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

УДК 004

ДИПЛОМНИЙ ПРОЕКТ

Тема: Розробка програмного забезпечення для шкіл, ліцеїв, гімназій

Спеціальність: 121 Інженерія програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДП.ІПЗ-12.ПЗ
(позначення)

Рецензент
ст.викл Савка І.Я.
(посада) (підпис) (дата) (розшифровка підпису)

Студент
ІПЗ-41 Матіїв Т.Р.
(шифр групи) (підпис) (дата) (розшифровка підпису)

Нормоконтролер
ст.викл Савка І.Я.
(посада) (підпис) (дата) (розшифровка підпису)

Керівник дипломного проекту
зав.кафедри Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Завідувач кафедри

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 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри Козленко М.І.
_____
„_____” _____ 20___р.

ЗАВДАННЯ
НА ВИКОНАННЯ ДИПЛОМНОГО ПРОЕКТУ

Студенту Матіїву Тарасу Романовичу
(прізвище, ім’я, по батькові студента)

1. Тема проекту Розробка програмного забезпечення для шкіл, ліцеїв, гімназій затверджена розпорядженням по факультету математики та інформатики від „25” жовтня 2019 р.№7
2. Термін здачі студентом закінченого проекту 22 травня 2020 р.
3. Вихідні дані до дипломного проекту вимоги до дистанційного навчання, технологія програмування серверної частини – Ruby on Rails, Ruby технології програмування клієнтської частини – HTML, CSS, JavaScript, Технології розгортання програмного забезпечення – Capybara, Heroku CLI
4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати) Огляд сфери шкільної освіти та ефективність дистанційного навчання для шкіл та гімназій різних ступенів, постановка задач які повинен вирішувати, програмний продукт, моделювання структури програмного забезпечення , практична реалізація програмного забезпечення дистанційного навчання учнів, бізнес-план для програмного забезпечення
5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень) 1) Загальний опис веб-додатку (шкільного сайту)
2) Основні завдання даного веб застосунку
3) Актуальність, переваги та недоліки конкурентів
4) Переваги даного веб додатку,
5) Основані завдання даної роботи
6) Огляд загальної структури бази даних
7) UML діаграми
7. Дата видачі завдання 11.09.2019

Керівник _____ Козленко М.І.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Матіїв Т.Р.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
Огляд сфери освіти та уже існуючих готових рішень для шкіл та гімназії	02.02.2020	Виконав
Опис вимог програмного забезпечення визначення ефективності дистанційного навчання для шкіл та гімназій	04.03.2020	Виконав
Практична реалізація програмного забезпечення визначення веб-додатку для шкіл	08.04.2020	Виконав
Бізнес-план для програмного забезпечення визначення веб-додатку для шкіл	10.05.2020	Виконав
Оформлення пояснювальної записки	18.05.2020	Виконав

Студент _____ Матіїв Т.Р.
(підпис) (розшифровка підпису)

Керівник проекту _____ Козленко М.І.
(підпис) (розшифровка підпису)

РЕФЕРАТ

Пояснювальна записка: 87 сторінок (без додатків), 17 рисунків, 3 таблиць, 16 джерел, 4 додатків на 9 сторінках.

Ключові слова: RUBY, RUBY ON RAILS, ФРЕЙМВОРК, CSS, HTML, MVC, MEMCACHED, REDIS, SIDEKIQ, MOODLE, E-SCHOOL, COURSERA, МІЙКЛАС, GOOGLE CLASSROOM, СУБД, POSTGRESQL, API, ГЕМИ, ВАЛІДАЦІЇ, ОПП, BOOTSTRAP.

Об`єктом дослідження є розробка веб-сервісу для можливості дистанційного навчання та контролю за процесом навчання.

Мета роботи: розробити веб застосунок, який зможе зберігати домашні та дистанційні завдання для учнів, а також розклад занять зі сповіщення в месенджер Telegram.

Стислий опис тексту пояснювальної записки:

Дана дипломна робота містить опис моделювання, розробки, тестування веб застосунку на мові програмування Ruby для шкіл та гімназій. Для роботи з даними було вибрано систему керування базами даних Postgresql. Для користувацького інтерфейсу було використано технології: HTML, CSS, Bootstrap, JavaScript.

ABSTRACT

Explanatory note: 87 pages (without appendices), 17 figures, 3 tables, 16 sources, 4 appendix on 9 pages.

Keywords: RUBY, RUBY ON RAILS, ФРЕЙМВОРК, CSS, HTML, MVC, MEMCACHED, REDIS, SIDEKIQ, MOODLE, E-SCHOOL, COURSERA, МІЙКЛАС, GOOGLE CLASSROOM, СУБД, POSTGRESQL, API, ГЕМИ, ВАЛІДАЦІЇ, ОПП, BOOTSTRAP.

The object of research is the development of a web service for the possibility of the distance learning and control over the learning process.

Purpose:

To develop a web application that can store homework and distance work for students, as well as save the class schedule with notification in the Telegram messenger.

Brief description of the text of the explanatory note:

This thesis contains a description of the modeling, development, testing of a web application in the Ruby programming language for schools and gymnasiums. The Postgresql database management system was chosen to work with the data. Technologies were used for the user interface: HTML, CSS, Bootstrap, JavaScript.

ЗМІСТ

ВСТУП

Освіта в інтернеті набирає великого поширення. У інших країнах, кожна школа вводить у програму навчання можливість здійснення освітніх процедур через мережу, а саме проведення відео уроків тощо. Для системи освіти такі «апгрейти» можуть суттєво вплинути на підвищення рівня освіти.

Враховуючи пандемію COVID-19, освітні заклади потребують швидкого переходу на дистанційну форму навчання. Тому веб ресурси, які надають таку можливість, набули неабиякої популярності. Люди стали частіше проводити час в дома, а школярі і зовсім повинні залишатися в дома, проте така довга перерва у навчанні не повинна впливати на обізнаність учнів. Тому в такий час ефективність сайту для школи з можливістю публікувати, перевіряти домашнє завдання є високою.

На даний час є багато уже готових рішень в інтернет просторі, проте вони є комерційними проектами і для того, щоб їх розгорнути потрібно спочатку за них заплатити, або ж вони є не під лаштованими під роботу школи.

Технологія Ruby on Rails на даний час нам дозволяє реалізувати такий проект за короткий термін. Так як її перевагами є швидкість розробки і низький поріг входу. Для інтерфейсу користувача існують різні фреймворки і бібліотеки, які неабияк скорочують час розробки веб додатку.

1. ОСНОВНІ ПОНЯТТЯ ТА ОСОБЛИВОСТІ ОСВІТИ В ІНТЕРНЕТ ПРОСТОРІ

1.1 Основні поняття та роль освіти

Освіта – це діяльність в ході, якої людина набуває нових умінь та навичок, як наслідок людина опановує якусь галузь. На даних час освіта є важливою складовою життя людини. Вона дає можливість знайти своє призначення, досягти своєї цілі [1].

Природні таланти людина повинна розвивати за допомогою самоосвіти. Активне читання книжок, наукової літератури, інтернет ресурсів допомагає здобути нові знання, які можуть знадобитись людині в ході її життя.

Фахівцем у певній галузі можна стати лише після здобуття професії, спеціальності. Якщо після здобуття фахової спеціалізації пройшло чимало часу, не зайвим буде відвідування кваліфікаційних курсів. Взагалі знання та досвід людина здобуває усе життя, адже світ не стоїть на місці. З віком ви можете зацікавитися тією сферою, яка не цікавила у молоді роки.

Чим освіченіша людина, тим їй легше розібратися в навколишньому світі, вирішити проблеми, які виникають у кожної окремої особистості в житті. Якщо у людини високий рівень освіти, то вона може застосувати свої знання, вміння та навички на практиці, використовуючи різні інформаційні ресурси.

На освітню систему виділяється велика кількість грошей, зокрема на школи, ліцеї, гімназії, університети тощо. Проте не завжди якість оправдується вкладеними коштами. Сьогодні існує безліч систем освіти, які в свою очередь надають різного рівня знання.

На даний час найкращими освітніми системами прийнято вважати в такий країнах як:

- Сінгапур

- Південна Корея
- Японія
- Фінляндія
- Швейцарія
- Канада
- Нідерланди
- Німеччина

В цих країнах спостерігається висока ступінь писемності, проте ці результати досягнуті різними освітніми реформами, які в ту чи іншу степінь покращую рівень знань учнів. В реальному житті в різних країнам ці форми навчання реалізуються по-різному, і не рідко ознаки певної форми навчання на практиці дуже суттєво відрізняються від теоретичної бази. Тому насамперед потрібно оцінити теоретичні засади кожної з форм, після чого оцінити їхню реалізацію в реаліях нашого життя і встановити, яка форма краще дає змогу оволодіти знаннями дисципліни.

Але реформи в освіті не головне. Якщо розглядати українську систему освіти, є звісно негативні і позитивні аспекти. Проте вона характеризується низьким прагненням учнів до навчання. Цьому є низка причини.

Однією з найголовніших, є те, що учні не мають розуміння мети навчання. У школах вчителі інколи забуваються донести учню, цінність навчання, а також роль знань для учня.

У різних країнах цю проблему вирішують різними методами. І один з них, це можливість навчатися дистанційно через комп'ютер. Як уже відомо школярі, студенти вишів багато часу проводять в інтернет просторі. Проводять час за комп'ютером в соціальних мережах, шукають там відповіді на свої запитання або ж находять інформації.

Тому у такому випадку освітні системи провідних країн, запровадили дистанційне навчання. Тобто, студент може займатися навчанням, знаходити додаткові матеріали по темах, які викладаються в закладах освіти.

Перевагами дистанційного навчання можна назвати:

- можливість в повній мірі засвоювати навчальний матеріал;
- можливість батьків контролювати процес навчання;
- можливість самостійно вибирати місце і час навчання;
- велика зацікавленість учнів у процесі навчання в інтернеті;
- навчатися не залежно від стану здоров'я;
- дешевизна, менша кількість витрачених грошей на утримання шкіл і персоналу;
- можливість ознайомитись з комп'ютерними технологіями.

Незважаючи на наведені вище переваги в такій системі є вагомі недоліки, які суттєво можуть перекрити переваги.

Основними недоліками дистанційного навчання є:

- велика затрата часу на процес навчання;
- нерегулярність у заняттях;
- відсутність вчителя;
- відсутність повноцінного контролю за процесом навчання.

1.2 Особливості веб-сервісу для шкіл та гімназій

У теперішній час із розвитком комунікаційних технологій такий як інтернет, стає можливим онлайн навчання для школярів.

Інтернет – це необмежене джерело різної інформації, як для учителя так і для учня. Це дає можливість ознайомлюватись із новою інформацією, а також і ділитись нею в простий спосіб. Якщо пропустити процес навчання в школах через призму інтернет технологій, можна дійти висновку що процес навчання, контролю навчання можна легко автоматизувати та покращити.

Основними завданнями проекту є:

- надання учню особистого кабінету для редагування особистої сторінки;

- забезпечення учня можливістю отримувати домашнє завдання на кожен день;
- можливість вчителя створювати дистанційні завдання (без живого уроку) для учнів. А учні, отримувати сповіщення про створення вчителем завдання;
- можливість прикріплювати відео та фото матеріали до дистанційних завдань, які будуть корисні для учні в ході виконання ним дистанційного завдання;
- надання можливості вчителя оцінювати роботу учня, яку від по завершення виконання завантажує на сайт і відправляє вчителю на перевірку;
- можливість активувати сповіщення в месенджері Telegram. Тобто учень може отримувати сповіщення про те, що він отримав оцінку за надіслане ним завдання, про створення дистанційного завдання для його класу, добавлення домашнього завдання;
- надання можливості будувати графіки по даних успішності студента за заданими критеріями.

Архітектура програмного забезпечення розбита на три основні складові: БД, сервер та клієнт

Користувачі сайту розділені на три групи:

- адміністратор;
- вчитель;
- учень.

Адміністратор має мати змогу реєструвати у сервісі вчителів та учнів, змінювати особисту інформацію кожного із користувачів окрім паролів, при потребі видаляти дані користувачів, а також створювати, редагувати, видаляти та читати дані на сайті.

Вчителі мають мати змогу створювати власні дистанційні завдання, додавати домашні завдання та оцінювати роботу учнів.

Учні мають мати змогу отримувати завдання від учителя, проходити відповідний тест, відправляти розв’язки завдання і переглядати результати.

1.3 Існуючі програмні рішення для автоматизації процесів у школі

З невпинним розвитком інтернет технологій, з’являється можливість інтеграції їх у навчальний процес. Для автоматизації деякий часозатратних процесів. Для прикладу домашнє завдання або ж дистанційні завдання можна публікувати на платформі Classroom компанії Google.

Рисунок 1.1 – Логотип Google Classroom

Google Classroom – безплатний веб-сервіс розроблений компанією Google для закладів освіти з метою перенесення навчального процесу в інтернет. Головна мета сервісу — пришвидшити процес поширення інформації між вчителями та студентами. Логотип Classroom зображено на рисунку 1.1.

Для того, щоб запросити учня до класу, потрібно знати його електронну пошту, на яку можна відправити лист із запрошенням, яке учень у свою очередь

повинен прийняти. Кожний клас по замовчуванню містить свій каталог, де учень може прикріпляти роботу, яку вчитель матиме змогу переглянути. Вчителі можуть публікувати завдання для учнів, стежити за процесом їх виконанням а також оцінити виконану роботу учнем роботу, у разі наявності помилок у виконаному учнем завданні, вчитель може повернути завдання із поясненням.

Існує також мобільний додаток Classroom, який є у доступі на двох найпоширеніших мобільних платформах (iOS, Android). Логотип веб-сервісу Google Classroom зображений на рисунку 1.1.

Перша версія Classroom-у вийшла у 2014. Відтоді платформа отримала багато нововведень. Такий як можливість коментувати учнем завдання які опублікував учитель, а також учень може створювати свої завдання у «класі», який створив учитель [2].

Для сторонніх веб-аплікацій Classroom містить API. За допомогою нього сторонні проекти можуть легко створювати свої завдання, а також зберігати їх на платформі Classroom і подальшому використовувати їх в свої веб-додатках

Рисунок 1.2 – Логотип Coursera

Courséra – американська онлайн-платформа для навчання, заснована в 2012 році професорами Стенфорда Ендрю Нґ та Дафни Коллер, яка пропонує широкі відкриті онлайн-курси, спеціалізації та ступені.

Coursera пропонує користувачам безліч безкоштовних онлайн-курсів із різноманітних дисциплін, у випадку успішного закінчення цих курсів користувачеві видається відповідний сертифікат [3].

Також ця онлайн-платформа співпрацює із багатьма вищими навчальними закладами для викладання цих онлайн курсів. Зараз Coursera пропонує курси в різних сферах освіти.

Найпопулярніші курси на Coursera:

1. Соціальна психологія;
2. Подумай двічі: як аргументу та переконувати;
3. Вступ до програмування: основи;
4. Біткойн та криптовалютні технології;
5. Теорія ігор;
6. Інженерія стартапів;

7. Фінансові ринки;
8. Креативність, інновації та зміни;
9. Аналіз даних;
10. Китайська мова для початківців.

Всі курси на Coursera існують на безкоштовній основі, але сертифікація у на різних курсах є платною, і варіюється в залежності від університету та складності самого курсу. Тривалість проходження навчання різна. Під час навчання студенти можуть переглядати лекції, читати різноманітну інформацію що часто прикріплена у вигляді посилань на додаткові джерела, яка надається кураторами курсу. Також студент повинен виконувати домашні завдання, у вигляді тестів або ж завдань які потрібно виконати вручну. Для перевірки знань студент проходить тестування і виконує різноманітні завдання. На даний час деякі курси є в доступі на українській мові.

Рисунок 1.3 – Логотип інформаційно-освітньої системи МійКлас

МійКлас – українська електронна інформаційно-освітня система. Функціонал системи дозволяє вчителям використовувати вже розроблені завдання шкільної програми з 1 по 11 класи або розробити власну навчальну програму для дистанційного навчання учнів. Логотип системи МійКлас зображено на рисунку 1.4

МійКлас – це відкрита система, нею можна користуватись будь-коли, будь-де та на будь-якому пристрої, який має підключення до інтернету. На сайті є ІКТ-курс, що дає змогу вчителям самостійно і за короткі терміни впровадити дистанційне навчання.

Основні функції інформаційно-освітньої системи МійКлас:

- різні завдання із предметів інваріантної складової навчальних планів 1-11 класів;
- можливість вчителя впроваджувати дистанційне навчання;
- можливість дистанційного проведення домашніх, самостійних та контрольних робіт;
- отримання адміністрацією закладу освіти звіту про використання даної системи вчителями та учнями;
- контроль з боку батьків.

Дана система дає змогу впроваджувати ігрові технології у навчальний процес. Учні завжди змагаються за першість серед однокласників, що посилює їхню зацікавленість в навчанні та підвищує успішність на 20% [4].

Основними недоліками цих сервісів є складний інтерфейс користувача та їх громіздкість. Також вони специфічні у плані налаштування.

Оскільки веб-сервіси, які задіяні у сфері освіти характеризуються великою популярністю, реалізація зручного користувацького інтерфейсу є пріоритетною задачею.

Рисунок 1.4 – Логотип освітньої системи e-school

E-school – це безкоштовна платформа для створення шкільних сайтів. Ця система досить зручна в інтерфейсі, так як створена для простоти в користуванні. Платформа офіційно дозволена для її використання в процес шкільного навчання. Вона сертифікована освітніми системами різних країн, зокрема українською та чеською. Логотип платформи зображено на рисунку 1.3.

Сервіс e-schools.info призначений:

- для обробки і надання в зручному електронному вигляді інформації про успішність учнів;
- розміщення сайтів закладів освіти на даній платформі.

Переваги проекту:

- Інформація надається у різному форматі: журнал, щоденник, таблиця успішності, графіки успішності за кожним класом і учнем, звіти для вчителя після закінчення навчального періоду.
- Для освітньої установи – електронні журнали і централізована база оцінок з усіх предметів.
- Для учнів і батьків – розклад, оцінки та домашні завдання, записані самими вчителями, завжди під рукою.
- Персональний вхід для кожного користувача.
- Приватне і публічне спілкування, голосування, чат для учнів.

Вся інформація знаходиться на серверах, які розташовуються на території України. За замовчуванням сайт має типовий дизайн, над удосконаленням якого ведеться постійна робота. Деякі аспекти зовнішнього вигляду можна налаштувати індивідуально. Безкоштовна платформа для створенню сайту.

Електронний освітній проект «E-schools» схвалений до використання Інститутом модернізації змісту освіти Міністерства освіти і науки України та успішно впроваджується в школах та дошкільних закладах України в межах виконання державної програми «Нова українська школа» [6].

Рисунок 1.5 – Логотип освітньої системи Moodle

Moodle - це безкоштовна онлайн-система управління навчанням, яка дає змогу викладачам розгорнути власний навчальний сайт, наповнений різноманітними та динамічними курсами, що дають змогу навчатися в будь-який час і в будь-якому місці. Логотип Moodle зображено на рисунку 1.4.

Незалежно від ролі самого користувача, Moodle може задовольнити потреби у навчанні. Надзвичайно легкий у використанні Moodle має багато стандартних функцій.

Moodle має широкий набір функціональності, притаманний тільки додаткам електронних навчальних систем, системам управління курсами (CMS), системам управління навчанням (LMS) або віртуальним навчальним середовищам (VLE). Moodle дає можливість вчителям створювати ефективні сайти для реалізації онлайн навчання. Moodle можна використовувати у різних видах освітніх закладів, це може стосуватись як учнів, студентів так і самих вчителі, наприклад для підвищення їх кваліфікації [7].

Типова функціональність Moodle включає:

- Виконання завдань;
- Форум;
- Завантаження різнотипних документів;
- Оцінювання виконаних завдань;

- Онлайн чат;
- Календар подій;
- Анонсування подій;
- Реалізація онлайн тестування;
- Корисна документація.

Великою перевагою Moodle є можливість розробнику створювати додаткові модулі з додатковою функціональністю. Moodle підтримує різні типи додаткових модулів:

- Види діяльностей (наприклад навчальні ігри)
- Ресурсні типи
- Типи тестових питань
- Робота з базою даних
- UI Теми
- Різного виду аутентифікації
- Різні способи запрошення на курс
- Валідація контенту

Використовуючи цю можливість розробники розробили багато сторонніх модулів, які вносять більше можливостей для даної платформи. Стандартно Moodle містить в собі модуль TCPDF, який дозволяє переводити HTML сторінки в PDF документи.

У середовищі Moodle студенти отримують:

- доступ до навчальних матеріалів (тексти лекцій, завдання до практичних/лабораторних та самостійних робіт; додаткові матеріали, книги, довідники, посібники, методичні розробки та засобів для спілкування і тестування «24 на 7»;
- засоби для групової роботи (Вікі, форум, чат, семінар, вебінар);
- можливість перегляду результатів проходження дистанційного курсу студентом;

- можливість перегляд результатів проходження тесту;
- можливість спілкування з викладачем через особисті повідомлення, форум, чат;
- можливість завантаження файлів з виконаними завданнями;
- можливість використання нагадувань про події у курсі.

Викладачам надається можливість:

- можливість на базі навчального закладу створювати власні курси;
- розміщення власної навчальної інформації (лекцій, додаткова інформація до практичних та лабораторних робіт;
- Можливість додавати матеріали у вигляді електронних підручників в форматах doc, docx, pdf тощо;
- додавання різноманітних додатків до курсу;
- можливість модифікувати прикріплені до курсу матеріали;
- створення, оновлення, видалення власних тестів;
- зручне формування онлайн тестів;
- автоматичний підрахунок балів за виконані тести;.

2 ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ САЙТУ ШКОЛИ

2.1 Проектування бази даних

Одним з перших етапів розробки аплікації є створення та моделювання структури бази даних. Потрібно обдумати кожну сутність, що будуть використовуватися в веб-сервісі, і звісно, всі властивості створених сутностей. В більшості випадків, одну сутність відображають, як одну реальну таблицю бази даних. Проте інколи розділяють на декілька таблиць для більш ширших можливостей використання даних, які вони будуть в собі містити.

Напевне найбільш частою сутністю, яка зустрічається у кожній веб аплікації - це таблиця під назвою users. У ній як можна зрозуміти веб аплікація буде зберігати користувачів. Важливою частиною сутності User є її поля в базі даних.

На сьогоднішній день стандартними полями в такій табличці є email, password. Як показує практика їх використовують для збереження даних, які користувач зможе запам’ятати, а в майбутньому за допомогою них увійти в систему.

Основними і необхідними полями таблиці users є:

- Id;
- Email – пошта користувача;
- Login – логін користувача;
- Password – пароль користувача;
- Confirmation_Password – поле для підтвердження паролю;

Перше поле, яке повинно бути в кожній таблиці це поле id, його ще називають первинним ключем таблиці. Первинний ключем розуміють поле, яке однозначно ідентифікують кожен запис в таблиці. Первинний ключ повинен бути мінімально достатнім: в ньому не повинно бути полів, видалення яких з первинного ключа не відіб'ється на його унікальності.

Ще один важливим полем є Email, щоб визначити користувача, чи є він реальною особою чи ні, так як у кожного є свою електронна скринька. А найбільш простим способом використання пошти - це ввід її як логін при вході користувача. Випадок використання пошти як логін користувача зустрічається дуже часто, так як більшість людей пам’ятають свою пошту. І з легкістю зможуть через деякий час увійти в систему знаючи свою пошту і пароль.

Зараз пошта як поле в таблиці бази даних і як інформація про користувача використовується для відсилання листів або ж для сповіщення важливої інформація, яку користувач може переглянути і отримати інформацій, наприклад про публікацію нової новини на сайті або ж добавлення нової фото.

Бувають випадки, коли користувач забуває пароль. У цьому випадку пошта може виконати рятувальну функцію, якщо користувач не може увійти у систему по причині того, що він забув пароль. Деякі веб-сайти надають можливість скинути пароль (Reset Password). Ось тут і ще один випадок використання пошти. Бо скидання паролю відбувається за допомогою того, що на пошту користувача надсилається лист із генерованою перед цим системою, унікального посилання, яке буде вести користувача на спеціальне посилання на якому буде розміщене форма для створення нового паролю.

Наступним важливим полем є login. Воно виконує ту ж функцію що і пошта, проте воно не є настільки універсальнм. По свої суті це поле, яке повинне бути користувачем знайоме або ж він легко може його запам’ятати. Проте на деякий веб-сайтах за допомогою нього можна змінити пароль у випадку його якщо він був забутий. Проте ця практика показує що це не зовсім безпечно. Часто в ролі логіну можу виступати поле email.

Напевно найважливішим полем в таблиці users є поле password. Його значення в на всіх веб-сайтах дуже високе. По своїй суті це просте текстове поле в якому буде збережено просто символі котрі будуть приховані в браузері, і можуть бути відкриті на серверній частині веб аплікації, що не є безпечним.

На даний час пароль стараються записувати у виді хешу для більшої безпеки даних. А під час входу в свій кабінет пароль і логін, який був введений користувачем надсилається на сервер, а він у свою очередь шукає користувача з таким логіном (це може бути і email), якщо запис в базі даних існує. Програма витягує цей запис з бази даних і перевіряє чи є цей хеш валідним з введеним користувачем паролем, якщо ні то хеш-функція (як прийнято називати її) поверне false. Що призведе до того, що програма поверне повідомлення з тим, що пароля для цього користувача не є вірним.

Ще одним досить важливим полем є confirmation_password. Його роль не є дуже важливою, проте його використовують. Суть його полягає в тому, що він створений для підтвердження паролю введеним користувачем при реєстрації або ж під час скидання паролю. У формах реєстрації часто зустрічається поле password і confirmation password. Це по свої суті одинакові поля і зберігаються вони одні і ті ж дані. Проте вони потрібні для того, щоб запевнитись, що користувач добре пам’ятає пароль і через деякий час користувач зможе знову успішно автентифікуватись на веб-сайті.

Якщо ж розглядати веб аплікація для шкіл, то таблиця users має бути приблизно такого вигляду. Поля таблиці users зображено на рисунку 2.1.

Users		
id	integer	
first_name	string(255)	
last_name	string(255)	
email	string(255)	
class_room	string(255)	
password	string	
confirmation_password	string	
Add field		

Рисунок 2.1 – Поля таблички users

Як бачимо окрім обов’язкових полів присутні такі поля як first_name, last_name, class_room. Це ті поля, які будуть містить особисту інформацію про користувача. First_name - буде містити ім’я користувача, last_name – прізвище і class_room – клас до якого буде належати користувач (в даному випадку учень).

Найпоширенішою і найпростішою табличкою в базі даних є новини, тобто Post. Новини ввиводяться на основній сторінці веб-застосунку, іншими слова на root шляху (localhost:3000). В даному випадку сутність Post - це просто новини школи, які будуть публікуватись на сайті. Створювати та редагувати новини (пости) може тільки користувач, який в системі є як адміністратор. Таблиця posts зображена на рисунку 2.2.

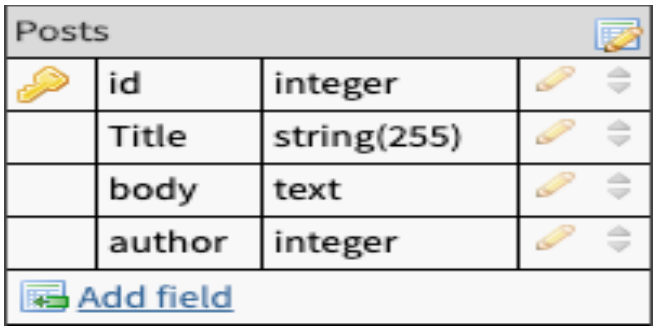

Рисунок 2.2 – Поля таблички posts

«Posts» містить наступні поля:

- Ідентифікатор (id);
- Заголовок (title);
- Тіло (body);
- вкладений файл (logo);
- Дата створення (created_at);
- Дата редагування (updated_at).

Кожна новина містить поле body в якому буде зберігатися самий текст новини. Не менш важливим полем є заголовок, який буде висвітлювати суть

новини в кількох речення. Також новина містить поле logo, яке відповідає за збереження назви картинки, яка уже виводитися разом із заголовком, текстом і яка в свою очередь буде зберігатися на сервері.

Розглядаючи функціонал шкільного сайту, структура бази даних набагато складніша і складається з багатьох таблиці, властивостей тощо. Тому, якщо детально розібратись, то повинні бути такі таблиці:

- home_tasks (домашні завдання);
- remote_tasks (завдання для віддаленого навчання);
- class_rooms (табличка де будуть зберігатися користувачі (учні), які відносяться до того чи іншого класу);
- schedules (табличка розкладу для кожного з класів);
- roles (таблиця для збереження прав, які буде мати кожен з користувачів (student, admin, teacher));
- reports (таблиця що буде містити відгуки учителя про учня).

Перша табличка (home_tasks) потрібна для збереження домашніх завдань, які будуть озвучені учителем в кінці уроку, і будуть відображуватись для учнів відповідного класу. Її вигляд зображено на рисунку 2.3.

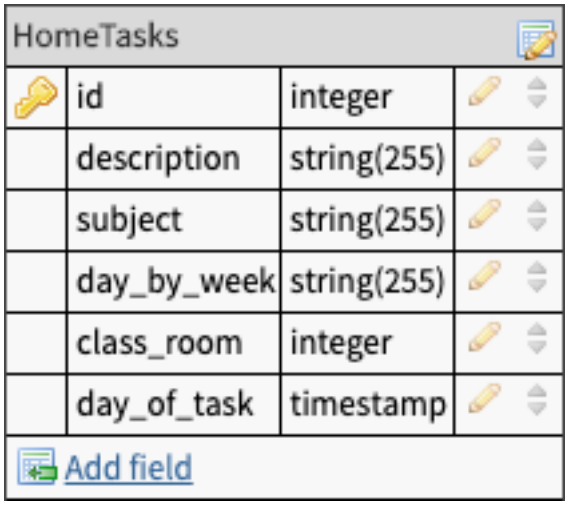

Рисунок 2.3 – Поля таблички home_tasks

Таблиця *home_tasks* містить поле id, яке в свою очередь є стандартним первинним ключем, які буде ідентифікувати кожен запис в цій таблиці. Другим полем є description (опис). Це поле буде містити самий текст домашнього завдання, що буде заповнюватися на вчителем після або під час уроку. Тип цього поля string з обмеженою кількістю символів (255).

Третім полем таблиці є поле subject. Ціль його поля зберігати вторинні ключі таблички subjects. Тобто кожне завдання буде мати зв'язок з іншою табличкою, а саме з subjects, що забезпечує зв'язок один до багатьох між домашніми завданнями і таблицею subjects.

Четвертим полем є day_of_week. Це поле містить день тижня (Понеділок, Вівторок, Середа…). Це поле потрібно що мати змогу сортувати або ж вибирати завдання для конкретного дня.

П'яте поле class_room. Воно як і поле subject містить вторинний ключ на табличку class_room. Ці дані можна використати для вибірки домашніх завдань для конкретного класу.

Шостим полем є day_of_task. Воно призначене для того, щоб містити дату того домашнього завдання на якій його треба виконати учню. Тип цього поля timestamp, що означає що він буде містити тільки дати будь якого формату.

Як видно таблиця *home_tasks* містить два вторинних ключа, які в майбутньому за допомогою INNER JOIN зможемо вибирати домашні завдання для конкретного класу або ж для конкретного шкільного предмету.

Друга таблиця в базі даних з назвою remote_tasks зберігатиме завдання, які вчитель буде публікувати поза урочний час, який у свою очередь так чи інакше повинен буде виконаний учнями. Ці завдання дуже схожі за своєю суттю і досить схожі за структурою із простими домашніми завдання, які публікуються вчителем. Проте їх мета різна, так як дистанційні завдання призначені для віддаленого навчання, тобто без участь в процесів навчання самого учителя.

Як і з простими домашніми завданнями, віддалені теж містять два вторинних ключа, ті ж самі що в home_task. Отож, графічне зображення полів таблиці зображено на рисунку 2.4.

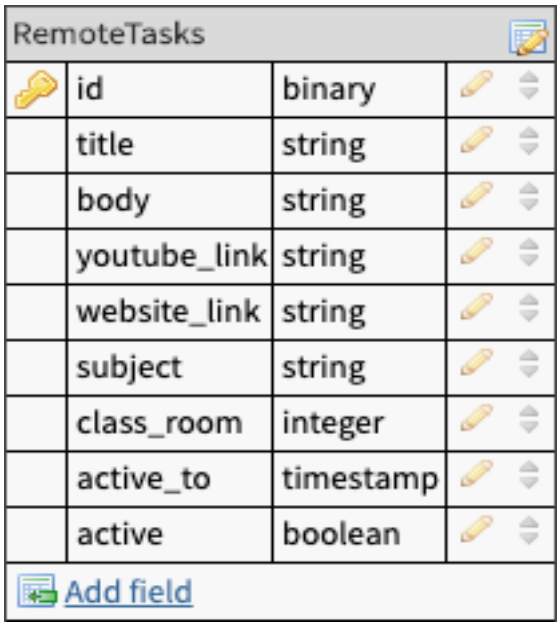

Рисунок 2.4 – Поля таблички remote_tasks

Перше поле під назвою title буде містити заголовок. Тобто вчитель може в заголовку приблизно писати про що йдеться в цьому завданні або ж описати тему, яку учень зможе опрацювати під час виконання даного завдання.

Друге поле body є полем, де вчитель зможе докладно описати суть дистанційного завдання, а учень в свою чергу, зможе глибше розібратись з суттю завдання, що допоможе йому краще, скоріше і більш ефективніше виконати дистанційне завдання.

Поле youtube_link є третім полем в таблиці remote_tasks. Воно буде містити посилання на відео матеріал, як приклад в назві вказано відео платформу YouTube. Тож вчитель може прикріплювати до дистанційного завдання відеоматеріал, який учень зможе використати для успішного виконання завдання.

Четверте поле website_link досить суттєво пов’язане з полем youtube_link тим, що ці два поля зберігають посилання на джерела з інших сайтів. Проте це поле призначене для збереження посилань на текстові джерела інформації.

П’ятим та шостим полем як і в таблиці home_tasks буде містити вторинні ключі. Тобто, як прості домашні завдання дистанційні завдання теж асоціюються з предметом та з класом, для кого він зроблений.

Сьоме поле active_to слугує для збереження дати, до якої учень матиме доступ до дистанційного завдання. Тобто, в учителя є можливість добавити «дедлайн», тобто дату, до якого учню потрібно зробити завдання, після дистанційне завдання стане не доступним. І щоб зробити його доступним, учителю доведеться поновлювати дату проходження.

Останнім полем в таблиці remote_tasks є поле active, тип цього поля є boolean, яке буде містити мітку чи є дистанційне завдання активним. А активне воно тоді коли, пройшов термін активності самого завдання

Напевне найважливішою таблицею, що є в базі є class_rooms. Вона потрібна для того, щоб користувач з правами student міг отримувати домашнє завдання, дистанційне завдання, розклад занять, сповіщення. Так як всі ці сутності будуть прив’язані саме до цієї таблиці. Отож, вона містить поля, зображені на рисунку 2.5.

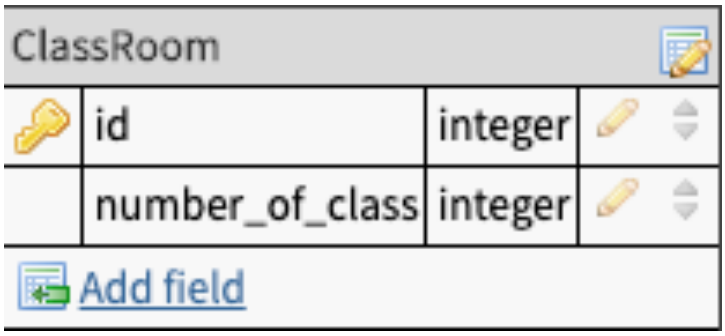

Рисунок 2.5 – Поля таблички class_rooms

Ця таблиця містить всього два поля, проте вона дуже важлива. Перше поле це первинний ключи для ідентифікації запису в таблиці. Друге поле буде містити назву або ж номер класу, так як класи в школах та гімназіях мають числове найменування.

Ще однією таблицею є schedules. Вона буде містити розклад занять для кожного з класів і буде доступна тільки для користувачів, які перебувають у якомусь класі.

Усі таблиці пов’язані між собою різними типами зв’язків. Є 3 основні зв’язки:

- Один до багатьох (one-to-many);
- Один до одного (one-to-one);
- Багато до багатьох (many-to-many).

Найбільш часто задіяний зв'язок один до багатьох. Практично кожна таблиця реалізовує в собі цей зв’язок. Загальний вигляд бази даних зображено на рисунку 2.6.

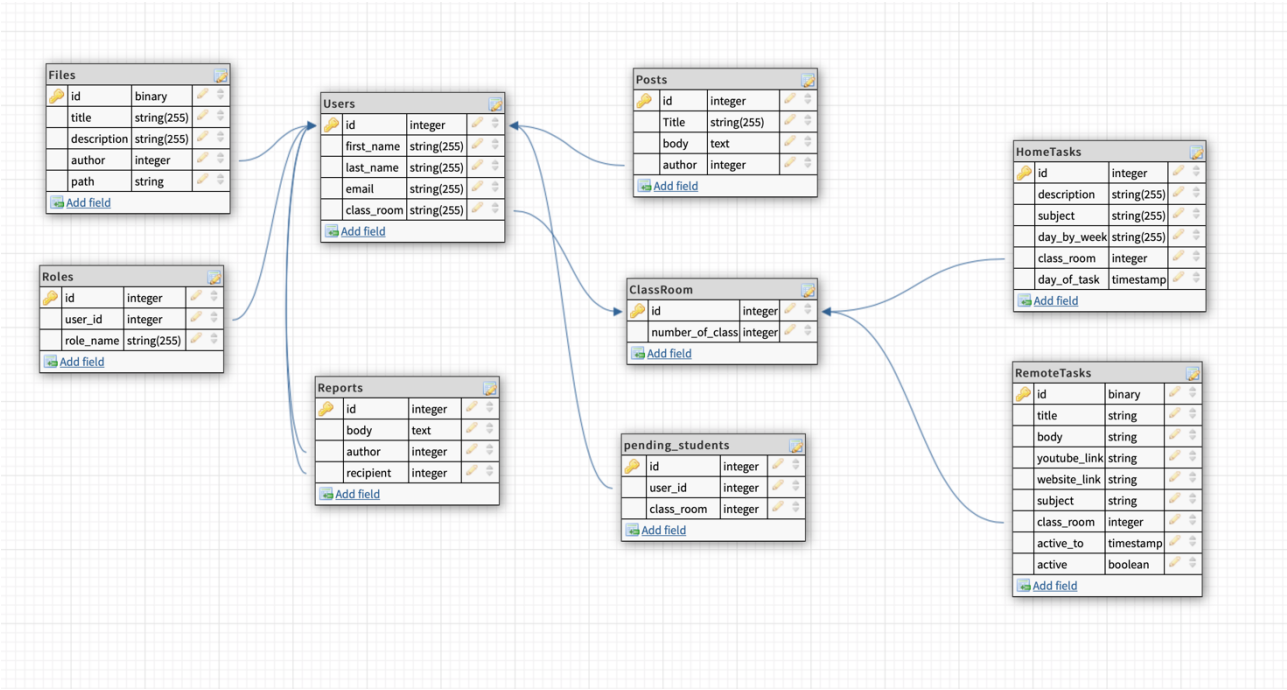

Рисунок 2.6 – Зображення структури бази даних

2.2 Структура проекту

Веб-сервіс шкільного сайту розроблений на мові програмування Ruby та на її фреймворку Ruby on Rails. При генерації проекту створюється стандартний набір папок. Структура веб-аплікації розділяє бізнес логіку, вигляд, і роботу з базою даних по відповідних директоріях. В даному випадку розглядається структура шкільного сайту.

Щоб зрозуміти структуру каталогів проекту, потрібно наочно побачити як він виглядає. Вигляд структури проекту зображено на рисунку 2.7.

Рисунок 2.7 - Структура Rails аплікації

App – містить ввесь код програми. Початкова структура проекту генерується самостійно, вона слідує MVC структурі, тобто вони розміщені в різних директоріях.

Bin – містить в собі Rails код, який буде запускати Rails аплікацію. Також каталог bin може включати скрипти для розгортання аплікації на віддаленому сервері (Heroku).

Config – має невеликий обсяг коду конфігурації, який знадобиться цьому додатку, включаючи конфігурацію бази даних (в файлі database.yml), структуру середовища Rails (environment.rb) і маршрутизацію запитів на веб-сервер (routes.rb). Там описуються усі конфігурації Rails додатку.

DB – у цьому каталозі зберігаються міграції проекту, а також два файли. Перший має назву seeds.rb. У ньому міститься код, який буде добавляти тестові дані для production так і для test середовищ розробки. Для запуску цього файлу є спеціальна команда, а саме rake db:seed. Другим файлом є schema.rb, у ньому збережена вся схема бази даних, він формується автоматично, та змінюється коли запускається міграції і коли ця міграції успішно ввійшла в дію.

Lib – тут ви можете зберігати код інших бібліотек. Тобто тут розміщується сторонній код, який реалізовує код будь якої з бібліотек

Log – тут зберігаються логи роботи rails аплікації. Rails створює скрипти, що допомагають керувати різними журналами помилок. Для кожного середовища розробки (production, development, test) є окремий файл логів.

Public – у цьому каталозі зберігаються статичні файли проекту. Наприклад картинки, відео тощо. Бібліотека carrierwave в стандартній конфігурації зберігає файли саме в цьому каталозі.

Spec – цей каталог є згенерованою папою бібліотеки rspec. В ньому зберігаються тести на різні частити програми, а саме на контролер, модель, мейлери, хелпери тощо.

Tmp – цей каталог містить в собі тимчасові файли. Наприклад файл pid, в ньому є зберігається id процесу в системі.

Окрім каталогів, Rails застосунок генерує кілька файлів. Основними: README, Gemfile та Rakefile.

Gemfile – це файл в якому міститься геми які потрібні для розробки. Bundler – це менеджер для управління сторонніми бібліотеками в ruby додатках. Ця бібліотека дозволяє встановлювати потрібні геми для застосування, дає можливість встановлювати геми специфічних версій. Bundler включили в Rails 3.0 за замовчуванням і тепер, саме він використовується для управління залежностями гемів в даній версії фреймворка. Його вигляд подано в додатку А.

Rakefile – цей файл допомагає при складанні, упаковці та тестуванні коду Rails, створенні, видаленні бази даних, чи навіть виведення всіх URL шляхів застосунку. Утилітою rake поставляється разом з установкою Ruby.

README – цей файл потрібен для відображення всієї документації про проект. На git репозиторії це файли буде відображуватися нижче усіх кореневого каталогу.

Вище наведено опис кореневого каталогу rails аплікації. Ввесь код зберігається в папці app.

- app / controllers – У ньому міститься всі контролери rails аплікації.
- app / models – Цей каталог містить файли моделей. У ньому rails аплікації зберігає файли бізнес логіки, які rails роблять доступними в усіх папках директорії.
- app / view – Тут зберігається HTML код, який буде рендеритись контролером. Для кожного контролера в цій папці є окрема папка, яка буде названа його іменем, і яка буде містити html файли для кожного з екшнів контролера.
- app / view / layouts – Так як Rails аплікації побудована на базі Single Page Application. Цей каталог буде містити html код, який буде однаковий для усіх ресурсів аплікації.
- app / helpers – підкаталог helpers містить всі допоміжні класи, які будуть використовуватися у файлах вигляду а також у контролера, для того щоб

не нагромаджувати код. Це вирішує проблему нагромадження моделей, контролерів і файлів вигляду лишнім кодом.

- app / assets – цей каталог містить JavaScript та CSS файли. Який за допомогою assets pipeline буде зібрано в один файл і віддано клієнту на обробку. Це зроблено для підвищення продуктивності rails аплікацій.
- app / javascript – каталог містить файли з розширенням js, для кожного контролера буде генеруватись окремий файл, в якому можна обробляти вигляд HTML сторінки.

Каталог app містить в собі основний код rails аплікації при цьому реалізовуючи шаблон проектування MVC.

2.3 Інструменти розробки веб додатку

2.3.1 Мова програмування Ruby

Ruby - це інтерпретована, повністю об'єктно-орієнтована мова програмування з динамічною типізацією змінних. Мова вирізняється швидкою та динамічною розробкою програм і унаслідувала усе чим пишались програмісти таких мов як: Perl, Java, Python, Smalltalk, Eiffel, Ada і Lisp. Ruby реалізовує в собі Perl-подібний синтаксис із об'єктно-орієнтованим підходом мови програмування Smalltalk. Також Ruby наслідує деякий “цукор” від таких програм як Python, Lisp, Dylan та CLU.

В Ruby майже все є об'єктом. Для кожної частини коду є визначені власні властивості і методи. В об'єктно-орієнтованому програмуванні властивості називаються змінними об'єкта. Найчастіше об'єктно-орієнтований підхід Ruby може бути продемонстрований парою рядків коду, в яких проводиться дія над числом. У інших мовах програмування, числа та інші примітиви не є об'єктами. Ruby під впливом мови Smalltal, унаслідувався підхід того, що примітиви теж є

об’єктами і теж мають свої властивості і методи. Це спрошує використання Ruby, так як правила застосовні до об'єктів - застосовані буквально до всього рубі загалом Ruby.

Ruby дуже гнучка мова, вона дозволяє переписувати готовий код, що дозволяє модифікувати ruby під свої потреби. Основні частини Ruby можуть бути видалені або перевизначені за бажанням. Ruby старається ні в чому не обмежувати розробника. Наприклад, метод додавання виконується операцією плюс (+). Але розробника надається додати до класу Numeric свій власний клас, в якому він змову створити свій метод plus і за допомогою нього додавати до об’єкту ще одне число .

Блоки в Ruby теж є відмінним джерелом гнучкості. Розробник може додати блоки до кожного методу, реалізувати його так, як цей метод повинен діяти. Замикання – це по суті блок коду, що є однією з найпопулярніших конструкцій для тих, хто прийшов у світ Ruby зі світу імперативних мов програмування, таких як PHP або Visual Basic. Створення блоків було взято від функціональних мов програмування. Юкіхіто Мацумото (розробник Ruby) говорив: «замиканнями в Ruby я хотів віддати данину поваги культурі мови Lisp.»

Ruby навмисно не дозволяє множинне наслідування. Але Ruby замінює це можливістю створення модулів. Модулі – це окрема сутність, в якій містяться методи і змінні. Класи можуть вільно імпортувати собі модуль і все що він в собі містить, тобто його методи та змінні. В основному, програмісти що працюють з мовою програмування Ruby охоче цим користуються, так як це є більш безпечнішим ніж множинне спадкування, яке може бути досить складним і в майбутньому викликати серйозні проблеми.

Оскільки в Ruby знаки пунктуація зустрічається досить рідко і, зазвичай, в якості слів для іменування використовується англійська мова, деякі знаки пунктуації використовуються для кращого розуміння Ruby коду. Ruby не потребує явне оголошення змінних. У ньому загально прийняті угоди по іменування коду, щоб розділити на області видимості змінних, наприклад:

- variable - може бути локальною змінною.

- @variable - змінна об'єкта.
- $variable - глобальна змінна.

Дана символіка підвищує читабельність коду, дозволяючи програмісту легко зрозуміти, яка саме логіку написаного коду.

Ruby сповнений іншими особливостями і конструкціями, і ось деякі з них:

- В Ruby є конструкції для обробки exceptions, як в Java, які дозволяють обробляти помилки, які виникають під час роботи програми;
- В Ruby існує carbage colector. Це збірник сміття в пам’ять. Тобто він очищує пам’ять він лишніх об’єктів.
- Ruby дає можливість розробникам підключати сторонні бібліотеки на мові програмування C, що дає можливість підвищення продуктивності виконання програми.
- Ruby може загружати додаткові сторонні бібліотеки на «льоту», якщо це дозволяє операційна система.
- В Ruby реалізована можливість роботи з потоками, які залежать від операційної системи.

Мова програмування Ruby надає декілька різних своїх реалізацій. Вони є дуже корисними в різних випадках, надають можливість інтегруватися з різними мовами та оточеннями.

Список інших реалізацій мови Ruby:

- JRuby – це Ruby, яка в більшості реалізована на мові програмування Java. Це реалізації запускається на JVM (Java Virtual Machine).
- Rubinius – це «Ruby написаний на Ruby». Реалізовано на основі LLVM - віртуальній машині, на який створено й інші відомі мови.
- IronRuby - це реалізація яка пов’язується з мовою програмування C#.
- Cardinal - це «компілятор Ruby для віртуальної машини Parrot» (Perl).

Ruby — об'єктно-орієнтована мова програмування. Кожен тип даних є об'єктом, включно з типами та класами, котрі в багатьох інших мовах реалізовані як примітиви (такі як «integer» або «null»). Кожна функція є методом.

Змінні Ruby містять не самі об'єкти, а посилання на них. Присвоєння — це не присвоєння значення, а копіювання посилання на комірку пам’яті на об'єкти. Для поширених гібридизованих мов програмування, наслідки такого вирішення можуть здаватись не звичними. Наприклад:

тобто при зміні значення змінної *a* неявно змінилось і значення *b*, оскільки вони містять посилання на один об'єкт. З іншого боку, це логічніше, ніж ситуація, коли для змінних різних типів присвоєння діє по різному (наприклад в Object Pascal).

В Ruby немає можливості успадковуватися від кількох класів, але це замінене існуючим, потужним механізмом MixIn. Всі класи (безпосередньо або через інші класи) успадковані від класу Object, тому, кожен об'єкт буде містити методи які є в класі Object (наприклад, class, to_s, nil?). Процедурний стиль також підтримується, але всі процедури приховано залишаються методами екземпляру класу Object.

Ruby був задуманий як мультипарадигмова мова програмування. Він підтримує процедурну (кожна функція або змінна, об’явлена поза межами класу, автоматично стає методом або зміною класу Object, що є батьківським для всіх інших класів), об'єктно-орієнтовану (все що є, є об'єктом), або функціональну парадигму програмування. Ruby підтримує динамічну типізацію даних та поліморфізм. Ruby зараз підтримує Unicode, хоча частково реалізована підтримка UTF-8.

2.3.2 Фреймворк Ruby on Rails на базі мови Ruby

Ruby on Rails — об'єктно-орієнтований готовий каркас (*фреймворк*) для створення веб-додатків, написаний на мові програмування Ruby. Ruby on Rails реалізовує парадигму модель-вид-контролер (Model-View-Controller) для

веб-застосунків, а також забезпечує їхню безпосередню інтеграцію з веб-сервером і базою даних.

Ruby on Rails був створений Девідом Гайнемаєр Генссоном на основі його роботи над інструментом керування проектами Basecamp і був зарелізаний в липні 2004 року. Ruby on Rails є open sourse програмним забезпеченням і розповсюджується безплатно [7].

Модель надає решті компонентів програми об'єктно-орієнтоване представлення даних (таких як пости або список домашнього завдання). Об'єкти моделі здійснюють витягування, збереження, оновлення та видалення даних в базі даних.

Завдяки можливостям оголошення змінної без необхідності оголошувати тип в мові Ruby, розробникові досить унаслідувати свою моделі від класу ActiveRecord::Base. Ruby on Rails зможе автоматично прив’язати класи моделі до таблицями в базі даних і створює атрибути об'єктів для відповідних полів таблиці.

Вид реалізовує інтерфейс користувача для відображення отриманих від моделі даних. Вид також передає HTTP запити користувача на маніпуляцію даними в контролер, який у свою чергу працює з класами моделей (як правило, вид не змінює безпосередньо дані з моделі).

У Ruby on Rails вид описується за допомогою модифікованого шаблону RHTML. Це є прості файли HTML з можливостями включення фрагментів коду Ruby (Embedded Ruby або ERb). Вивід, згенерований вставленим кодом Ruby, вставляється в текст шаблону сторінки HTML, яка після цього повертається користувачеві у вигляді response. Види можуть використовувати фрагменти інших видів і, у свою чергу, бути включеними в шаблон (layout) вищого рівня.

Контролер — основний компонент, що відповідає за взаємодію з сервера користувачем. Контролер витягує необхідні дані з моделі і використовує їх для відображення, а також зберігає отримані дані які прийшли від клієнта.

Усі контролери в Ruby on Rails є класами, які унаслідуванні від ActionController::Base. Відкриті методи контролера є так званими діями тобто

actions (екшни). Action часто відповідає окремому видові. Наприклад, по запиту користувача admins буде викликаний екшн index класу AdminsController і потім використаний вид index.html.erb.

Варто зазначити те, що на мові програмування Ruby працюють в основному професіонали: поріг входження у мову є високим, тому програмісти на Ruby приходять, уже маючи за спиною будь який досвід в програмуванні. Тому навіть початківцем в Ruby можна рахувати досвідченого веб-розробника з досить великим запасом знань і досвіду. Для мови Ruby найпопулярніший фреймворк - це Rails, більше 90% Ruby веб-застосунків, які написані на Ruby, реалізовані саме Rails.

Основними правилами розробки на Rails є:

- Принцип DRY (Do not repeat yourself) – фреймворк для правильного написання коду. Він надає механізм щоб протидіяти повторенню коду в rails аплікаціях;
- Принцип Convention over configuration - за замовчуванням у фреймворку використовуються численні конфігурації, які зустрічаються і в інших веб фреймворках. Це дуже полегшує створення додатків, так як не стандартна конфігурації проекту потрібно тільки в нестандартних випадках;
- Автоматизоване тестування - в складі RoR поставляються інструменти для повністю автоматичного тестування, а ідеологія Ruby on Rails передбачає використання методології розробки тестування (TDD - Test Driven Development). Все це робить реалізовані аплікації надійними;
- Можливість розширення фреймворка Ruby on Rails.

Ruby on Rails складаєть велика система з додаткових бібліотек або ж готових рішень з відкритим вихідним кодом («джемів», gems), які реалізують найбільш потрібні функції. «Джеми» бувають різні: від бібліотекою на C, що відповідають за якийсь внутрішній аспект роботи аплікації, до високорівневих, що представляють собою окремі модулі для вирування великого спектру завдань.

Використання системи підключення сторонніх бібліотек стало однією з основних причин високої популярності фреймворка - можливість вибірково підключати різні компоненти і бібліотеки дуже сильно прискорює процес розробку, а той факт, що використовувані бібліотеки добре протестовані і «дебажиться» роками, забезпечує надійність рішень які використовують ці бібліотеки.

2.3.3 СКБД PostgreSQL

PostgreSQL — об'єктно-реляційна система керування базами даних (СКБД). Є альтернативою як комерційним СКБД (Oracle Database), так і СКБД з відкритим кодом (MySQL).

Якщо порівнювати з іншими СКБД, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією, над її розробкою було задіяно велика кількість розробників і компанії. Які вирішували свої потреби, і ставити завдання перед цією базою даних виходячи тільки зі своїх потреб.

PostgreSQL реалізовано на мові програмування C. Зазвичай розповсюджується у вигляді набору текстових файлів із відкритий кодом. Для інсталяції необхідно виконати компілювання файлів на своєму локальному комп'ютері і скопіювати в деякий каталог. Весь процес детально описується в документації до неї.

Функції дозволяють виконувати код безпосередньо самою базою даних. Ці функції можуть бути написані на чистому SQL, який має деякі стандартні програмні оператори, такі як розгалуження та цикли. Але гнучкішою буде функція нереалізована на одній із доступних мов програмування, з якими PostgreSQL може працювати. До таких мов належать:

- Вбудована мова, яка зветься PL/pgSQL, подібна до мови написання запитів PL/SQL компанії Oracle;
- Мови розробки сценаріїв: PL/Perl, PL/Python, PL/Tcl, PL/Ruby, PL/sh;
- Класичні мови програмування C, C++, Java (за допомогою PL/Java).

Функції можуть виконуватись від імені користувача з правами, який її викликав. У PostgreSQL є підтримка індексацій наступних видів: B-дерево, хеш, R-дерево, GiST, GIN. Проте є можливість реалізувати свої типи індексації [8].

PostgreSQL підтримує одночасне керування базою даних декількома користувачами за допомогою механізму Multiversion Concurrency Control (MVCC).

PostgreSQL підтримує великий набір вбудованих типів даних:

- Цілочисельні типи;
- Цілі;
    - З фіксованою крапкою;
    - З нефіксованою крапкою;
- Грошовий тип;
- Символьні типи;
- Двійкові типи (включаючи BLOB);
- Типи часу;
- Бінарний тип;
- Перерахування;
- Геометричні примітиви;
- Мережеві типи;
    - IP і IPv6-адреси;
    - CIDR-формат;
    - MAC-адреса;
- UUID-ідентифікатор;

- XML;
- JSON;
- Array;
- OID-типи;
- Псевдотипи.

Крім того, розробник має можливість створювати власні, потрібні йому типи та програмувати для них механізми індексування даних за допомогою GiST. PostgreSQL може бути розширено користувачем для власних потреб практично в будь-якому аспекті. Є можливість додавати власні:

- Перетворення типів;
- Типи даних;
- Домени імена;
- Дефініції (включаючи агрегатні);
- Індекси;
- Оператори;
- Процедурні мови.

Таблиці мають можливість бути успадковувати від інших таблиць (батьківських). Дані в дочірній таблиці при додаванні будуть брати участь і в запитах до батьківської таблиці. Ця функція є ще на стадії розробки.

Тригери визначаються як функції, що ініціюються DML-операціями. Наприклад, операція INSERT може запускати тригер, що перевіряє доданий запис на відповідність певним умовам. Тригери можна писати різними мовами програмування. Вони пов'язані з визначеною таблицею. Множинні тригери виконуються в алфавітному порядку.

- Дотримання принципів ACID;
- Відповідність стандартам ANSI SQL-92 і SQL-99;
- Підтримка запитів з OUTER JOIN, UNION, UNION ALL, EXCEPT і підзапитів;

- Послідовності;
- Контроль цілісності;
- Реплікація;
- Загальні табличні вирази й рекурсивні запити;
- Аналітичні функції;
- Підтримка Unicode (UTF-8);
- Підтримка регулярних виразів у стилі Perl;
- Вбудована підтримка SSL і Kerberos;
- Протокол поділюваних блокувань;
- Завантажувані розширення, підтримують SHA1, MD5, XML і іншу функціональність (API відкритий);
- Засоби для генерації сумісного з іншими системами SQL-коду та імпорту з інших систем.

2.3.3 Redis, Memcached, Puma, Sidekiq

Рисунок 2.8 – Логотип Memcached

Memcached — комп'ютерний сервіс кешування даних в оперативній пам'яті на основі хеш-таблиці. Логотип Memcached зображено на рисунку 2.8.

З допомогою клієнтської бібліотеки (для Perl, PHP, Python, Java та ін.) дозволяє кешувати дані в оперативній пам’ять одного або декількох серверів. Розподіл даних реалізується по значенню хеш ключа. Використовуючи ключ для даних, бібліотека визначає його хеш і використовує його для вибору окремого сервера. Ситуація, коли не знайдено сервер, проявляється як не існування кеша. Це дозволяє проводити заміну серверів.

В API memcached є тільки потрібні функції: вибір сервера, настройка з'єднання, додавання, знищення, оновлення і витягування об'єкта. Для кожного з існуючих об'єків встановлюється час активності, починаючи з початку існування до нескінченності. При переповненні оперативної пам'яті старі об'єкти автоматично будуть знищуються [9].

Сервер memcached було рреалізовано для сайта LiveJournal з метою ззменшення навантаження на базу даних.

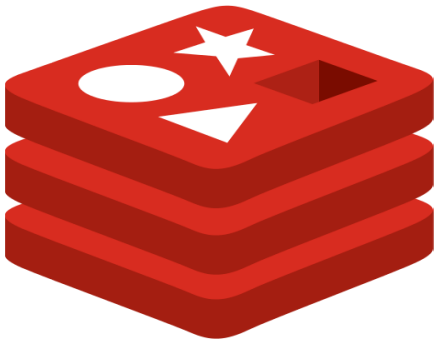

Рисунок 2.9 – Логотип Redis

Redis —сховище яке базується на парах ключ-значення, які зберігаються в оперативній пам'яті, з можливістю забезпечувати довготривалого зберігання яке можна налаштувати в конфігураційних файлах. Це програмне забезпечення з відкритим кодом написане на мові програмування C. Розробка Redis профінансована компанією VMware. Sourse code проекту поширюється в межах ліцензії BSD. Логотип Redis зображено на рисунку 2.9.

Redis також розширений підтримкою структурованих даних, таких як послідовні списки, хеш і множини. На відміну від Memcached, Redis забезпечує постійне перезбереження даних чим самим гарантуючи збереження даних бази даних у разі аварійного завершення роботи. Існують багато бібліотек які дозволяють багатьом мовам програмування працювати з Redis, а саме: Python, PHP, Java, Ruby [10].

Є підтримка передач даних, що дозволяють виконати за один крок кілька пов’язаних команд, гарантуючи відсутність несумісносте і послідовностей (команди від інших запитів не можуть вклинитися) виконання даного набору різнотипних команд, а в разі проблем дозволяє повернути внесені зміни.

Зберігання даних в оперативній пам'яті комп’ютера допомагає досягнути великої продуктивності.

Для управління даними підтримуються такі команди, як інкремент/декремент, стандартні операції над списками і множинами (об'єднання, перетин), перейменування ключів хешів, множинні перебори даних та функції сортування даних. Можлива реалізація master-slave дуплікації даних на кілька окремих серверів, здійснювана в простому режимі.

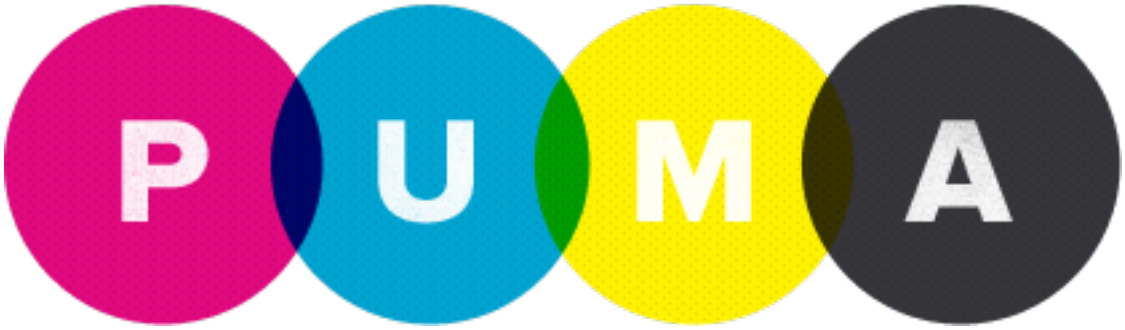

Рисунок 2.10 – Логотип Puma веб-серверу

Puma обробляє HTTP-запити, використовуючи розширення Ragel, яке забезпечує швидкісний, точний розбір протоколу HTTP 1.1 на частини. Потім Puma обслуговує запит, використовуючи пул потоків протоколу. Кожен HTTP-запит подається в окремому потоці, тому інші реалізації Ruby (JRuby, IronRuby) будуть використовувати всі вільні ядра процесора.

Puma був розроблений як сервер для підтримки Рубінія, але також може працювати з JRuby та MRI. Логотип Puma зображено на рисунку 2.10.

На MPT є глобальний VM Lock (GVL), який буде слідкувати, за тим щоб лише один потік зміг запустити Ruby код. Але якщо ви заблокуєте IO наприклад, HTTP-запити до зовнішніх API, наприклад Twitter, Facebook. Puma покращує пропускну здатність MRI, дозволяючи потокам працювати паралельно, без особливих затримок [11].

Puma входить до списку рубінових веб-серверів. Основним елементом, який залишається, є великий HTTP-аналізатор Mongrel, написаний в ragel і виконаний на C або Java, залежно від реалізації ruby.

По-перше, світ Ruby майже повністю змінився, щоб виконувати Rack завдання як основний інтерфейс для веб аплікацій. Mongrel був написаний у світі перед самим Rack. По-друге, Puma розроблений для виконання і реалізації Ruby, яка дозволяє реалізувати «concurrency», такий як на інших версіях Rubinius та JRuby

Сьогодні Puma працює з усіма існуючими реалізаціях Ruby, але завжди буде найкращим та реалізації, яка забезпечує дійсно найкраще реалізований

паралелізм та надійність. Може забезпечити легкий і продуктивний конвеєр із запиту / відповіді до Rack аплікацій, що дозволяє використовувати його для майже всіх існуючих рубінових веб-аплікацій.

Рисунок 2.11 – Логотип Puma веб-серверу

Sidekiq - це структура з відкритим вихідним кодом, яка забезпечує ефективну обробку фонових завдань для додатків Ruby. Логотип Redis зображено на рисунку 2.11. Redis використовує як структуру даних, пам'яті комп’ютера для зберігання всіх своїх важливих та цінних даних. Sidekiq за замовчуванням не виконує планування завдань планування, а тільки їх виконує самі завдання. Ми можемо реалізувати планування завдань за допомогою об’єкта Resque, яка уже є в Redis [12]. Він надає готові рішення для синхронного виконання будь-яких поставлених завдань або для планування їх виконання, додаючи їх у чергу завдань. Він також виконує повторний запуск завдань після їх невдалих спроб виконання.

Отже, Sidekiq є платформою для виконання завдань в асинхронній послідовності.

3 РОЗРОБКА ШКІЛЬНОГО САЙТУ

3.1 Сторонні Ruby бібліотеки для Rails аплікації

Під час розробки веб додатку для школи важливим є реалізувати функціонал, що дозволить користувачам авторизовуватися на сайті. Спільнота мови програмування Ruby створило бібліотеку або ж як воно називається мовою Ruby – gem (гем). Ця бібліотека називається devise. За цього гему ми можемо здійснивши кілька команд в консольній вікні, створити протестований та надійний готовий функціонал для авторизації користувачів.

Devise – це Ruby бібліотека, що надає можливості для автентифікації в rails-аплікаціях. Devise працює на базі іншого гему, який в свою чергу надає механізм для аутентифікації і авторизації ruby програмах зокрема rails. Основні особливості Devise описані нижче:

- Реалізовано на базі Rack бібліотеці;
- є закінченим MVC-рішенням;
- дозволяє вхід в систему використовуючи кілька моделей (M);
- Використовує тільки те, що потрібно.
- Легко піддається переписуванню.

Даний гем встановлюється разом з декількома іншими гемами. Наприклад, гем для шифрування паролів «bcrypt-ruby». Він надає просту обгортку для роботи з паролями. В основі лежить хеш-функція bcrypt(). А гем «orm_adapter» – надає єдину точку входу для використання основних функцій Ruby ORM. Також разом з «devise» установлюється гем «thread_safe» (надає потоко-безпечні колекції і утиліти для Ruby), та «railties» (внутрішні компоненти Rails, такі як завантажувачі додатки, плагіни, генератори і rake-завдання та job-завдання). Для завантажування всіх сторонніх бібліотек для цього гему потрібно тільки вказати його назву в файлі «Gemfile» та запустити в терміналі бібліотеку, яка

завантажить всі інші бібліотеки з віддалених репозиторіїв, а саме запустити команду: *bundle install* (в новіших версія просто *bundle*).

В даний момент, гем був викачаний з віддалених репозиторіїв, але ще необхідно згенерувати модель, яка буде прив’язана до rails аплікації. Для виконання імплементацію бібліотеки Devise потрібно виконати в терміналі команду: *rails generate devise:install*. Після виконання цієї команди, у вашому проекті гем згенерує файли які потрібні для успішної роботи «devise». Наступним кроком була імплементація моделі User (users). За замовчуванням буде підключено 6 модулів. Але ми можемо відредагувати цей список:

- Database Authenticatable: надає можливість входу в систему на основі зашифрованого паролю який захешованим зберігається в базі даних. Вхід може бути виконаний за допомогою відправки POST-запиту з HTML форми, яку Devise автоматично згенерує.
- Omniauthable: додає авторизація через соціальні мережі (github, twitter, facebook, google, instagram)
- Confirmable: додає можливість відправляти лист для підтвердження аккаунта після його реєстрації.
- Recoverable: дозволяє відновлювати пароль (reset password). Надсилає інструкції по відновленню паролю на пошту.
- Registerable: управляє реєстрацією користувачів на сайті, дозволяє маніпулювати акаунтами.
- Rememberable: дозволяє запам'ятовувати користувачів за допомогою cookies, управляє створенням і видаленням авторизаційних токенів.
- Trackable: веде статистику кількості кожного користувача.
- Timeoutable: слідкує за тривалістю сесії активності користувача на сайті;
- Validatable: надає можливість валідувати дані користувача при реєстрації, або ж під час логіну користувача.
- Lockable: дає можливість блокувати акаунта користувача, якщо кількість спроб буде більшою ніж зазначеною в конфігурацій. Дає можливість при

блокуванні акаунта розблокувати його за допомогою email або ж блокування пропаде після якогось часу.

Вигляд моделі devise-у подана в додатку Г. Так як devise є веб бібліотекою вона чудово працює у зв'язці з rails фреймворком. Бібліотека настільки популярна, що її використовують сторонні бібліотеки такі як CanCanCan.

CanCanCan - це бібліотека авторизації для Rails аплікації, яка обмежує доступ до ресурсів devise користувача.

Усі доступи можна надавати в одному або декількох файлах можливостей і не дублюватись через контролери, вигляд окремих даних та запити до бази даних, зберігаючи бізнес логіку доступів в одному, зручного для обслуговування та тестування місці.

Функціонал бібліотеки складається з двох основних частин:

- Бібліотека авторизації, яка дозволяє оголосити доступу до різних ресурсів аплікації, і надає так званих хелпер-методів для перевірки цих дозволів;
- Хелпер-методи Rails для скорочення коду в Rails Controllers, виконуючи завантаження та перевірку доступів моделей автоматично при цьому керуючись принципом «Don`t repeat myself».

По суті за допомогою неї можна розширити роботу devise. Реалізовує функціонал ролей на сайті. Тобто є можливість добавляти різних типів користувачів. В даному випадку student, admin, teacher.

Обидві бібліотеки легко співпрацюють. Добавивши в Gemfile «gem 'cancancan'» після цього виконавши команду в командному рядку «bundle install». Ми отримаємо стабільну версію цієї бібліотеки на свою локальну машину. Далі для створення взаємодії між devise і cancancan, потрібно згенерувати файли необхідні бібліотеці для коректної роботи в проекті. Для генерації файлів потрібно в консолі прописати, «rails g cancan:ability». Ця

команда згенерує class Ability, в якому буде зберігатися настройка доступа окремої групи користувачів до окремих ресурсів веб-сервісу.

Коли ми згенерували доступи до для окремих груп користувачів, треба реалізувати самі ролі. Для цього спільнотою rails було створено рішення, а саме бібліотека Rolify. За допомогою її ми можемо з легкістю реалізувати на рівні бази, контролерів, моделей можливість погрупувати всіх користувачів на групи.

Всього на сайті будуть користувачі трьох типів, а саме: адміністратор, вчитель, учень. Вони зможуть:

Таблиця 3.1 – Доступність ресурсів для кожної групи користувачів

Дія	Адміністратор	Учитель	Учень
Переглядати домашнє завдання	+	+	-
Створювати, редагувати, видаляти домашнє завдання	+	+	-
Додавати дистанційне завдання	+	+	-
Переглядати дистанційне завдання та відправляти відповіді на них	+	-	+

Переглядати новини сайту	+	+	+
Створювати, редагувати, видаляти новини	+	-	-
Переглядати прикріплені файли	+	+	+
Створювати, редагувати, видаляти файли	+	-	-
Переглядати зауваження учня	+	-	+
Створювати зауваження для учня	+	+	-
Особистий кабінет	+	+	+
Перегляд фотогалереї	+	+	+
Добавляння нових фотографій для школи	+	-	-

Кінець таблиці 3.1

Можливість приймати заявки на вступ до класу	+	+	-
Переводити на новий рік учнів	+	+	-
Ставити оцінки за дистанційні завдання	-	+	-
Видаляти користувачів	+	-	-

Як і інші геми ми добавляємо rolify до файлу Gemfile і запускаємо команду «bundle install». Після цього генеруємо файли для роботи rolify. Генерація створить міграцію, яка в свою очередь буде добавляти нову таблицю в базу даних під назвою roles. У ній будуть зберігатися групи користувачів (адміністратор, вчитель, учень). А також в rails аплікації буде доступні методі

Таблиця 3.2 – Доступні методи в бібліотеці Rolify

Методи	опис
has_role?	Метод викликається на об’єкті user. За допомогою цього можна перевірити чи належить даний користувач до однієї з груп. У нашому випадку є три варіанти використання цього методу: has_role? :admin

Кінець таблиці 3.2

add_role	Метод викликається на об’єкті user. За допомогою нього можна змінити групу для користувача. Приклад використання методу: add_role :teacher
remove_role	Метод викликається на об’єкті user. За допомогою нього можна видалити ролі для користувача. Приклад використання методу: add_role :teacher. Метод видалить роль, і установить роль по замовчуванню (student)

Ще одною корисною бібліотекою є carrierwave, за допомогою якої реалізовано можливість публікувати фото матеріал. Цю задачу і буде вирішувати carrierwave. Він допомагає зберігати зображення на сервері і назву файлу в базі. Його установка така ж як і для інших бібліотек.

В даному проекті у папці app/uplouders зберігатимуться файли, в яких будуть прописуватися конфігурації для збереження картинок. Для кожної моделі, для якої повинно прикріплювати фотографії, буде окремий файли, який у свою чергу відповідатиме за конфігурації саме для неї. Отож, згенерувався файл у папці uploaders, ми можемо відкрити його і побачити, що в ньому є загальний код, який оголошує якого типу картинки можуть приходити від клієнта, розмір до якого вони будуть конвертуватися, коли зберігатимуться на диску сервера, а також різного типу валідації і тд.

Після створення файлу конфігурації ми можемо з’єднати саму моделі із конфігураційним файлом. Це робитися за допомогою одного рядку коду. В моделі в яка містити картинку, тобто поле типу string, добавляємо

mount_uploader :назва_поле_в_моделі, НазваКласуКонфігурацій. Тепер наша модель зможе в собі зберігати назви файлів.

Сама бібліотека також містить важливу конфігурацію, де ми будемо зберігати файли, тобто в якій директорії, а саме в функції default_url ми вписуємо шлях, куди будуть зберігатися файли. По замовчуванню, це шлях буде містити назву самої моделі, первинний шлях, а також назву самого файлу. Отож коли ми зберегли файли, в папці public буде творено шлях, наприклад: user/1/назва_файлу. В даному шляху присутнє назва моделі з маленької букви, а також первинний ключ, для якого запису в базі цей файл прикріплений.

Одною функцією конфігураційного файлу є можливість збереження файлів на віддаленому сервері (aws, azure, google platform). Тобто, бібліотека дає можливість не зберігати картинки на сервері, а відправляти їх на віддалений сервер. Для реалізації цього, потрібна додаткова бібліотека, яка цим всім буде керувати, а саме fog. За допомогою нього можна відправляти файли будьякого розміру на віддалені сервера. У нього є різні версії, тобто якому ми хочемо працювати з платформою aws нам потрібен fog-aws.

Отож установка його не займе багато часу, після цього в файлі конфігурації ми повинні розкоментувати рядок *storage :fog* і закоментувати *storage :file.* Цим ми дали зрозуміти бібліотеці, що вона повинна використовувати бібліотеку fog для відправки файлів на віддалений сервер. Текст конфігураційного файлу подано в додатку Д.

Після цього потрібно реалізувати конфігурацію самого fog. В даному прикладі ми використовувати віддалені сервери aws. Тому зареєструвавшись на цьому сервісі, ми отримаємо два хеш-ключа. Ці ключи будуть давати нам доступ до aws акаунта. Сервіс, на який будуть відправлятися файли, називаються S3. На ньому повинно бути створено bucket, який буде отримувати файли і зберігати їх в собі. Отримавши ці ключі ми повинні їх вставити в конфігураційний файли fog бібліотеки, так як вона повинна знати куди вона має надсилати файли. В конфігураційному файлі ми повинні вказати назву bucket-а, а ще два секретні

ключі, після цього carrierwave зможе отримувати та зберігати файли не в каталогах самого проекту, а на віддаленому сервер. Причому ієрархія папок буде так сама, що прописана в конфігурації самого carrierwave.

Ще однією корисною бібліотекою є rspec. Вона потрібна для тестування rails аплікації. Хоча в rails додатку по замовчуванню встановлено фреймворк для тестування. Проте спільнота ruby не рекомендує використовувати те тестування, що йдеться з коробки. Воно не надає достатньо інструментів для якісного тестування аплікації.

Отож, установивши цю бібліотеку і запустивши команду *bundle exec rspec:install,* rspec буде встановлено. Бібліотека створить в кореневій папці проекту каталог spec, в ній будуть знаходитись тести на різні частити rails аплікації. А також два конфігураційні файли rspec_helpers.rb та rails_helpers.rb, вони потрібні для того, щоб настроювати саму бібліотеку rspec і безпосередньо її поведінку в rails аплікації.

Сама бібліотека може тестувати будьякий ruby код. В даному випадку rspec буде тестувати:

- модель (User, Post, HomeTask)
- контролери (PostsController, UsersController)
- mailers
- helpers
- workers
- uploads

Усі вище перераховані бібліотеки тим чи іншим чином спрощують розробку самої rails аплікації. Що дає зможу біль детально акцентувати свою увагу на саме на структурі та архітектурі проекту.

3.2 Розробка контролерів, моделей та вигляду сайту

3.2.1 Імплементація контролерів

Action Controller - це rails бібліотека відповідає за таку частину аплікації як контролери, це по суті буква С в абревіатурі MVC. Після того, як запит потрапить на роутер, він визначить, який саме контролер потрібно використати для обробки надісланого запиту. Контролеру відповідальному за обробку запиту від клієнта (браузера) і генерацію відповіді (response). Action Controller робить велику частину роботи і використовує уже готовий код, щоб зробити обробку запитів і відповідей був якомога простішим.

Для rails додатків, які реалізовані на RESTful, контролер отримує request, зчитує або зберігає дані в моделі і використовує їх для створення response у вигляді HTML. Проте якщо потрібно реалізувати іншу поведінку для контролерів, то всі існуючі поведінки уже є в «коробці».

Тому контролер можна розуміти як посередник між моделями і виглядом. Він робить дані моделі доступними в файлах вигляду, так що він може виводити ці дані для користувача і приймати дані від користувача для подальшого збереження в моделі [13].

Іменування контролерів в Rails повинно закінчувати множиною, хоча строго це не потрібно (наприклад, ApplicationController). Наприклад, PostsController більш кращий ніж PostController, HomeTasksController більш кращий, ніж HomeTaskController.

Якщо дотримуватися угоди найменування контролерів, Rails дозволяє використовувати генератори маршрутів за замовчуванням (наприклад, resources і т.п.) без необхідності визначати кожен шлях для кожного з екшнів у контролері.

Контролер - це клас Ruby, який успадковується від ApplicationController і містить методи, у даному випадку їх називають «екшнами». Коли додаток отримує запит, маршрутизація rails аплікації визначає, який саме контролер та

екшни потрібно запустити, потім Rails створює екземпляр цього контролера і запускає метод з ім'ям, як у екшна.

```
class PostsController < ApplicationController
    def index
    end
end
```

Якщо наприклад, користувач перейде по URL шляху /posts в rails додатку, Rails створить екземпляр PostsController і викличе метод index. У вище наведеному коді, метод буде працювати, так як Rails за замовчуванням відрендерить вигляд index.html.erb, проте в самому екшні можна це змінити. При створенні нового поста, метод index створює змінну @posts, після чого він ця змінна буде доступна у відповідному файлі вигляду:

```
def index
    @posts = Post.all
end
```

ApplicationController унаслідуваний від ActionController::Base, який визначаєкорисний функціонал для цього класу.

Тільки public методи можуть бути викликані маршрутизатором як екшени. Хорошою практикою є здійснення обмеження доступності методів (за допомогою модифікаторів доступу private або protected), які не призначені бути екшенами, до них можна віднести допоміжні методи і фільтри.

Важливою частиною ActionController є можливість обробляти дані, які прийшли від клієнту, вони доступні в методі params. Існує два типи параметрів, доступних у rails додатках. Перший - це параметри, послані в самому URL, звані параметрами в рядку запиту. Параметри в рядку запиту завжди слід перечислювати після знаку "?" в URL.

Другий тип параметрів зазвичай згадується як дані POST. Ця інформація як правило приходить з HTML форми, що заповнюється користувачем. Так як можуть бути послані тільки як body-частина HTTP-запиту методу POST, Rails не робить будь-яких відмінностей між строковими параметрами і параметрами POST, і всі вони є доступними в params в будь-якому контролері: в даному випадку для реалізації проекту нам потрібні такі контролери. Вони зображені на рисунку 3.1.

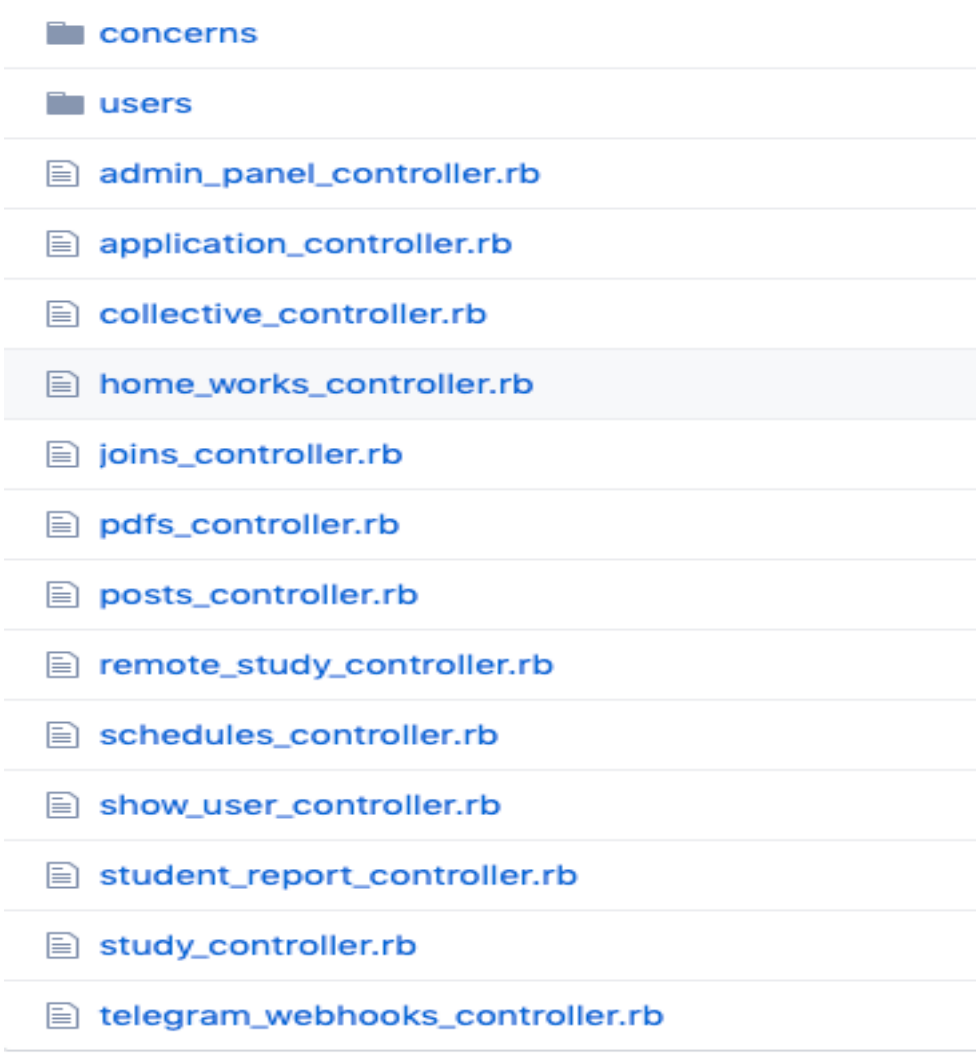

Рисунок 3.1 – реалізовані контролери

Кожен з цих контролерів працює з окремою сутністю, тобто контролер posts_controller.rb буде працювати із сутністю Post. Як видно існує контролер application_controller.rb, він є спільним для кожного з існуючих контролерів. Він буде містити код який буде спільним для кожного з існуючих контролерів.

Реалізація контроллера posts_controller.rb подана в Додатку Б.

Як бачимо кожен публічний метод цього класу буде мітитись як екшн цього контролера, по стандарту вони мають назви create, new, index, show, edit, destroy. Також контролер містить один приватний метод, він потрібний для того, щоб контролер фільтрував параметри, які приходять йому з клієнту, це потрібно для більшої безпеки аплікації. У більшість контролерів похожі у своєму вигляді, так як їхня стандартна структура є універсальною.

Проте один контролер є специфічним, а саме telegram_webhooks_controller.rb і існує для реалізації телеграм бота. Кожин екшн цього контролера є окремою командою для телеграм бота. Отож коли користувач в телеграм напише в чаті бота команду /start, бот відправить запит на rails аплікацію, маршрутизатор переправить на цей контролер а саме на екшн start!.

```
class TelegramWebhooksController < Telegram::Bot::UpdatesController
        use_session!

        def start!(*)
                respond_with :message, text: 'Привіт!'
        end
end
```

Важливою деталлю є те, що команда start в телеграм боті буде співставленою зі екшном з тією ж самою назвою, проте зі знаком оклику.

3.2.2 Моделі

Active Record - це M в MVC - модель – бізнес-логіки аплікації та робота з даними. ActiveRecord використовує створені та використання об'єкти, що потребують безпосереднього зберігання в базі даних. Сама по собі реалізація паттерна ActiveRecord описують і реалізовують ORM (Object Relational Mapping).

ActiveRecord є по своїй суті просто об'єктами, що постійно перебувають у активній роботі, тобто воно постійно витягуються, додаються нові або ж видаляються. ActiveRecord забезпечує логіку, що доступна у вигляді даних, як частини об'єкта показують користувачеві, як витягувати та записувати у базу даних.

ОРВ (об'єктивно-реляційне відображення), зазвичай використовуються як абревіатура ORM, це технологія, з'єднуючи різні об'єкти з таблицями в базі даних, тобто це означає, що об’єкти зможуть зберігатися в базі, а властивостями об’єкта будуть колонками в таблиці бази даних. З підтримкою ORM, властивості і взаємозв'язків цих об'єктів можна використовувати без прямої її реалізації, збереження даних і витягування з бази даних без прямого написання SQL коду, і при цьому зменшується суммарний код, що допомагає не нагромаджувати код [14].

Для розуміння бібліотеки ActiveRecord, потрібно розуміти базові знання і поняття реляційних систем управління базою даних (RDBMS) а також мову створення запитів SQL.

ActiveRecord пропонує нам декілька механізмів, які є важливими у розробці rails додатку:

- Представлення моделі та її даних;
- Представлення взаємозв'язку між різними моделями;
- Представлення ієрархічного успадкування за допомогою зв’язаний між собою моделей;
- Валідації даних для моделей, перед тим як вони будуть записані в базу даних;
- Виконання операцій із даними моделей як з об’єктами.

За замовчуванням ActiveRecord має деякі умови щодо іменування моделей та таблиць в базі даних, щоб пам'ятати, як потрібно створити зв’язок між моделями та таблицями в базі даних. Rails повинні бачити що існує множинна назва для іменні таблиці, щоб знайти відповідну сутність в моделях. Так, для моделі Book повинна бути створена таблиця в базі даних під назвою books.

Rails, зокрема ActiveRecord здатні формувати множинне (і єдине) число як для правильних, а також для неправильних форм іменування. При використанні іменування для моделі, створеної з двох і більш слів, імена класу моделей повинні зберігати узгодження для мови рубі Ruby, тобто використовувати форму іменування CamelCase. Тобто назва таблиці для такої моделі буде розділювати нижньою лінією.

Таблиця 3.2 – Найменування класів моделей і таблиць в базі даних

Модель	Таблиця в базі даних
Post	posts
LineItem	line_items
Mouse	mice

Active Record використовує узгодження іменування також і для стовбців таблиці в базі даних, в залежності від призначення цих самих стовбців.
Зовнішній ключ - це поле повинно обов'язкове мати назву з іменем моделі на запис якої веде цей ключ (singularized_table_name_id - тобто, post_id, book_id). ActiveRecord створюють належний зв’язок між вашими моделями та таблицями в базі даних.

Первинний ключ - по замовчуванням Active Record для кожної таблиці використовує назву id і тип bigint для ключа. Коли буде створюватись міграції для моделі, це поле автоматично буде створюватися, для таблиці зв’язаної з цією моделлю, про те є ще конфігурацііі, які дозволяють змінити тип поля з Integer на String. Є ще кілька колонок для таблиці, які ActiveRecord пропонує для імплементування, вони є стандартними і ідуть з «коробки»:

- created_at - автоматично будуть записуватися поточні дані про час при створенні запису.
- updated_at - автоматично будуть записуватися поточні дані і під час кожного раз, коли буде створено або оновлено запис.
- lock_version - додає можливість блокування для моделям.
- type – указує на то, що модель використовується як STI.
- (type_name) _type – зберігатиме тип поліморфного зв’язку.
- (Таблиця_назви) _count - використовується для кешування кількох прив’язаних до запису об’єктів.

Важливою складовою ActiveRecord є можливість створювати міграції. Міграції – це механізм, за допомогою якого ми можемо без написання SQL коду міняти структуру бази даних. Тобто ми можемо створити, видалити оновите таблиці в базі даних, при цьому не написавши жодного рядка SQL коду.

Причому можливість написати SQL код теж присутня. Міграцію можна створити двома способами через командний рядок. При генерації моделі буде створюватися міграція (rails g model User). Якщо створити міграції генеруючи моделі через консоль, то в назва міграції буде міститись дата створення міграція, а також обов’язкове слова create (тільки якщо це буде створення міграції через створення моделі), а також назва моделі в множині.

Самий файл міграції буде створюватися в db/migrate. Коли створено саму міграцію ми можемо модифікувати, тобто добавити поля які буде містити таблиця, а як заодно і властивості моделі класу. Коли ми створили потрібні нам поля, щоб змінить структуру бази даних цією міграцією її треба запустити командою *rake db:migrate*. ActiveRecord переведе її в SQL код та відправить її до бази даних, де вона добавить нову табличку або ж змінить щось в уже існуючій структурі бази даних.

ActiveRecord реалізовує зв’язки між двома класами моделі. Вони існують для того щоб зробити код простішим тим самим уникаючи нагромадження коду. Наприклад, розглянемо простий випадок який зустрічається в проекті, в якому

існують дві моделі користувач(User) та новини(Post). Кожен автор може мати багато постів. Це зв’язок називаються один до багатьох. Класи моделей при цьому будуть виглядати ось так:

```
class Author < ApplicationRecord
    has_many :posts
end

class Post < ApplicationRecord
    belongs_to :author
end
```

При такому зв’язку модель посту(Post) буде містити вторинний ключ, який буде називатись author_id. Так як логічніше зберігати ключі в тих моделях, де вони приналежать одній сторонній моделі. Список всіх існуючих моделей зображено на рисунку:

Таблиця 3.3 – Існуючі моделі в школі

Модель	Існуючі зв’язки
HomeTask	One-to-many – цей зв’язок на реалізовано два рази. Перший раз це зв'язок з таблицею користувача, тобто домашнє завдання має свого автора Другий раз це зв'язок використовується з моделлю class-room, так як домашнє завдання стосується тільки одного конкретного класу
RemoteTask	Реалізовую тіж самі зв’язки що і модель домашніх завдань. Проте ще і зв’язок із моделлю Mark

Кінець таблиці 3.3

User	Ця сутність зв’язана з таблицею class_room, так як користувач може існувати тільки в одному класі
Report	Відгук мають зв’язок на сутність User два рази, перша на автора який створив відгук, а другий а користувача на якого це відгук був створений.
Role	Ця сутність має зв'язок з сутністю User, так як кожен користувач має зв'язок тільки з однією роллю. Ця таблиця по замовчуванню буде містити всього три ролі.
Schedule	Модель зв’язана з сутністю class_room. Так як розклад є спільним для кожного окремого класу.

3.2.3 Вигляд. HTML

Вигляд в rails аплікації обробляється за допомогою бібліотеки ActiveView. За допомогою неї можна з’єднувати ruby код з мовою розмітки тексту HTML. Якщо генерувати контролер через rake завдання разом із ним згенерується і файли для кожного з екшнів контролера. Тобто, в каталозі view буде створений ще один каталог, який буде називатись як і контролер. В ній будуть містись файли вигляду, наприклад, там буде існувати файл index.html.erb, який автоматично rails аплікацією буде автоматично рендеритись, якщо буде задіяний екшн index в контролері.

Також у файлі вигляду будуть доступні всі змінні екземпляру (@posts). Тобто, якщо в контролері змінна @posts буде містити всі записи з бази, то у файлі

вигляду ми зможемо за допомогою циклу вивести кожен пост з цього тієї змінної [15].

Також важливим аспектом rails аплікації є те, що в файлах вигляду для кожного еншна контролера ми можемо описувати тільки той код, який буде відповідати тільки за конкретний вивід саме масиву, а не цілої сторінки. Так як ActionView реалізовує парадигму Single Page Application, у каталозі layout є вигляд, який буде однаковим для кожної сторінки сайту, тобто для кожного ресурсу. Під час розробки сайту ця парадигма і була основою проекту. В самому файлі layout-а є спецсимвол yield. Замість нього rails аплікація буде замінювати її на html код якогось екшна контролера. Загальний вигляд головного layout-а подано в додатку В.

Також вигляд сайту реалізовується на CSS фрейморку як Bootstrap. На проект він підключений за допомогою CDN. Тобто, самі файли вигляду зберігаються на віддаленому сервері. І під час входу на сайт будуть завантажуються клієнту в браузер. Ця бібліотека дає зможу простими html класами та готовими стилями швидко реалізувати зовнішній вигляд сайту. Головна сторінка сайту зображено на рисунку 3.2.

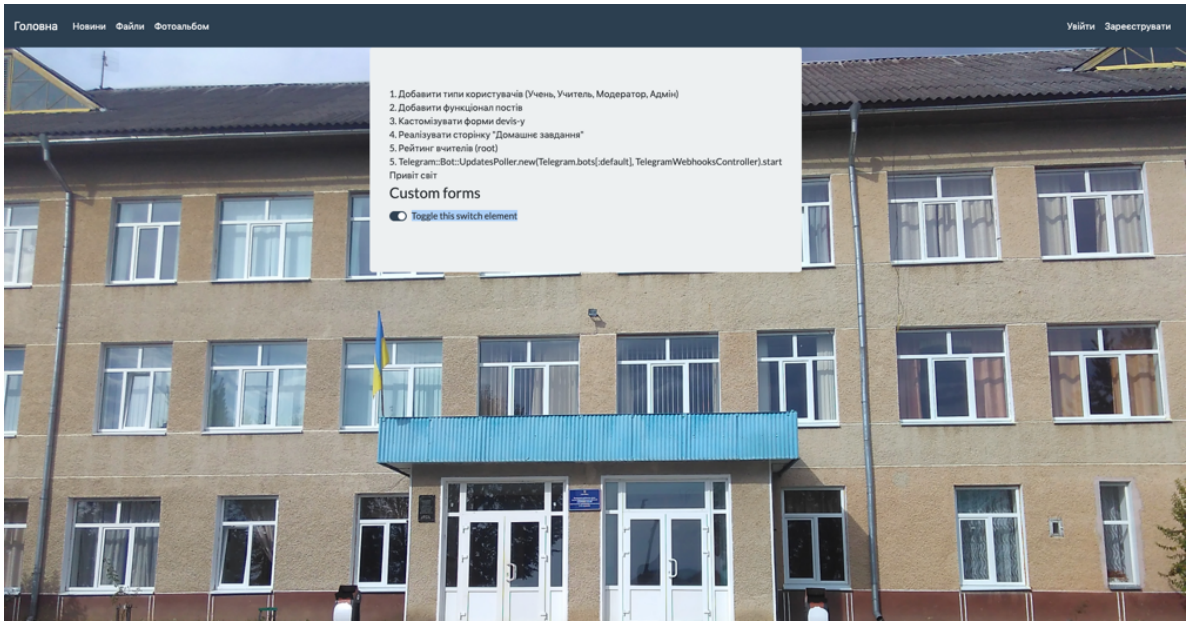

Рисунок 3.2 – Головна сторінка аплікації

З самого верху сторінки знаходиться меню, який реалізоване за допомогою Bootstrap. За допомогою нього можна перейти на сторінку з новина школи на файли, які можна скачати і переглянути онлайн, а також на галерею школи. Там зберігаються фотографії зі шкільних свят, вистав тощо. Зправа знаходяться два посилання на форму входу і реєстрації. Якщо ж увійти в особистий кабінет замість посилань на форми входу і реєстрації з'являється фото користувача, його ім'я та кнопка виходу з облікового запису. А якщо перейти на сторінку новин з правої сторони, з'являється список з посиланнями, через яке користувач може потрапити на сторінку домашнього завдання, дистанційного завдання, розкладу, списку користувачів, які в тому ж класі що і сам користувач, сторінка відгуків про користувача.

Звісно, якщо користувач є адміністратором то, з'являється додаткове меню, в якому ми можемо перейти до адміністраторської панелі, а також посилання на сторінку, де ми можемо переглянути користувачів, які хочуть надіслали заявку на прийняття до якось класу. Зправа розмішується контент, який рендерить контролер. Тобто, це можуть бути html сторінки екшнів контролера. Наприклад: index (index.html.erb), edit (edit.html.erb). Сторінку авторизованого користувача зображено на рисунку 3.3.

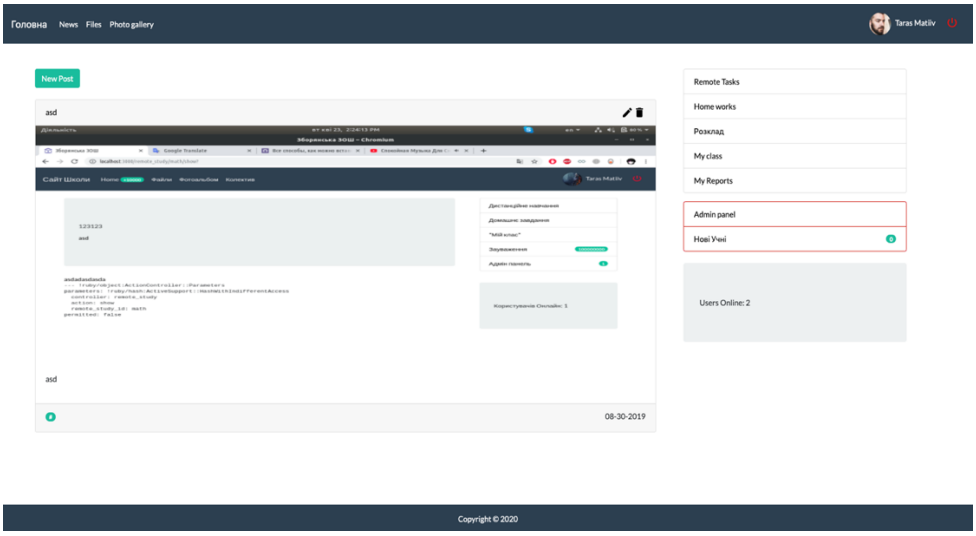

Рисунок 3.3 – UI авторизованого користувача

4 БІЗНЕС ПЛАН ВЕБ-ДОДАТКУ ДЛЯ ШКІЛ

4.1 Загальний огляд

Веб додаток для школи зберігатиме домашнє завдання, яке задається на кінці уроку і яке постить учителем, а також є можливість створювати дистанційні завдання та розклад занять для кожного класу з авто сповіщенням початку уроку.

Цільовою аудиторією цього веб-додатку є люди від 6 до 60 років, які мають пристрої з можливістю підключитись до мережі Інтернет.

Для створення веб аплікації для школи необхідна сума в розмірі від 20 000 до 35 000 грн.

Для розробки необхідно залучити двох працівників (розробник і тестер).

Очікуваний мінімальний місячний дохід додатку становить 2 000 грн.

Чистий прибуток в рік становитиме від 24 000 грн.

4.2 Маркетингові дослідження

Можливості проекту

Веб сайт має можливість зберігати оцінки кожного учня для конкретного предмету або ж для дистанційних завдань.

Також є така функція як сповіщення в телеграм про зміни в контенті сайту. Коли вчитель добавляє нове завдання (дистанційне), учням класу, для якого було створене це завдання, прийде сповіщення про те, що створено нове завдання.

Сайт може будувати графіки успішність учня. Це корисно, коли оцінки учнів починають зменшуватися, це допомагає контролювати успішність самого учня, та не допускати подальшого спаду рівня знань.

Веб додаток також має можливість розробки для сторонніх розробників. За допомогою API і спеціального коду ми можемо отримати успішність учня. Причому код для кожного учня є унікальним.

Дослідження ринку

Для пошуку інформації про шкільні сайти використовувались інформаційні джерела мережі Інтернет, а також створювалось опитування учнів і учителів, що вони хотіли б бачити на сайті.

Оцінка попиту

На даний час попит на такий сайт є великою, так як у зв’язку з появою COVID-19, різко збільшилось потреба саме в інтернет ресурсах, які допомагають перенести процес навчання в інтернет простір.

Конкуренти

- У інтернеті існує багато готових рішень, проте вони всі потребують затрат на їх розгортання;
- Сторонні веб додатки характеризуються популярністю через їх велику потребу;
- Більшість готовий рішень потребують помітних капіталовкладень;
- До явних конкурентів можна віднести: МійКлас, e-school, moodle, Google Class.

Оптимістичний сценарій розвитку

- Багато шкіл у зв’язку з появою пандемію потребуватимуть сайту, за допомогою якого можна буде здійснювати дистанційне навчання;

- Так як веб додаток є повністю безкоштовним, ним може скористатись кожна школа;
- Кількість навчальних закладів, які будуть замовляти сайт, буде невпинно рости;
- Розширення З часом сайт можна буде покращити та добавити більший функціонал, наприклад чат та відео-чат.

Песимістичний сценарій розвитку

- Можливо, протягом великого часу школі відмовляться від дистанційного навчання;
- Більша вигідність існуючих проектів;
- Перехід на платний хостинг, що призведе до помітних затрат;
- Реклама виявиться недостатньо ефективною;
- Погані відгуки про сайт від користувачів;
- Занепад процесу розробки нового функціоналу.

Реалістичний сценарій розвитку

- Школи, в бюджеті яких не закладено витрати на розробку свого власного сайту, зможуть безкоштовно використовувати цей веб-додаток;
- Буде форум, де будуть обговорвати можливий новий функціонал ;
- Реклама в соціальних мережах залучатиме нові школи;
- Реальна собівартість проекту буде включати тільки саму розробку сайту та зарплати працівникам.

Маркетингова підтримка

На початку піару проекту буде виділено 2 000 грн. Так як уже є школи, які готові взяти проект собі як частину навчального процесу.

Проте є можливість просування сайту через різні соціальні мережі, щоб все більше навчальних закладів змогло замовити сайт та його розгортку саме для себе.

Встановлення рівня цін

Ціна самого продукту, тобто веб-сайту по своїй суті низька, так як додаток буде монетизовуватись тільки на рекламі. Самі розробники зможуть ділитись коштами з реклами зі школами та придбати на ті гроші якісь корисні речі.

План реалізації

План збуту веб додатку буде проводитись через залучення все більше шкіл. Це буде відбуватися за допомогою комунікації між школами. На це є велика надія, що інші школи теж зможуть захотіти такий же проект для себе. Є навіть можливість змінити дизайн самого додатку, щоб додаток був унікальний для кожної зі шкіл.

План SEO просування проекту

Найбільшим вагомою стратегією, яка може бути реально дієвою. Це просування через комунікацію між школами. Так як ця комунікація існує, коли вчителі спілкуються між собою або ж обмінюються досвідом. Там вони можуть обговорювати нововведення у шкільні процеси. Зокрема мова може піти про можливість дистанційного навчання через особистий сайт школи.

4.3 Фінансові необхідності

Виробничі потужності

Веб-додаток буде розгорнуто на одному із найпопулярніших платформ для хостингу Heroku. Віддаленим сервером для збереження картинок буде сервіс AWS S3.

Режим роботи сайту на хостингу – цілодобовий.

Потрібно також передбачити співпрацю з хостингами, що пропонують невелику кількість можливостей за меншу ціну. Такими хостингами можуть стати місцеві провайдери, які готові дати сервера для розгортання веб-додатку.

Також потрібно реалізувати бібліотеку для реалізації коду, який зможе автоматично підготувати сервер для розгортання на нього веб аплікації. Для Rails аплікації є бібліотека для автоматизації процесу розгортання.

Затрати проекту

Таблиця 4.1 – Вартість послуг для утримання веб-сайту

Оренда сховищ для зображень:	≈ 350 грн (AWS S3)
Оренда (білого) IP-адресу	≈ 0 грн (pp.ua)
Ціна послуг хостингу	≈ 250 грн (Heroku)
Сума:	**≈ 600 - 700 грн**

Працівники та оплата їх праці

Після розміщення веб-додатку на Heroku, не потрібно залучати програмістів так як є готовий функціонал, який виконає розгортання аплікації самостійно. Коли набереться певна кількість користувачів, потрібно все ж таки найняти розробника для підтримки веб-застосунку.

Для економії коштів можна залучити студента, компетентного в підтримці сайтів. Пізніше, коли все більше шкіл замовить сайт і він приноситиме більший прибуток, можна залучити кілька програмістів, які будуть розробляти новий функціонал і підтримувати уже старий код, а також продуктивно тестувати додаток на наявність усіх видів помилок.

З подальшою розширюваність веб-додатку потрібно буде більше програмістів які зможуть реалізувати функціонал який поможе в майбутньому повністю перенести школу в веб простір.

4.4 Правові аспекти

Організаційно-правова форма проекту

Проект буде рахуватись як приватне підприємство.

Організаційний аспект

Ведення фінансів буде виконуватись директором або ж бухгалтерією школи.

4.5 Бюджет

Джерела фінансування

Джерелом фінансування на початку зародження проекту буде проводитись через пожертвування та через кредит у банку, який буде записаний на приватне підприємство. З часом коли сайт буде уже розгорнуто та запущено, на ньому буде реклама, гроші з якої будуть іти на розроблю нового функціонала, частина на благо устрій школи.

Загальні витрати

Для створення веб-аплікації для сайту школи необхідно: 2-3 розробника, біля 3 місяців розробки та сума в розмірі від 35 000 до 45 000 грн.

Витрати на підтримку сайту становлять від 3 000.

Очікуваний мінімальний місячний дохід аплікації становить 4 000 грн.

Чистий прибуток в рік становитиме від 48 000 грн.

Термін окупності: 12~14 місяців.

Оцінка ризиків

- Різке падіння попиту;
- Завершення карантину;
- Стрибки рівня інфляції;
- Нестабільна робота сайту;
- Недостатнє фінансування;
- Скорочення штату працівників;
- Перехід хостингу на платну основу.

ВИСНОВКИ

У процесі розробки веб аплікації було використано такі технології: Ruby, Ruby on Rails, Bootstrap, Redis, Memcached, Heroku. Перелічені технології на даний час є трендовими. Основними перевагами їх є швидкість та якість розробки. За допомогою їх реалізовано веб додаток для школи, який розгорнуто на популярному та безплатному хостингу Heroku.

Актуальність даного проекту є великою, оскільки існує необхідність дистанційного навчання у освітніх закладах. Такого переходу потребує вся освітня система, яка на даний час базується тільки на реальній взаємодії вчителя та учня. Виходячи з реалій, шкільна освіта була не в змозі швидко адаптуватися до можливості навчання з дому, оскільки для цього в школах, ліцеях та гімназіях не було ні програмного забезпечення, ні плану проведення такого навчання. В таких випадках проект дає можливість працювати вчителю з учнем дистанційно.

На даний час проекту знаходиться у використанні однієї школи, яка використовує його для розповсюдження розкладу онлайн занять, створення дистанційних завдання та публікації оцінок для кожного учня. Розробка проекту перебуває у постійному процесі. Окрім цього, йде активна розробка нового функціоналу і удосконалення існуючого.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

REFERENCES

1. ActiveController як частина Rails фреймворка. URL: https://apidock.com/rails/v4.2.7/ActionController/Base (дата звернення: 20.02.2020)
2. Веб сервер Puma. URL: https://github.com/puma/puma (дата звернення: 20.02.2020)
3. Сервіс Memcached. URL: https://uk.wikipedia.org/wiki/Memcached (дата звернення: 20.02.2020)
4. Redis як база даних. URL: https://redis.io (дата звернення: 20.02.2020)
5. Сайт з корисними курсами для само розвитку. URL: https://en.wikipedia.org/wiki/Coursera (дата звернення: 20.02.2020)
6. Реляційна база даних. URL: https://uk.wikipedia.org/wiki/PostgreSQL (дата звернення: 20.02.2020)
7. Українська онлайн система тестування учнів. URL: https://uk.wikipedia.org/wiki/Мійклас (дата звернення: 20.02.2020)
8. Документація про фреймворк Ruby on Rails. URL: https://web-creator.ru/articles/about_ruby_on_rails (дата звернення: 20.02.2020)
9. Загальний опис фреймворка Ruby on Rails. URL: https://uk.wikipedia.org/wiki/Ruby_on_Rails (дата звернення: 20.02.2020)
10. Обробник завдань. URL: https://sidekiq.org (дата звернення: 20.02.2020)
11. Гнучка система для розробки власних курсів. URL: https://uk.wikipedia.org/wiki/Moodle (дата звернення: 20.02.2020)

12. Освіта в нашому житті. URL:https://www.0512.com.ua/list/56573 (дата звернення: 20.02.2020)
13. E-school. URL: https://www.kristti.com.ua/v-ukrayini-vprovadzhuyetsya-elektronnyj-osvitnij-proekt-e-schools/ (дата звернення: 20.02.2020)
14. Туторіал по ActionController. URL: https://ruby-on-rails.programmingpedia.net/en/tutorial/2838/actioncontroller (дата звернення: 20.02.2020)
15. ActionView. URL: http://rusrails.ru/action-view-overview (дата звернення: 20.02.2020)
16. Федорук П. І., Дутчак М.С. Побудова бази знань адаптивних систем дистанційного навчання на основі фреймової та продукційної моделей представлення знань. Управляючі системи і машини (УСіМ). Київ, 2012. №5. С.3-10.
17. Пікуляк М. В. Інформаційна технологія побудови моделі адаптивного тестування в системах дистанційного навчання. Вісник Хмельницького національного університету. Серія: Технічні науки. 2018. № 4. С. 153–158.

ДОДАТОК А

Файл Gemfile

```
source 'https://rubygems.org'

git_source(:github) do |repo_name|
  repo_name = "#{repo_name}/#{repo_name}" unless
repo_name.include?('/')
  "https://github.com/#{repo_name}.git  "
end

gem 'rspec-rails'

gem 'i18n'
gem 'rubycritic'

gem 'letter_opener'

gem 'telegram-bot'

gem 'redis'
gem 'sidekiq'
gem 'sidekiq-cron'

gem 'carrierwave'
gem 'mini_magick'
gem 'fog-aws'

gem 'will_paginate', '~> 3.1', '>= 3.1.6'
gem 'will_paginate-bootstrap'

gem 'cancan'
gem 'devise'
gem 'dotenv'
gem 'rolify'

gem 'pg'

gem 'jquery-rails'

gem 'bootstrap', '~> 4.1.3'
gem 'twitter-bootstrap-rails'
# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '~> 5.1.6'
# Use Postgres as the database for Active Record
# gem 'sqlite3'
# Use Puma as the app server
gem 'puma', '~> 3.7'
```

```
# Use SCSS for stylesheets
gem 'sass-rails', '~> 5.0'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'
# See https://github.com/rails/execjs#readme for more supported
runtimes
# gem 'therubyracer', platforms: :ruby

# Use CoffeeScript for .coffee assets and views
gem 'coffee-rails', '~> 4.2'
# Turbolinks makes navigating your web application faster. Read
more: https://github.com/turbolinks/turbolinks
gem 'turbolinks', '~> 5'
gem 'jquery-turbolinks'

# Build JSON APIs with ease. Read more:
https://github.com/rails/jbuilder
gem 'jbuilder', '~> 2.5'
# Use Redis adapter to run Action Cable in production
# gem 'redis', '~> 4.0'
# Use ActiveModel has_secure_password
# gem 'bcrypt', '~> 3.1.7'
#gem 'webpacker'
#gem 'yarn'
# gem 'capistrano-rails', group: :development

group :development, :test do
  # Call 'byebug' anywhere in the code to stop execution and get a
debugger console
  # Adds support for Capybara system testing and selenium driver
  gem 'capybara', '~> 2.13'
  gem 'selenium-webdriver'
end
group :development do
  # Access an IRB console on exception pages or by using <%=
console %> anywhere in the code.
  gem 'web-console', '>= 3.3.0'
end

# Windows does not include zoneinfo files, so bundle the tzinfo-
data gem
gem 'tzinfo-data', platforms: %i[mingw mswin x64_mingw jruby]
```

ДОДАТОК Б

Текст контролера Posts

```
class PostsController < ApplicationController
  def create
    @post = Post.create(post_params)
    redirect_to posts_path, notice: 'Новина була успішно створена'
if @post.save
  end
  def new
    @post = Post.new
  end
  def update
    @post = Post.find(params[:id])
    if @post.update_attributes(post_params)
      redirect_to @post
    else
    end
  end
  def edit
    @post = Post.find(params[:id])
  end
  def destroy
    @post = Post.find(params[:id])
    @post.destroy
    redirect_to posts_path, notice: 'Новина була успішно видалена'
  end
  def index
    @shows = Post.order(created_at: :desc).paginate(page:
params[:page], per_page: 15)
  end
  def show
    @shows = Post.find(params[:id])
  end
  private
  def post_params
    params.require(:post).permit(:title, :body, :who, :logo)
  end
end
```

ДОДАТОК В

Layout веб застосунку

```
<!DOCTYPE html>
<html>
<head>
  <title>Зборянська ЗОШ</title>
  <%= csrf_meta_tags %>
  <%= javascript_include_tag 'application', media: 'all', 'data-
turbolinks-track' => 'reload'%>
  <%= stylesheet_link_tag 'bootswatch', media: 'all', 'data-
turbolinks-track' => 'reload' %>
</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-primary"
style="z-index:999;">
  <div id="myModal" class="modal" >
    <div class="modal-content border-success card" style="max-
width: 400px;">
      <div class="close text-right material-icons">cancel</div>
      <p style="max-width: 400px;">
        <div class="card text-white bg-danger mb-3">Заповнюйте
форму тільки, якщо ви ЯВЛЯЄТЕСЬ учнем школи</div>
        <%= form_for Join.new do |f| %>
          <%= f.label 'Ваш Клас' %>
          <%= f.text_field :class_room, class: 'form-
control'%><br>

          <%= f.submit_tag "Submit", class: "btn btn-primary" %>
        <% end %>
      </p>
    </div>
  </div>
  <a class="navbar-brand" href="<%= root_path %>">Сайт Школи</a>
  <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarColor01" aria-
controls="navbarColor01" aria-expanded="false" aria-label="Toggle
navigation">
    <span class="navbar-toggler-icon"></span>
  </button>

  <div class="collapse navbar-collapse" id="navbarColor01">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item">
        <a class="nav-link" href="<%= posts_path %>"><%= t('news')
%>  <span class="badge badge-primary badge-
pill">+10000</span><span class="sr-only">(current)</span></a>
```

```
      </li>
      <li class="nav-item">
        <a class="nav-link" href="<%= pdfs_path %>"><%= t('file')
%></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#" onclick="an()"><%= t('pg')
%></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="<%= collective_index_path
%>"><%= t('colective') %></a>
      </li>
    </ul>
    <div id="root " >
      <ul class="navbar-nav pull-xs-right background">
        <% if user_signed_in? %>
          <li class="nav-item">
            <%= link_to image_tag(current_user.avatar.url, class:
'bl', style: 'background-color: #18BC9C;'),
edit_user_registration_path %>
          </li>
          <li class="nav-item">
            <%= link_to current_user.name,
edit_user_registration_path, class: "nav-link" %>
          </li>
          <li class="nav-item">
            <%= button_to "power_settings_new",
destroy_user_session_path, :method => :delete, class: "btn
material-icons btn-link", style: "color: red" %>
          </li>
        <% else %>
          <li class="nav-item">
            <%= link_to t('login'), new_user_session_path, class:
"nav-link", id: 'login', :@click => "showLoginModal=true" %>

          </li>
          <li class="nav-item">
            <%= link_to t('reg'), new_user_registration_path,
class: "nav-link" %>
          </li>
        <% end %>
      </ul>
    </div>
  </div>
</nav>
<div id='dv' class="d-none" style=""><%= notice %></div>

<div class="row container-fluid bg" style="width: 100%">
<!--  <div class="bg">-->
  <div class="<%= user_signed_in? ? "col-lg-8 mr-4 ml-5 mt-5 " :
'col-lg-11 mr-4 ml-5 mt-5' %>">
```

```
    <%= yield %>
  </div>
  <div class="<%= user_signed_in? ? "col-lg-3 ml-0 mt-5" : 'text-
hide col-lg-0 ml-0 mt-0 d-none' %>">
    <% if user_signed_in? and check_student %>
    <ul class="list-group" >
      <li class="list-group-item d-flex justify-content-between
align-items-center" >
          <%= link_to t("join_class"), '#', class: 'pulse', style:
"color: green;", id: "myBtn" %>
      </li>
    </ul><br>
    <% end %>
    <ul class="list-group">
      <li class="list-group-item d-flex justify-content-between
align-items-center">
        <%= link_to t("remote_study"), remote_study_index_path %>
        <span class="badge badge-primary badge-pill"></span>
      </li>
      <li class="list-group-item d-flex justify-content-between
align-items-center">
        <%= link_to t('dz'), home_works_path %>
        <span class="badge badge-primary badge-pill"></span>
      </li>
      <li class="list-group-item d-flex justify-content-between
align-items-center">
        <%= link_to "Розклад", schedules_path %>
        <span class="badge badge-primary badge-pill"></span>
      </li>
      <li class="list-group-item d-flex justify-content-between
align-items-center">
        <%= link_to t('my_class'), my_class_path %>
        <span class="badge badge-primary badge-pill"></span>
      </li>
      <li class="list-group-item d-flex justify-content-between
align-items-center">
        <%= link_to t("report"), student_report_index_path %>
        <span class="badge badge-primary badge-
pill">100000000</span>
      </li>
    </ul>

    <p></p>
    <ul class="list-group">
      <li class="list-group-item d-flex justify-content-between
align-items-center" style="border-color: #CC0000;" >
        <%= link_to_if (user_signed_in? and current_user.has_role?
:admin), t("admin_panel"), show_user_path %>
      </li>
      <li class="list-group-item d-flex justify-content-between
align-items-center" style="border-color: #CC0000;" >
```

```
        <%= link_to_if (user_signed_in? and current_user.has_role?
:admin), 'Нові Учні', joins_path %>
        <span class="badge badge-primary badge-pill"><%=
number_willing_to_join %></span>
      </li>
    </ul>
    </br>
    <div class="jumbotron">
      <%= t('user_online') + ": #{User.online_users.count}" %>
    </div>

  </div>
<!--  </div>-->
</div><br>
<footer class="page-footer mt-5" style="height: 50px;">
  <nav class="navbar navbar-expand-lg navbar-dark bg-primary"
style="z-index:999;width: 100%">
    <div style="color: white" class="center-block">Copyright © <%=
Date.today.year %></div>
  </nav>
</footer>
<script>
    var modal = document.getElementById("myModal");

    // Get the button that opens the modal
    var btn = document.getElementById("myBtn");

    // Get the <span> element that closes the modal
    var span = document.getElementsByClassName("close")[0];

    // When the user clicks on the button, open the modal
    btn.onclick = function() {
        modal.style.display = "block";
    }

    // When the user clicks on <span> (x), close the modal
    span.onclick = function() {
        modal.style.display = "none";
    }

    // When the user clicks anywhere outside of the modal, close
it
    window.onclick = function(event) {
        if (event.target == modal) {
            modal.style.display = "none";
        }
    }

</script>

</body>
</html>
```

ДОДАТОК Г

Модель User

```
class User < ApplicationRecord
  has_many :schedules
  has_many :reports
  has_one :user_role
  validates :name, :email, presence: true

  mount_uploader :avatar, AvatarUploader
  rolify

  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable, :trackable and
:omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :validatable, :confirmable,
:trackable
  scope :online_users, -> { pluck(:id).delete_if { |user_id|
!User.find(user_id).online? } }
  def online?
    if current_sign_in_at.present?
      last_sign_out_at.present? ? current_sign_in_at >
last_sign_out_at : true
    else
      false
    end
  end
end
```

ДОДАТОК Д

Файл для загрузки аватарок на сервер

```
class AvatarUploader < CarrierWave::Uploader::Base
  include CarrierWave::MiniMagick

  # Choose what kind of storage to use for this uploader:
 # storage :file
  storage :fog

  # Override the directory where uploaded files will be stored.
  # This is a sensible default for uploaders that are meant to be
mounted:
  def store_dir
    'public/my/upload/directory'
  end

  # Provide a default URL as a default if there hasn't been a file
uploaded:
  def default_url(*args)
     # For Rails 3.1+ asset pipeline compatibility:
    "fallback/" + [version_name, "avatar.png"].compact.join('_')
   end

  # Process files as they are uploaded:
  # process scale: [200, 300]
  #
  # def scale(width, height)
  #   # do something
  # end

  # Create different versions of your uploaded files:
  # version :thumb do
```

```
  #   process resize_to_fit: [50, 50]
  # end

  # Add a white list of extensions which are allowed to be
uploaded.
  # For images you might use something like this:
   def extension_whitelist
     %w(jpg jpeg gif png)
   end

  # Override the filename of the uploaded files:
  # Avoid using model.id or version_name here, see
uploader/store.rb for details.
  # def filename
  #   "something.jpg" if original_filename
  # end
end
```