ApplicationRefreshTokenProvider”, наслідується від системного класу “AuthenticationTokenProvider” і змінює логіку виконання його методів.

В методі CreateAsync(), отримується вхідний контекст Google OAuth, використовуючи “AuthRepository”, в базу даних заноситься новий token оновлення, який хешується за допомогою методу ReceiveAsync(). В подальшому, його буде використано для оновлення існуючого token-у доступу. Так як token оновлення не має часу життя, для того, щоб зберігати безпеку, окрім його хешування, йому задано час життя в 24 години.

Ввесь перечислений вище функціонал є middleware, які починають свою роботу з моменту старту програми, та взаємодіють між собою по стандартних схемах. Цього досягнуто, через використання системних класів авторизації та автентифікації, як базису, для створених класів.

Також в контролері “AccountController”, було створено наступні методи:

Метод GetExternalLogins(), створений для вибору потрібного провайдеру аутентифікації. В ньому отримуються доступні провайдери, та добавляються дані про них, після чого, надсилається відповідь.

Метод GetExternalLogin(), створений для авторизації користувача. В ньому користувача перенаправляє на кінцеву точку аторизації потрібного провайдеру, і в разі успішного входу, користувач вноситься в базу даних, як новий, якщо це його перший вхід, або до нього добавляються права на доступ. В методі, на початку, розміщено перевірку на помилки під час входу, якщо вони виявлені, то за допомогою методу ChallengeResult(provider, this), відбувається редірект на потрібний провайдер. Далі, за допомогою властивості “User.Identity.IsAuthenticated”, продовжується аутентифікаця користувача. Якщо він не аутентифікований , виконується редірект на провайдер аутентифікації. Для отримання даних провайдера, в сервісі “ExternalLoginDataService”, створено метод GetFromIdentity(), який приймає клас “ClaimsIdentity”. Далі, з клеймів отримуються потрібні дані, та формується модель “ExternalLoginDataViewModel”. Якщо в методі не отримано жодних даних від провайдеру, редірект повторюється до введення користувачем правильних даних. Далі, з конфігурації, отримуються дозволені домени для входу. Якщо вони занесені в конфігурацію, перевіряється, чи збігається email користувача, з дозволеними доменами, якщо не збігається, то проводиться редірект для введення правильних даних. Далі, за допомогою “UserManager”, отримується користувач з системи, і якщо він не null, то він вже був зареєстрований. Якщо користувач зареєстрований, то очищаються всі клейми користувача та проходить його оновлення в базі даних. Якщо користувач не зареєстрований, то отримуються всі “Claim” з Owin Middleware, для даного користувача. З них отримуються базові дані, надані провайдером аутентифікації та заносяться в модель “ApplicationUser”. Відбувається внесення користувача в базу даних, за допомогою методу CreateAsync(), класу “UserManager”. Якщо створення користувача успішне, йому додається роль “User” та додаткові дані. Після цього,

за допомогою методів сервісу “CalendarService”, перевіряється наявність календаря і при потребі користувачу створюється новий календар. На цьому вхід користувача завершується.

Для отримання даних про користувача, створено метод GetUserInfo(), в якому, за допомогою описаних раніше методів проводиться отримання основних даних про користувача та формування моделі “UserInfoViewModel”. Також, в методі використовується метод SetUserVacation(), який описаний раніше, для того, щоб перерахувати нараховані дні відпусток.

3.3 Репорти

3.3.1 Репорт по користувачу

Для таймлогів потрібно розробити окремий контролер, тому створено контролер під назвою “TimeLogController”, де є метод Get, для отримання репорту по користувачу. В методі задано “Rout” в заголовок якого задані такі параметри:

- userId - унікальний ідентифікатор користувача, по якому створюється репорт;
- startDate - стартова дата, по якій робиться репорт;
- endDate - кінцева дата, по якій робиться репорт.

При прийомі данних, використовується перевірка стану моделі запиту на помилки, за допомогою властивості “ModelState.IsValid” класу “ApiController”. При виявленні помилок, створюється помилка “ArgumentException”, яка свідчитиме про помилки під час прийняття запиту на контролер. Після перевірки моделі, використовуючи платформу Identity, я дізнаюсь Id користувача який надіслав данний запит. Так як дата надходить в контролер в типі string, потрібно перевірити коректність формату дати та перевести в тип “DateTime” для роботи з нею. Для цього використовується метод TryParse(), структури “DateTime”, який

спробує перевести string в “DateTime”, і в залежності від результату, поверне певне буліанівське значення.

Для розробки самого методу, створення репорту, потрібно спочатку визначитись з моделюю данних, яка буде повернена. Від моделі залежить дуже багато, адже при погано спроектованій структурі моделі, ми не зможемо застосувати алгоритми, які зроблять репорт швидким. Так як дата по якій робиться репорт буде відображатись по місяцях, потрібно на верхньому рівні моделі розмістити місяць, по якому зроблений репорт, а також передати ім’я та Id користувача, по якому зроблений репорт. Так як один користувач може бути на декількох проектах одночасно, модель також повинна містити масив репортів по проекту з підсумками.

Також, враховуючи те, що в користувача можуть бути дні відпусток, в репорті, потрібно розробити модель, яка віддображатиме вихідні дні, відпустки та свята. Ці дні не відносяться до конкретного проекту, тому створюється окрема модель для їх відображення.

Модель репорту по одному проекту, міститиме назву та Id проекту, а також поля підсумків, по окремих таймлогах та загальний підсумок по проекту. Також, модель по проекту, міститиме моделі днів, які стануть основою для подальшого відображення та створення нових таймлогів.

Модель дня, натомість міститиме день в форматі “DateTime”, моделі таймлогів, підсумок по дню, та буліанівське значення того, чи це активний день. Цих данних має бути достаньо, для коректної роботи репорту а моделі таймлогів задані не полями а окремими типами, що дозволить зменшити об’єм пам’яті яка виділяється. Підсумок по всіх проектах міститиме загальний підсумок по всіх днях та по окремому дню. Підсумок по конкретному дні міститиме день в форматі “DateTime”, буліанівське значення того, чи це вихідний та підсумок.

Після проектування моделі данних репорту, потрібно розробити інтерфейс, через який відбуватиметься взаємодія з нашим TimelogService.

Інтерфейс міститиме метод GetTimeLogReportByUser(), з необхідною структурою.

В самій реалізації методу, спочатку потрібно розділити стартову та кінцеву дату вибірки по місяцях. Для цього розроблено метод, який приймає стартову і кінцеву дату вибірки і повертає масив стартової дати кожного місяця, типу “DateTime”. Далі проводиться вибірку таймлогів користувача, які входять в дату вибірки. Краще зробити вибірку з бази данних, саме за ввесь період а не по місяцях окремо, адже в базі данних можуть бути тисячі таймлогів і потрібно зробити янайменше запитів на вибірку, що покращить швидкодію. Коли вже вибрані всі таймлоги, які входять в вибірку, можна працювати з ними на рівні коду і робити опирації поділу по місяцях, та вибірки по умовах. Далі проведена вибірка з бази данних самого користувача, по якому робиться репорт. Коли отримали необхідні данні, спільні для всіх проектів, які можуть бути в користувача, починаємо перебирати масив отриманих місяців.

Спочатку за допомогою методу DaysInMonth() структури “DateTime”, отримуємо число днів в місяці. Отримане значення, передається в конструктор структури, для отримання коректного формату кінцевої дати. Для можливості подальшого розширення методу і уникнення дублювання коду, потрібно створити ще один метод, який повертатиме масив репортів по проекту конкретного місяця першим параметром і підсумок другим. В методі спочатку отримаєм масив всіх проектів на яких знаходиться користувач. Якщо в користувача нема проектів, то повертаємо тип null, в іншому випадку створюємо медель, яку буде повернено. Важливо передбачити умови, при яких модель не доведеться створювати. При кожному створенні моделі, виділяється память в кучі, що в подальшому призводить до виклику збору мусору і взагалі це не раціональне використання памяті. Також, до того як ми будем робити репорт по окремому проекті, створюємо масив в який будемо поміщати спільний для всіх проектів підсумок по днях і створюємо змінну isFirst, яка міститиме значення

того, чи це перший проект перебирається. Вона необхідна для того, щоб могти заповнити масив з підсумками днів, які спільні для всіх проектів.

Під час перебору масиву проектів, спочатку створюється модель репорту по одному проекту і добавляється назва та Id проекту. Після цього створено цикл, який буде перебирати дні в місяці. Далі, створено модель дня та заповнено значення Day, та за допомогою властивості “DayOfWeek”, заповнено “IsWeekend”. Далі поміщаємо в блок умови, який спрацює при isFirst == true, добавлення дня в модель підсумків по днях. Значення “IsActiveDay” набуватиме true, якщо день більший за стартову дату проекту і менший за кінцеву.

Далі, потрібно покласти блок умови, при якому ітерація циклу буде пропускатись, якщо день виявиться не активний або час репорту більший за сьогоднішній, адже в подальшому проходженні немає сенсу і воно ресурсозатратне. Якщо ж потік виконання піде дальше, то за допомогою методу GetTimelogsReport(), добавляється моделі таймлогів. Після проходження всіх днів, добавляємо підсумок та вносимо властивості “isFirst”, значення false. Так повторюється з усіма проектами, після чого добавляється підсумок і дані повертаються з методу. Дані, які повторюється з методу, добавляються до моделі місяця. Також, добавляємо репорт з типами днів. Він організовується через перебір днів місяця і додавання в модель всіх відпусток, вихідних та свят.

3.3.2 Репорт по проектах

Репорт по проекту, розроблений за допомогою побудованої попередньо логіки. В контролері створено метод GetListUsersTimeLogByProject(), в якому виконано ті самі дії, що і при репорті по користувачу, але з викликом іншого методу GetTimeLogProjectReport(). В методі, спочатку, з бази данних отримується потрібний проект. Дату вибірки, звіряю з датою старту та закінчення проекту. Далі, використано описаний в репорті по користувачу метод GetMonths(), для поділу дати вибірки на місяці. Далі, дані передаються в метод

GetTimeLogProjectReport(), сервісу “ProjectTimeLogReportCreator”. В методі відбувається перебір масиву місяців, створюючи для кожного, модель репорту за місяць “GetTimeLogProjectReportViewModel”. Створено окремі методи для формування тіла репорту і підсумку. Метод GetMonthUserProjectModels(), для тіла репорту, повертає масив моделей “MonthUserProjectReportModel”. В методі отримується масив свят, які занесені в базу даних на даний місяць. Також, отримуються користувачі, які знаходяться на проекті, та сортуються по імені. Після цього починаємо формувати репорт по користувачам. Для кожного користувача з бази даних отримуються його відпустки, та поступово відбувається заповнення даними спроетованої моделі. Для створення моделі даних по днях, створено окремий метод GetProjectDayInfoByUser(). В методі отримую всі таймлоги користувача, і створюю цикл перебору днів в місяці. Кожен день в циклі, перетворюється в формат “DateTime” для роботи з датою. З вибраних раніше свят і відпусток, за допомогою методу CheckDayTypeInfo(),присвоюється моделі дня, тип і опис дня. Після отримання опису дня, викликається метод CreateUserReportByDay(), для формування репорту по одному дню. В ньому створюється модель дня та ініціалізується стандартними даними. З списку таймлогів за місяць, вибираються тількі ті, які створені на сьогоднішній день, і додаються в модель дня. Після чого, добавляється підсумок дня, закінчується ітерація, і формується підсумок по всім днях. На цьому етапі закінчується виконання методу, повертаючи сформований репорт. Після цього, формуються подальші підсумки, і повертається відповіддю вже сформований репорт. При розробці репорту, було враховано розроблену на стадії планування модель даних та алгоритми роботи, що допомогло без проблем виконати поставлену задачу.

Детальний репорт по проекту. Для створення детального репорту по проекту, створено метод GetTimeLogSingleProjectReport(), який приймає масив місяців та проект, по якому створюється репорт. Спочатку в методі створюється масив моделей “SingleProjectViewModel”, кожна з яких є репортом по одному

місяцю. Починається перебір місяців і створення дати останнього дня місяця. Після чого, модель заповнюється назвою проекту, місяця та датою першого дня місяця. Далі за допомогою методу GetUsersByProjectId(), сервісу “ProjectService”, отримуємо Id всіх користувачів, які були в команді проекту в даний місяць, окрім акаунтменеджера, який не входить в репорти. За допомогою GetUsersByIds(), отримуємо користувачів з попередньо знайденого масиву Id користувачів. Метод SortUsers(), сортує користувачів по імені. Після чого створюється масив моделей “UserInSingleProjectViewModel”, і відбувається перебір користувачів. Для кожного користувача, отримуються створені залоговані години на даний проект і місяць. Далі отримуються відпустки, за допомогою методу GetUserVacationByDate(), сервісу “VacationService”, який описувався в звичайному репорті по проекті. Після отримання цих даних, за допомогою методу GetUserDaysReport(), отримуємо репорт по всіх днях. В методі перевіряємо дату закінчення проекту, якщо вона менша за останній день місяця, то назначаємо останнім днем, дату закінчення проекту. Отримуємо всі вихідні на даний місяць, за допомогою методу GetHolidayEntityByDate(), описаного в звичайному репорті по проекті. Далі отримуємо тип дня за допомогою методу CheckDayTypeInfo(), описаного раніше, і добавляємо в залежності від типу додатковий опис, та опис залогованого часу. Після створення репорту по всіх днях, за допомогою методу колекції Sum(), створюємо підсумки. Так повторюється виконання для кожного користувача, після чого виконання створення репорту завершується.

Репорт по проектах адміна. Для створення репорту по проектах адміністратора, було створено метод GetReportByAdminProjects(), який приймає стартову та кінцеву дату вибірки та Id адміністратора, перевіряє дані та викликає метод GenerateAdminProjectsReport(). В ньому відбувається отримання масиву місяців, та виклик методу GetLeadUserTimelogProjects(), в якому отримуються всі проекти адміністратора. Далі відбувається перебір місяців та отримання всіх проектів за конкретний місяць, і якщо вони наявні, то створюю необхідні моделі,

та викликаю метод створення репорту по одному проекту, для кожного з його проектів.

Репорт по всіх проектах. Для створення репорту по всіх проектах було створено метод GetAllProjectsReport(), в контролері “TimelogController”, в якому викликається метод GenerateAllProjectsReport(). В методі відбувається перебір місяців вибірки, та для кожного місяця проводиться вибірка всіх проектів, на яких є акаунтменеджери, та за допомогою методу колекції GroupBy(), групую по акаунтменеджерах. Ящо вони є, починається перебір акаунтменеджерів, заповнюються необхідні дані моделі та відбувається перебір всіх проектів. Для кожного проекту буде викликано метод, для створення репорту по одному проекту.

3.3.3 Загальний репорт

Для створення репорту по всіх користувачах системи за певний період, було розроблено метод GetAllUsersReport(), в контролері “TimeLogController”. В методі параметрами приймаються стартова та кінцева дати вибірки, та перетворення їх до типу “DateTime”. Далі за допомогою описаного раніше методу, отримується масив місяців, та передача їх в метод GetAllUsersTimeLogReport(), сервісу “TimelogService”. В методі починається перебір місяців і виклик методу GetTotalUsersReport(), в який передається місяць даної ітерації. В методі з бази даних оримуються всі активні користувачі системи, відсортовані по імені, та за допомогою методу GetPlanByMonth(), формується місячний план. Далі ініціалізується змінна “userNumber”, якій задається значення 1, та отримуються затверджені таймлоги користувачів на даний місяць. Далі, відбувається перебір користувачів, та отримання з вибраних раніше таймлогів, лише ті, які належать даному користувачеві. Далі з таймлогів користувача, отримуються тільки звичайні години та години очікування на проектах. Якщо такі є, то за допомогою методу колекції Sum(), отримуємо суму

всіх годин, та викликаємо метод GetProjectHours(), для отримання суми годин по кожному. Метод в параметрах отримує колекцію timelog-ів та буліанівське значення того, чи проводиться вибірка по проекту. В ньому отримуються Id всіх проектів, до яких відносяться передані таймлоги, та для кожного отримую суму годин та його назву. Такі самі дії, проводяться для понаднормових годин, пресейлів, відділів та іншої активності. Для отримання плану та відпусток користувача, я створив метод GetUserVacationHours(), в якому отримую всі відпустки користувача на даний місяць. Далі створюється змінна “userTotalVacations”, яку буде повернуто з методу, та вказується їй нульове значення. Далі, відбувається перебір днів, і якщо на даний день, в користувача є відпустка, до “userTotalVacations” добавляється число 8. Далі вираховується підсумок по всьому, відсоток виконання від плану та різницю між підсумком та планом та заносяться, всі отримані раніше дані, в модель. Після цього виконується пошук коментаря до даного користувача, і якщо він є, то додається в вихідну модель.

3.4 Excel репорти

3.4.1 Експорт репорту по користувачу

В контролері “TimeLogController”, бууло створено метод GetReportInExcel(), для формування репорту по користувачу, в форматі excel файлу. Спочатку відбувається перевірка моделі та перетворення формату дати. Після чого в сервісі “UserExcelExporter”, створено метод GetUserExcelReport(), який стане перехідною ланкою між формуванням описаного репорту по користувачу, та методом, який перетворить репорт в формат excel файлу.

Отже в методі, в змінну “userTimeLogReport”, було передано сформований репорт, яку передав в метод GenerateExcelUserReport() сервісу “GenerateUserExcelTableService”. В методі використовуючи вибрану бібліотеку

для формування excel файлів “OfficeOpenXml”, створюється внутрішня модель файлу “ExcelPackage”. Далі відбувається перебір моделі репорту, для занесення в файл, почавши з верхнього рівня місяців. На початку ітерації, для кожного місяця, створено окремий лист в excel документі, за допомогою методу Add(), створюється змінна з стартовим рядком. Створення таблиці відображення репорту, я було розділено на три методи:

- AddHeaderToWorkheet() – який відповідатиме за формування верхньої частини таблиці;
- AddTotalToWorkheet() – який відповідатиме за формування підсумків в таблиці;
- AddProjectsToBodyWorkheet() - який відповідатиме за формування основного тіла таблиці;

Також створив додаткові методи:

- AddVacationsToBodyWorkheet() - який відповідатиме за додавання в тіло таблиці відпусток;
- AddPlanToWorkheet() - який відповідатиме за додавання робочого плану в таблицю;
- AddStyleToWorkheet() - який доповнюватиме стилізацію таблиці.

В методі AddHeaderToWorkheet(), за допомогою властивості Cells[] нашого робочого листа, я вибираю клітинку в excel документі. Після вибору клітинки, я маю змогу добавляти в неї текст, значення, формули та стилізацію. Метод LoadFromText(), дозволяє загрузити данні в форматі string і використовую його для занесення імені користувача. За допомогою властивості Style, добавляються стилі. Тому, щоб не робити всю стилізацію, після занесення даних, я це роблю одразу. За допомогою властивості Border, я встановлюю рамку для клітинки, а за допомогою Fill, тип та колір заливки. Після занесення сталих даних, починаю заносити кількість днів в заголовок, за допомогою циклу for(), в кожній ітерації циклу зміщаючи значення column вправо. Після цього добавляю підсумок, переходжу на один рядок вниз та повертаю його з методу.

В методі AddTotalToWorkheet(), відбувається формування підсумку по проектах. В методі перебираються підсумки по днях і вносяться в тому випадку, якщо вони не дорівнюють нулю, після чого добавляється основний підсумок.

В методі AddProjectsToBodyWorkheet(), відбувається формування всіх проектів репорту. Для початку створюється змінна “column”, якій я присвоюю початкове значення 1. Після чого починаю перебір всіх проектів користувача в репорті в методі foreach(). Спочатку заношу назву проекту, далі перебираю дні, вносячи години тільки в тому випадку, якщо вони не дорівнюють 0. Для кожного дня при потребі добавляю стилізацію, для вихідних присвоюю сірий колір клітинки.

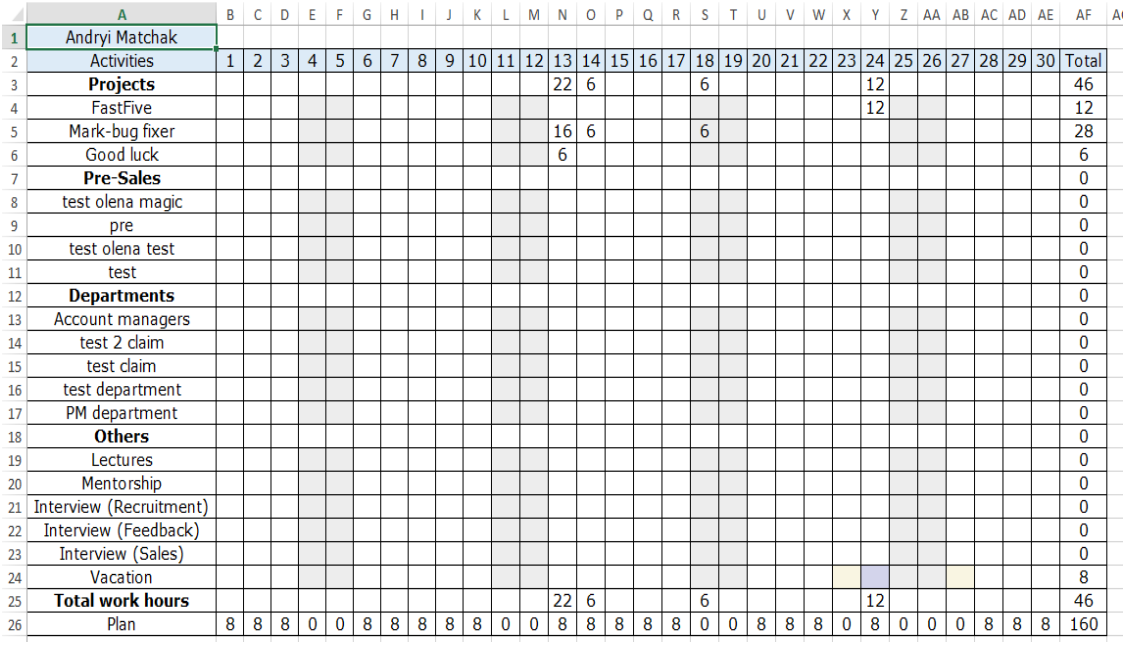
В методі AddVacationsToBodyWorkheet(), додаються всі відпустки, вихідні, та святкові дні які є в репорті. Для цього роблю перебір масу з типами днів, та за допомогою switch(), роблю перевірку на відповідне значення, вносячи необхідну стилізацію.

В методі AddPlanToWorkheet(), через перебір масиву з днями, записую план виконання.

В методі AddStyleToWorkheet(), за допомогою властивості робочого листа SelectedRange[], вибираю ввесь репорт, та задаю йому стилі шрифта.

Коли буде повністю внесений репорт, зберігаю зміни та перетворюю його в тип “MemoryStream’’, який повертаю з методу.

В контролері, створюю відповідь про успішне виконання, з типом файлу та його назвою, після чого виконання повністю завершується (рис. 3.2).



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF
1	Andryi Matchak																															
2	Activities	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	Total
3	**Projects**													22	6				6						12							46
4	FastFive																								12							12
5	Mark-bug fixer													16	6				6													28
6	Good luck													6																		6
7	**Pre-Sales**																															0
8	test olena magic																															0
9	pre																															0
10	test olena test																															0
11	test																															0
12	**Departments**																															0
13	Account managers																															0
14	test 2 claim																															0
15	test claim																															0
16	test department																															0
17	PM department																															0
18	**Others**																															0
19	Lectures																															0
20	Mentorship																															0
21	Interview (Recruitment)																															0
22	Interview (Feedback)																															0
23	Interview (Sales)																															0
24	Vacation																															8
25	**Total work hours**													22	6				6						12							46
26	Plan	8	8	8	0	0	8	8	8	8	8	0	0	8	8	8	8	8	0	0	8	8	8	0	8	0	0	0	8	8	8	160

Рисунок 3.2 – Excel репорт по користувачу

3.4.2 Експорт репорту по проектах

Excel репорт по проекту. При створенні цього функціоналу, розроблено такий самий початок, як і при репорті по користувачу. В класі “ProjectReportExcelExporter”, розділено створення репорту та занесення в репорт. В методі формування excel документу GenerateExcelProjectReport(), створюється новий документ та робочий лист з назвою місяця. Створюються змінні з стартовим рядком, місяцем репорту та кількістю днів в місяці. Для побудови репорту використовуються такі методи:

- AddHeaderToWorkheet() – використовується для добавлення заголовку до репорту;
- AddAllUsersToBodyWorkheet() - використовується для добавлення основного тіла репорту з користувачами;
- AddTotalByProject() – використовується для добавлення підсумку по проекту;

- AddStyleToTable() – використовується для стилізації excel документу.

В методі AddHeaderToWorkheet(), вноситься сталий заголовок з кількістю днів, в залежності від місяця. Використовується той самий принцип, що і при формуванні заголовку, в репорті по користувачу.

В методі AddAllUsersToBodyWorkheet(), відбувається перебір користувачів, та внесення основних данних про них. Використовується стилізація в залежності від типу дня. Якщо при занесенні дня, в користувача є оплачувана відпустка, підсумок дня буде більший на 8 годин, тому вони віднімаються. Для того щоб при зміні даних, в excel документі, мінялись значення відповідно до формул, використовується властивість клітинки “FormulaR1C1[]”. В цій властивості, задається така ж сама формула, що і в excel, з можливістю вказувати клітинки відносно вибраної. Через R[], вказується зміщення по рядках, а через C[], по колонках. Так задається, що підсумок по годинах повинен містити підсумок по всіх днях, а загальний підсумок, кількість робочих годин та відпусток. За допомогою методу Calculate(), відбувається підрахунок даних по вказаних формулах.

В методі AddTotalByProject (), вносяться підсумки, перебираючи дні, та для кожного дня, добавляючи формулу, для перерахунку підсумку по днях у всіх користувачів. В формулі, відбувається зсув на кількість користувачів в репорті вгору. Після чого, стилізаця, відповідно до типу дня та для кожного загального підсумку, задається формула, яка базується на сумі підсумків всіх користувачів репорту.

Подальші дій, такі самі, як і при репорті по користувачу (рис. 3.3).

A1 | FastFive

FastFive	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	Hours + Overtime	Overtime	Others	Vacation	Total	Plan
Білик Марк																								14			8	8			30	0	0	0	30	160
Ivakhiv Bohdan																															0	0	9	0	9	160
Chief Maxim																															0	0	0	0	0	160
Івахів Богдан																															0	0	0	0	0	160
bi mark																															0	0	0	0	0	160
Total	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	0	0	8	8	0	0	30	0	9	0	39	800

Рисунок 3.3 – Excel репорт по проекту

Excel експорт детального репорту по проекті

Для створення детального репорту по проекті, в контролері, було створено метод GetReportInExcelBySingleProject(), який приймає Id проекту, стартову і кінцеву дату репорту. Далі отримується Id користувача, який надіслав запит, та перевіряється право на отримання репорту за допомогою методу CheckIfHaveAccessToProjectReport(), сервісу “TimeLogService”. Якщо права в доступі підтверджено, відбувається конвертація дат в формат “DateTime”, та виклик методу GetSingleProjectExcelReport(), для генерації репорту в excel документі. В методі виконують описані раніше дії, для формування детального репорту по користувачу, після чого, він передається параметрами в метод GenerateExcelSingleProjectReport(), сервісу “GenerateProjectExcelTableService”. В методі, створено змінну “counter”, для визначення користувача, так як в репорті, вони ітимуть в дві колонки. Далі, відбувається створення самого документу та перебір репортів по окремому місяці, створюючи для кожного окремий робочий лист та задаючи стартову колонку та стовпчик. Починається перебір користувачів, для кожного формуючи окрему таблицю, за допомогою таких методів:

- AddHeaderToWorkheet() – відповідає за добавлення заголовку до таблиці, і є перезагрузкою методу для створення заголовку, для звичайного репорту по проекті;
- AddUserToBodyWorkheet() – відповідає за додавання даних по залогованих годинах та опису днів користувача до таблиці;
- AddTotalByUser() – відповідає за добавлення підсумку по конкретному користувачу до таблиці.

В методі AddHeaderToWorkheet(), задається ім’я користувача, одразу зі стилізацією. За допомогою властивості “Merge”, поєднуються 4 колонки в одну, задається стиль шрифту та рамка. Далі створюється заголовок з такими параметрами: Date, Work Description, Hours, Overtime. Для цього, ініціалізується

масив, з цими заголовками, після чого заноситься в документ за допомогою циклу, додаючи для кожного стилізацію.

В методі AddUserToBodyWorkheet(), відбувається перебір днів користувача, для занесення дня з формату “DateTime”, клітинці задається потрібний формат, за допомогою стилізації “Numberformat.Format”. Далі, вноситься опис та години, після чого, через блоки умови, задається відповідний до типу дня колір клітинок. Після занесених днів, з методу повертається рядок, на якому зупинилось виконання.

В методі AddTotalByUser(), без жодних циклів та блок умов, вноситься підсумок по користувачу.

Так як ширина кожної таблиці користувача є фіксованою, в кінці циклу перевіряємо, чи є наша змінна “counter”, парним числом, і в залежності від цього, присвоюємо стартову позицію для наступного користувача.

Після формування таблиць для всіх користувачів, за допомогою методу AddTotalByAllUsers(), добавляємо підсумок для всіх користувачів.

В методі AddTotalByAllUsers(), об’єднуться 4 колонки, для яких задається текст "SUBTOTAL" та стилізація. Далі, знову починається перебір підсумків по користувачу, з внесенням імені.

Після створення таких репортів для всіх місяців, відбувається збереження документу та ті самі дії, що і в звичайному репорті по проекті (рис. 3.4) (рис. 3.5).

	A	B	C	D	E	F	G	H	I
1	Білик Марк					Ivakhiv Bohdan			
2	Date	Work Description	Hours	Overtime		Date	Work Description	Hours	Overtime
3	03.04.2020		0	0		03.04.2020		0	0
4	04.04.2020		0	0		04.04.2020		0	0
5	05.04.2020		0	0		05.04.2020		0	0
6	06.04.2020		0	0		06.04.2020		0	0
7	07.04.2020		0	0		07.04.2020		0	0
8	08.04.2020		0	0		08.04.2020		0	0
9	09.04.2020		0	0		09.04.2020		0	0
10	10.04.2020		0	0		10.04.2020		0	0
11	11.04.2020		0	0		11.04.2020		0	0
12	12.04.2020		0	0		12.04.2020		0	0
13	13.04.2020		0	0		13.04.2020		0	0
14	14.04.2020	Day Off;	0	0		14.04.2020		0	0
15	15.04.2020		0	0		15.04.2020		0	0
16	16.04.2020		0	0		16.04.2020		0	0
17	17.04.2020		0	0		17.04.2020		0	0
18	18.04.2020		0	0		18.04.2020		0	0
19	19.04.2020		0	0		19.04.2020		0	0
20	20.04.2020		0	0		20.04.2020		0	0
21	21.04.2020		0	0		21.04.2020		0	0
22	22.04.2020		0	0		22.04.2020		0	0
23	23.04.2020	Holiday;	0	0		23.04.2020	Holiday;	0	0
24	24.04.2020	bkjbk;	8	0		24.04.2020		0	0
25	25.04.2020		0	0		25.04.2020		0	0
26	26.04.2020		0	0		26.04.2020		0	0
27	27.04.2020	Holiday;	8	0		27.04.2020	Holiday;	0	0
28	28.04.2020	fkgujfj;	8	0		28.04.2020		0	0
29	29.04.2020		0	0		29.04.2020		0	0
30	30.04.2020		0	0		30.04.2020		0	0
31		Total:	24	0			Total:	0	0

Рисунок 3.4 – Користувачі в детальному Excel репорті по проекту

95	Total:	0	0
96			
97	SUBTOTAL		
98	Білик Марк	24	0
99	Ivakhiv Bohdan	0	0
100	Chief Maxim	0	0
101	Івахів Богдан	0	0
102	bi mark	0	0
103			

Рисунок 3.5 – підсумок в детальному Excel репорті по проекту

Ексель репорт по проектах адміністратора. Для формування репорту по проектах адміністратора, розроблено метод GetAdminProjectsExcelReport(). В ньому, отримується масив місяців вибірки та за допомогою описаного раніше методу GenerateAdminProjectsReport(), формується репорт, та передається в метод GenerateAdminExcelProjectReport(), для занесення його в excel документ.

В методі виконуються ті самі методи та алгоритми для формування репорту, що і при занесенні репорту по одному проекту.

Ексель репорт по всіх проектах. Для формування репорту по проектах адміністратора, я створено метод GenerateExcelAllProjectsReport(). В ньому, отримується масив місяців вибірки, та за допомогою описаного раніше методу GenerateAllProjectsReport(), фомується репорт, та передається в метод GenerateExcelAllProjectsReport(), для занесення його в excel документ. В методі створюються робочі листи та починається перебір АМ-ів, для кожного вносячи данні, по тому самому принципу, що і для репорту по проектах адміністратора.

3.4.3 Експорт загального репорту

Для створення репорту по всіх користувачах, створено метод GetAllUsersExcelReport(), в якому спочатку генерується репорт по всіх користувачах, а потім передається в метод GenerateExcelAllUsersReport()(таб. 3.1).

Далі, репорт перебирається по місяцях, для кожного дадається робочий лист, з назвою місяця, та викликаються на ступні методи:

AddHeaderToWorkheet() – метод, для створення заголовку репорту. В ньому, розділені на дві окремі частини, фіксовані заголовки, які будуть стояти до і після заголовку з іншою активністю користувача. Визначається кількість інших активностей, та занесення їх в змінну ”countOthers”, після чого, перебір першої частини заголовку. Для кожного значення, необхідно поєднати клітинку з нижньою, за допомогою властивості “Merge”. Після занесення, вноситься заголовок, і з’єднюються клітинки вбік, на кількість інших активностей, а внизу, вони виводяться за допомогою циклу. Після цього, вноситься другу частину заголовку. Після з’єднання колонок, стає відсутньою можливість автопідбору ширини, тому для кожної позиції, задається ширина вручну.

AddAllUsersToBodyWorkheet() – метод, для занесення всіх користувачів в тіло таблиці. Так як всі значення сталі, задано фіксовану позицію для коментарів, та стартову позицію, для занесення користувачів. Далі, відбувається перебір всіх користувачів, вносячи дані в таблицю, добавляючи стилізацію. Було створено цикл для перебору підсумку, і в задежності від значення, в ньому добавляється колір до тексту.

AddTotalToWorkheet() – метод, для занесення підсумку по всіх користувачах. В методі, відбувається внесення підсумку, для кожного поля, встановлюються формули, для динамічної зміни значень, та додається стилізація.

AddFormulas() – метод, для занесення формул для кожного користувача, для цього, робиться перебір всіх користувачів, для кожного додаючи формули для підсумків.

Таблиця 3.1 – Загальний Excel репорт

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1							Others										
2	№	Name	Project Hours	Project Overtime	Pre-Sales	Departments	Interview (Feedback)	Interview (Recruitment)	Interview (Sales)	Lectures	Mentorship	Vacation	Total	Plan	Total out of Plan (%)	Difference	Comments
3	1	AM Olena	39	0	8	19	1	2	1	0	3	0	73	160	45,6%	-87	гшгншгнш
4	2	Bi M	0	0	0	3	0	0	0	0	0	0	3	160	1,9%	-157	try rublia
5	3	bi mark	21	5	0	0	0	0	0	0	0	0	26	160	16,3%	-134	asdS
6	4	BIbi BIBIbi	0	0	0	0	0	0	0	0	0	0	0	160	0,0%	-160	
7	5	Bordun Ihor	261,5	0	13	8	0	0	0	0	0	8	290,5	160	181,6%	130,5	sdfsd
8	6	Chief Maxim	0	0	0	0	0	0	0	0	0	0	0	160	0,0%	-160	
9	7	Chipak Ihor	0	0	0	0	0	0	0	0	0	0	0	160	0,0%	-160	
10	8	Developer Olena	33	9	0	4	0	0	0	0	0	8	54	160	33,8%	-106	негенгнег
11	9	HR Olena	0	0	0	2	0	0	0	0	0	0	2	160	1,3%	-158	fsdfsef
12	10	HR Наталя	17	0	0	0	0	0	0	0	0	16	33	160	20,6%	-127	
13	11	Husnay Tatiana	0	0	0	0	0	0	0	0	0	0	0	160	0,0%	-160	
14	12	Ipsum Lorem	0	0	0	0	0	0	0	0	0	0	0	160	0,0%	-160	
15	13	Ivakhiv Bohdan	153	5	24	23	3	5	0	1	0	0	214	160	133,8%	54	ппавпавпвап
16	14	Marian - Makar Bilyk	256,5	5	0	68,5	59	50	55	49	67	0	610	160	381,3%	450	SDfSD
17	15	Matchak Andryi	21	0	0	0	0	0	0	0	0	0	21	160	13,1%	-139	
18	16	Matchak Andryi	78	0	0	40	0	0	0	0	0	8	126	160	78,8%	-34	opopioppo
19	17	PM Olena	6	3	0	0	0	0	0	0	0	0	9	160	5,6%	-151	zsdgzdfdfvdfv
20	18	PM Тарас	28,5	11	0	8	0	0	0	3	0	16	66,5	160	41,6%	-93,5	
21	19	Portal InVerita	0	0	0	0	0	0	0	0	0	0	0	160	0,0%	-160	ghhfghfghgfh
22	20	Pylypchuk Taras	67	2	0	0	0	0	0	0	0	8	77	160	48,1%	-83	некнекнкен
23	21	Shevchenko Mike	23	0	0	0	0	0	0	0	0	0	23	160	14,4%	-137	
24	22	Білик Марк	69,5	0	0	6	0	0	0	0	0	0	75,5	160	47,2%	-84,5	uyytutyuu
25	23	Івахів Богдан	16	0	0	0	0	0	0	0	0	0	16	160	10,0%	-144	
26	24	Куціль Микола	0	0	0	0	0	0	0	0	0	0	0	160	0,0%	-160	
27	25	Пилипчук Тарас	24	0	0	52	15	0	8	7	0	0	106	160	66,3%	-54	
28			**1114**	**40**	**45**	**233,5**	**78**	**57**	**64**	**60**	**70**	**64**	**1826**	**4000**	**45,6%**	**-2174,5**	

3.5 Логування часу

Для того щоб логувати робочі години в контролері “TimeLogController”, було створено два методи.

Метод Create(), приймає модель даних “TimeLogIncomeModel”, перевіряє на помилки, дізнається Id користувача, який надіслав запит. Після чого за допомогою методу CheckIfUserHaveAccess(), робиться перевірка на те чи можливе логування часу. Сам метод CheckIfUserHaveAccess(), розміщено в сервісі “TimeLogService”. Спочатку в методі, за допомогою методу CheckIfTimeLogPeriodClosed(), який приймає день на який логується час, дізнаємося чи дозволено логувати час на даний день. Далі, з бази данних отримуємо проект, на який логуються години. За допомогою блоків умов генеруємо помилки про відмову в доступі. Якщо ж жоден з блоків умов не спрацював, виконання методу завершується, і вважається що логування часу дозволено.

Далі, викликається сам метод логування часу Create(), сервісу “TimeLogService”. В методі перебирається масив таймлогів, для кожного, за допомогою методу CheckIfHoursValid(), проходить валідація часу. Далі, таймлоги логуються в базу даних, за допомогою методу CreateTimelogEntities(). В методі, створюється масив, в який в подальшому додаватиму створені таймлоги. Спочатку проводиться перевірка, чи потрібно створювати основні години, якщо потрібно, створюється модель “TimeLog”, та додається в створений масив. Після цього, в блоці умови, потрібно також перевірити необхідність створення понаднормових годин, і при потребі створити. Якщо ж у моделі немає звичайних годин, але є понаднормові, то спочатку перевіряється, чи створені звичайні години, якщо так, тоді логуються понаднормові, в іншому випадку генерується помилка. При створенні понаднормових годин, за допомогою методу GetAdminFromProject(), визначається адміністратор, відповідальний за підтвердження годин, і якщо він є, то присвоюється статус квитка “Pending”, в іншому випадку “Approved”. З годинами очікування, роблю ту саму перевірку і

створення. Далі всі таймлоги з масиву додаються в базу даних, методом AddRange(), і фіксую зміни. Після цього, з методу повертаються 3 таймлоги. Якщо він був створений, то повернеться сам таймлог, якщо ні, то null.

Після створення, якщо створювались звичайні години, оновлюються дні відпустки користувача. Якщо створювались понаднормові і їх потрібно підтвердити адміністратором, за допомогою створених раніше методів, створюються сповіщення для адміністраторів та розсилаються поштові повідомлення.

Далі, відбувається створення моделі відповіді, в яку поміщається Id таймлогів та повідомлення, через хаб, відповідальних людей, після чого надсилається відповідь.

Метод Update(), приймає модель даних “UpdateTimeLogIncomeModel”, і спочатку виконує ту саму перевірку на дозвіл оновлення таймлогу, що і при створенні. Далі, перевіряється чи були створені тайлоги, які потрібно оновити, якщо ні, то створюється помилка з відповідним текстом. Далі, відбувається оновлення самого таймлогу, і ті самі дії, з надсиланням сповіщень та оновлення днів відпусток користувача, що і при створенні, надсилання відповіді.

3.6 Відпустки та Свята

В “VacationController”, було створено методи Create(), Update(), Delete(), UpdateStatus(), GetAllUserVacation() і GetUserVacationsForAdmin().

В методі Create(), отримується вхідна модель створення відпускти, перевірка запиту на наявність помилок, та за допомогою платформи “Identity”, отримання Id користувача, який відправив запит. Після цього, дані передаються в метод CreateVacation(), сервісу “VacationService”. На початку методу, розміщено блоки умов, для перевірки, чи задовольняють вхідні данні запланованим вимогам. Далі, якщо відпустка оплачувана, то потрібно перевірити, чи після її створення, не буде перевищений максимальний ліміт днів.

Якщо відпуска включає додаткові дні, то виконуєм ті ж самі дії, але з врахуванням додаткових днів користувача. Якщо відпустка не оплачувана, додаткових перевірок не потрібно. В разі, якщо якась з наведених вище перевірок не задовольняється, використовуємо розроблений нами статичний клас “ObjectExistenceChecker” для генерації помилок. В разі задоволення всіх вимог, створюємо нову модель відпустки, присвоюємо їй потрібні данні та добавляємо її в базу даних. Після успішного створення відпуски, потрібно визначити відповідальних людей, які повинні дозволити користувачу дану відпустку. Для цього, створено метод GetAdminUsersOnActiveProjects(), в якому за допомогою блоків умови, визначаються адміністратори, на проектах користувача. Після визначення відповідальних користувачів, видаляються ті Id, які повторюються і повертаються з методу. Далі, надсилаю повідомлення на пошту за допомогою розробленого сервісу “EmailService”, і створюю сповіщення за допомогою сервісів “NotificationService” і “UserNotificationService”. Після повернення даних з основного методу CreateVacation(), використовуємо розроблений нами “TicketNotificationsHub”, для того щоб проінформувати клієнт частину адміністраторів, про створення квитка підтвердження відпустки.

В методі Update(), виконуємо ті самі перевірки і дії, що при створенні, тільки замість створення нової відпуски, оновлюємо вже створену.

В методі Delete(), видаляємо створену раніше відпустку і інформуємо відповідальних людей про це, за допомогою розроблених сервісів.

Метод UpdateStatus(), відповідає за підтвердження адміністраторами відпуски. Він працюватиме на основі розробленого раніше рішення, з зміною статусу моделі “TicketAdminStatus”, і при потребі зміною статусу моделі “Ticket”. В методі контролера перевіряється модель та викликається метод UpdateVacationStatusByAdmin(), в який передаються вхідні данні. В самому методі, з бази отримуємо користувача та відпустку, після чого створюємо модель “UpdateVacationTicketStatusModel”, яку передаємо в сервіс “TicketService”, для зміни статусу квитка. Далі перераховуємо користувачу доступні дні відпустки за

допомогою методу SetUserVacation(). Метод працює по спроектованому раніше алгоритму нарахування, де спочатку визначає кількість нарахованих днів за допомогою методу CalculateUserVacationCount(), а потів по умові розділяє на теперішній та минулий роки за допомогою методу UpdateUserVacations(). Після оновлення днів відпусток, відбувається надсилання сповіщень про підтвердження відпустки.

Через метод GetAllUserVacation(), отримуються всі відпустки, які є в користувача, кожна з яких представлена моделлю “VacationViewModel”.

Через метод GetUserVacationsForAdmin(), отримуються всі квитки підтвердження відпусток для даного користувача. Так як в основі лежать квитки, а не відпустки, то для початку потрібно отримати з бази даних всі квитки адміністраторів, які прив’язані до даного користувача. Потім перебираю отриманий масив квитків в циклі foreach(), для кожного формуючи модель ”VacationModelForAdmin”. Сформований масив моделей відправляється назад.

Свята. Для свят, було створено окремий контролер “HolidayController”, в якому є методи для створення, оновлення, видалення та отримання всіх вихідних.

Метод GetList(), перевіряє запит на помилки і якщо їх не виявлено, віддає модель всіх вихідних, посортованих по даті створення.

Метод Create(), створює в базі данних нове свято, після чого надсилає повідомлення про створення через пошту. та створює нове сповіщення в базі данних всім незаблокованим користувачам. До цього методу надається доступ лише тим користувачам, які мають Holidays Management Claim. Модель яку приймає данний метод містить назву свята, стартову та кінцеву його дату. На назві свята, я поклав обмеження в 200 символів.

Метод Update(), оновлює в базі данних свято, після чого надсилає повідомлення про оновлення через пошту. та створює нове сповіщення в базі данних всім незаблокованим користувачам. До цього методу надається доступ лише тим користувачам, які мають Holidays Management Claim. Модель яку

приймає данний метод містить ті ж самі данні що і при створенні але добавилось лише Id свята.

Метод Delete() видаляє в базі данних свято, після чого надсилає повідомлення про видалення через пошту. та створює нове сповіщення в базі данних всім незаблокованим користувачам. До цього методу надається доступ лише тим користувачам, які мають Holidays Management Claim. В параметрах метод приймає “holidayId”.

3.7 Сповіщення

3.7.1 Створення сповіщень в базі

Для створення сповіщень в сервісі “NotificationService”, було створено метод CreateNotificationForUsers(), який приймає в параметрах масив користувачів, для яких потрібно створити сповіщення, текст та тип сповіщення. Для більш лаконічного представлення типу сповіщення, використовується “Enum”. В методі створюється саме сповіщення, заголовок якого визначається по типу сповіщення в методі GetTitleFromType(). Після успішного створення сповіщення, потрібно прикріпити його до конкретних користувачів, тому в сервісі “UserNotificationService” створюємо метод CreateList(). В методі для кожного користувача створюється “UserNotification”, де вказується Id попередньо створеного сповіщення.

Для того щоб отримувати сповіщення, в контролері створюємо метод GetList(). В методі будемо використовувати пагінацію, для того щоб віддавати данні частинами. Тому, в самому методі отримаємо номер теперішньої сторінки і кількість елементів на одну сторінку. Після цього, передаємо ці данні в метод GetList(), сервісу “UserNotificationService”. В методі отримуємо всі сповіщення, які створені на даного користувача, відсоротовані по даті створення і

пропускаємо від початку потрібну кількість елементів. Для кожного сповіщення створюємо вихідну модель даних та відправляємо в відповіді.

Також в контролері добавляємо два методи для видалення сповіщень. В одному прийматиметься Id сповіщення і відбуватиметься видалення одного сповіщення. В іншому прийматиметься масив Id і вібуватиметься видалення кількох сповіщень.

3.7.2 Поштові повідомлення

Для роботи з поштою, вибрано клас SmtpClient. Він надає можливість надсилання пошти та контролю процесу.

Конфігурацію цього сервісу, було записано в Web.config, де описано host, підтимку SSL сертифыкату та порт.

Після цього, створено метод InitializeSmtpClient(), який повертає ініціалізований SmtpClient, з настроєною конфігурацією. Метод SendEmailToGroup(), відповідає за відправку повідомлення. Він приймає заголовок і тіло повідомлення, які присвоює класу “MailMessage”. Клас “MailMessage”, являє собою саме повідомлення, основу якого складає тіло і додаткові параметри.

Формування шаблону повідомлення відбувається за допомогою методів сервісу “EmailTemplateService”, та HTML шаблонів. Було розроблено наступні методи для вирішення поставлених завдань:

- GetHolidayTemplate() – метод формування повідомлення, які стосуються свят;
- GetProjectTemplate() – метод формування повідомлення, які стосуються проектів;
- GetTimelogTemplate() - метод формування повідомлення, які стосуються таймлогів;

- GetVacationTemplate() - метод формування повідомлення, які стосуються відпусток.

Кожен метод повертає індивідуальну структуру повідомлення, використовуючи вхідні параметри та клас “EmailNotificationResources”, в якому заданий текстові ресурси для повідомлень.

3.7.3 SignalR

SignalR це технологія, яка дозволяє взаємодіяти з іншими клієнтами в режимі реального часу. В її основі стоять хаби, які постають в ролі ретрансляторів повідомлень і працюють через сокети. Хаби побудовані по патерну Singelton, і він зберігає спільний контекст для всіх запитів.

Розробку SignalR, розпочато з класу “ConnectionMapping”, який стане основою для контролю користувачів та стану інших хабів. В ньому створено методи для отримання, добавлення та видалення користувачів хабів.

Наші хаби, наслідуватимуться від базового класу “Hub” та інтерфейсу, через який надаватиметься доступ до нього. Базовий клас “Hub”, надає методи для ретрансляції повідомлень, контроль груп та інший функціонал.

“TicketNotificationsHub”, міститиме два методи для ретрансляції повідомлень.

Перший метод, SentTimelogTicketNotification(), відповідає за відправку повідомлення адміністратору, на якого створено квиток підтвердження залогованих понаднормових годин. Метод приймає Id адміністратора, Id таймлогу, та тип операції над залогаваними годинами. Спочатку, щоб не розміщати перевірку, чи Id адміністратора не порожній, в місцях де викликається метод, я розмістив її в метод. Це допоможе скоротити дублювання коду. Далі створюємо модель повідомлення “HubMessageViewModel”, в яку присвоюємо отримані в методі дані. Далі пробуємо отримати зв'язок з данним адміністратором, якщо адміністратор на данний момент підключений до нашого

хабу, то відправляємо повідомлення, в іншому випадку, завершуємо виконання методу.

Другий метод я назвав SentVacationTicketNotification(). Він відповідає за відправку повідомлення адміністраторам, на яких створено квиток підтвердження відпустки. Метод приймає Id адміністраторів, Id відпуски, та тип операції над відпускою. Спочатку, щоб не розміщати перевірку, чи список Id адміністраторів не порожній, в місцях де викликається метод, я розмістив її в метод. Це допоможе скоротити дублювання коду. Далі створюємо модель повідомлення “HubMessageViewModel”, в яку присвоюємо отримані в методі дані. Далі починаємо пребір Id в масиві, і для кожного Id виконуємо ті ж самі дії, що і в першому методі.

“NewsNotificationsHub” міститиме два методи для ретрансляції повідомлень.

Перший метод я назвав SendNewsNotification(). Він відповідає за відправку повідомлення всім користувачам, які підключені до хабу, окрім користувача який ініціював створення сповіщення. Метод приймає Id користувача який ініціював створення сповіщення. Спочатку створюємо модель повідомлення “ApiResponseModel” з повідомленням успішного виконання. Далі отримуємо всі підключення, і з них за допомогою методу AllExcept(), видаляємо Id даного користувача. Після цього надсилаємо повідомлення всім користувачам з масиву.

Перший метод я назвав SendNewsNotificationToGroup(). Він приймає масив Id користувачів, яким потрібно відправити повідомлення, створює модель повідомлення “ApiResponseModel” з повідомленням успішного виконання і виконує такі ж самі дії, що і в методі SentVacationTicketNotification() класу “TicketNotificationsHub”.

3.8 Google Calendar

При використанні Google Calendar, BackEnd нашої програми стає посередником для доступу до Google Calendar Api. Для доступу до Google Calendar Api, використовується Google.Apis.Calendar.v3, яку надає Google.

Для ініціалізації данного сервісу, створено окремий метод Init(). Спочатку потрібно дадати в клас GoogleCredential, данні нашого сервісного акаунту, та ключ нашої програми. Також добавляємо “Scope” на відкриття на зчитування, які формують наші права при доступі до Google Calendar Api.

Метод GetList(), отримує Id календаря та дати вибірки, перевіряє запит на помилки, і якщо їх не виявлено, ініціалізує Calendar сервіс та відправляє запит на отримання всіх подій, які створені на календар з данним Id. Після чого методом Execute(), відбирає тільки ті події які входять в вибірку.

Метод Create(), отримує модель данних для створення, перевіряє запит на помилки, і якщо їх не виявлено, ініціалізує Calendar сервіс. Проходить перетворення вхідної моделі в модель Event, яка надана бібліотекою Google.Apis.Calendar.v3.Data. Після чого методом Insert(), відправляє запит на створення.

Метод Update(), отримує модель данних для оновлення, перевіряє запит на помилки і якщо їх не виявлено, ініціалізує Calendar сервіс. Проходить перетворення вхідної моделі в модель Event, яка надана бібліотекою Google.Apis.Calendar.v3.Data. Після чого методом Patch(), відправляє запит на оновлення.

Метод Delete(), отримує Id календаря та Id події, перевіряє запит на помилки і якщо їх не виявлено, ініціалізує Calendar сервіс та відправляє запит на видалення події з данним Id, які створені на календар з данним Id.

Також, сервіс “CalendarService”, містить метод для перевірки наявності календаря, та метод його створення.

Метод CheckIfCalendarExists(), приймає параметрами email користувача, ініціалізує сервіс. Після цього виконується запит на отримання календарів

користувача. Якщо відповіді немає, це означає що в користувача відсутній календар, і викликається метод Create().

Метод Create() приймає параметрами email користувача, ініціалізує сервіс. Далі створюється модель календаря з, namespace “Google.Apis.Calendar.v3.Data”, вноситься часова зона, загальні дані та тип. Після цього надсилається запит на створення календаря та добавляється отримане Id календаря до відповідного поля в моделі користувача.

3.9 Активності

3.9.1 Проекти

Для керування проектами, в контролері “ProjectController”, Було створено наступні методи:

- GetNamesList() – метод, для отримання назв існуючих проектів;
- GetCountriesList() – метод, для отримання назв країн;
- GetIndustriesList() – метод, для отримання назв існуючих індустрій;
- GetList() – метод, для отримання всіх проектів в моделі “GetProjectViewModel”;
- Get() – метод, для отримання одного проекту в моделі “ProjectViewModel”;
- Create() – метод, для створення нового проекту;
- Update() – метод, для оновлення існуючого проекту;
- Delete () – метод, для видалення існуючого проекту.

У кожному методі, викликається відповідний метод сервісу “ProjectService”.

Для методу GetNamesList(), викликається метод GetProjectsNamesList(), в якому з бази даних вибираються всі проекти, посортовані по імені та статусі. Для

того, щоб зробити правильне сортування, використано метод Concat(), який дозволяє об’єднати запити.

Для методу GetCountriesList(), викликається метод GetCountriesList(), в якому отримуютється, створений раніше, масив країн.

Для методу GetIndustriesList(), викликається метод GetIndustriesList(), в якому отримуютється, створений раніше, масив індустрій.

Для методу GetListi(), викликається метод GetListByFilter(), в якому використовується пагінація через вхідну модель. Спочатку отримуємо масив проектів з бази даних, відсортовані по даті створення та типу. Після чого, робиться вибірка з масиву, для кожної властивості вхідної моделі, задається можливість бути null, якщо ж вона є, то фільтрується по ній. Далі пропускається необхідна кількість елементів, та для тих, що потрібно повернути, відбувається створення моделі “ProjectViewModel”, та задаються додаткові параметри про них, за допомогою методу колекції Count().

Для методу Get(), викликається метод GetNotDeletedProject(), в якому за допомогою методу GetProjectById(), отримується проект, а за допомогою GetProjectViewModelByProject(), формується модель “ProjectViewModel”. При формуванні моделі використовуються методи GetUserName(), GetAllTechnologyByProjectId(), GetProjectMembersInfo().

Метод GetUserName(), приймає параметром Id користувача і отримує з бази даних його назву.

Метод GetAllTechnologyByProjectId(), приймає параметром Id проекту і отримує з бази даних, назви технологій які використовуються.

Метод GetProjectMembersInfo(), приймає параметром Id проекту і якщо на проекті є користувачі, за допомогою методу GetUserIdListFromTeamByRole(), вибираю Id всіх звичайних користувачів типу “Member”. Після чого формує потрібну модель даних “ProjectUserModel”.

Для методу Create(), викликається метод CreateProject(), який приймає параметром вхідну модель “CreateProjectIncomeModel” та Id користувача. В

методі, в базі даних створюється проект, та фіксуються зміни. Після успішного створення самого проекту, відбувається створення його команди, за допомогою методу CreateTeamForProject(). В методі, за допомогою методу CreateTeamEntity(), сервісу “TeamService”, відбувається створення команди, до якої додаються відповідні користувачі, з їх ролями, за допомогою методу AddUserInProjectTeam(), сервісу “TeamUserService”. Після створення проекту та команди, команда повідомляється про їх залучення до проекту, за допомогою сповіщень та надсилання пошти.

Для методу Update (), викликається метод UpdateProject(), який приймає параметром вхідну модель “UpdateProjectIncomeModel” та Id користувача. В методі, з бази даних отримуємо проект і перевіряємо, чи він уже створений. Далі, за допомогою методу CheckUserRightsForModifyingProject(), перевіряю права на оновлення проекту. Якщо це суперадмін, або користувач з встановленою на стадії планування роллю на проекті то дозволяю оновлення.

Після цього, проводимо перевірку на існування команди даного проекту, і якщо вона не існує в базі, генерується помилка, за допомогою класу “ObjectExistenceChecker”. Далі, відбувається оновлення даних проекту з вхідної моделі, та фіксація змін. Найважчим моментом, було реалізувати оновлення команди. Для цього, було створено метод UpdateUsersOnProjects(), в якому з бази даних обираються всі користувачі на даному проекті. Ці користувачі розділяються на звичайну команду та масив AM-ів, PM-ів. За оновлення звичайної моделі, відповідає метод UpdateUsersInTeam(). В ньому отримуються id користувачів, які внесені в базу даних, та Id користувачів команди, з моделі для оновлення. За допомогою методу колекції Except(), знаходиться різність вибірок, тобто користувачів, яких необхідно видалити. Далі, починається перебір цих користувачів, і якщо вони не були занесені в базу даних, як вже видалені, встановлюються даті видалення та даті останньої модифікації, теперішній час. За оновлення одного користувача відповідає метод AddUserToTeamUsers(), який приймає в параметрах Id користувача, якого потрібно оновити або створити, його

роль в команді та список користувачів. В методі, є блоки умови і в першому, повертається “Guid.Empty”, в другому, якщо користувач уже був в команді, встановлюється нова роль, дату модифікації, та занесення полю “DeleteAt” значення null. В іншому випадку, створюється новий учасник команди за допомогою методу CreateTeamUser(), додається в базу та фіксуються зміни. В кожному з блоків повертаються Id користувача, якого оновлювали. Отже, метод AddUserToTeamUsers(), я використовую для керівника команди та звичайних учасників, заносячи Id які повертаються в попередньо створений масив “newUsers”. Також оновлюються АМ та РМ, і так само добавляються в відповідний масив. Після оновлення команди, визначаються користувачі, які були видалені, додані, та ті, що залишились без змін. Для кожного з цих масивів користувачів, створено окремий блок умови, в якому вибирається відповідний текст з класів “NotificationTextResources”, “EmailNotificationResources ” та надсилаються сповіщення, почтові повідомлення.

Для методу Delete(), викликається метод DeleteProject(), в якому отримується проект та перевіряється його наявність базі. Після цього, в методі GetAllUsersEmailsByProject(), отримується Email всіх користувачів на проекті, та за допомогою методу GetAllUsersOnProject(), самих користувачів. Ні сам проект, ні команда, фактично не видаляються. Для них встановлюється дата видалення теперішнім часом, а користувачів повідомляється через сповіщення та поштове повідомлення.

3.9.2 Команди

Для керування командами, в контролері “TeamController”, було створено наступні методи:

- Create() – метод, для створення команди;
- Delete() – метод, для видалення команди;
- GetTeam() – метод, для отримання існуючої команди;

- GetList() – метод, для отримання всіх існуючих команд;
- Update() – метод, для оновлення команди;
- GetTeamByProjectId() – метод, для отримання команди певного проекту.

У кожному методі викликається відповідний метод сервісу “TeamService”.

Для методу Create(), викликається метод CreateTeam(), в якому в базі створюється команда з відповідними параметрами. Далі до неї додаються відповідні користувачі з ролями, відповідно до вхідної моделі, та фіксуються змани. Після створення, за допомогою розроблених методів для сповіщення, повідомляю доданих користувачів, про створення команди.

Для методу Delete(), викликається метод DeleteTeam(), в якому з бази отримуються потрібна команда та користувачі в ній. Далі, для команди та всіх користувачів встановлюється дата видалення теперішнім числом, та за допомогою розроблених методів для сповіщення, повідомляю доданих користувачів, про видалення команди.

Для методу GetTeam(), викликається метод GetNotDeletedTeam(), в якому з бази отримую потрібну команду за допомогою методу GetTeamByTeamId(). Далі, за допомогою методу Get(), формую вихідну модель, та повертаю її відповіддю.

Для методу GetList(), викликається метод GetList(), в якому з бази даних отримую всі команди, які не привязані до проекту та не видалені. Після цього починаю перебір колекції, для кожної команди створюю відповідну модель “GetListTeamViewModel”, та повертаю колекцію команд.

Для методу Update(), я використовую таку саму логіку, що і при оновленні команди проекту.

3.10 Додатковий функціонал

3.10.1 Міграції та Sead метод

При роботі з EF через підхід Code first, який було обрав, ми можемо оновлювати нашу базу данних за допомогою міграцій. Після команди в консолі менеджера пакетів “update-database”, яка оновлює структуру таблиць відповідно до моделей, запускається метод Sead(), який проводить певні зміни з данними в таблицях. Тому для того, щоб при першому створенні бази данних, були автоматично занесені такі данні як Claims, Roles, Skills, Specialties та створення акаунту суперадміна, створюємо методи які відповідатимуть за їх занесення.

В методі AddOrUpdateClaims(), доступний список Claims, які повинні бути автоматично занесены в базу. Спочатку в методі витягуються всі Claims з бази данних, і перевіряє чи співпадають вони зі списком. Ті данні, яких немає в базі, заносяться в неї зі списку.

В методі CreateRoles(), добавляються ролі User, Admin, SuperAdmin.

В методі CreateSuperAdmin(), добавляється акаунт суперадміна, і після чого до цього користувача приєднюється роль SuperAdmin, та добавляються всі доступні Claims.

В методі SetSpecialtiesAnsSkills(), добавляються зі списку спеціальності та навички, які не занесені до бази данних.

В методі SetSystemSetting(), добавляються початкові системні настройки.

3.10.2 Фонові задачі

Для реалізації фонових задач на платформі .NET, було обрано Quartz.NET. Quartz.NET, представляє відкритий фреймворк для виконання дій за розкладом, в середовищі ASP.NET. Для початку було додано Nuget-пакет, який називається по імені фреймворка. Треба визначити тригер, який буде керувати

виконанням роботи, запускати її та конфігурувати. Клас роботи, повинен реалізувати інтерфейс IJob, який визначає метод Execute (), власне виконує деяку роботу. Потім створюємо саму роботу, яка буде виконуватися у вигляді об'єкта IJobDetail.

Кілька слів з приводу можливих налаштувань відправки. Замість хвилинного інтервалу виконання ми можемо поставити ще ряд інших:

- WithInterval (milliseconds): інтервал виконання в мілісекундах;
- WithIntervalInSeconds (seconds): інтервал в секундах;
- WithIntervalInHours (hours): інтервал в годинах;
- WithRepeatCount (number): визначає кількість повторів.

Момент запуску, який задається в тригері викликом StartNow (), також має різні варіанти:

- StartNow (): запуск відразу ж після початку виконання;
- StartAt (): визначає час, коли тригер починає запускати роботу;
- EndAt (): визначає час, коли тригер перестає запускати роботу.

І щоб робота почала виконуватися за розкладом зі стартом додатки, змінив клас Global.asax.cs.

Cron-Expressions - це рядки, які фактично складаються із семи підвиразів, що описують окремі деталі розкладу. Ці суб-вирази відокремлені пробілом та являють собою:

- секунди;
- хвилини;
- години;
- день місяця;
- місяць;
- день тижня;
- рік (необов’язкове поле).

За допомогою описаної вище інстукції, я створив в класі “Scheduler” наступні “Cron” операції:

MoveUsersNotUsedVacationDaysJob() – операція яка виконується першого вересня кожного року. В ній викликається метод MoveUsersNotUsedVacationDays(), в якому для всіх користувачів онуляються поля, які відповідають за нарахування відпусток, а не використані відпустки переносяться в поле кількості відпусток відпусток за минулий рік.

CloseProjects() – операція яка виконується кожного дня о 8 годині. В ній викликається метод CloseProjectsEndTime(), в якому з бази даних отримую всі проекти, в яких дата закінчення стоїть теперішньою датою, та кожному виставляю статус Finished.

CreateNotificationToCloseProject() – операція яка виконується кожного дня о 8 годині. В ній викликається метод CreateNotificationToCloseProject(), в якому отримуємо дату, на місяць меншу ніж теперішній час, та вибираються проекти, в яких від цієї дати не логувалися години. Для оптимізації запиту я використовую підзапит до таблиці з таймлогами. Якщо отриманий масив проектів не порожній, починаю їх перебір в методі foreach(), для кожного створюючи сповіщення з просьбою закрити проект.

CloseAllProjects – операція яка виконується кожного дня о 8 годині. В ній викликається метод CloseProjects(), в якому спочатку отримуємо дату, на місяць меншу ніж теперішній час, та вибираються проекти, в яких від цієї дати не логувалися години. Для оптимізації запиту, було використано підзапити до таблиці з таймлогами. Якщо отриманий масив проектів не порожній, відбувається їх перебір в методі foreach(). В ньому, за допомогою блоків умов, перевідяється наявність менеджера та акаунт менеджера, створюючи для них сповіщення та надсилаю повідомлення на пошту про закриття проекту. Для цього використовую розроблені раніше відповідні методи. В кінці кожної ітерації, зазначаю статус проекту Finished, та дату закриття встановлюю як дату в момент виконання.

UserOnboarding() – операція яка виконується кожного дня о 5 годині. В ній викликається метод UserOnBoarding(), в якому з бази даних вибираються

користувачі, в яких на дану дату закінчується випробувальний термін. Якщо такі користувачі є, для кожного встановлюю дату закінчення випробувального як null, а буліанівському значенню, яке відповідає за те, чи є користувач на випробувальному встановлюю як false.

UpdateUsersVacation() – операція яка виконується кожного дня о 6 годині. В ній викликається метод SetAllUsersVacation(), в якому для всіх користувачів викликається описаний раніше метод нарахування днів відпусток.

Birthday() – операція яка виконується кожного дня о 10 годині. В ній викликається метод SendHappyBirthday(), в якому з бази даних отримуються всі користувачі, в яких день і місяць дня народження, співпадають з теперішньою датою. Для кожного, використовуючи розроблені раніше методи, створюю сповіщення та надсилаю привітання на пошту.

StartNotifyAboutEndOfTrialPeriod() – операція яка виконується кожного дня о 9 годині. В ній викликається метод NotifyAboutEndOfTrialPeriod(), в якому створюються сповіщення та відбувається надсилання поштових повідомлень користувачам, в яких сьогодні закінчується випробувальний термін.

CreateNotifcationTeammateOnVacation() – операція яка виконується кожного дня о 21 годині. В ній викликається метод NotifyAboutTeammateOnVacation(), в якому з бази отримуються користувачі, та перевіряю, чи є в них погоджена відпустка, в якої стартова дата є завтрашнім числом. Якщо є, отримую Id користувачів, які знаходяться з даним в одній команді на проекті, і кожному надсилаю сповіщення з потрібним текстом.

4 ЕКОНОМІКА ПРОЕКТУ

4.1 Загальний огляд процесу розробки та збуту ПЗ

Один з найбільших підводних каменів - хибне уявлення про легкість впровадження корпоративного порталу. Перша і головна проблема - неготовність співробітників користуватися сервісом на всі 100%. Безглуздо вносити в CRM інформацію про половині проведених зустрічей з клієнтами, а другу половину записувати в свій блокнот.

Якщо планується інсталяція серйозного комерційного продукту, то вартість впровадження, іноді може перевищувати вартість самого ПЗ. Такі портали зазвичай охоплюють весь функціонал компанії - від бухгалтерії до служби охорони, - а це означає, що необхідно навчити всіх співробітників компанії роботі з новим інструментом. Таке навчання зажадає найму додаткового персоналу, що теж коштує грошей.

Недешево обходиться і технічна сторона питання - оренда або покупка серверів, оплата послуг технічної підтримки. Запуск будь-якого корпоративного порталу вимагає наявності як мінімум одного системного адміністратора, виділеного на підтримку користувачів. З розвитком сервісу та залученням все більшої кількості співробітників кількість таких адміністраторів буде тільки рости. Оновлення продукту, в більшості випадків платне, про що не варто забувати при плануванні бюджету.

Щоб економічно обгрунтувати дане ПЗ необхідно скласти бізнес план. Створене ПЗ вирішує проблему наявністі експертів для оцінки характеристик програмних продуктів. Основною проблемою такого підходу, є низька інформативність згаданих методів оцінювання та суб’єктивність оцінки. Існуючі методи оцінювання, не викликають позитивних асоціацій і приводять до однобокого сприйняття інформації щодо якості, зокрема у випадках використання програмних продуктів.

Створене ПЗ, можна використовувати для визначення чітких і зрозумілих чисельних вагових коефіцієнтів в межах [0...1], замість застарілих і нечітких лексичних виразів для опису важливості (Низький, Середній, Високий).

ПЗ створюється для широкого спектру компаній які розробляють ПЗ, компанії, які займаються експертною оцінкою програмного забезпечення та сторонніх осіб, які мають бажання перевірити правильність виставлення, або виставити вагові коефіцієнти.

Виробничі потужності - оскільки програмна реалізація є одноразовою, приміщення не вимагається. Передбачається виробнича кооперація з дизайнерськими компаніями, для розробки інтерфейсу потрібного для конкретного користувача. Накладними витратами, є витрати на електроенергію, підтримку робочого середовища, хостинг сервера, інтернет та оплату виробничої кооперації.

Найкраще працюють канали збуту через інтернет. Вони є найдешевшими, оскільки потрібно оплатити тільки хост сервер. При наявності власного сервера необхідний додатковий персонал для його обслуговування та оплата інтернет-з'єднання з "білою" IP-адресою.

4.2 Визначення основних витрат

Трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д$$

t_o - витрати праці на підготовку й опис поставленої задачі;

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{відл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p)$$

q - передбачуване число операторів;

C - коефіцієнт складності програми;

p - коефіцієнт кореляції програми в ході її розробки.

4.3 Моделі для визначення вартості ПЗ

Більшість моделей для визначення вартості ПО може бути зведене до функції п'яти основних параметрів: розміру, процесу, персоналу, середовища і необхідної якості.

1. Розмір кінцевого продукту (для компонентів, написаних вручну), який зазвичай вимірюється числом рядків вихідного коду або кількістю функціональних точок, необхідних для реалізації даної функціональності.

2. Особливості процесу, використовуваного для отримання кінцевого продукту, зокрема його здатність уникати непродуктивних видів діяльності (переробок, бюрократичної тяганини, витрат на взаємодію).

3. Можливості персоналу, який бере участь в розробці ПЗ, особливо його професійний досвід і знання предметної області проекту.

4. Середовище, яке складається з інструментів і методів, використовуваних для ефективної розробки ПЗ і автоматизації процесу. Необхідну якість продукту, що включає в себе його функціональні можливості, продуктивність, надійність і адаптованість. Співвідношення між розраховується вартістю і цими параметрами, може бути записано таким чином :

Трудомісткість = (Персонал) (Середовище) (Якість) (Розмір).

Для оцінки вартості ПЗ, створено кілька параметричних моделей. Всі вони, можуть бути зведені до такої форми. Один з важливих аспектів економіки створення ПЗ (як це представляється в сучасних моделях визначення вартості ПО) полягає в тому, що зв'язок між роботою і розмірами, визначає плату за великий масштаб. Плата за великий масштаб, при розробці ПЗ, є результатом того, що показник експоненти процесу більше одиниці. На відміну від більшості виробничих процесів, чим більше ПО створюється, тим дорожче воно обходиться в перерахунку на одну одиницю. Наприклад, для деякого довільного застосування програмне рішення об'ємом в 10 ТОВ рядків обійдеться дешевше в перерахунку на один рядок, ніж програмне рішення об'ємом 100 ТОВ рядків. Припустимо, що для створення 100 000-рядкової системи потрібно 900 людино-місяців, або близько 111 рядків за один людино-місяць, або 1.37 години на один рядок. Якби те ж саме система складалася з 10 000 рядків при незмінних інших параметрах, то проект оцінювався б приблизно в 62 людино-місяці, або 175 рядків за один людино-місяць, або. 0.87 години на один рядок. Вартість одного рядка для меншого додатку, виявляється набагато нижче, як для більшого додатку. Причина цього полягає насамперед у складності управління міжособистісними взаємодіями в міру того, як число членів команди (і відповідно число цілей, умов їх досягнення, технічних переваг) зростає. Ця плата за великий масштаб характерна для будь-якого дослідницького проекту, продуктом якого є єдиний в своєму роді, об'єкт інтелектуальної власності. Однією з важливих проблем при оцінці вартості ПО, є відсутність добре документованих практичних прикладів проектів, в яких застосовувалася ітераційна розробка. Хоча автори моделей оцінки вартості і заявляють, що їх інструментарій, призначений для оцінки проектів, що використовують ітераційну розробку, лише деякі з них грунтуються на емпіричних даних проектів, в яких ітераційна розробка була успішною. Більш того, оскільки індустрія ПО оперує суперечливими метриками і основними одиницями виміру, то дані по конкретним проектам виявляються досить підозрілими з точки зору їх

несуперечності і можливості порівняння. Збір однорідних даних по проекту, в рамках однієї організації, виявляється досить складним. Надзвичайно складний збір однорідних даних по різних організаціях, які використовують різні процеси, мови, підходи і т.д. Наприклад, фундаментальне поняття - одиниця виміру розміру (рядок вихідного коду або функціональна точка) - обчислюється всюди по-різному. Здається дивним, що стандарти сучасних мов (таких, як Ada 95 і Java) не мають визначення поняття рядка вихідного коду для підрахунку їх компілятором. Те, яке саме визначення буде застосовано, не настільки важливо, як і конкретна довжина фути або метри, може бути абсолютно довільної.

ВИСНОВКИ

Створена програма є простішою у використанні, та дає змогу працювати віддалено, оскільки знаходиться в вільному доступі, для працівників компанії, в мережі. Програмне забезпечення може використовуватися як в ІТ компаніях, так і у інших сферах, з метою автоматизації та покращення якості основних внутрішніх процесів.

Основною перевагою розробленого порталу, є можливість гнучкого управління внутрішніми процесами компанії, при найменших затратах часу на це. Можливість отримувати звітність, по роботі кожного з працівників, в зручному електронному форматі. Забезпечує безпечне зберігання даних, необхідних для роботи компанії, та доступ до них в мережі. Розроблена архітектура програми, дозволяє легко розширювати програмний продукт, а використання системних налаштувань, підлаштувати до потреб конкретної компанії. А через інтеграцію розробленого продукту з Google сервісами, дозволяє керувати порталом через додаткові програми компанії Google.

В результаті виконання даної дипломної роботи, було успішно засвоєно методи проектування, розробки та реалізації back-end частини ПЗ для внутрішньой роботи ІТ компанії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

REFERENCES

1. Адам Фримен. ASP.NET Core MVC с примерами на C# для профессионалов. 6-е издание. 2017р.
2. Ендрю Троелсен, Філіп ЯпіксеPro. C# 7: With .NET and .NET Core. 2017у.
3. Роберт Мартин. Чистая архитектура. Искусство разработки программного обеспечения. 2018р.
4. Юрген Аппело Agile-менеджмент. Лидерство и управление командами. 2018р.
5. Патерни. URL: https://metanit.com/sharp/patterns/ (дата звернення: 15.01.2020)
6. Entity Framework. URL: https://metanit.com/sharp/entityframework/ (дата звернення: 10.01.2020)
7. ASP.NET. URL: https://metanit.com/sharp/mvc5/ (дата звернення: 15.01.2020)
8. Корпоративний портал для компанії. URL: https://salesap.ru/blog/korporativnyj-portal-dlya-kompanii-chto-eto-i-kakaya-ot-nego-polza/ (дата звернення: 15.03.2020)
9. Управління проектами. URL: https://project.dovidnyk.info/index.php/home/upravlenieproektamiposozdaniyu programmnogoobespecheniya/67-evolyuciya_ekonomiki_razrabotki_po (дата звернення: 10.03.2020)
10. Класифікація та короткий огляд сучасних субд. URL: https://studfile.net/preview/5118185/page:28/ (дата звернення: 10.04.2020)
11. Google OAuth. URL: https://developers.google.com/identity/protocols/oauth2 (дата звернення: 20.03.2020)

12. Корпоративний портал. URL: http://www.sx-ua.com/ua/rishennya/rishennya-dlya-biznesu/korporativnij-portal/ (дата звернення: 15.04.2020)
13. Системи для управління підприємством. URL: https://inagro.com.ua/ua/programmy/korp-portal/ (дата звернення: 25.01.2020)
14. Бітрікс24. URL: https://www.bitrix24.ua/ (дата звернення: 25.12.2019)
15. Впровадження порталу. URL: https://quantum-int.com/uk/opis-biznes-procesiv-pri-vprovadzhenni-wms/ (дата звернення: 25.01.2020)
16. Джеймс Чамберс, Дэвид Пэкетт, Саймон Тиммс ASP.NET Core Application Development: Bulding an Application in Four Sprints (Developer Reference). 2015p.
17. M. Kozlenko, V. Tkachuk, and M. Dutchak, "Software implementation of microcomputer based intrusion detection and prevention system with binary neural network," in Proc. 2nd International Scientific-Practical Conference "Problems of Cyber Security of Information and Telecommunication Systems" (PCSITS), O. Oksiiuk et al, Eds. Taras Shevchenko National University of Kyiv, Kyiv, Ukraine, Apr. 11-12, 2019, pp. 371-373.
18. M. Kozlenko and A. Bosyi, "Performance of spread spectrum system with noise shift keying using entropy demodulation," 2018 14th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), Slavske, 2018, pp. 330-333, doi: 10.1109/TCSET.2018.8336213.
19. П. Федорук і М. Дутчак, “Побудова бази знань адаптивних систем дистанційного навчання на основі фреймової та продукційної моделей представлення знань,” Управляючі системи і машини (УСіМ), №5, с.35-42, 2012.

ДОДАТОК А

Основні зв’язки з користувачами

А.1 Права та ролі

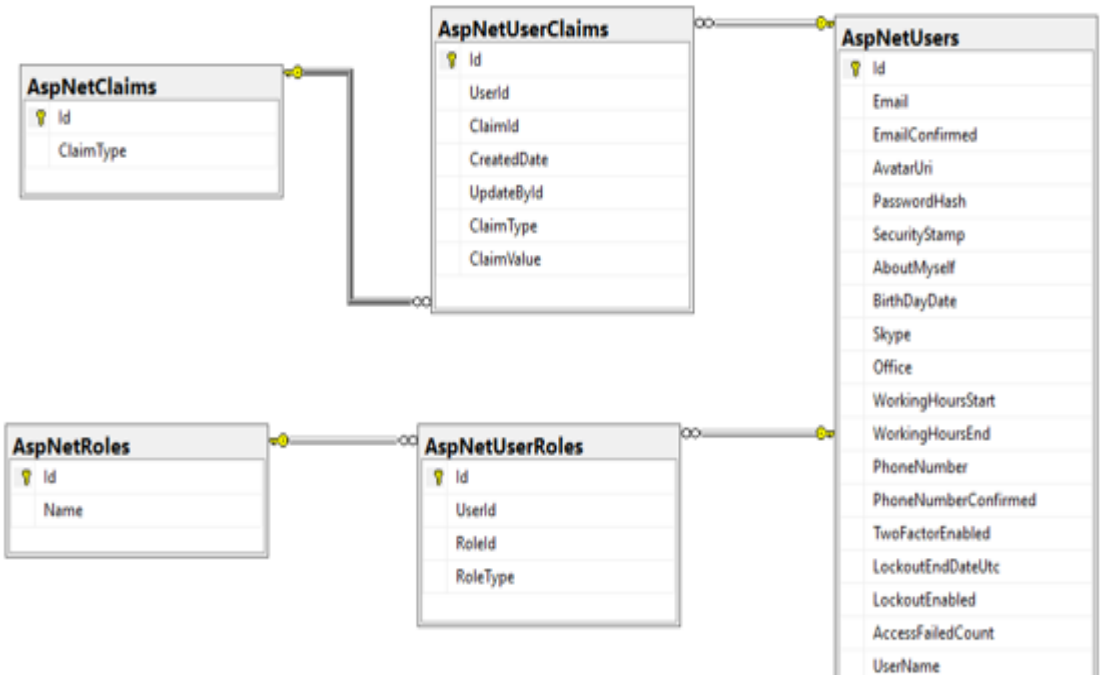


Рисунок А.1 – Зв’язки контролю доступу

А.2 Основні зв’язки

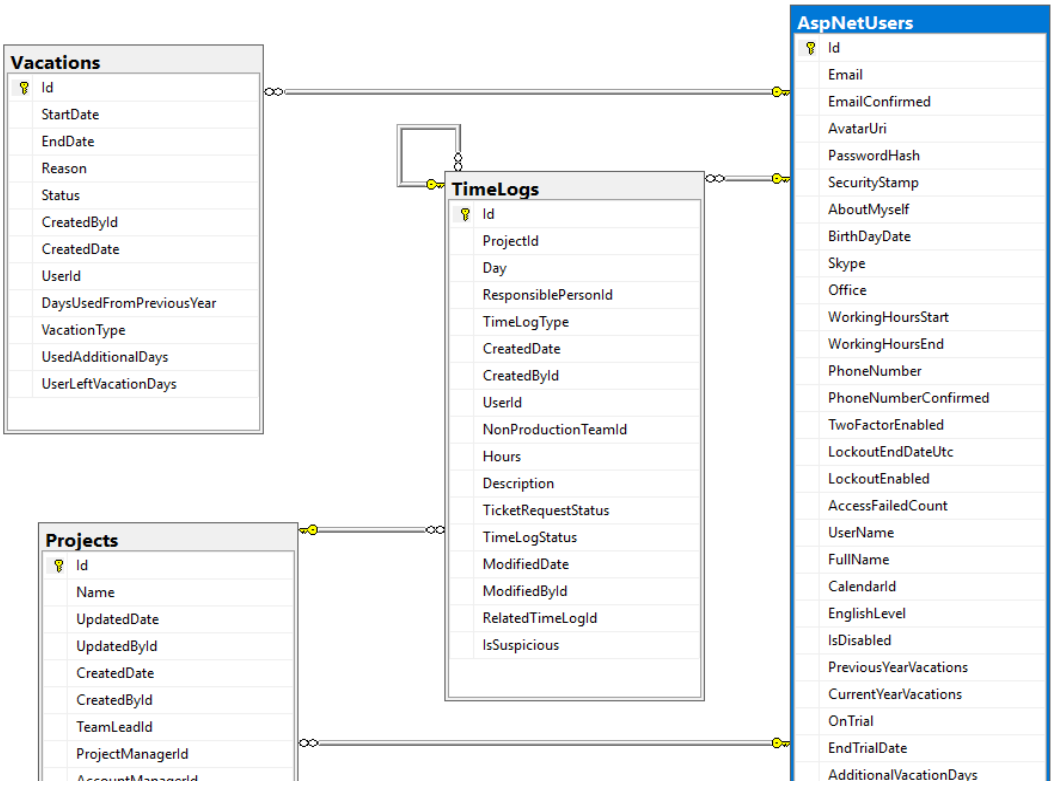


Рисунок А.2 – Основні звязки в роботі порталу

А.3 Навички та спеціальності

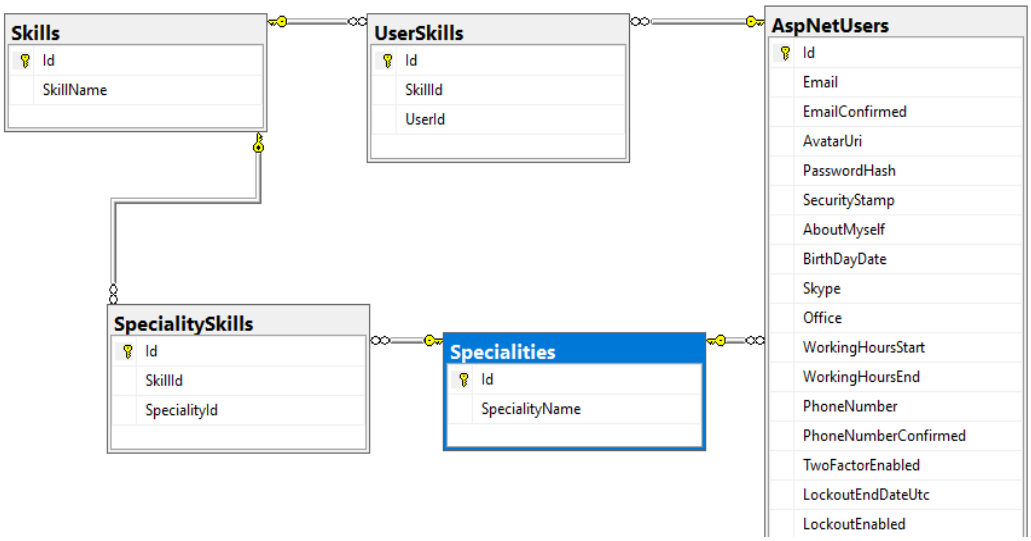


Рисунок А.3 – Зв’язки користувачів з навичками та спеціальностями

ДОДАТОК В

Код програми

B.1 User Controller

```
[Authorize]
[RoutePrefix("api/User")]
public class UserController : BaseController
{
    private readonly IUserService _userService;
    private readonly ISkillService _skillService;
    private readonly ISpecialityService _specialtyService;
    private readonly ITeamUserService _teamUserService;
    private readonly IUserExcelExport _userExelExport;

    public UserController(IUserService userService,
ISkillService skillService,
        ISpecialityService specialtyService, ITeamUserService
teamUserService, IUserExcelExport userExelExport )
    {
        _userService = userService;
        _skillService = skillService;
        _specialtyService = specialtyService;
        _teamUserService = teamUserService;
        _userExelExport = userExelExport;
    }

    [HttpGet]
    [Route("Onboarding/{userId}")]
    public IHttpActionResult OnBoarding(Guid userId)
    {
        _userService.InformHRsAboutNewUser(userId);
```

```
            _userService.InviteUserToOtherActivity(userId);


            return Ok();
        }


        [HttpPatch]
        [Route("UpdateByUser")]
        public IHttpActionResult
Update(UpdateUsersInformationByUserIncomeModel model)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            var userId = new Guid(User.Identity.GetUserId());


            _userService.UpdateUsersInformationByUser(model,
userId);


            return OkResult();
        }




        [HttpPatch]
        [Route("UpdateByAdmin/{userId}")]
        public IHttpActionResult
Update(UpdateUsersInformationByAdminIncomeModel model, Guid
userId)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));
```

```
            _userService.UpdateUsersInformationByAdmin(model,
userId, new Guid(User.Identity.GetUserId()));


            return OkResult();
        }


        [HttpGet]
        [Route("{userId}")]
        public IHttpActionResult GetUser(Guid userId)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            var user = _userService.GetUserById(userId);
            user.CurrentYearVacations =
Math.Truncate(user.CurrentYearVacations);
            return OkResult(user);
        }


        [HttpPost]
        [Route("list")]
        public IHttpActionResult
GetList([FromBody]GetPaginatedListUserIncomeModel model)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            return OkResult(_userService.GetListByFilter(model));
        }
```

```
        [HttpGet]
        [Route("Status/{userId}")]
        public IHttpActionResult GetUserStatus(Guid userId)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            return OkResult(_userService.GetUserStatus(userId));
        }

        [HttpGet]
        [Route("Projects/{userId}")]
        public IHttpActionResult GetUserProjects([FromUri]Guid
userId)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            return OkResult(_userService.GetUserProjects(userId));
        }

        [HttpGet]
        [Route("Skills")]
        public IHttpActionResult GetAllUsersSkills()
        {
            return OkResult(_skillService.GetAllSkills());
        }
```

```
        [HttpGet]
        [Route("OtherTeams")]
        public IHttpActionResult GetOtherTeams()
        {
            return OkResult(_teamUserService.GetOtherTeams());
        }

        [HttpPatch]
        [Route("DisableUser")]
        [ClaimsAuthorization(ClaimType = ClaimTypeResources.Auth,
ClaimValue = ClaimTypeResources.PartialProfileManagementPersonal)]
        public IHttpActionResult
DisableUser([FromBody]DisablingUserIncomeModel model)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            _userService.DisableUser(model);
            return OkResult();
        }

        [HttpGet]
        [Route("specialitySkills/{specialtyId}")]
        public IHttpActionResult
GetSkillsBySpeciality([FromUri]Guid specialtyId)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            return OkResult(_skillService.GetAllSkills());
```

```
        }

        [HttpGet]
        [Route("specialities")]
        public IHttpActionResult GetAllSpecialities()
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            return
OkResult(_specialtyService.GetAllSpecialities());
        }

        [HttpGet]
        [Route("AllBirthdays")]
        public IHttpActionResult AllBirthdays()
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            return OkResult(_userService.GetAllUsersBirthday());
        }

        [HttpGet]
        [Route("GetYearStatistics")]
        public IHttpActionResult GetYearStatistics()
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));
```

```
            return OkResult(_userService.GetYearStatistics());
        }

        [HttpGet]
[Route("GetHrStatisticExelReport/excel/{startDate}/{endDate}")]
        [ClaimsAuthorization(ClaimType = ClaimTypeResources.Auth,
ClaimValue = ClaimTypeResources.HolidaysManagement)]
        public HttpResponseMessage GetHrStatisticExelReport(string
startDate, string endDate)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            DateTime startBy;
            DateTime endBy;

            if (!(DateTime.TryParse(startDate, out startBy) |
DateTime.TryParse(endDate, out endBy)))
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            var userStatistic =
_userService.GetUserListForStatistic(startBy, endBy);
            var responseData =
_userExelExport.GetHrStatisticExelReport(userStatistic);

            var response =
Request.CreateResponse(HttpStatusCode.OK);
            response.Content = new StreamContent(responseData);
            var mediaType =
```

```
                new
MediaTypeHeaderValue("application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet");

            response.Content.Headers.ContentType = mediaType;

            response.Content.Headers.ContentDisposition =

                new ContentDispositionHeaderValue("report") {
FileName = "HR_Statistic_Report.xlsx" };


            return response;

        }

    }
```

B.2 Timelog Controller

```
    [Authorize]

    [RoutePrefix("api/timelog")]

    public class TimeLogController : BaseController

    {

        private readonly ITimeLogService _timeLogService;

        private readonly IProjectExcelExporter
_projectExcelExporter;

        private readonly IUserTimeLogReportCreator
_userTimeLogReportCreator;

        private readonly IUserExcelExporter _userExcelExporter;

        private readonly IProjectTimeLogReportCreator
_projectReportCreator;


        public TimeLogController(

            ITimeLogService timeLogService,

            IProjectExcelExporter projectExcelExporter,
IUserExcelExporter userExcelExporter,
```

```
            IUserTimeLogReportCreator userTimeLogReportCreator,
IProjectTimeLogReportCreator projectReportCreator)
        {
            _timeLogService = timeLogService;
            _projectExcelExporter = projectExcelExporter;
            _userExcelExporter = userExcelExporter;
            _userTimeLogReportCreator = userTimeLogReportCreator;
            _projectReportCreator = projectReportCreator;
        }

        [HttpGet]
        [Route("userReport/{userId}/{startDate?}/{endDate?}")]
        public IHttpActionResult GetReportByUserProject(Guid
userId, string startDate, string endDate)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            var adminUserId = new Guid(User.Identity.GetUserId());

            DateTime startBy;
            DateTime endBy;

            if (!(DateTime.TryParse(startDate, out startBy) |
DateTime.TryParse(endDate, out endBy)))
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            _timeLogService.CheckIfHaveAccessToUserReport(userId,
adminUserId);
```

```
            return
OkResult(_userTimeLogReportCreator.GetTimeLogReportByUser(userId,
startBy, endBy));

        }


        [HttpGet]

        [Route("project/{projectId}/{startDate?}/{endDate?}")]

        public IHttpActionResult GetListUsersTimeLogByProject(Guid
projectId, string startDate, string endDate)

        {

            if (!ModelState.IsValid)

                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            var adminUserId = new Guid(User.Identity.GetUserId());



_timeLogService.CheckIfHaveAccessToProjectReport(projectId,
adminUserId);


            DateTime startBy;

            DateTime endBy;


            if (!(DateTime.TryParse(startDate, out startBy) |
DateTime.TryParse(endDate, out endBy)))

                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            return
OkResult(_timeLogService.GetTimeLogProjectReport(projectId,
startBy, endBy));

        }


        [HttpPost]
```

```
        public IHttpActionResult
Create([FromBody]TimeLogIncomeModel model)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            var user = new Guid(User.Identity.GetUserId());
            _timeLogService.CheckIfUserHaveAccess(model, user);
            var returnModel = _timeLogService.Create(model, user);

            var responseModel =
CreateResponseModelWithSuccessMessage(
                new { returnModel.TimeLogId,
returnModel.OvertimeId, returnModel.WaitingId });

            if (returnModel.ResponsibePersonId != null)
            {
                var ticketHub = new TicketNotificationsHub();

ticketHub.SentTimelogTicketNotification(returnModel.ResponsibePers
onId,
                    returnModel.OvertimeId,
HubTicketNotificationType.CreateTimelog);
            }

            return OkResult(responseModel);
        }

        [HttpPatch]
        public IHttpActionResult
Update([FromBody]UpdateTimeLogIncomeModel model)
        {
```

```
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            var user = new Guid(User.Identity.GetUserId());
            _timeLogService.CheckIfUserHaveAccess(model, user);
            var returnModel = _timeLogService.Update(model, user);

            var responseModel =
CreateResponseModelWithSuccessMessage(
                new { returnModel.TimeLogId,
returnModel.OvertimeId, returnModel.WaitingId });

            var ticketHub = new TicketNotificationsHub();

ticketHub.SentTimelogTicketNotification(returnModel.ResponsibePers
onId,
                    returnModel.OvertimeId,
HubTicketNotificationType.UpdateTimelog);

            return OkResult(responseModel);
        }

        [HttpPatch]
        [Route("AdminUpdateOvertimeRequest")]
        [ClaimsAuthorization(ClaimType = ClaimTypeResources.Auth,
ClaimValue = ClaimTypeResources.TimelogManagement)]
        public IHttpActionResult
UpdateOvertimeRequest([FromBody]UpdateTimeLogStatusIncomeModel
model)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));
```

```
            var responseAdminsData =
_timeLogService.UpdateTimeLogRequestStatusByAdmin(model, new
Guid(User.Identity.GetUserId()));
            var responseModel =
CreateResponseModelWithSuccessMessage(responseAdminsData);


            var ticketHub = new TicketNotificationsHub();

ticketHub.SentTimelogTicketNotification(responseAdminsData.Respons
ibePersonId,
                    responseAdminsData.OvertimeId,
HubTicketNotificationType.UpdateTimelog);


            return OkResult(responseModel);
        }


        [HttpGet]

[Route("ReportByUser/excel/{userId}/{startDate}/{endDate}")]
        public HttpResponseMessage GetReportInExcel(Guid userId,
string startDate, string endDate)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            var adminUserId = new Guid(User.Identity.GetUserId());


            _timeLogService.CheckIfHaveAccessToUserReport(userId,
adminUserId);


            DateTime startBy;
            DateTime endBy;
```

```
            if (!(DateTime.TryParse(startDate, out startBy) |
DateTime.TryParse(endDate, out endBy)))
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            var responseData =
_userExcelExporter.GetUserExcelReport(userId, startBy, endBy);

            var response =
Request.CreateResponse(HttpStatusCode.OK);
            response.Content = new StreamContent(responseData);
            var mediaType =
                new
MediaTypeHeaderValue("application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet");
            response.Content.Headers.ContentType = mediaType;
            response.Content.Headers.ContentDisposition =
                new ContentDispositionHeaderValue("report") {
FileName = "User_Report.xlsx" };

            return response;
        }

        [HttpGet]
        [Route("ReportByAllUsers/{startDate}/{endDate}")]
        [ClaimsAuthorization(ClaimType = ClaimTypeResources.Auth,
ClaimValue = ClaimTypeResources.CompanyStaffReport)]
        public IHttpActionResult GetAllUsersReport(string
startDate, string endDate)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));
```

```
            DateTime startBy;
            DateTime endBy;

            if (!(DateTime.TryParse(startDate, out startBy) |
DateTime.TryParse(endDate, out endBy)))
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            var monthList =
BaseExcelExporter.GetMonthByRange(startBy, endBy);

            return
OkResult(_userTimeLogReportCreator.GetAllUsersTimeLogReport(monthL
ist));
        }

        [HttpGet]
        [Route("ReportByAllUsers/excel/{startDate}/{endDate}")]
        [ClaimsAuthorization(ClaimType = ClaimTypeResources.Auth,
ClaimValue = ClaimTypeResources.CompanyStaffReport)]
        public HttpResponseMessage
GetReportInExcelByAllUsers(string startDate, string endDate)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            DateTime startBy;
            DateTime endBy;

            if (!(DateTime.TryParse(startDate, out startBy) |
DateTime.TryParse(endDate, out endBy)))
```

```
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            var responseData =
_userExcelExporter.GetAllUsersExcelReport(startBy, endBy);


            var response =
Request.CreateResponse(HttpStatusCode.OK);
            response.Content = new StreamContent(responseData);
            var mediaType =
                new
MediaTypeHeaderValue("application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet");
            response.Content.Headers.ContentType = mediaType;
            response.Content.Headers.ContentDisposition =
                new ContentDispositionHeaderValue("report") {
FileName = "All_Users_Report.xlsx" };


            return response;
        }


        [HttpGet]

[Route("ReportByProject/excel/{projectId}/{startDate}/{endDate}")]
        public HttpResponseMessage GetReportInExcelByProject(Guid
projectId, string startDate, string endDate)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            var adminUserId = new Guid(User.Identity.GetUserId());
```

```
_timeLogService.CheckIfHaveAccessToProjectReport(projectId,
adminUserId);


            DateTime startBy;

            DateTime endBy;


            if (!(DateTime.TryParse(startDate, out startBy) |
DateTime.TryParse(endDate, out endBy)))

                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            var responseData =
_projectExcelExporter.GetProjectExcelReport(projectId, startBy,
endBy);


            var response =
Request.CreateResponse(HttpStatusCode.OK);

            response.Content = new StreamContent(responseData);

            var mediaType =

                new
MediaTypeHeaderValue("application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet");

            response.Content.Headers.ContentType = mediaType;

            response.Content.Headers.ContentDisposition =

                new ContentDispositionHeaderValue("report") {
FileName = "Project_Time_Log_Report.xlsx" };


            return response;

        }


        [HttpGet]

[Route("ReportBySingleProject/excel/{projectId}/{startDate}/{endDa
te}")]
```

```
        public HttpResponseMessage
GetReportInExcelBySingleProject(Guid projectId, string startDate,
string endDate)

        {

            if (!ModelState.IsValid)

                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            var adminUserId = new Guid(User.Identity.GetUserId());



_timeLogService.CheckIfHaveAccessToProjectReport(projectId,
adminUserId);


            DateTime startBy;

            DateTime endBy;


            if (!(DateTime.TryParse(startDate, out startBy) |
DateTime.TryParse(endDate, out endBy)))

                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            var responseData =
_projectExcelExporter.GetSingleProjectExcelReport(projectId,
startBy, endBy);


            var response =
Request.CreateResponse(HttpStatusCode.OK);

            response.Content = new StreamContent(responseData);

            var mediaType =

                new
MediaTypeHeaderValue("application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet");

            response.Content.Headers.ContentType = mediaType;

            response.Content.Headers.ContentDisposition =
```

```
                new ContentDispositionHeaderValue("report") {
FileName = "Time_Report_My_Project_Month.xlsx" };


            return response;

        }


        [HttpGet]

[Route("ReportByAdminProjects/excel/{adminId}/{startDate}/{endDate
}")]
        public HttpResponseMessage
GetReportInExcelByAdminProjects(Guid adminId, string startDate,
string endDate)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            DateTime startBy;
            DateTime endBy;


            if (!(DateTime.TryParse(startDate, out startBy) |
DateTime.TryParse(endDate, out endBy)))
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            var responseData =
_projectExcelExporter.GetAdminProjectsExcelReport(adminId,
startBy, endBy);


            var response =
Request.CreateResponse(HttpStatusCode.OK);
            response.Content = new StreamContent(responseData);
            var mediaType =
```

```
                new
MediaTypeHeaderValue("application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet");

            response.Content.Headers.ContentType = mediaType;

            response.Content.Headers.ContentDisposition =

                new ContentDispositionHeaderValue("report") {
FileName = "Projects_Time_Log_Report.xlsx" };


            return response;

        }


        [HttpGet]

[Route("ReportByAdminProjects/{adminId}/{startDate}/{endDate}")]

        public IHttpActionResult GetReportByAdminProjects(Guid
adminId, string startDate, string endDate)

        {

            if (!ModelState.IsValid)

                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            DateTime startBy;

            DateTime endBy;


            if (!(DateTime.TryParse(startDate, out startBy) |
DateTime.TryParse(endDate, out endBy)))

                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            return
OkResult(_projectReportCreator.GenerateAdminProjectsReport(adminId
, startBy, endBy));

        }
```

```
        [HttpGet]
        [Route("ReportByAllProjects/excel/{startDate}/{endDate}")]
        [ClaimsAuthorization(ClaimType = ClaimTypeResources.Auth,
ClaimValue = ClaimTypeResources.CompanyStaffReport)]
        public HttpResponseMessage
GetReportInExcelByAllProjects(string startDate, string endDate)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            DateTime startBy;
            DateTime endBy;

            if (!(DateTime.TryParse(startDate, out startBy) |
DateTime.TryParse(endDate, out endBy)))
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));

            var responseData =
_projectExcelExporter.GetAllProjectsExcelReport(startBy, endBy);

            var response =
Request.CreateResponse(HttpStatusCode.OK);
            response.Content = new StreamContent(responseData);
            var mediaType =
                new
MediaTypeHeaderValue("application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet");
            response.Content.Headers.ContentType = mediaType;
            response.Content.Headers.ContentDisposition =
                new ContentDispositionHeaderValue("report") {
FileName = "All_Projects_Time_Log_Report.xlsx" };
```

```
            return response;
        }


        [HttpGet]
        [Route("ReportByAllProjects/{startDate}/{endDate}")]
        [ClaimsAuthorization(ClaimType = ClaimTypeResources.Auth,
ClaimValue = ClaimTypeResources.CompanyStaffReport)]
        public IHttpActionResult GetAllProjectsReport(string
startDate, string endDate)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            DateTime startBy;
            DateTime endBy;


            if (!(DateTime.TryParse(startDate, out startBy) |
DateTime.TryParse(endDate, out endBy)))
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            var monthList =
BaseExcelExporter.GetMonthByRange(startBy, endBy);


            return
OkResult(_projectReportCreator.GenerateAllProjectsReport(monthList
));
        }
    }
```

B.3 Vacation Controller

```
[Authorize]
[RoutePrefix("api/Vacation")]
public class VacationController : BaseController
{
    private readonly IVacationService _vacationService;

    public VacationController(IVacationService
vacationService)
    {
        _vacationService = vacationService;
    }

    [HttpGet]
    [Route("VacationsForAdmin")]
    public IHttpActionResult GetUserVacationsForAdmin()
    {
        if (!ModelState.IsValid)
            throw new
ArgumentException(System.Web.Helpers.Json.Encode(ModelState));

        return
OkResult(_vacationService.GetUserVacationsForAdmin(new
Guid(User.Identity.GetUserId())));
    }

    [HttpGet]
    [Route("{vacationId}")]
    public IHttpActionResult GetVacation([FromUri]Guid
vacationId)
    {
```

```
            if (!ModelState.IsValid)
                throw new
ArgumentException(System.Web.Helpers.Json.Encode(ModelState));

            return
OkResult(_vacationService.GetVacationViewModelByVacationId(vacatio
nId));
        }

        [HttpGet]
        [Route("user/{userId}")]
        public IHttpActionResult GetAllUserVacation([FromUri]Guid
userId)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(System.Web.Helpers.Json.Encode(ModelState));

            return
OkResult(_vacationService.GetAllUserVacations(userId));
        }

        [HttpPost]
        public IHttpActionResult
Create([FromBody]CreateVacationIncomeModel model)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(System.Web.Helpers.Json.Encode(ModelState));

            var user = new Guid(User.Identity.GetUserId());
            var returnData =_vacationService.CreateVacation(model,
user);
```

```
            var ticketHub = new TicketNotificationsHub();

ticketHub.SentVacationTicketNotification(returnData.Item1,
returnData.Item2, HubTicketNotificationType.CreateVacation);


            return OkResult(returnData.Item2);
        }


        [HttpDelete]
        [Route("{vacationId}")]
        public IHttpActionResult Delete([FromUri]Guid vacationId)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(System.Web.Helpers.Json.Encode(ModelState));
            var user = new Guid(User.Identity.GetUserId());
            var returnData =
_vacationService.DeleteVacation(vacationId, user);


            var ticketHub = new TicketNotificationsHub();

ticketHub.SentVacationTicketNotification(returnData.Item1,
returnData.Item2, HubTicketNotificationType.DeleteVacation);


            return OkResult();
        }


        [HttpPatch]
        [Route("updateVacation")]
        public IHttpActionResult
Update([FromBody]UpdateVacationIncomeModel model)
        {
            if (!ModelState.IsValid)
```

```
                throw new
ArgumentException(System.Web.Helpers.Json.Encode(ModelState));


            var returnData =
_vacationService.UpdateVacation(model, new
Guid(User.Identity.GetUserId()));


            var ticketHub = new TicketNotificationsHub();

ticketHub.SentVacationTicketNotification(returnData.Item1,
returnData.Item2, HubTicketNotificationType.UpdateVacation);


            return OkResult();
        }


        [HttpPatch]
        [Route("updateVacationStatus")]
        [ClaimsAuthorization(ClaimType = ClaimTypeResources.Auth,
ClaimValue = ClaimTypeResources.TicketsManagement)]
        public IHttpActionResult
UpdateStatus([FromBody]UpdateVacationStatusIncomeModel model)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(System.Web.Helpers.Json.Encode(ModelState));


            var returnData =
_vacationService.UpdateVacationStatusByAdmin(model, new
Guid(User.Identity.GetUserId()));
            var responseModel =
CreateResponseModelWithSuccessMessage(returnData);


            var ticketHub = new TicketNotificationsHub();
```

```
ticketHub.SentVacationTicketNotification(returnData.Item1,
returnData.Item2, HubTicketNotificationType.UpdateVacation);


            return OkResult(responseModel);
        }
    }
```

B.4 Projects Controller

```
 [RoutePrefix("api/Project")]
    [Authorize]
    public class ProjectController : BaseController
    {
        private readonly IProjectService _projectService;

        public ProjectController(IProjectService projectService)
        {
            _projectService = projectService;
        }

        [HttpGet]
        [Route("names")]
        public IHttpActionResult GetNamesList()
        {
            return
OkResult(_projectService.GetProjectsNamesList());
        }

        [HttpGet]
        [Route("countries")]
```

```
        public IHttpActionResult GetCountriesList()
        {
            return OkResult(_projectService.GetCountriesList());
        }

        [HttpGet]
        [Route("industry")]
        public IHttpActionResult GetIndustriesList()
        {
            return OkResult(_projectService.GetIndustriesList());
        }

        [HttpPost]
        [Route("list")]
        public IHttpActionResult
GetList([FromBody]GetPaginatedListProjectIncomeModel model)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(System.Web.Helpers.Json.Encode(ModelState));

            return
OkResult(_projectService.GetListByFilter(model));
        }

        [HttpGet]
        [Route("{projectId}")]
        public IHttpActionResult Get(Guid projectId)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(System.Web.Helpers.Json.Encode(ModelState));
```

```
            var result =
_projectService.GetNotDeletedProject(projectId);


            return OkResult(result);
        }


        [HttpPost]
        [ClaimsAuthorization(ClaimType = ClaimTypeResources.Auth,
ClaimValue = ClaimTypeResources.ProjectsManagement)]
        public IHttpActionResult
Create([FromBody]CreateProjectIncomeModel model)
        {
            if (!ModelState.IsValid || model.AccountManagerId ==
Guid.Empty)
                throw new
ArgumentException(System.Web.Helpers.Json.Encode(ModelState));


            var result = _projectService.CreateProject(model, new
Guid(User.Identity.GetUserId()));


            return OkResult(result);
        }


        [HttpPatch]
        [ClaimsAuthorization(ClaimType = ClaimTypeResources.Auth,
ClaimValue = ClaimTypeResources.ProjectsManagement)]
        public IHttpActionResult Update([FromBody]
UpdateProjectIncomeModel model)
        {
            if (!ModelState.IsValid || model.AccountManagerId ==
Guid.Empty)
                throw new
ArgumentException(System.Web.Helpers.Json.Encode(ModelState));
```

```
            _projectService.UpdateProject(model, new
Guid(User.Identity.GetUserId()));


            return OkResult();
        }


        [HttpDelete]
        [Route("{projectId}")]
        [ClaimsAuthorization(ClaimType = ClaimTypeResources.Auth,
ClaimValue = ClaimTypeResources.ProjectsManagement)]
        public IHttpActionResult Delete(Guid projectId)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(System.Web.Helpers.Json.Encode(ModelState));


            _projectService.DeleteProject(projectId, new
Guid(User.Identity.GetUserId()));


            return OkResult();
        }
    }
```

B.5 Quartz Jobs

```
public static class Scheduler
    {
        public static void StartUpdatingVacationByYear()
        {
            var kernel = KernelBinder.InitializeJobFactory();
```

```
            var scheduler = kernel.Get<IScheduler>();

            scheduler.Start();
            scheduler.ScheduleJob(
JobBuilder.Create<UpdateVacationCoeffiecientJob>().Build(),
                TriggerBuilder.Create()
                    .WithCronSchedule("0 0 1 1 9 ? *")
                    .Build());
        }

        public static void
StartMovingUsersNotUsedVacationDaysByYear()
        {
            var kernel = KernelBinder.InitializeJobFactory();
            var scheduler = kernel.Get<IScheduler>();

            scheduler.Start();
            scheduler.ScheduleJob(
JobBuilder.Create<MoveUsersNotUsedVacationDaysJob>().Build(),
                TriggerBuilder.Create()
                    .WithCronSchedule("0 0 1 1 9 ? *")
                    .Build());
        }

        public static void CloseProjects()
        {
            var kernel = KernelBinder.InitializeJobFactory();
            var scheduler = kernel.Get<IScheduler>();
```

```
    scheduler.Start();
    scheduler.ScheduleJob(
        JobBuilder.Create<CloseProjects>().Build(),
        TriggerBuilder.Create()
            .WithCronSchedule("0 0 8 1 * ? *")
            .Build());
}

public static void CreateNotificationToCloseProject()
{
    var kernel = KernelBinder.InitializeJobFactory();
    var scheduler = kernel.Get<IScheduler>();

    scheduler.Start();
    scheduler.ScheduleJob(
        JobBuilder.Create<NotifyToCloseProject>().Build(),
        TriggerBuilder.Create()
            .WithCronSchedule("0 0 8 1 * ? *")
            .Build());
}

public static void CloseAllProjects()
{
    var kernel = KernelBinder.InitializeJobFactory();
    var scheduler = kernel.Get<IScheduler>();

    scheduler.Start();
    scheduler.ScheduleJob(
        JobBuilder.Create<CloseAllProjects>().Build(),
        TriggerBuilder.Create()
```

```
            .WithCronSchedule("0 0 8 1 * ? *")
            .Build());
}

public static void UserEndTrial()
{
    var kernel = KernelBinder.InitializeJobFactory();
    var scheduler = kernel.Get<IScheduler>();

    scheduler.Start();
    scheduler.ScheduleJob(
        JobBuilder.Create<UserEndTrial>().Build(),
        TriggerBuilder.Create()
            .WithCronSchedule("0 0 6 ? * * *")
            .Build());
}

public static void UpdateUsersVacation()
{
    var kernel = KernelBinder.InitializeJobFactory();
    var scheduler = kernel.Get<IScheduler>();

    scheduler.Start();
    scheduler.ScheduleJob(
        JobBuilder.Create<UpdateUsersVacation>().Build(),
        TriggerBuilder.Create()
            .WithCronSchedule("0 0 6 1/1 * ? *")
            .Build());
}
```

```
        public static void Birthday()
        {
            var kernel = KernelBinder.InitializeJobFactory();
            var scheduler = kernel.Get<IScheduler>();

            scheduler.Start();
            scheduler.ScheduleJob(
                JobBuilder.Create<Birthday>().Build(),
                TriggerBuilder.Create()
                    .WithCronSchedule("0 0 10 1/1 * ? *", cron =>
cron.InTimeZone(TimeZoneInfo.FindSystemTimeZoneById("FLE Standard
Time")))
                    .Build());
        }

        public static void StartNotifyAboutEndOfTrialPeriod()
        {
            var kernel = KernelBinder.InitializeJobFactory();
            var scheduler = kernel.Get<IScheduler>();

            scheduler.Start();
            scheduler.ScheduleJob(

JobBuilder.Create<NotifyAboutEndOfTrialPeriod>().Build(),
                TriggerBuilder.Create()
                    .WithCronSchedule("0 0 9 1/1 * ? *", cron =>
cron.InTimeZone(TimeZoneInfo.FindSystemTimeZoneById("FLE Standard
Time")))
                    .Build());
        }
    }
```

B.6 Authorization

```
[Authorize]
    [RoutePrefix("api/Account")]
    public class AccountController : BaseController
    {
        private const string LocalLoginProvider = "Local";
        private ApplicationUserManager _userManager;
        private readonly IAuthorizationService
_authorizationService;
        private readonly ICalendarService _calendarService;
        private readonly IUserService _userService;
        private readonly IVacationService _vacationService;

        private IAuthenticationManager Authentication =>
Request.GetOwinContext().Authentication;

        public ApplicationUserManager UserManager
        {
            get => _userManager ??
Request.GetOwinContext().GetUserManager<ApplicationUserManager>();
            private set => _userManager = value;
        }

        public ISecureDataFormat<AuthenticationTicket>
AccessTokenFormat { get; }

        public AccountController(IAuthorizationService
authorizationService, ICalendarService calendarService,
            IUserService userService, IVacationService
vacationService)
        {
            _userService = userService;
```

```
            _vacationService = vacationService;
            _authorizationService = authorizationService;
            _calendarService = calendarService;
        }

        public AccountController(ApplicationUserManager
userManager,
            ISecureDataFormat<AuthenticationTicket>
accessTokenFormat,
            IAuthorizationService authorizationService,
ICalendarService calendarService, IVacationService
vacationService)
        {
            _authorizationService = authorizationService;
            _calendarService = calendarService;
            _vacationService = vacationService;
            UserManager = userManager;
            AccessTokenFormat = accessTokenFormat;
        }


[HostAuthentication(DefaultAuthenticationTypes.ExternalBearer)]
        [Route("UserInfo")]
        public UserInfoViewModel GetUserInfo()
        {
            var claimIdentity = User.Identity as ClaimsIdentity;
            var externalLogin =
_authorizationService.GetExternalLoginDataFromIndentity(claimIdent
ity);
            var identity =
(ClaimsPrincipal)Thread.CurrentPrincipal;
            var claims = identity.Claims.Where(c => c.Type ==
"Auth").Select(c => c.Value).ToList();
```

```
            var user =
_userService.GetUserById(User.Identity.GetUserGuid());
            var userRole =
UserManager.GetRoles(user.Id).FirstOrDefault();
            var isFirstLogin =
_userService.IsUserFirstLogin(user);
            _vacationService.SetUserVacation(user.Id);


            return new UserInfoViewModel
            {
                Id = user.Id,
                FullName = user.FullName,
                UserName = User.Identity.GetUserName(),
                Email =
UserManager.GetEmail(User.Identity.GetUserGuid()),
                CalendarId = user.CalendarId,
                HasRegistered = externalLogin == null,
                LoginProvider = externalLogin?.LoginProvider,
                UserImageUri = user.AvatarUri,
                SpecialityName = user.SpecialityName,
                SpecialityId = user.SpecialityId,
                ClaimValue = claims,
                OnTrial = user.OnTrial,
                EndTrialDate = user.EndTrialDate,
                HasAdditionalVacationDays =
user.AdditionalVacationDays > 0,
                UserRole = userRole,
                IsFirstLogin = isFirstLogin
            };
        }


        [HttpGet]
```

```
        [Route("Logout")]
        public IHttpActionResult Logout()
        {

Authentication.SignOut(CookieAuthenticationDefaults.Authentication
Type);
            return Ok();
        }


        // GET
api/Account/ManageInfo?returnUrl=%2F&generateState=true
        [Route("ManageInfo")]
        public async Task<ManageInfoViewModel>
GetManageInfo(string returnUrl, bool generateState = false)
        {
            var user = await
UserManager.FindByIdAsync(User.Identity.GetUserGuid());

            if (user == null)
                throw new ObjectNotFoundException();

            var logins = user.Logins.Select(linkedAccount => new
UserLoginInfoViewModel
            {
                LoginProvider = linkedAccount.LoginProvider,
                ProviderKey = linkedAccount.ProviderKey
            }).ToList();

            if (user.PasswordHash != null)
            {
                logins.Add(new UserLoginInfoViewModel
                {
```

```
                    LoginProvider = LocalLoginProvider,
                    ProviderKey = user.UserName,
                });
            }

            return new ManageInfoViewModel
            {
                LocalLoginProvider = LocalLoginProvider,
                Email = user.UserName,
                Logins = logins,
                ExternalLoginProviders =
GetExternalLogins(returnUrl, generateState)
            };
        }

        [Route("AddExternalLogin")]
        public async Task<IHttpActionResult>
AddExternalLogin(AddExternalLoginIncomeModel model)
        {
            if (!ModelState.IsValid)
                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


Authentication.SignOut(DefaultAuthenticationTypes.ExternalCookie);


            var ticket =
AccessTokenFormat.Unprotect(model.ExternalAccessToken);


            if (ticket?.Identity == null ||
ticket.Properties?.ExpiresUtc != null &&
ticket.Properties.ExpiresUtc.Value < DateTimeOffset.UtcNow)
                return BadRequest("External login failure.");
```

```
            var externalData =
_authorizationService.GetExternalLoginDataFromIndentity(ticket.Ide
ntity);


            if (externalData == null)

                return BadRequest("The external login is already
associated with an account.");


            var result = await
UserManager.AddLoginAsync(User.Identity.GetUserGuid(),

                new UserLoginInfo(externalData.LoginProvider,
externalData.ProviderKey));


            return ValidateIdentityResult(result);

        }


        [Route("RemoveLogin")]

        public async Task<IHttpActionResult>
RemoveLogin(RemoveLoginIncomeModel model)

        {

            if (!ModelState.IsValid)

                throw new
ArgumentException(WebJson.Json.Encode(ModelState));


            IdentityResult result;


            if (model.LoginProvider == LocalLoginProvider)

                result = await
UserManager.RemovePasswordAsync(User.Identity.GetUserGuid());

            else

                result = await
UserManager.RemoveLoginAsync(User.Identity.GetUserGuid(),
```

```
                    new UserLoginInfo(model.LoginProvider,
model.ProviderKey));

            return ValidateIdentityResult(result);
        }

        [AllowAnonymous]
        [OverrideAuthentication]
        [Route("ExternalLogin", Name = "ExternalLogin")]

[HostAuthentication(DefaultAuthenticationTypes.ExternalCookie)]
        public async Task<IHttpActionResult>
GetExternalLogin(string provider, string error = null)
        {
            if (error != null)
                return new ChallengeResult(provider, this);

            if (!User.Identity.IsAuthenticated)
                return new ChallengeResult(provider, this);

            var externalLoginModel =
_authorizationService.GetExternalLoginDataFromIndentity(User.Ident
ity as ClaimsIdentity);

            if (externalLoginModel == null)
                return new ChallengeResult(provider, this);

            var accessibleDomains =
CustomConfigurationManagerService.AccessibleDomainsValidation;

            if (accessibleDomains != null)
            {
```

```
                var getUserEmail = externalLoginModel.UserEmail;
                var indexOfSplitting = getUserEmail.IndexOf('@');
                var userEmailDomain =
getUserEmail.Substring(indexOfSplitting + 1);

                if (!userEmailDomain.Any())
                    return new ChallengeResult(provider, this);
            }

            if (externalLoginModel.LoginProvider != provider)
            {

Authentication.SignOut(DefaultAuthenticationTypes.ExternalCookie);
                return new ChallengeResult(provider, this);
            }

            var applicationUser = await UserManager.FindAsync(new
UserLoginInfo(externalLoginModel.LoginProvider,
                externalLoginModel.ProviderKey));
            var hasRegistered = applicationUser != null;

            var apiRequestUri =
                new
Uri($"https://www.googleapis.com/oauth2/v2/userinfo?access_token={
externalLoginModel.AccessToken}");
            string userImageUri;

            using (var webClient = new System.Net.WebClient())
            {
                var json =
webClient.DownloadString(apiRequestUri);
                dynamic responseResult =
JsonConvert.DeserializeObject(json);
```

```
                userImageUri = responseResult.picture;
            }

            if (hasRegistered)
            {
                if (applicationUser.IsDisabled)
                    return NotFound();

Authentication.SignOut(DefaultAuthenticationTypes.ExternalCookie);
                applicationUser.AvatarUri = userImageUri;
                applicationUser.LastUserLogin = DateTime.UtcNow;
                UserManager.Update(applicationUser);

                await SignInUser(applicationUser,
externalLoginModel.AccessToken, externalLoginModel.RefreshToken,
false);
            }
            else
            {
                var externalIdentity =
Request.GetOwinContext().Authentication.GetExternalIdentityAsync(D
efaultAuthenticationTypes.ExternalCookie);
                var email =
externalIdentity.Result.Claims.FirstOrDefault(c => c.Type ==
ClaimTypes.Email)?.Value;
                var userName =
externalIdentity.Result.Claims.FirstOrDefault(c => c.Type ==
ClaimTypes.Name)?.Value;

                applicationUser = new ApplicationUser
                {
                    Id = Guid.NewGuid(),
                    UserName = email,
```

```
                    Email = email,
                    AvatarUri = userImageUri,
                    FullName = userName,
                    IsDisabled = false,
                    LastUserLogin = DateTime.UtcNow
                };

                var foundSuperAdmin = await
UserManager.FindByNameAsync(applicationUser.UserName);

                if (foundSuperAdmin == null)
                {
                    var result = await
UserManager.CreateAsync(applicationUser);

                    if (!result.Succeeded)
                        return ValidateIdentityResult(result);

                    var userLogin = new
UserLoginInfo(externalLoginModel.LoginProvider,
externalLoginModel.ProviderKey);

                    await
UserManager.AddLoginAsync(applicationUser.Id, userLogin);

                    await SignInUser(applicationUser,
externalLoginModel.AccessToken, externalLoginModel.RefreshToken,
true);

                    await
UserManager.AddToRoleAsync(applicationUser.Id,
Enum.GetName(typeof(RoleType), RoleType.User));
                }
                else
```

```
                {

                    var adminLogin = new
UserLoginInfo(externalLoginModel.LoginProvider,
externalLoginModel.ProviderKey);


                    await
UserManager.AddLoginAsync(foundSuperAdmin.Id, adminLogin);


                    await SignInUser(foundSuperAdmin,
externalLoginModel.AccessToken, externalLoginModel.RefreshToken,
false);


                    await
UserManager.AddToRoleAsync(foundSuperAdmin.Id,
Enum.GetName(typeof(RoleType), RoleType.SuperAdmin));

                }


                _calendarService.CheckIfCalendarExists(email);

            }

            return Ok();

        }


        // GET
api/Account/ExternalLogins?returnUrl=%2F&generateState=true

        [AllowAnonymous]

        [Route("ExternalLogins")]

        public IEnumerable<ExternalLoginViewModel>
GetExternalLogins(string returnUrl, bool generateState = false)

        {

            var descriptions =
Authentication.GetExternalAuthenticationTypes();


            string state;
```

```
            if (generateState)
            {
                const int strengthInBits = 256;
                state =
_authorizationService.GenerateRandomOAuthState(strengthInBits);
            }
            else
                state = null;


            return descriptions.Select(description => new
ExternalLoginViewModel
            {
                Name = description.Caption,
                Url = Url.Route("ExternalLogin", new
                {
                    provider = description.AuthenticationType,
                    response_type = "token",
                    client_id = Startup.PublicClientId,
                    redirect_uri = new Uri(Request.RequestUri,
returnUrl).AbsoluteUri,
                    state
                }),
                State = state
            })
                .ToList();
        }


        private async Task SignInUser(ApplicationUser
applicationUser, string googleAccessToken, string
googleRefreshToken, bool isFirstLogin)
        {
```

```
            var oAuthIdentity = await
applicationUser.GenerateUserIdentityAsync(UserManager,
                OAuthDefaults.AuthenticationType,
googleAccessToken, googleRefreshToken, isFirstLogin);
            var cookieIdentity = await
applicationUser.GenerateUserIdentityAsync(UserManager,
                CookieAuthenticationDefaults.AuthenticationType,
googleAccessToken, googleRefreshToken, isFirstLogin);


            var properties =
ApplicationOAuthProvider.CreateProperties(applicationUser.FullName
);
            var ticket = new AuthenticationTicket(oAuthIdentity,
properties);
            var context = new
AuthenticationTokenCreateContext(Request.GetOwinContext(),
                    Startup.OAuthOptions.RefreshTokenFormat,
ticket);


            await
Startup.OAuthOptions.RefreshTokenProvider.CreateAsync(context);
            properties.Dictionary.Add("refresh_token",
context.Token);


            Authentication.SignIn(properties, oAuthIdentity,
cookieIdentity);
        }


        protected override void Dispose(bool disposing)
        {
            if (disposing && _userManager != null)
            {
                _userManager.Dispose();
                _userManager = null;
            }
```

```
            base.Dispose(disposing);
        }

        private IHttpActionResult
ValidateIdentityResult(IdentityResult result)
        {
            if (result == null)
                return InternalServerError();

            if (result.Succeeded)
                return Ok();

            if (result.Errors != null)
                foreach (var error in result.Errors)
                    ModelState.AddModelError("", error);

            if (ModelState.IsValid)
                return BadRequest();

            return BadRequest(ModelState);
        }
    }
```