МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)
травня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Інформаційна технологія керування персоналом ІТ-індустрії»
здобувача групи ІН-м.21н Шелеста Сергія Миколайовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Сергій ШЕЛЕСТ
(підпис)

Керівник,
доцент, кандидат технічних наук,
в.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми «Інформатика»
здобувача групи ІН-м.21н Шелеста Сергія Миколайовича

1. Тема роботи: «Інформаційна технологія керування персоналом ІТ-індустрії»
затверджую наказом по СумДУ від *№ 0517-VI від 13.05.2024 року*
2. Термін здачі здобувачем кваліфікаційної роботи *до 14.05.2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Аналіз об'єкту дослідження. 3) Розробка інформаційного та програмного забезпечення для керування персоналом ІТ-індустрії. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Аналіз об'єкту дослідження*		
3	*Розробка інформаційного та програмного забезпечення для керування персоналом ІТ-індустрії*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 103 стор., 10 рис., 7 таблиць, 1 додаток, 14 цифрових джерела. використаних джерел.

Обґрунтування актуальності теми роботи – Тема "Інформаційна технологія керування персоналом IT-індустрії" надзвичайно актуальна у зв'язку зі стрімким розвитком сучасної технологічної сфери. Зростання конкуренції, постійні зміни та потреба у висококваліфікованих кадрах вимагають ефективного управління персоналом. Інформаційні технології надають інструменти для оптимізації процесів підбору, навчання, мотивації та управління співробітниками, що є критичним для успіху в IT-індустрії.

Об'єкт дослідження — процес керування персоналом IT-індустрії.

Мета роботи — створення інформаційної технології керування персоналом IT-індустрії

Методи дослідження — методи побудови інформаційних систем, методи формування та нормалізації структури баз даних, методи проєктування та імплементації API, методи тестування програмного забезпечення.

Результати — Було проведено детальний аналіз методик та інструментів, необхідних для реєстрації та цифровізації даних, а також для обміну командами та даними між програмами, написаними на різних мовах програмування та платформах. Вивчено специфіку застосування цих технологій у серверній частині програм, для різноманітних категорій користувачів. За результатами дослідження було розроблено алгоритм нового програмного забезпечення та його втілення у формі API. Ця розробка забезпечує ефективне управління та доступ до реєстру персоналу. Реалізація проекту була здійснена за допомогою мови програмування Ruby і фреймворку Ruby on Rails.

ТЕХНОЛОГІЧНИЙ СЕКТОР, УПРАВЛІННЯ КАДРАМИ, АВТОМАТИЗАЦІЯ, ЕФЕКТИВНІСТЬ, ОПТИМІЗАЦІЯ

ЗМІСТ

ВСТУП

Актуальність. Тема "Інформаційна технологія керування персоналом IT-індустрії" надзвичайно актуальна у зв'язку зі стрімким розвитком сучасної технологічної сфери. Зростання конкуренції, постійні зміни та потреба у висококваліфікованих кадрах вимагають ефективного управління персоналом. Інформаційні технології надають інструменти для оптимізації процесів підбору, навчання, мотивації та управління співробітниками, що є критичним для успіху в IT-індустрії.

Об’єкт дослідження –процес керування персоналом IT-індустрії.

Предмет дослідження – моделі, методи та засоби інформаційної технології керування персоналом IT-індустрії

Суперечність, що вирішується у роботі - У роботі вирішується суперечність між необхідністю керування персоналом IT-індустрії та відсутністю підходів, які враховують особливості IT-галузі.

Гіпотеза – Адаптація технологій керування персоналом з урахуванням специфіки IT-індустрії дозволить підвищити ефективність та оперативність відповідних процесів.

Новизна – Запропонований комплекс інформаційного, алгоритмічного та програмного забезпечення для керування персоналом дозволяє надати інформаційну підтримку та супроводження відповідних процесів з урахуванням специфіки IT-індустрії.

Структура роботи - Кваліфікаційна робота магістра складається зі вступу, інформаційний огляду, що завершується постановкою задачі, опису вибору методів реалізації та результатів їх застосування при проєктуванні методів та засобів інформаційної технології, прогрманої реалізації засобів інформаційної технології та їх тестування, висновків, списку використаних джерел та додатків.

Впровадження – впровадження частини запропонованих засобів було впроваджено на сучасному ІТ-підприємстві.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Сучасні тенденції в сфері керування персоналом

Із кожним днем спостерігається постійне зростання попиту на використання електронних реєстрів, що робить вкрай важливим питання їхньої розробки та оптимізації, особливо з використанням сучасних інтернет-технологій. Цей тренд особливо посилився у світлі глобальної пандемії COVID-19 у 2020 році, яка серйозно вплинула на економічний стан багатьох країн, зокрема на малі підприємства та державні установи [1-5]. Деякі з цих організацій зіткнулися з необхідністю тимчасово або навіть остаточно закрити свої двері для відвідувачів, що спричинило потребу в пошуках альтернативних способів ведення документації та обслуговування клієнтів.

Запроваджені карантинні обмеження значно прискорили процеси цифровізації та збільшили популярність онлайн-рішень для документообігу, стимулюючи державні органи та приватні компанії до впровадження інноваційних технологій. Особливу увагу було приділено розробці систем, які дозволяють дистанційно оформляти різні типи документів, вести реєстри людей, обробляти запити громадян та забезпечувати взаємодію між урядовими структурами та населенням без потреби особистих зустрічей. Такі системи стали вирішальним чинником у підтримці роботи багатьох секторів економіки під час пандемії [5, 8, 10].

Відповідно до цих потреб, дана робота зосереджена на розробці API для управління IT-персоналом, що дозволить через веб-додаток або мобільний додаток на будь-якій платформі не тільки спростити, але й значно прискорити процес цифровізації персональних даних, їх обробки та управління. Розроблена система буде характеризуватися [7, 11, 15] не тільки простотою в користуванні, але й високим рівнем доступності для різних категорій користувачів, що зробить її ідеальним рішенням для широкого спектру застосувань, від малого бізнесу до великих корпоративних структур та державних установ.

Ключовою перевагою розроблюваного API є його універсальність та гнучкість, які дозволяють інтегрувати систему з існуючими IT-рішеннями організацій, а також адаптувати її під специфічні потреби та вимоги користувачів. Така інтегрована система надасть користувачам зручні та ефективні інструменти для ведення електронного документообігу, зберігання даних та їх захисту, забезпечуючи при цьому високий рівень безпеки та конфіденційності інформації.

Завершуючи, імплементація цього API може суттєво підвищити ефективність взаємодії між державними установами та громадянами, а також між різними відділами в межах однієї організації. Це зменшить паперовий документообіг, спростить процедури обслуговування клієнтів і в кінцевому результаті призведе до більш швидкого та якісного обслуговування.

1.2 Веб-фреймворк

Фреймворк у сфері програмування — це комплексна архітектурна структура, що надає розробникам систематизовану основу і багатий набір інструментів для створення програмного забезпечення. Така структура дозволяє зосередитися на суттєвих елементах проєкту, ігноруючи рутинну роботу з деталями, які не впливають на загальний результат. Фреймворк спрощує багато процесів у програмуванні, пропонуючи стандартизовані методи роботи, що забезпечують якість та уніформність кінцевих продуктів [10-15].

Концепція абстракції, яка лежить у фундаменті використання фреймворків, є ключовою не лише у світі програмування, але й у багатьох життєвих аспектах. Вона допомагає людям спрощувати складні системи та ефективно управляти різноманітними процесами, не вникаючи у всі технічні деталі.

Звернімося до повсякденного прикладу водіння автомобіля. Водій не потребує глибоких знань у механіці чи фізиці для того, щоб керувати

автомобілем. Головне — зрозуміти основні принципи управління: як керувати кермом, використовувати педалі та спостерігати за приладовою панеллю. Ця спрощеність досягається через абстракцію складних систем автомобіля до базових елементів управління, що дозволяє зосередити увагу на водінні, а не на розумінні внутрішніх процесів [20, 22, 24, 25].

Подібно до керування автомобілем, у програмуванні розробники використовують інтерфейси та абстракції, які дозволяють їм легко взаємодіяти зі складними системами. Наприклад, програмісти можуть використовувати API, щоб взаємодіяти з базою даних, не вникаючи в деталі її внутрішньої реалізації. Це спрощує розробку та дозволяє зосередитися на створенні функціональності, яка є важливою для кінцевого користувача.

Фреймворки в програмуванні відіграють ключову роль, надаючи шаблони та методики, що полегшують впровадження стандартних рішень і практик, забезпечуючи високу продуктивність, надійність та масштабованість продукту. Такі інструменти, як фреймворки, не тільки спрощують роботу розробників, але й відкривають можливість для колаборації та спільного використання коду, що значно прискорює процес розробки та сприяє інноваціям [16-18].

Водночас важливо розуміти різницю між фреймворками та бібліотеками. Бібліотека пропонує набір функцій, які можна викликати за потреби, тоді як фреймворк задає структуру проекту та спосіб реалізації програми, що часто передбачає дотримання певних правил та методологій. Ця структурна вимога з фреймворками може забезпечити більшу консистенцію та якість розробки, але також накладає певні обмеження на творчу свободу розробника.

1.3 Розгляд існуючих рішень

Оглянемо функціональні аспекти трьох лідируючих веб-фреймворків, які вирішують схожі завдання і активно конкурують на ринку.

Django, розроблений на Python, є вільнодоступним і використовує структуру модель-шаблон-перегляд (MTV). Його підтримує некомерційна організація з США – Django Software Foundation.

Laravel – це також безкоштовний веб-фреймворк з відкритим кодом на PHP, заснований на Symfony. Він слідує архітектурі модель-вид-контролер (MVC) і має модульну систему пакування з менеджером залежностей, забезпечує різні методи доступу до реляційних баз даних і включає утиліти для розгортання та обслуговування додатків [28-30].

Ruby on Rails (Rails) написаний на Ruby та ліцензований за MIT. Це серверний програмний продукт для веб-додатків з архітектурою модель-вид-контролер (MVC). Rails спрощує використання веб-стандартів, таких як JSON або XML для передачі даних та HTML, CSS, і JavaScript для інтерактивності з користувачами. Фреймворк також активно використовує відомі шаблони програмування, включно з принципами "convention over configuration" (CoC) і "don't repeat yourself" (DRY).

Основна відмінність цих фреймворків полягає у мовах програмування: Django на Python, Laravel на PHP, і Rails на Ruby. Кожен фреймворк вимагає розуміння відповідної мови для розробки веб-додатків.

Всі три фреймворки мають архітектурні схожості (MVC, а Django також MTV), вони фокусуються на легкості розповсюдження коду і читабельності, забезпечують автоматизований доступ до баз даних і не вимагають ручного написання SQL запитів. Їхні маршрутизаційні системи прості та безпечні, а динамічне відображення веб-сторінок, шаблонні системи, наповнені фільтрами і вбудованими функціями, сприяють гнучкості та інтеграції з сучасними технологіями [29-32].

Давайте детальніше розглянемо кожен із трьох веб-фреймворків, щоб краще зрозуміти їхні унікальні особливості та переваги.

Django (Python)

Особливості:

1. "Batteries included" філософія: Django містить множину готових рішень "з коробки", які спрощують розробку веб-додатків, таких як аутентифікація, сесії, адміністративний інтерфейс, і багато іншого.
2. MTV (Model-Template-View) архітектура: Схожа на MVC, але у Django шаблон відповідає за відображення, а view — за логіку додатку.
3. ORM (Object-Relational Mapping): Django ORM дозволяє легко взаємодіяти з базою даних за допомогою Python-коду.
4. Сильна безпека: Django надає вбудовані інструменти для захисту веб-додатків, такі як захист від CSRF, SQL-ін'єкцій, XSS, і багато іншого.
5. Масштабованість та гнучкість: Підходить для розробки як маленьких, так і великих веб-додатків.

Laravel (PHP)

Особливості:

1. Eloquent ORM: Надзвичайно виразний, зручний у використанні ORM, який дозволяє взаємодіяти з базами даних за допомогою PHP.
2. Blade шаблонізатор: Blade дозволяє легко створювати шаблони даних із простим та інтуїтивним синтаксисом.
3. Міграції баз даних: Функція міграцій дозволяє контролювати версії бази даних та спрощує спільну роботу над проектом.
4. Artisan CLI: Командний інтерфейс Laravel, який дозволяє виконувати багато задач розробки прямо з командного рядка.
5. Laravel Echo, Scout, і Socialite: Пакети для реалізації складного функціоналу, такого як реалізація WebSockets, повноцінний пошук та інтеграція соціальних мереж.

Ruby on Rails (Ruby)

Особливості:

1. Convention over Configuration (CoC): Rails стимулює використання стандартних конвенцій, щоб мінімізувати кількість спеціального коду, який розробник має написати.

2. Don’t Repeat Yourself (DRY): Кожна частина інформації повинна мати єдине, чітке представлення в системі, що допомагає зменшити дублювання коду.
3. Active Record ORM: Провідний компонент Rails, який спрощує створення та використання баз даних за допомогою Ruby.
4. Action Pack: Інтеграція контролерів та видів, яка дозволяє створювати веб-запити та відповіді без необхідності використання сторонніх шаблонів або контролерів.
5. Rich Libraries and Gems: Розширена екосистема бібліотек і додатків (gems), яка охоплює майже всі аспекти веб-додатків.

Кожен із цих фреймворків розроблений так, щоб задовольнити певні потреби розробників і проектів, пропонуючи унікальні інструменти і підходи, що роблять їх ідеальними для певних видів додатків або команд розробників [30-31].

1.4 Переваги та недоліки Ruby on Rails

Ruby on Rails, часто просто званий Rails, є веб-фреймворком написаним на мові програмування Ruby. Це відкрите програмне забезпечення, яке ліцензується згідно з ліцензією MIT. Rails є фреймворком, що підтримує архітектурний патерн модель-вид-контролер (MVC), і призначений для забезпечення стандартної структури для баз даних, веб-служб та веб-сторінок. Фреймворк активно заохочує використання веб-стандартів, таких як JSON або XML для передачі даних, і HTML, CSS, та JavaScript для інтерфейсів користувача [12, 2, 8, 9].

Rails також славиться своєю філософією, що включає такі принципи як "Convention over Configuration" (CoC), який зменшує кількість рішень, які розробники повинні приймати, не втрачаючи гнучкість. Принцип "Don't Repeat Yourself" (DRY) спонукає розробників писати мінімально можливу кількість дублюваного коду. Фреймворк також відомий своєю "магією", яка дозволяє автоматично виконувати багато завдань без зайвого конфігурування. Ці

особливості зробили Rails популярним серед розробників, особливо для швидкої розробки складних веб-додатків.

Розглянемо спочатку переваги даного фреймворку:

1. Швидкість розробки: Однією з основних переваг Rails є швидкість розробки, що забезпечується завдяки принципам CoC (Convention over Configuration) та DRY (Don't Repeat Yourself). Ці принципи допомагають уникнути багатьох зайвих конфігурацій і спрощують написання коду, що робить Rails ідеальним вибором для стартапів і проектів з обмеженими термінами [8].
2. Спільнота та бібліотеки: Rails має велику та активну спільноту, яка створила велику кількість бібліотек (gems), які можна використовувати для додавання функціональності або вирішення специфічних завдань без необхідності писати код з нуля [9, 13, 14].
3. Інтеграція тестування: Rails підтримує тестування "з коробки" із використанням бібліотеки MiniTest, яка дозволяє легко писати та виконувати тести для забезпечення якості коду [15, 19, 20].
4. Ресурс-орієнтований підхід: Rails сприяє створенню RESTful додатків, де кожний ресурс має стандартний набір операцій, що спрощує розуміння та взаємодію з веб-додатками.

Тепер перейдемо до недоліків даного фреймворку:

1. Час відгуку: Через велику кількість абстракцій та автоматизації, Rails може виявитися повільнішим у порівнянні з іншими фреймворками, особливо при обробці великої кількості запитів [4, 5, 9].
2. Використання пам'яті: Rails вимагає більше системних ресурсів, зокрема оперативної пам'яті, що може стати проблемою при розгортанні на об

Як і будь-яка технологія, Ruby on Rails не позбавлений недоліків. Одна з головних проблем — це порівняно низька швидкість обробки. Наприклад, фреймворки та середовища розробки, такі як Laravel або Django, можуть працювати швидше за Ruby on Rails. Прикладом цього може слугувати Twitter,

який виявив, що ефективність Ruby on Rails знизилася після того, як соціальна мережа здобула значну популярність. Хоча Twitter і не відмовився повністю від Ruby on Rails, компанії довелося замінити деякі компоненти обробки запитів і серверної логіки. Також, час завантаження проектів у Rails може бути довшим у порівнянні з такими фреймворками, як Django або Laravel. Однак, незважаючи на ці недоліки, Ruby on Rails було обрано через його стабільність та постійно високі показники використання серед розробників Ruby [19, 21].

1.5 Постановка задачі

Створення веб-додатку для керування персоналом у IT-компанії за допомогою фреймворку Ruby on Rails передбачає розробку системи, яка дозволить ефективно управляти інформацією про співробітників, їхні проекти, завдання та робочі години. Ось детальна постановка задачі для такого проекту:

Цільові функціональності додатку:

1. Реєстрація та Аутентифікація Користувачів:
   - Реалізація системи реєстрації та входу для різних ролей користувачів (адміністратори, менеджери проектів, співробітники).
   - Можливість відновлення пароля через електронну пошту.
2. Профілі Співробітників:
   - Створення, редагування та перегляд профілів співробітників з можливістю додавання фотографії, контактних даних, досвіду роботи та освіти.
   - Відстеження статусу співробітників (активний, відпустка, лікарняний тощо).
3. Управління Проектами:
   - Створення та керування проектами, призначення співробітників на проекти.
   - Можливість відстежувати прогрес проектів, призначення завдань співробітникам та контроль термінів виконання.

4. Відстеження Часу:
    - Інтеграція засобу для обліку часу, який співробітники витрачають на кожен проект або завдання.
    - Генерація звітів про використаний час за проектами, клієнтами або співробітниками.
5. Звітність та Аналітика:
    - Автоматичне створення звітів про продуктивність співробітників, статуси проектів, і використання ресурсів.
    - Дашборди з графіками та аналітикою для кращого розуміння робочих процесів.
6. Налаштування та Інтеграції:
    - Інтеграція з зовнішніми системами для управління ресурсами, такими як HR платформи, CRM системи, та інші бізнес-інструменти.
    - Кастомізація інтерфейсу згідно з корпоративними стандартами та брендбуком компанії.

Технічна Реалізація:

1. Моделі Бази Даних: Організація баз даних з таблицями для користувачів, ролей, проектів, завдань, часових записів тощо.
2. MVC Архітектура: Використання моделі, видів та контролерів для структурування коду та бізнес-логіки додатку.
3. Тестування та Безпека: Розробка тестів для забезпечення стабільності та безпечної роботи всіх компонентів системи.

Кінцеві Кроки:

1. Розгортання: Налаштування серверів та деплоймент додатку в продакшн.
2. Документація: Підготовка користувацьких мануалів та технічної документації для спрощення використання та подальшого розвитку системи.

Програмні засоби інформаційної технологій, створениі на основі фреймворку Ruby on Rails, спеціально призначений для забезпечення

ефективного керування ресурсами в IT-компаніях, де критично важливо мати злагоджені та автоматизовані процеси управління. Вони допоможуть підвищити продуктивність на різних рівнях організації шляхом забезпечення керівникам потужних інструментів для планування, моніторингу та аналізу роботи співробітників і проектів.

2 ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ

2.1 Ruby on Rails як технологія для створення веб-додатка

Архітектура додатка на базі Ruby on Rails, як це видно на схемі (рис. 2.1), відображає архітектурний підхід Model – View – Controller (MVC). Вона забезпечує чітке розділення не тільки між бекендом та фронтендом, але також виділяє моделі, які управляють даними, відображення (view), яке забезпечує візуалізацію інформації, та контролери, які обробляють взаємодію з користувачем і керують моделями щодо змін [27, 28, 30].

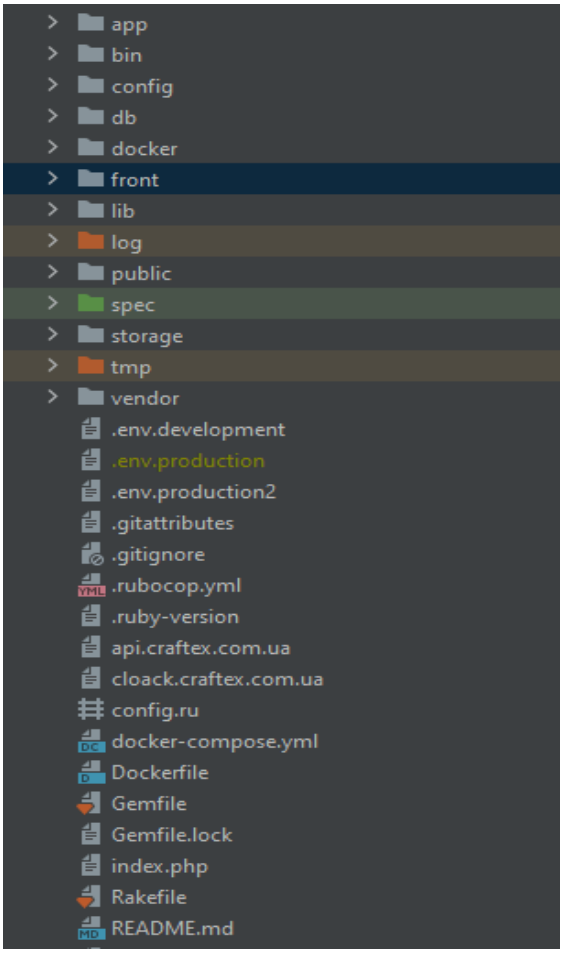
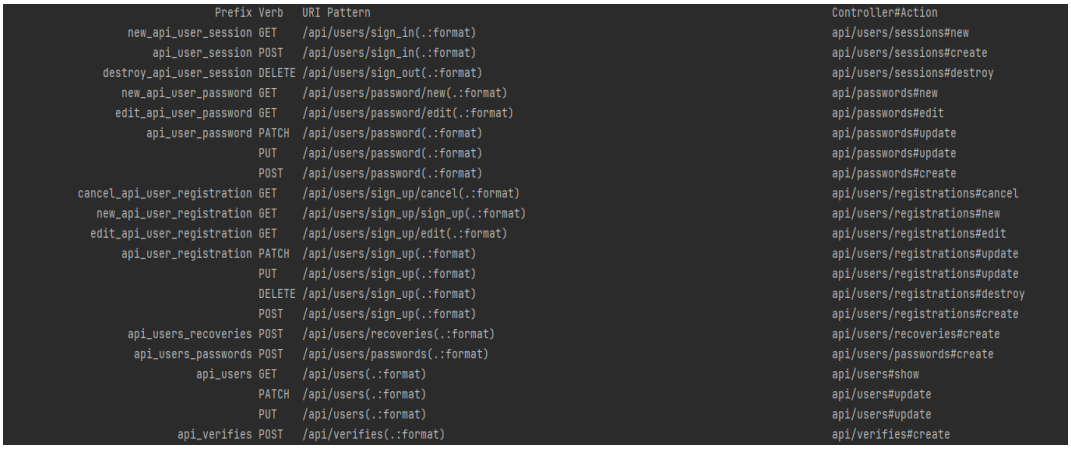


Рисунок 2.1 – Структура RoR додатку

Ресурсний роутинг у Ruby on Rails дозволяє ефективно встановлювати стандартні маршрути для контролера, що керує ресурсами (рисунок 2.2). Кожен метод у цьому контексті є запитом для виконання дії з ресурсом. Конфігурація

маршрутів здійснюється через файл config/routes.rb, де ресурсний маршрут асоціюється з низкою дій у відповідному контролері.



Prefix	Verb	URI Pattern	Controller#Action
new_api_user_session	GET	/api/users/sign_in(.:format)	api/users/sessions#new
api_user_session	POST	/api/users/sign_in(.:format)	api/users/sessions#create
destroy_api_user_session	DELETE	/api/users/sign_out(.:format)	api/users/sessions#destroy
new_api_user_password	GET	/api/users/password/new(.:format)	api/passwords#new
edit_api_user_password	GET	/api/users/password/edit(.:format)	api/passwords#edit
api_user_password	PATCH	/api/users/password(.:format)	api/passwords#update
	PUT	/api/users/password(.:format)	api/passwords#update
	POST	/api/users/password(.:format)	api/passwords#create
cancel_api_user_registration	GET	/api/users/sign_up/cancel(.:format)	api/users/registrations#cancel
new_api_user_registration	GET	/api/users/sign_up/sign_up(.:format)	api/users/registrations#new
edit_api_user_registration	GET	/api/users/sign_up/edit(.:format)	api/users/registrations#edit
api_user_registration	PATCH	/api/users/sign_up(.:format)	api/users/registrations#update
	PUT	/api/users/sign_up(.:format)	api/users/registrations#update
	DELETE	/api/users/sign_up(.:format)	api/users/registrations#destroy
	POST	/api/users/sign_up(.:format)	api/users/registrations#create
api_users_recoveries	POST	/api/users/recoveries(.:format)	api/users/recoveries#create
api_users_passwords	POST	/api/users/passwords(.:format)	api/users/passwords#create
api_users	GET	/api/users(.:format)	api/users#show
	PATCH	/api/users(.:format)	api/users#update
	PUT	/api/users(.:format)	api/users#update
api_verifies	POST	/api/verifies(.:format)	api/verifies#create

Рисунок 2.2 – Маршрути додатку, що використовуються для інтеграції

Маршрутизація дозволяє не лише створювати внутрішні посилання у проекті, а й вводити нові методи, що дає можливість одразу спостерігати за внесеними змінами у функціоналі. Однак, ключовим інструментом для розробки у Ruby on Rails є Rubygems. Кожен gem визначається своїм іменем, версією та сумісною платформою. Gems призначені для використання з Ruby і оптимізовані для певної платформи, засновані на архітектурі процесора та версії операційної системи [1-4].

2.2 Локалізація

Інтернаціоналізація, або I18n, це ключовий процес у розробці програмного забезпечення, що підготовлює продукт до легкої адаптації під різні культурні та мовні умови. Оскільки мови мають різні особливості, створення універсальних інструментів для вирішення всіх мовних питань є складним завданням. Інтернаціоналізоване програмне забезпечення ефективно адаптується до особливостей місцевих ринків, відповідаючи місцевим стандартам і вимогам, та забезпечуючи вищу задоволеність користувачів.

Локалізація в контексті веб-додатків на Ruby on Rails включає налаштування додатка таким чином, щоб текстові ресурси могли бути легко

перекладені на бажані мови. На головній сторінці такого додатка користувачам надається можливість вибору локалізації зручно і інтуїтивно.

Інструмент інтернаціоналізації у Ruby on Rails, Ruby gem I18n, складається з двох частин:

1. Публічний API I18n, який є модулем Ruby з визначеними публічними методами, керуючи загальними налаштуваннями інтернаціоналізації.
2. Бекенд, який виконує вказані методи, реалізуючи потрібну функціональність.

Завдяки принципу "Convention over Configuration" (перевага угод перед конфігурацією), Rails автоматично використовує стандартні налаштування перекладів, але при цьому дозволяє користувачам легко перевизначати ці налаштування за потребою, щоб точніше відповідати локальним вимогам [11, 14, 18, 31, 32].

2.3 База даних

Для забезпечення максимально ефективного управління ресурсами та проектами в IT-секторі, дуже важливо мати добре сплановану та структуровану базу даних. Це допомагає оптимізувати процеси в компанії та забезпечує швидкий доступ до необхідної інформації для різних користувачів. Нижче представлена модель бази даних для веб-додатку на базі Ruby on Rails і PostgreSQL, який реалізує функціональність управління персоналом і проектами [8-12]. Ось приклад такої структури:

Таблиці бази даних:

1. **users** - Таблиця для збереження інформації про користувачів:
    - id (integer, PK)
    - email (string, unique)
    - password_digest (string)
    - role (string) - наприклад, 'admin', 'project_manager', 'employee'
    - created_at (datetime)
    - updated_at (datetime)

2. **profiles** - Таблиця для збереження профілів користувачів:
    - id (integer, PK)
    - user_id (integer, FK to users)
    - first_name (string)
    - last_name (string)
    - photo_url (string)
    - contact_number (string)
    - status (string) - наприклад, 'active', 'on_leave', 'sick'
    - education (text)
    - experience (text)
    - created_at (datetime)
    - updated_at (datetime)
3. **projects** - Таблиця для збереження проектів:
    - id (integer, PK)
    - name (string)
    - description (text)
    - start_date (date)
    - end_date (date)
    - status (string) - наприклад, 'active', 'completed', 'on_hold'
    - created_at (datetime)
    - updated_at (datetime)
4. **assignments** - Таблиця для збереження призначень співробітників на проекти:
    - id (integer, PK)
    - project_id (integer, FK to projects)
    - user_id (integer, FK to users)
    - role (string) - роль в проекті, наприклад, 'developer', 'tester' assigned_date (date)

- release_date (date)
- created_at (datetime)
- updated_at (datetime)

5. **time_entries** - Таблиця для обліку часу, витраченого на проекти:
    - id (integer, PK)
    - project_id (integer, FK to projects)
    - user_id (integer, FK to users)
    - date (date)
    - hours_spent (decimal)
    - description (text)
    - created_at (datetime)
    - updated_at (datetime)
6. **reports** - Таблиця для автоматичних звітів:
    - id (integer, PK)
    - title (string)
    - content (text)
    - generated_date (datetime)
    - user_id (integer, FK to users)
    - created_at (datetime)
    - updated_at (datetime)

Відносини між таблицями:

- **Users** мають один до одного відношення з **Profiles**.
- **Users** можуть мати багато **Assignments** і **TimeEntries**.
- **Projects** мають багато Assignments і **TimeEntries**.

Ця база даних створена для підтримки гнучкого і масштабованого веб-додатку, що дозволяє не тільки ефективно керувати ресурсами і проектами, а й надає потужні інструменти для аналізу та звітності, забезпечуючи адаптивність до змінних вимог бізнесу та вдосконалення внутрішніх процесів.

2.4 Пакетний менеджер

Для досягнення оптимальної продуктивності додатків розроблених на базі Ruby on Rails, інструментом незамінної важливості є менеджер пакетів Bundler. Ця утиліта спеціалізується на управлінні залежностями gem, забезпечуючи розробникам значний контроль над бібліотеками, що використовуються в проекті. Bundler дозволяє автоматизувати процес інсталяції необхідних gem-ів, не звертаючи уваги на ті, що вже присутні в системі, що істотно спрощує управління проектними залежностями [3-5].

Залежності gem-ів в Rails традиційно визначаються через файли конфігурації, такі як config.gem та enviroment.rb. Однак, Bundler пропонує більш ефективне та зручне рішення для цих завдань, автоматично вирішуючи конфлікти версій та залежності між різними gem-ами. Це забезпечує високу ступінь стабільності та зручності при розробці.

Починаючи з версії Rails 3.0, Bundler став інтегрованим за замовчуванням менеджером залежностей, що є свідченням його важливості і надійності у екосистемі Ruby on Rails. Завдяки його універсальності, Bundler не лише поліпшує процес розробки на Rails, але й є корисним управлінням залежностями в будь-яких інших проектах на Ruby, дозволяючи розробникам зосередитися на самій розробці, мінімізуючи технічні перепони, пов'язані з підтримкою програмного забезпечення [4-8].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Опис маршрутів та доступів

Одна з найважливіших аспектів роботи програми - це налагодження відповідних маршрутів додатку. Це визначає, як користувачі можуть взаємодіяти з системою та які операції вони можуть виконувати. Для цього нам допомагає обраний нами фреймворк Ruby on Rails (RoR) та вбудована система маршрутизації.

RoR пропонує зручний та ефективний спосіб визначення маршрутів за допомогою файлу `routes.rb`, де ми можемо вказати URL-адреси та зв'язати їх з контролерами та діями, що відповідають за обробку запитів. Це дозволяє нам легко організувати структуру нашого додатку та забезпечити зручний та логічний доступ до різноманітних функцій і функціональностей.

Система маршрутизації RoR дозволяє також здійснювати динамічне маршрутизування та використання параметрів в URL-адресах, що дозволяє створювати більш гнучкі та потужні маршрути для нашого додатку. Це особливо важливо у великих проектах зі складною логікою маршрутизації, де необхідно точно керувати розподілом запитів та забезпечувати їх відповідну обробку.

Таким чином, завдяки RoR та його вбудованій системі маршрутизації, ми можемо легко керувати маршрутами нашого додатку, забезпечуючи його ефективну та правильну роботу для користувачів.

Давайте детальніше пройдемося по описаних маршрутах – див. Додаток Є. Цей код маршрутів визначає структуру нашого Rails додатку та визначає, які URL-шляхи відповідають кожному ресурсу або функціоналу. Давайте розглянемо кожен рядок коду та пояснимо його:

1. ‘mount Rswag::Api::Engine => '/api-docs'’: Цей рядок встановлює маршрут для генерації документації API за допомогою гему Rswag. Документація буде доступна за адресою /api-docs.

2. ‘mount Rswag::Ui::Engine => '/api-docs'’: Цей рядок встановлює маршрут для інтерфейсу користувача генератора документації API. Інтерфейс буде доступний за адресою /api-docs.
3. ‘namespace :api’: Цей блок визначає простір імен для всіх наших API маршрутів. Усі маршрути у цьому блоку будуть починатися з /api.
4. ‘devise_for :users’: Цей рядок встановлює маршрути для автентифікації користувачів за допомогою гему Devise. Він також вказує на те, що реєстрація буде доступна за /api/users/sign_up, вхід - за /api/users/sign_in, а вихід - за /api/users/sign_out.
5. ‘namespace :users’: Цей блок визначає простір імен для маршрутів, пов'язаних з користувачами.
6. ‘resources :recoveries, only: %i[create]’ та ‘resources :passwords, only: %i[create]’: Ці рядки встановлюють маршрути для створення запитів на відновлення паролю та створення нового паролю. Вони обмежені лише дією створення.
7. ‘resource :users, only: %i[show update]’: Цей рядок встановлює маршрут для отримання та оновлення інформації про користувача. Він дозволяє отримувати дані про користувача та оновлювати їх.
8. ‘resources :projects’, ‘resources :reports’, ‘resources :time_entries’, ‘resources :assignments’: Ці рядки встановлюють маршрути для керування проектами, звітами, відомостями про витрати часу та призначеннями. Вони дозволяють створювати, зчитувати, оновлювати та видаляти записи для кожного з цих ресурсів.

Цей код визначає основну структуру маршрутів для нашого додатку, яка дозволяє користувачам взаємодіяти з різними частинами системи за допомогою HTTP-запитів.

3.2 Бібліотеки (Gems)

Ключовою складовою реалізації веб-додатка на Ruby on Rails є використання gem-файлів, що забезпечують функціональність різних бібліотек.

Під час розробки веб-додатка були використані наступні gem-файли – див. Таблицю 3.1.

Таблиця 3.1 – Бібліотеки які використовує проект

Найменування	**Функціонал, який забезпечується даною біблотекую**
devise	Цей гем надає гнучку та потужну аутентифікацію та авторизацію користувачів у Rails додатках. Він дозволяє швидко створити рішення для реєстрації, увіходу в систему, відновлення пароля та інші пов'язані функції.
devise-jwt	Цей гем розширює функціонал гему devise, дозволяючи створювати та перевіряти JWT (JSON Web Tokens) для аутентифікації користувачів.
devise-two-factor	Цей гем додає підтримку двофакторної аутентифікації до гему devise, забезпечуючи додатковий рівень безпеки для вхідних в систему користувачів.
dotenv-rails	Цей гем дозволяє завантажувати змінні середовища з файлу .env у Rails додатках, що дозволяє керувати конфігурацією середовища для різних етапів розробки.
httparty	Цей гем надає простий та зручний спосіб здійснення HTTP запитів у Rails додатках, що дозволяє взаємодіяти з зовнішніми API.
pagy	Цей гем забезпечує розширений функціонал пагінації для Rails додатків, що дозволяє створювати зручні та ефективні сторінки для відображення великої кількості даних.
pg	Цей гем забезпечує підтримку PostgreSQL баз даних у Rails додатках, що дозволяє використовувати всі можливості цієї потужної реляційної системи управління базами даних.
pg_Search	Цей гем надає додаткові можливості для пошуку та індексації даних у PostgreSQL базах даних, що полегшує роботу з пошуковими функціями в Rails додатках.
rack-cors	Цей гем додає підтримку CORS (Cross-Origin Resource Sharing) для Rails додатків, що дозволяє здійснювати безпечну взаємодію з ресурсами на інших доменах.

Продовження Табл. 3.1

pundit	Цей гем надає простий та ефективний спосіб виконання авторизації в Rails додатках на основі правил та політик, що дозволяє керувати доступом користувачів до різних ресурсів.
factory_bot_rails	Цей гем дозволяє створювати фабрики для тестування моделей у Rails додатках, що полегшує створення та управління тестовими даними.
faker	Цей гем надає зручний механізм для генерації фейкових даних у Rails додатках, що допомагає заповнювати базу даних тестовими даними або для розвитку.
rails-controller-testing	Цей гем надає зручний інтерфейс для тестування контролерів у Rails додатках, що дозволяє перевіряти правильність реалізації контролерів у різних сценаріях.
rspec	Цей гем є популярним фреймворком для тестування у Rails додатках, що надає зручний та ефективний спосіб написання й виконання тестів.
rspec-collection_matchers	Цей гем розширює функціонал RSpec, дозволяючи перевіряти колекції даних у тестах з більшою гнучкістю та зручністю.
rspec-its	Цей гем додає підтримку ізольованих тестів для атрибутів об'єктів у RSpec, що полегшує тестування різних властив

3.3 Написання програмної реалізації

Перейдемо до створення основної частини застосунку. В першу чергу нам потрібно створити відповідні моделі, які будуть відображати основні сутності системи.

1. User (Користувач): Ця модель відповідає за реєстрацію та управління користувачами системи.
2. Project (Проект): Модель, яка представляє собою проекти, над якими працюють користувачі.

3. Assignment (Призначення): Модель, яка відображає зв'язок між користувачами і проектами, вказуючи, хто працює над певним проектом.
4. TimeEntry (Відомість про витрату часу): Модель, що відображає інформацію про час, витрачений на різні завдання або проекти.
5. Report (Звіт): Модель, яка представляє собою звіти, створені користувачами на основі відомостей про витрату часу або інших даних.
6. Profile (Профіль): Модель, що містить додаткову інформацію про користувачів, яка не включена в стандартну модель користувача.

Для створення цих моделей ми скористаємося командою ‘rails g migration <Найменування> -m’, яка допоможе створити відповідну модель та міграційну таблицю в базі даних – див. Рисунок 3.1

```
rails g migration User -m
rails g migration Project -m
rails g migration TimeEntry -m
rails g migration Report -m
rails g migration Assignment -m
rails g migration Profile -m
```

Рисунок 3.1 – Перелік команд для створення відповідних моделей та міграцій

Ці команди створять відповідні файли міграцій, в яких буде описано створення таблиць для кожної з моделей, а також їхніх атрибутів. Після цього ми зможемо визначити необхідні атрибути для кожної моделі та виконати міграцію для створення таблиць в базі даних – див. Додаток Ж.

Після створення моделей нам потрібно створити відповідні контролери для кожної з них. Контролери відповідають за обробку запитів користувачів та

взаємодію з моделями. Ось як ми можемо створити контролери для кожної з описаних вище моделей – див. Рисунок 3.2.

```
rails g controller Api::Users
rails g controller Api::Projects
rails g controller Api::Assignments
rails g controller Api::TimeEntries
rails g controller Api::Reports
rails g controller Api::Users::Passwords
rails g controller Api::Users::Recoveries
rails g controller Api::Users::Registrations
rails g controller Api::Users::Sessions
```

Рисунок 3.2 – Перелік команд для створення відповідних контролерів

Ці команди створять відповідні файли контролерів у каталозі ‘app/controllers’. Після створення контролерів ми можемо визначити методи дій (action methods), такі як ‘index’, ‘show’, ‘create’, ‘update’, ‘destroy’, які будуть відповідати за обробку різних типів запитів до кожної моделі.

Крім того, ми можемо визначити власні методи дій для обробки специфічних випадків взаємодії з моделями. Наприклад, для моделі ‘TimeEntry’ ми можемо створити метод дії для обчислення загального часу, витраченого на всі проекти – див. Додаток Ж.

Після створення контролерів та визначення методів дій ми зможемо встановити маршрути, що вказують, які URL-шляхи будуть відповідати кожному з цих методів дій. Таким чином, ми зможемо забезпечити доступ користувачам до різних функцій та ресурсів нашого додатку через відповідні URL-шляхи.

Таким чином, користувачі зможуть взаємодіяти з даними через додатки, які реалізовані на базі нашого API, з використанням створених контролерів та маршрутів. За допомогою цього API користувачі матимуть доступ до різних функцій і ресурсів системи, таких як створення, читання, оновлення та видалення записів користувачів, проектів, відомостей про витрату часу та інших даних.

Додатки, що використовують наше API, зможуть взаємодіяти з ним через стандартні HTTP-запити, використовуючи методи, які ми визначили у контролерах. Це дозволить розробникам створювати різноманітні клієнтські додатки, включаючи веб-додатки, мобільні додатки та інші програми, які можуть використовувати нашу систему.

3.4 Створення інтеграційних тестів

Для зручної та швидкої інтеграції будемо використовувати бібліотеку Rswag. Ця бібліотека допоможе нам описати інтеграційні тести та створити документацію для нашого API. Давайте розглянемо кожен інтерфейс окремо, щоб визначити його функціональність та очікувані результати.

Для початку розглянемо один з найважливіших інтерфейсів нашого додатку - "Users". Для цього ми повинні створити файл з інтеграційними тестами, який допоможе нам перевірити коректність роботи цього інтерфейсу.

Для цього перейдімо у папку **rspec** та створимо нову директорію під назвою **integration**. Потім нам потрібно створити сам файл з тестами. Назвемо його **users_spec.rb**. У цьому файлі ми будемо описувати та тестувати різні маршрути, пов'язані з інтерфейсом "Users" – див. Таблицю 3.2.

Ці тести допоможуть нам переконатися, що кожен маршрут працює правильно і надає очікувану відповідь. Наприклад, ми можемо перевірити, чи коректно працює маршрут для отримання інформації про користувача, чи чи відбувається додавання нового користувача у систему. Ці тести дозволять нам впевнитися в коректності роботи інтерфейсу "Users" та відповідності його функціональності вимогам нашого додатку.

Таблиця 3.2 – Перелік маршрутів для Users інтерфейсу

Маршрут	**Опис**
/api/users/sign_up	Створення користувача
/api/users/sign_in	Авторизація користувача
/api/users/recoveries	Відновлення паролю користувача
/api/users/passwords	Встановлення нового паролю користувача
/api/users	Отримання інформації про користуча

Зважаючи на важливість тестування поведінки нашого інтеграційного інтерфейсу, ми також розглянемо всі можливі HTTP статус коди, які може повертати наш інтерфейс – див. Таблицю 3.3

Під час написання наших інтеграційних тестів ми будемо переконуватися, що кожен маршрут нашого інтерфейсу повертає правильний статус код у відповідь на запит. Наприклад, при успішному виконанні запиту на отримання інформації про користувача, ми очікуємо отримати статус код '200 OK'. У разі невдачі, наприклад, при неправильних вхідних даних або відсутності ресурсу, ми можемо очікувати інші статуси, такі як '400 Bad Request' або '404 Not Found'.

Цей підхід дозволить нам не лише перевірити правильність роботи кожного маршрут, а й забезпечити коректну поведінку нашого інтерфейсу у всіх можливих ситуаціях, що дозволить нам зробити наш додаток більш надійним та стійким до помилок.

Таблиця 3.3 – Перелік статус кодів

Код	**Опис**
200	Успішне виконання з поверненням об'єктів з поверненням об'єктів
204	Успішне виконання з пустою відповідью
201	Успішне створення з поверненням об'єктів у відповідь
401	Помилка авторизації відправ
422	Помилка в прийнятті передаваємих параметрів
404	Ресурс не знайдено
400	Невірний запит

Вище ми детально описали маршрути та HTTP статус-коди, які потрібно протестувати та задокументувати. Тепер перейдемо до самої практичної реалізації цих тестів. У створеному файлі users_spec.rb давайте реалізуємо наступне:

1. Написання тестів для кожного маршрута: Почнемо з написання тестів для кожного маршрута, описаного у нашій специфікації. Для кожного маршрута ми

будемо перевіряти коректність отриманих відповідей та відповідність їх очікуваним HTTP статус-кодам.

2. Налаштування фіктивних даних: Створимо фіктивні дані для виконання наших тестів. Це можуть бути тестові користувачі з різними параметрами, які допоможуть нам протестувати різні сценарії взаємодії з інтерфейсом "Users".

3. Запуск тестів і перевірка результатів: Після написання тестів ми запустимо їх і переконаємося, що кожен з них проходить успішно. Важливо перевірити, що отримані результати відповідають очікуваним результатам, описаним у нашій специфікації.

4. Створення документації: Після успішного виконання тестів Rswag автоматично збере всю інформацію та згенерує документацію на основі наших тестів. Ця документація буде містити опис кожного маршрута, його параметрів та очікуваних відповідей, що дозволить нам зробити наш API більш доступним та зрозумілим для інших розробників.

Цей підхід дозволить нам не лише ефективно протестувати наш інтерфейс "Users", але і автоматично згенерувати актуальну та зрозумілу документацію для нього, що є важливим аспектом у розробці сучасних веб-додатків.

Давайте розглянемо наступний тест, який створений для API шляху /api/users/sign_up за допомогою Rswag - див. рис. 3.3. Цей тест призначений для перевірки функціоналу створення нового користувача в системі.

Спочатку, у нашому описі ми вказуємо метод HTTP, який буде використовуватися для цього запиту, а саме POST. Це означає, що ми будемо надсилати дані на сервер для створення нового користувача.

Після цього ми вказуємо теги для цього шляху, які допомагають організувати наші тести і вказують на їхню призначеність. У цьому випадку, це тег 'Users', що позначає, що цей шлях пов'язаний з операціями, що стосуються користувачів.

Далі ми описуємо параметри, які необхідно передати для створення нового користувача. Це можуть бути дані, такі як email, пароль та

підтвердження паролю. Ми вказуємо їх тип, опис і чи вони обов'язкові для заповнення.

```
path '/api/users/sign_up' do
  post 'Create a user' do
    tags 'Users'
    consumes 'application/json'
    produces 'application/json'

    parameter name: :user,
              in: :body,
              description: 'Create a user',
              schema: {
                type: :object,
                properties: {
                  email: { type: :string },
                  password: { type: :string },
                  password_confirmation: { type: :string }
                },
                required: %w[email password password_confirmation]
              }

    response '201', 'OK' do
      schema type: :object,
             properties: {
               id: { type: :integer },
               email: { type: :string },
               role: { type: :string },
               profile: {
                 type: :object,
                 properties: {
                   id: { type: :integer },
                   first_name: { type: :string },
                   last_name: { type: :string },
                   photo_url: { type: :string },
                   contact_number: { type: :string },
                   status: { type: :string },
                   education: { type: :string },
                   experience: { type: :string },
                 }
               }
             }
      run_test!
    end

    response '401', 'Failed' do
      schema type: :object,
             properties: {
               error: { type: :string }
             }
      run_test!
    end
  end
end
```

Рисунок 3.3 – Інтеграційний тест для одно маршрутів

Після цього ми переходимо до опису очікуваних відповідей від сервера. Наприклад, у випадку успішного створення користувача, ми очікуємо код відповіді 201 та об'єкт з даними нового користувача, такими як id, email та роль.

Також ми описуємо можливі варіанти помилок, які можуть виникнути при створенні користувача, наприклад, якщо користувач введе невірні дані або вже існуючий email. У такому випадку сервер повинен повернути код відповіді 401 та повідомлення про помилку.

Після написання опису тесту ми запускаємо його, щоб переконатися, що він працює правильно. Якщо всі тести успішно пройдені, Rswag автоматично генерує документацію на основі цих тестів, що дозволяє розробникам краще розуміти функціонал

Тепер ми можемо зрозуміти, у якому форматі ми описуємо тестування інтерфейсу. Завдяки специфікації Rswag ми можемо зручно і чітко описати кожен шлях API, його параметри та очікувані відповіді сервера. Повністю ми можемо ознайомитися з описаним інтерфейсом - див. Додатку Б, де ми детально розглянули тести для різних операцій, таких як створення користувача, авторизація, відновлення паролю, встановлення нового паролю та отримання інформації про користувача. Це дозволяє нам усвідомити всі можливі варіанти використання нашого API і переконатися, що вони відповідають нашим потребам і очікуванням.

Вище ми розглянули порядок тестування наших інтерфейсів, відзначивши його важливість для забезпечення надійності та ефективності роботи програмного забезпечення. Тепер давайте продовжимо, описавши інші інтерфейси, з яких почнемо з інтерфейсу Reports - див. Додаток В.

Інтерфейс Reports зазвичай відповідає за управління звітами в системі. При його розробці ми маємо враховувати різноманітні сценарії, включаючи створення нових звітів, редагування вже існуючих, видалення, перегляд та фільтрацію за різними критеріями. Ми маємо наступний перелік маршрутів який ми повинні протестувати - див. Таблиця 3.4 Кожен з цих сценаріїв повинен

бути ретельно протестований для впевненості в його коректному функціонуванні.

Для цього ми створимо окремий файл інтеграційних тестів для інтерфейсу Reports, аналогічно до того, як ми робили це для інтерфейсу Users. У цьому файлі будуть описані різні тести, які перевіряють різні аспекти роботи інтерфейсу Reports. Наприклад, ми можемо перевірити, чи відбувається коректне створення нового звіту, чи можна видалити існуючий звіт, чи правильно відображаються дані у вигляді списку звітів тощо.

Такий підхід дозволить нам переконатися, що інтерфейс Reports працює належним чином та задовольняє всі вимоги функціональності, що вимагаються від нього.

Таблиця 3.4 – Перелік маршрутів для Reprots інтерфейсу

Маршрут	**Опис**
GET /api/reports	Отримання повного переліку репортів
POST /api/reports	Створення репорту
GET /api/reports/{id}	Отримати інформацію про певний репорт за його ІД
PUT /api/reports/{id}	Внесення змін в репорті за його ІД
DELETE /api/reports/{id}	Видалення репорту по його ІД

Далі ми маємо інтерфейс – Projects. Даний інтерфейс відповідає за управління проектами в системі. Тут ми можемо створювати нові проекти, редагувати існуючі, призначати користувачів для роботи над проектами, встановлювати терміни виконання та інші параметри, що стосуються керування проектами.

Також ми можемо ознайомитися з переліком ротів для тестування - див. Таблиця 3.5. Реалазація самого інтеграційного тесту - див. Додаток Г

Таблиця 3.5 – Перелік маршрутів для Projects інтерфейсу

Маршрут	**Опис**
GET /api/projects	Отримання повного переліку проекту
POST /api/projects	Створення проекту
GET /api/projects /{id}	Отримати інформацію про певний проекту за його ІД
PUT /api/projects/{id}	Внесення змін в проекту за його ІД
DELETE /api/projects/{id}	Видалення проекту по його ІД

Інтерфейс Assignments дозволяє нам призначати користувачам робочі завдання у межах проектів. Це можуть бути різні завдання, які включаються до проекту, і вони можуть мати різні властивості, такі як термін виконання, пріоритет та інші. Маршрути які ми будемо тестувати – див. Таблиця 3.6. Реалізація самого інтеграційного тесту - див. Додаток Ґ

Таблиця 3.6 – Перелік мар для Assignments інтерфейсу

Маршрут	**Опис**
GET /api/assignments	Отримання повного переліку назначень людей на ппроекти
POST /api/assignments	Створення нового назначення
GET /api/assignments /{id}	Отримати інформацію про певне назначення за його ІД
PUT /api/assignments /{id}	Внесення змін в назначення за його ІД
DELETE /api/assignments /{id}	Видалення назначення по його ІД

Інтерфейс TimeEntries призначений для реєстрації часу, витраченого користувачами на виконання робочих завдань або проектів. Цей інтерфейс дозволяє користувачам вказувати кількість витраченого часу, а також пов'язувати часові записи з конкретними завданнями чи проектами. Маршрути які ми будемо тестувати – див. Таблиця 3.7 Реалазація самого інтеграційного тесту - див. Додаток Д

Таблиця 3.7 – Перелік маршрутів для TimeEntries інтерфейсу

Маршрут	Опис
GET /api/time_entries	Отримання повного переліку витраченого часу користувачами
POST /api/time_entries	Створення запису витраченого часу
GET /api/time_entries/{id}	Отримати інформацію про витраченний час за його ІД
PUT /api/time_entries/{id}	Внесення змін в запис витраченого часу за його ІД
DELETE /api/time_entries/{id}	Видалення запису витраченого часу по його ІД

3.5 Тестування інтеграційних тестів

Описавши наші інтеграційні тести в попередньому підпункті, тепер ми можемо перейти до їхнього тестування. Для цього ми запускаємо набір тестів, які ми написали для кожного інтерфейсу, і перевіряємо їх на відповідність очікуваному результату. Щоб запустити автоматичне тестування описаних наших маршрутів треба виконати наступну команду:

‘rspec’ – Команда яку додає біблотека Rspec для покриття тестами проетктy

Після виконання команди в консолі додатку ми зможемо побачити скільки тестів було написано та які з них були завершені успішно – див. Рисунок 3.4

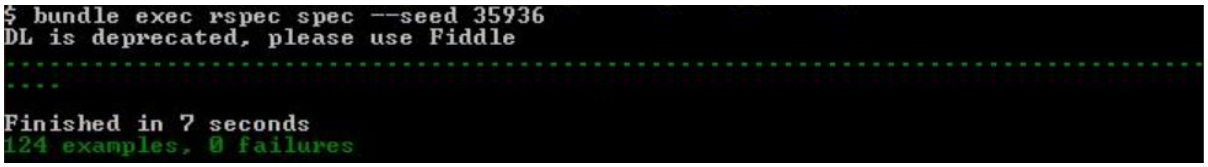


Рисунок 3.4 – Результати тестування

Важливо перевірити кожен аспект функціоналу, щоб переконатися, що наші тести покривають всі можливі сценарії використання інтерфейсів. Це допомагає забезпечити високу якість програмного забезпечення і виявити можливі проблеми або помилки в ранніх етапах розробки.

Після успішного завершення тестів ми можемо бути впевнені в тому, що наші інтерфейси працюють належним чином і готові для використання в продуктовому середовищі.

3.6 Документація інтерфейсів

Після успішного створення та вдалого пройдення тестів інтеграції для наших маршрутів ми отримали впевненість у їхній працездатності та відповідності очікуванням. Тепер настав час перейти до наступного етапу - ознайомлення зі згенерованою документацією. Ця документація відображає усі доступні маршрути, їх параметри, та очікувані відповіді. Вона є не лише корисним довідковим матеріалом для розробників, але й важливим ресурсом для команди, яка працює над проектом, дозволяючи швидко зорієнтуватися у функціоналі системи. Під час ознайомлення з документацією ми матимемо можливість перевірити, чи відповідають наші реалізації очікуванням, а також виявити можливі прогалини або покращення, які можна внести у систему. Такий підхід допомагає підтримувати наш проект на високому рівні якості та забезпечує ефективну співпрацю всіх учасників команди. UI документація зображена - див. Рисунок з 3.5 по 3.6

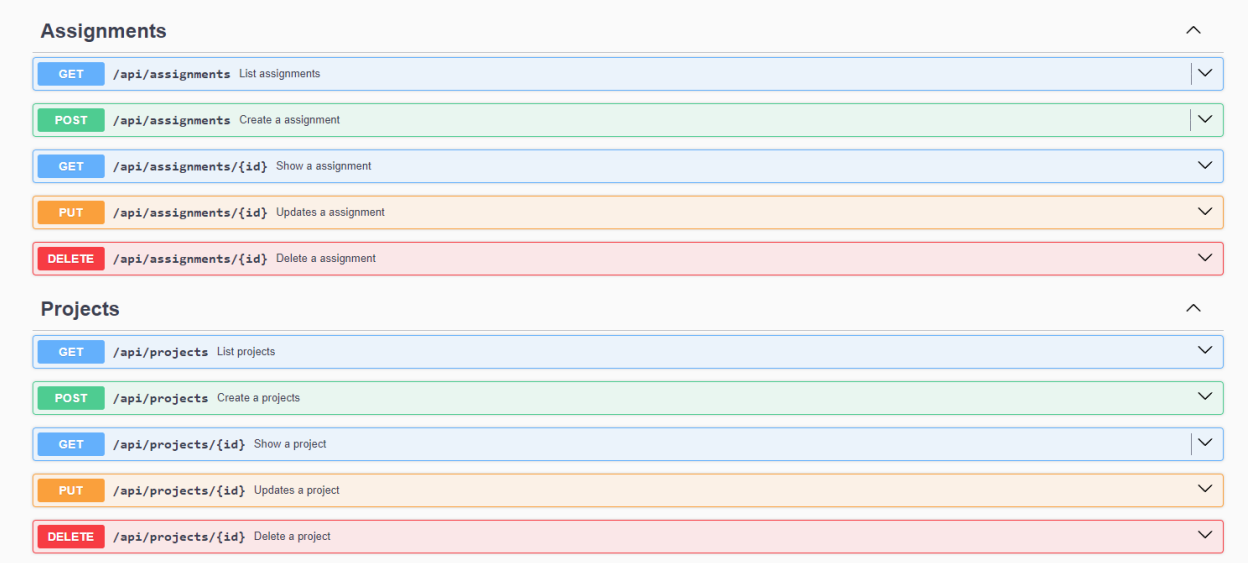


Рисунок 3.5 – UI документація маршрутів Assignments та Projects

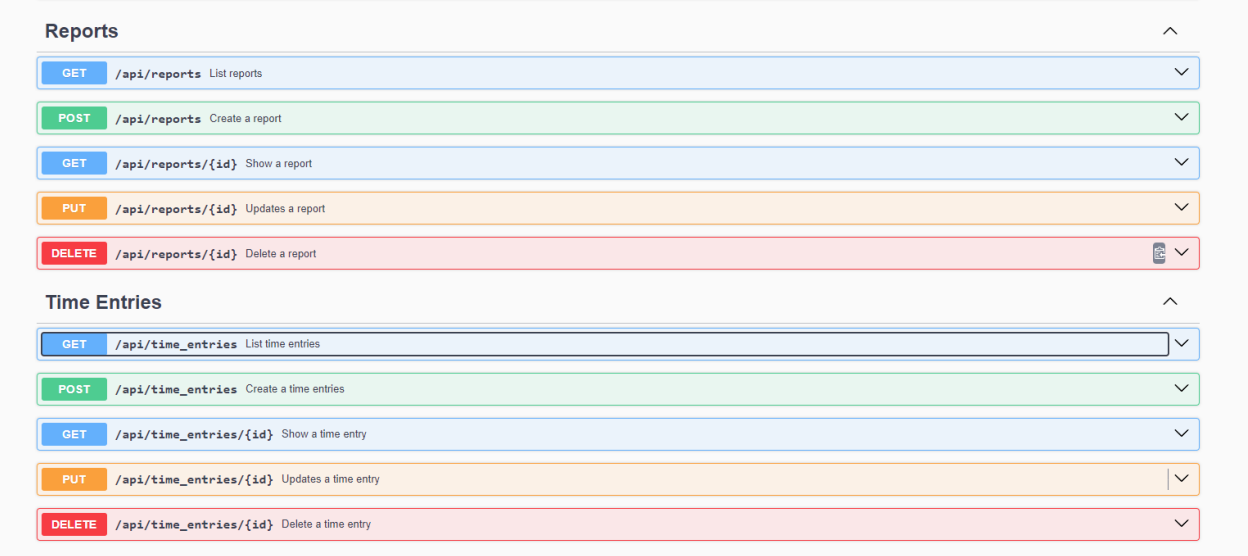


Рисунок 3.6 – UI документація маршрутів Reports та Time Entries

Давайте зануримося глибше у структуру одного з задокументованих маршрутів. Візьмемо, наприклад, інтерфейс користувачів (Users) та маршрут `/api/users/sign_up`. Цей маршрут призначений для реєстрації нового користувача в системі. При використанні цього маршруту ми маємо вказати необхідні параметри, щоб успішно створити обліковий запис користувача. Документація надає вичерпний опис цих параметрів, їх типів та обов'язковості (див. Рисунок 3.8).

Після того як ми відправимо запит за допомогою цього маршруту на сервер, ми очікуємо отримати певну відповідь, що залежить від того, які параметри були введені чи не введені. Ця відповідь також детально описана у документації, разом з прикладами можливих варіантів відповідей (див. Рисунок 3.9).

Ознайомлення з такими документованими маршрутами не лише допомагає розібратися у структурі системи, а й дозволяє розуміти очікувані результати взаємодії з сервером. Це важливо для розробників, які інтегруються з системою, а також для тестувальників, які перевіряють правильність роботи API. Ретельне дослідження документації маршрутів сприяє якості розробки та забезпечує послідовність у роботі всієї команди.

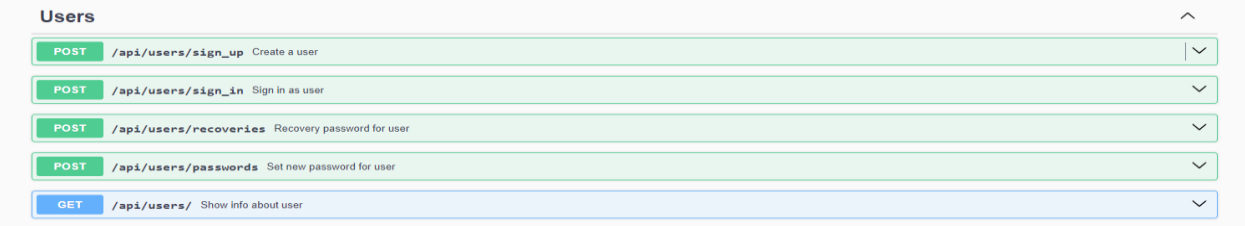


Рисунок 3.7 – UI документація маршрутів Users

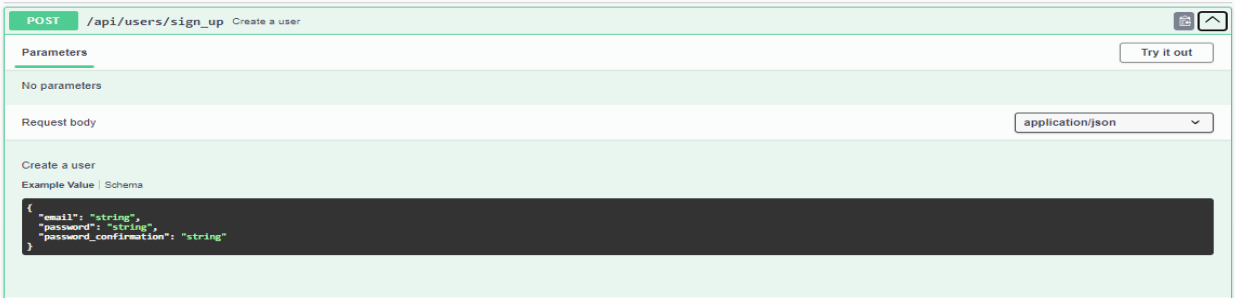


Рисунок 3.8 – Приклад параметрів для створення користувача

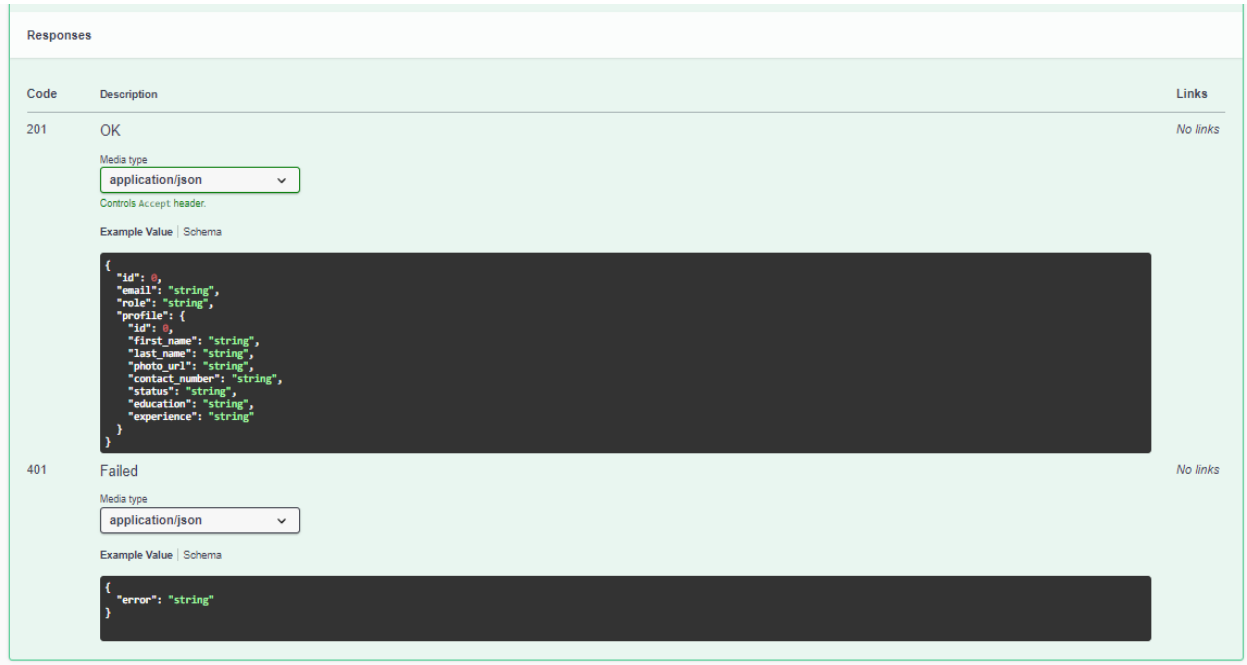


Рисунок 3.9 – Приклад відповіді від серверу після створення користувача

Подібну структуру, що включає опис параметрів, типів даних та очікуваних відповідей, мають і інші маршрути, які були розглянуті раніше. Використовуючи створену документацію, майбутні розробники отримують зручний та структурований ресурс, який дозволяє їм легко створювати свої додатки на основі написаної специфікації та взаємодіяти з системою.

Ця документація є важливим інструментом для будь-якого, хто працює з системою, включаючи нових розробників, які приєднуються до проекту, а також існуючих членів команди, які потребують швидкого доступу до інформації про маршрути та їхню взаємодію з системою.

Будучи докладним та досконалим джерелом інформації, ця документація сприяє розумінню принципів функціонування системи, її можливостей та обмежень. Вона створює платформу для ефективного спілкування між розробниками, тестувальниками та іншими учасниками проекту, сприяючи таким чином швидкому та точному розвитку програмного забезпечення.

ВИСНОВОК

У процесі розробки було створено API додаток, який вирішує безліч проблем у галузі управління людськими ресурсами (HRM). Розроблений додаток мовою програмування Ruby з використанням фреймворку Ruby on Rails та бази даних PostgreSQL є кроком вперед у сфері сучасного управління персоналом в IT-індустрії.

Ruby on Rails, опираючись на принципи "Конвенція перед конфігурацією" та "Суха реалізація", забезпечує швидку розробку, уникнення дублювання коду та легку підтримку. Заснований на архітектурному шаблоні MVC (Model-View-Controller), він дозволяє ефективно розділити логіку, дані та представлення, що спрощує керування складними системами.

PostgreSQL, використовуючи свої функціональні можливості, забезпечує безпечне зберігання та обробку великого обсягу інформації про персонал та їх проекти. Його функціональності, такі як транзакції, контроль доступу та масштабованість, забезпечують високу ефективність роботи системи.

У процесі розробки також було проведено тестування для забезпечення надійності та стабільності розробленого API додатку. Для цього було використано інструменти тестування, що включають у себе як юніт-тести, так і інтеграційні тести. Юніт-тести перевіряють окремі компоненти програми, включаючи моделі, контролери та сервіси. Інтеграційні тести перевіряють взаємодію між різними частинами програми, включаючи HTTP-запити та відповіді.

Крім того, була створена документація API за допомогою інструменту Rswag, яка надає докладний опис усіх доступних маршрутів, параметрів та відповідей. Ця документація допомагає іншим розробникам легко розуміти, як взаємодіяти з API, і забезпечує їм необхідну інформацію для успішної інтеграції з системою.

В цілому, завдяки тестуванню та документації API ми забезпечуємо якість та доступність нашого додатку для користувачів та інших розробників, що сприяє його успішному впровадженню та використанню.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Using Rails for API-only Applications [Електронний ресурс]. – Режим доступу: https://guides.rubyonrails.org/api_app.html– 12.05.2021р.
2. What is a Web Framework [Електронний ресурс]. – Режим доступу: https://www.goodfirms.co/glossary/web-framework/ – 15.10.2020р.
3. Why Blog? The Benefits of Blogging for Business and Marketing [Електронний ресурс]. – Режим доступу: https://blog.hubspot.com/marketing/the-benefits-of-business-blogging-ht/– 17.10.2020р.
4. Надзвичайно корисна перевага Ruby on Rails, яку ви повинні знати [Електронний ресурс]. – Режим доступу: https://mindstack.in/blog/advantages-ruby-rails/ – 15.10.2020р.
5. Плюси та мінуси Ruby on Rails [Електронний ресурс]. – Режим доступу: https://sloboda-studio.com/blog/pros-and-cons-of-ruby-on-rails/– 20.10.2020р.
6. Django проти Laravel проти Rails [Електронний ресурс]. – Режим доступу: https://www.flowkl.com/article/web-development/django-vs-laravel-vs-rails/ – 20.10.2020р.
7. Розуміння архітектури Модель-Вид-Контролер (MVC) в Rails [Електронний ресурс]. – Режим доступу: https://www.sitepoint.com/model-view-controller-mvc-architecture-rails/ – 01.11.2020р.
8. Знайомство з Laravel - Чому [Електронний ресурс]. – Режим доступу: https://laravel.com/docs/8.x#why-laravel/ – 05.02.2021р.
9. Про Django Software Foundation - Чому [Електронний ресурс]. – Режим доступу: https://www.djangoproject.com/foundation/ – 05.02.2021р.
10. Протоколи Інтернету TCP/IP [Електронний ресурс]. – Режим доступу: https://www.britannica.com/technology/TCP-IP – 10.05.2021р.
11. Що таке локалізація і коли вона потрібна? [Електронний ресурс]. – Режим доступу: https://blog.languageline.com/what-is-localization – 29.04.2021р.

12. Стек технологій Ruby on Rails [Електронний ресурс]. – Режим доступу: https://www.railscarma.com/technology-stack/ – 05.05.2021р.
13. What if I Tell You That Ruby on Rails Is Scalable? [Електронний ресурс]. – Режим доступу: https://rubygarage.org/blog/ruby-on-rails-is-scalable – 29.04.2021р.
14. Стан розвитку веб-додатків на Ruby on Rails на початку 2021 року [Електронний ресурс]. – Режим доступу: https://www.ideamotive.co/blog/state-of-ruby-on-rails-web-development – 11.05.2021р.
15. Adams Elliott. Learning Airtable: Building Database-Driven Applications with No-Code .- O'Reilly Media, Inc., 2024. — 382 p.
16. Acuna Pablo. Deploying Rails with Docker, Kubernetes and ECS, Apress, 2016. — 138 p
17. Advanced Database Programming with Rails and Postgres, Pganalyze. — 45 p
18. Agarwal S. Understanding Ruby Regexp: Example based guide to mastering Ruby regular expressions, Leanpub (ua), 2024. — 107 p
19. Agarwal Sudeep, Sehrawat Manoj. Learning Sinatra, Packt Publishing, 2016. — 164 p
20. Aimonetti Matt. MacRuby: The Definitive Guide, O'Reilly, 2011. - 244 p
21. Allen Andrew. Efficient Rails, Leanpub, 2016. — 227 p
22. Allsopp C. RubyMotion: iOS Development with Ruby, The Pragmatic Programmers, LLC., 2013. - 130 p
23. Andrews K., Klaus R., Le R., Bigg R. Active Rails, Independently published, 2021. — 522 p
24. Athayde J., Williams B. The Rails View: Creating a Beautiful and Maintainable User Experience, Pragmatic Programmers, LLC., 2012. - 264 p
25. Atkinson Andrew. High Performance PostgreSQL for Rails. Reliable, Scalable, Maintainable Database Applications, The Pragmatic Bookshelf, 2023. — Beta 3.0 — 288 p

26. Baird K.C. Ruby by Example. Concepts and Code, No Starch Press, 2007. – 281 p.
27. Balbaert Ivo, St. Laurent Simon. Programming Crystal. Create High-Performance, Safe, Concurrent Apps, The Pragmatic Bookshelf, 2019. — 244 p
28. Bates M. Distributed Programming with Ruby, Addison-Wesley Professional, 2009. 250 p.
29. Ben Hamou André. Practical Ruby for System Administration, Apress, 2007. — 262 p.
30. Benson E. The Art of Rails, Wrox, 2008 — 309 p
31. Berkopec Nate. The Complete Guide to Rails Performance, 2nd Edition. — Self-published, 2018. — 386 p
32. Berube D. Practical Ruby Gems, Apress, 2007. — 288 p.

ДОДАТОК А

Лістинг модуля - db/schema.rb

```
ActiveRecord::Schema[7.0].define(version: 2024_05_13_171636) do
  # These are extensions that must be enabled in order to support this database
  enable_extension "plpgsql"

  create_table "assignments", force: :cascade do |t|
    t.bigint "user_id"
    t.bigint "project_id"
    t.string "role"
    t.datetime "assigned_date"
    t.datetime "release_date"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["project_id"], name: "index_assignments_on_project_id"
    t.index ["user_id"], name: "index_assignments_on_user_id"
  end

  create_table "profiles", force: :cascade do |t|
    t.bigint "user_id"
    t.string "first_name"
    t.string "last_name"
    t.string "photo_url"
    t.string "contact_number"
    t.integer "status", default: 0
    t.string "education"
    t.string "experience"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["user_id"], name: "index_profiles_on_user_id"
  end

  create_table "projects", force: :cascade do |t|
    t.string "name"
    t.string "description"
    t.datetime "start_date"
    t.datetime "end_date"
    t.integer "status"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
  end

  create_table "reports", force: :cascade do |t|
    t.bigint "user_id"
    t.string "title"
    t.string "content"
    t.datetime "generated_date"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["user_id"], name: "index_reports_on_user_id"
  end

  create_table "time_entries", force: :cascade do |t|
    t.bigint "user_id"
    t.bigint "project_id"
    t.date "date"
    t.decimal "hours_spent"
    t.string "description"
```

```
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["project_id"], name: "index_time_entries_on_project_id"
    t.index ["user_id"], name: "index_time_entries_on_user_id"
  end

  create_table "users", force: :cascade do |t|
    t.string "email", default: "", null: false
    t.string "encrypted_password", default: "", null: false
    t.integer "role", default: 0, null: false
    t.string "jti"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.index ["email"], name: "index_users_on_email", unique: true
  end

  add_foreign_key "assignments", "projects"
  add_foreign_key "assignments", "users"
  add_foreign_key "profiles", "users"
  add_foreign_key "reports", "users"
  add_foreign_key "time_entries", "projects"
  add_foreign_key "time_entries", "users"
end
```

ДОДАТОК Б

Інтеграціний тест – users_spec.rb

```ruby
require 'swagger_helper'

describe 'Users' do
  path '/api/users/sign_up' do
    post 'Create a user' do
      tags 'Users'
      consumes 'application/json'
      produces 'application/json'

      parameter name: :user,
                in: :body,
                description: 'Create a user',
                schema: {
                  type: :object,
                  properties: {
                    email: { type: :string },
                    password: { type: :string },
                    password_confirmation: { type: :string }
                  },
                  required: %w[email password password_confirmation]
                }

      response '201', 'OK' do
        schema type: :object,
               properties: {
                 id: { type: :integer },
                 email: { type: :string },
                 role: { type: :string },
                 profile: {
                   type: :object,
                   properties: {
                     id: { type: :integer },
                     first_name: { type: :string },
                     last_name: { type: :string },
                     photo_url: { type: :string },
                     contact_number: { type: :string },
                     status: { type: :string },
                     education: { type: :string },
                     experience: { type: :string },
                   }
                 }
               }
        run_test!
      end

      response '401', 'Failed' do
        schema type: :object,
               properties: {
                 error: { type: :string }
               }
        run_test!
      end
    end
  end

  path '/api/users/sign_in' do
```

```
    post 'Sign in as user' do
      tags 'Users'
      consumes 'application/json'
      produces 'application/json'

      parameter name: :user,
                in: :body,
                description: 'Sign in as user',
                schema: {
                  type: :object,
                  properties: {
                    email: { type: :string },
                    password: { type: :string }
                  },
                  required: %w[email password]
                }

      response '201', 'OK' do
        schema type: :object,
               properties: {
                 id: { type: :integer },
                 email: { type: :string },
                 role: { type: :string },
                 profile: {
                   type: :object,
                   properties: {
                     id: { type: :integer },
                     first_name: { type: :string },
                     last_name: { type: :string },
                     photo_url: { type: :string },
                     contact_number: { type: :string },
                     status: { type: :string },
                     education: { type: :string },
                     experience: { type: :string },
                   }
                 },
               }
        run_test!
      end

      response '401', 'Failed' do
        schema type: :object,
               properties: {
                 error: { type: :string }
               }
        run_test!
      end
    end
  end

  path '/api/users/recoveries' do
    post 'Recovery password for user' do
      tags 'Users'
      consumes 'application/json'
      produces 'application/json'

      parameter name: :user,
                in: :body,
                description: 'Recovery password for user',
                schema: {
                  type: :object,
                  properties: {
```

```
                  email: { type: :string },
                  captcha_token: { type: :string }
                },
                required: %w[email captcha_token]
              }

      response '200', 'OK' do
        schema type: :object,
               properties: {
                 status: { type: :string },
                 message: { type: :string },
               }
        run_test!
      end

      response '422', 'Not Found' do
        schema type: :object,
               properties: {
                 error: { type: :string }
               }
        run_test!
      end
    end
  end

  path '/api/users/passwords' do
    post 'Set new password for user' do
      tags 'Users'
      consumes 'application/json'
      produces 'application/json'

      parameter name: :user,
                in: :body,
                description: 'Set new password for user',
                schema: {
                  type: :object,
                  properties: {
                    change_password_token: { type: :string },
                    password: { type: :string },
                    password_confirmation: { type: :string }
                  },
                  required: %w[change_password_token password
password_confirmation]
                }

      parameter name: :user,
                in: :body,
                description: 'Set new password for user',
                schema: {
                  type: :object,
                  properties: {
                    email: { type: :string },
                    description: { type: :string }
                  }
                }

      response '201', 'OK' do
        schema type: :object,
               properties: {
                 id: { type: :integer },
                 email: { type: :string },
                 role: { type: :string },
```

```
                profile: {
                  type: :object,
                  properties: {
                    id: { type: :integer },
                    first_name: { type: :string },
                    last_name: { type: :string },
                    photo_url: { type: :string },
                    contact_number: { type: :string },
                    status: { type: :string },
                    education: { type: :string },
                    experience: { type: :string },
                  }
                },
              }
      run_test!
    end

    response '404', 'Not Found' do
      run_test!
    end

    response '422', 'Unprocessable entity' do
      run_test!
    end
  end
end

path '/api/users/' do
  get 'Show info about user' do
    tags 'Users'
    description 'Delete a user'
    produces 'application/json'

    parameter name: :id,
              in: :path,
              type: :integer,
              required: true,
              description: 'User identification'

    response '201', 'OK' do
      schema type: :object,
             properties: {
               id: { type: :integer },
               email: { type: :string },
               role: { type: :string },
               profile: {
                 type: :object,
                 properties: {
                   id: { type: :integer },
                   first_name: { type: :string },
                   last_name: { type: :string },
                   photo_url: { type: :string },
                   contact_number: { type: :string },
                   status: { type: :string },
                   education: { type: :string },
                   experience: { type: :string },
                 }
               },
             }
      run_test!
    end
```

```
      response '404', 'Not Found' do
        run_test!
      end

      response '422', 'Unprocessable entity' do
        run_test!
      end
    end
  end
end
```

ДОДАТОК В

Інтеграційний тест – reports_spec.rb

```ruby
require 'swagger_helper'

describe 'Reports' do
  path '/api/reports' do
    get 'List reports' do
      tags 'Reports'
      description 'List all reports'
      produces 'application/json'

      response '200', 'OK' do
        schema type: :array,
               items: {
                 type: :object,
                 properties: {
                   id: { type: :integer },
                   user_id: { type: :integer },
                   title: { type: :string },
                   content: { type: :string },
                   generated_date: { type: :string },
                 },
               }
        run_test!
      end
    end
  end

  path '/api/reports' do
    post 'Create a report' do
      tags 'Reports'
      consumes 'application/json'
      produces 'application/json'

      parameter name: :user,
                in: :body,
                description: 'Create a report',
                schema: {
                  type: :object,
                  properties: {
                    user_id: { type: :integer },
                    title: { type: :string },
                    content: { type: :string },
                    generated_date: { type: :string },
                  },
                }

      response '201', 'OK' do
        schema type: :object,
               properties: {
                 id: { type: :integer },
                 user_id: { type: :integer },
                 title: { type: :string },
                 content: { type: :string },
                 generated_date: { type: :string },
               }
        run_test!
      end
```

```
    response '422', 'Unprocessable entity' do
      run_test!
    end
  end
end

path '/api/reports/{id}' do
  get 'Show a report' do
    tags 'Reports'
    description 'Show a reports'
    produces 'application/json'

    parameter name: :id,
              in: :path,
              type: :integer,
              required: true,
              description: 'Report identification'

    response '200', 'OK' do
      schema type: :object,
             properties: {
               id: { type: :integer },
               user_id: { type: :integer },
               title: { type: :string },
               content: { type: :string },
               generated_date: { type: :string },
             }
      run_test!
    end

    response '401', 'Unauthorized' do
      run_test!
    end

    response '404', 'Not Found' do
      run_test!
    end
  end
end

path '/api/reports/{id}' do
  put 'Updates a report' do
    tags 'Reports'
    description 'Updates a report'
    consumes 'application/json'
    produces 'application/json'

    parameter name: :id,
              in: :path,
              type: :integer,
              required: true,
              description: 'Report identification'

    parameter name: :user,
              in: :body,
              description: 'Updates a report',
              schema: {
                type: :object,
                properties: {
                  user_id: { type: :integer },
                  title: { type: :string },
```

```
                    content: { type: :string },
                    generated_date: { type: :string },
                  }
                }
      response '200', 'OK' do
        schema type: :object,
               properties: {
                 id: { type: :integer },
                 user_id: { type: :integer },
                 title: { type: :string },
                 content: { type: :string },
                 generated_date: { type: :string },
               }
        run_test!
      end

      response '404', 'Not Found' do
        run_test!
      end

      response '422', 'Unprocessable entity' do
        run_test!
      end
    end
  end

  path '/api/reports/{id}' do
    delete 'Delete a report' do
      tags 'Reports'
      description 'Delete a report'
      produces 'application/json'

      parameter name: :id,
                in: :path,
                type: :integer,
                required: true,
                description: 'Report identification'

      response '204', 'OK' do
        run_test!
      end

      response '404', 'Not Found' do
        run_test!
      end
    end
  end
end
```

ДОДАТОК Г

Інтеграціний тест – projects_spec.rb

```ruby
require 'swagger_helper'

describe 'Projects' do
  path '/api/projects' do
    get 'List projects' do
      tags 'Projects'
      description 'List all projects'
      produces 'application/json'

      response '200', 'OK' do
        schema type: :array,
               items: {
                 type: :object,
                 properties: {
                   id: { type: :integer },
                   name: { type: :string },
                   description: { type: :string },
                   start_date: { type: :string },
                   end_date: { type: :string },
                   status: { type: :string },
                 },
               }
        run_test!
      end
    end
  end

  path '/api/projects' do
    post 'Create a projects' do
      tags 'Projects'
      consumes 'application/json'
      produces 'application/json'

      parameter name: :user,
                in: :body,
                description: 'Create a projects',
                schema: {
                  type: :object,
                  properties: {
                    name: { type: :string },
                    description: { type: :string },
                    start_date: { type: :string },
                    end_date: { type: :string },
                    status: { type: :string },
                  },
                }

      response '201', 'OK' do
        schema type: :object,
               properties: {
                 id: { type: :integer },
                 name: { type: :string },
                 description: { type: :string },
                 start_date: { type: :string },
                 end_date: { type: :string },
                 status: { type: :string },
```

```
            }
      run_test!
    end

    response '422', 'Unprocessable entity' do
      run_test!
    end
  end
end

path '/api/projects/{id}' do
  get 'Show a project' do
    tags 'Projects'
    description 'Show a project'
    produces 'application/json'

    parameter name: :id,
              in: :path,
              type: :integer,
              required: true,
              description: 'Project identification'

    response '200', 'OK' do
      schema type: :object,
             properties: {
               id: { type: :integer },
               name: { type: :string },
               description: { type: :string },
               start_date: { type: :string },
               end_date: { type: :string },
               status: { type: :string },
             }
      run_test!
    end

    response '401', 'Unauthorized' do
      run_test!
    end

    response '404', 'Not Found' do
      run_test!
    end
  end
end

path '/api/projects/{id}' do
  put 'Updates a project' do
    tags 'Projects'
    description 'Updates a project'
    consumes 'application/json'
    produces 'application/json'

    parameter name: :id,
              in: :path,
              type: :integer,
              required: true,
              description: 'Project identification'

    parameter name: :user,
              in: :body,
              description: 'Updates a project',
              schema: {
```

```
                type: :object,
                properties: {
                  name: { type: :string },
                  description: { type: :string },
                  start_date: { type: :string },
                  end_date: { type: :string },
                  status: { type: :string },
                }
              }
    response '200', 'OK' do
      schema type: :object,
             properties: {
               id: { type: :integer },
               name: { type: :string },
               description: { type: :string },
               start_date: { type: :string },
               end_date: { type: :string },
               status: { type: :string },
             }
      run_test!
    end

    response '404', 'Not Found' do
      run_test!
    end

    response '422', 'Unprocessable entity' do
      run_test!
    end
  end
end

path '/api/projects/{id}' do
  delete 'Delete a project' do
    tags 'Projects'
    description 'Delete a project'
    produces 'application/json'

    parameter name: :id,
              in: :path,
              type: :integer,
              required: true,
              description: 'Project identification'

    response '204', 'OK' do
      run_test!
    end

    response '404', 'Not Found' do
      run_test!
    end
  end
end
end
```

ДОДАТОК Ґ

Інтеграціний тест – assignments_spec.rb

```ruby
require 'swagger_helper'

describe 'Assignments' do
  path '/api/assignments' do
    get 'List assignments' do
      tags 'Assignments'
      description 'List all project assignments'
      produces 'application/json'

      response '200', 'OK' do
        schema type: :array,
               items: {
                 type: :object,
                 properties: {
                   id: { type: :integer },
                   user_id: { type: :integer },
                   project_id: { type: :integer },
                 },
               }
        run_test!
      end
    end
  end

  path '/api/assignments' do
    post 'Create a assignment' do
      tags 'Assignments'
      consumes 'application/json'
      produces 'application/json'

      parameter name: :user,
                in: :body,
                description: 'Create a assignment',
                schema: {
                  type: :object,
                  properties: {
                    user_id: { type: :integer },
                    project_id: { type: :integer }
                  }
                }

      response '201', 'OK' do
        schema type: :object,
               properties: {
                 id: { type: :integer },
                 user_id: { type: :integer },
                 project_id: { type: :integer },
               }
        run_test!
      end

      response '422', 'Unprocessable entity' do
        run_test!
      end
    end
  end
```

```
path '/api/assignments/{id}' do
  get 'Show a assignment' do
    tags 'Assignments'
    description 'Show a assignment'
    produces 'application/json'

    parameter name: :id,
              in: :path,
              type: :integer,
              required: true,
              description: 'Assignment identification'

    response '200', 'OK' do
      schema type: :object,
             properties: {
               id: { type: :integer },
               user_id: { type: :integer },
               project_id: { type: :integer },
             }
      run_test!
    end

    response '401', 'Unauthorized' do
      run_test!
    end

    response '404', 'Not Found' do
      run_test!
    end
  end
end

path '/api/assignments/{id}' do
  put 'Updates a assignment' do
    tags 'Assignments'
    description 'Updates a assignment'
    consumes 'application/json'
    produces 'application/json'

    parameter name: :id,
              in: :path,
              type: :integer,
              required: true,
              description: 'Assignment identification'

    parameter name: :user,
              in: :body,
              description: 'Updates a assignment',
              schema: {
                type: :object,
                properties: {
                  user_id: { type: :integer },
                  project_id: { type: :integer },
                }
              }
    response '200', 'OK' do
      schema type: :object,
             properties: {
               id: { type: :integer },
               user_id: { type: :integer },
               project_id: { type: :integer },
```

```
                }
        run_test!
      end

      response '404', 'Not Found' do
        run_test!
      end

      response '422', 'Unprocessable entity' do
        run_test!
      end
    end
  end

  path '/api/assignments/{id}' do
    delete 'Delete a assignment' do
      tags 'Assignments'
      description 'Delete a assignment'
      produces 'application/json'

      parameter name: :id,
                in: :path,
                type: :integer,
                required: true,
                description: 'Assignment identification'

      response '204', 'OK' do
        run_test!
      end

      response '404', 'Not Found' do
        run_test!
      end
    end
  end
end
```

ДОДАТОК Д

Інтеграціний тест – time_entries_spec.rb

```ruby
require 'swagger_helper'

describe 'Time Entries' do
  path '/api/time_entries' do
    get 'List time entries' do
      tags 'Time Entries'
      description 'List all project time entries'
      produces 'application/json'

      response '200', 'OK' do
        schema type: :array,
               items: {
                 type: :object,
                 properties: {
                   id: { type: :integer },
                   user_id: { type: :integer },
                   project_id: { type: :integer },
                   date: { type: :string },
                   hours_spent: { type: :string },
                   description: { type: :string },
                 },
               }
        run_test!
      end
    end
  end

  path '/api/time_entries' do
    post 'Create a time entries' do
      tags 'Time Entries'
      consumes 'application/json'
      produces 'application/json'

      parameter name: :user,
                in: :body,
                description: 'Create a time entries',
                schema: {
                  type: :object,
                  properties: {
                    user_id: { type: :integer },
                    project_id: { type: :integer },
                    date: { type: :string },
                    hours_spent: { type: :string },
                    description: { type: :string },
                  }
                }

      response '201', 'OK' do
        schema type: :object,
               properties: {
                 id: { type: :integer },
                 user_id: { type: :integer },
                 project_id: { type: :integer },
                 date: { type: :string },
                 hours_spent: { type: :string },
                 description: { type: :string },
```

```
            }
      run_test!
    end

    response '422', 'Unprocessable entity' do
      run_test!
    end
  end
end

path '/api/time_entries/{id}' do
  get 'Show a time entry' do
    tags 'Time Entries'
    description 'Show a time entry'
    produces 'application/json'

    parameter name: :id,
              in: :path,
              type: :integer,
              required: true,
              description: 'Time entry identification'

    response '200', 'OK' do
      schema type: :object,
             properties: {
               id: { type: :integer },
               user_id: { type: :integer },
               project_id: { type: :integer },
               date: { type: :string },
               hours_spent: { type: :string },
               description: { type: :string },
             }
      run_test!
    end

    response '401', 'Unauthorized' do
      run_test!
    end

    response '404', 'Not Found' do
      run_test!
    end
  end
end

path '/api/time_entries/{id}' do
  put 'Updates a time entry' do
    tags 'Time Entries'
    description 'Updates a time entry'
    consumes 'application/json'
    produces 'application/json'

    parameter name: :id,
              in: :path,
              type: :integer,
              required: true,
              description: 'Time entry identification'

    parameter name: :user,
              in: :body,
              description: 'Updates a time entry',
              schema: {
```

```
                type: :object,
                properties: {
                  user_id: { type: :integer },
                  project_id: { type: :integer },
                  date: { type: :string },
                  hours_spent: { type: :string },
                  description: { type: :string },
                }
              }
    response '200', 'OK' do
      schema type: :object,
             properties: {
               id: { type: :integer },
               user_id: { type: :integer },
               project_id: { type: :integer },
               date: { type: :string },
               hours_spent: { type: :string },
               description: { type: :string },
             }
      run_test!
    end

    response '404', 'Not Found' do
      run_test!
    end

    response '422', 'Unprocessable entity' do
      run_test!
    end
  end
end

path '/api/time_entries/{id}' do
  delete 'Delete a time entry' do
    tags 'Time Entries'
    description 'Delete a time entry'
    produces 'application/json'

    parameter name: :id,
              in: :path,
              type: :integer,
              required: true,
              description: 'Time entry identification'

    response '204', 'OK' do
      run_test!
    end

    response '404', 'Not Found' do
      run_test!
    end
  end
end
end
```

ДОДАТОК Е

Сгенерованна документація до API

```
---
openapi: 3.0.1
info:
  title: API V1
  version: v1
paths:
  "/api/assignments":
    get:
      summary: List assignments
      tags:
      - Assignments
      description: List all project assignments
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  properties:
                    id:
                      type: integer
                    user_id:
                      type: integer
                    project_id:
                      type: integer
    post:
      summary: Create a assignment
      tags:
      - Assignments
      parameters: []
      responses:
        '201':
          description: OK
          content:
            application/json:
              schema:
                type: object
                properties:
                  id:
                    type: integer
                  user_id:
                    type: integer
                  project_id:
                    type: integer
        '422':
          description: Unprocessable entity
      requestBody:
        content:
          application/json:
            schema:
              type: object
              properties:
                user_id:
```

```
                type: integer
              project_id:
                type: integer
      description: Create a assignment
"/api/assignments/{id}":
  get:
    summary: Show a assignment
    tags:
    - Assignments
    description: Show a assignment
    parameters:
    - name: id
      in: path
      required: true
      description: Assignment identification
      schema:
        type: integer
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: integer
                user_id:
                  type: integer
                project_id:
                  type: integer
      '401':
        description: Unauthorized
      '404':
        description: Not Found
  put:
    summary: Updates a assignment
    tags:
    - Assignments
    description: Updates a assignment
    parameters:
    - name: id
      in: path
      required: true
      description: Assignment identification
      schema:
        type: integer
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: integer
                user_id:
                  type: integer
                project_id:
                  type: integer
      '404':
```

```
        description: Not Found
      '422':
        description: Unprocessable entity
    requestBody:
      content:
        application/json:
          schema:
            type: object
            properties:
              user_id:
                type: integer
              project_id:
                type: integer
      description: Updates a assignment
  delete:
    summary: Delete a assignment
    tags:
    - Assignments
    description: Delete a assignment
    parameters:
    - name: id
      in: path
      required: true
      description: Assignment identification
      schema:
        type: integer
    responses:
      '204':
        description: OK
      '404':
        description: Not Found
"/api/projects":
  get:
    summary: List projects
    tags:
    - Projects
    description: List all projects
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  id:
                    type: integer
                  name:
                    type: string
                  description:
                    type: string
                  start_date:
                    type: string
                  end_date:
                    type: string
                  status:
                    type: string
  post:
    summary: Create a projects
    tags:
```

```
    - Projects
    parameters: []
    responses:
      '201':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: integer
                name:
                  type: string
                description:
                  type: string
                start_date:
                  type: string
                end_date:
                  type: string
                status:
                  type: string
      '422':
        description: Unprocessable entity
    requestBody:
      content:
        application/json:
          schema:
            type: object
            properties:
              name:
                type: string
              description:
                type: string
              start_date:
                type: string
              end_date:
                type: string
              status:
                type: string
      description: Create a projects
"/api/projects/{id}":
  get:
    summary: Show a project
    tags:
    - Projects
    description: Show a project
    parameters:
    - name: id
      in: path
      required: true
      description: Project identification
      schema:
        type: integer
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
```

```
                id:
                  type: integer
                name:
                  type: string
                description:
                  type: string
                start_date:
                  type: string
                end_date:
                  type: string
                status:
                  type: string
      '401':
        description: Unauthorized
      '404':
        description: Not Found
  put:
    summary: Updates a project
    tags:
    - Projects
    description: Updates a project
    parameters:
    - name: id
      in: path
      required: true
      description: Project identification
      schema:
        type: integer
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: integer
                name:
                  type: string
                description:
                  type: string
                start_date:
                  type: string
                end_date:
                  type: string
                status:
                  type: string
      '404':
        description: Not Found
      '422':
        description: Unprocessable entity
    requestBody:
      content:
        application/json:
          schema:
            type: object
            properties:
              name:
                type: string
              description:
                type: string
```

```
              start_date:
                type: string
              end_date:
                type: string
              status:
                type: string
      description: Updates a project
  delete:
    summary: Delete a project
    tags:
    - Projects
    description: Delete a project
    parameters:
    - name: id
      in: path
      required: true
      description: Project identification
      schema:
        type: integer
    responses:
      '204':
        description: OK
      '404':
        description: Not Found
"/api/reports":
  get:
    summary: List reports
    tags:
    - Reports
    description: List all reports
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  id:
                    type: integer
                  user_id:
                    type: integer
                  title:
                    type: string
                  content:
                    type: string
                  generated_date:
                    type: string
  post:
    summary: Create a report
    tags:
    - Reports
    parameters: []
    responses:
      '201':
        description: OK
        content:
          application/json:
            schema:
              type: object
```

```
              properties:
                id:
                  type: integer
                user_id:
                  type: integer
                title:
                  type: string
                content:
                  type: string
                generated_date:
                  type: string
      '422':
        description: Unprocessable entity
    requestBody:
      content:
        application/json:
          schema:
            type: object
            properties:
              user_id:
                type: integer
              title:
                type: string
              content:
                type: string
              generated_date:
                type: string
      description: Create a report
"/api/reports/{id}":
  get:
    summary: Show a report
    tags:
    - Reports
    description: Show a reports
    parameters:
    - name: id
      in: path
      required: true
      description: Report identification
      schema:
        type: integer
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: integer
                user_id:
                  type: integer
                title:
                  type: string
                content:
                  type: string
                generated_date:
                  type: string
      '401':
        description: Unauthorized
      '404':
```

```
        description: Not Found
    put:
      summary: Updates a report
      tags:
      - Reports
      description: Updates a report
      parameters:
      - name: id
        in: path
        required: true
        description: Report identification
        schema:
          type: integer
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                type: object
                properties:
                  id:
                    type: integer
                  user_id:
                    type: integer
                  title:
                    type: string
                  content:
                    type: string
                  generated_date:
                    type: string
        '404':
          description: Not Found
        '422':
          description: Unprocessable entity
      requestBody:
        content:
          application/json:
            schema:
              type: object
              properties:
                user_id:
                  type: integer
                title:
                  type: string
                content:
                  type: string
                generated_date:
                  type: string
        description: Updates a report
    delete:
      summary: Delete a report
      tags:
      - Reports
      description: Delete a report
      parameters:
      - name: id
        in: path
        required: true
        description: Report identification
        schema:
          type: integer
```

```
    responses:
      '204':
        description: OK
      '404':
        description: Not Found
"/api/time_entries":
  get:
    summary: List time entries
    tags:
    - Time Entries
    description: List all project time entries
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: array
              items:
                type: object
                properties:
                  id:
                    type: integer
                  user_id:
                    type: integer
                  project_id:
                    type: integer
                  date:
                    type: string
                  hours_spent:
                    type: string
                  description:
                    type: string
  post:
    summary: Create a time entries
    tags:
    - Time Entries
    parameters: []
    responses:
      '201':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: integer
                user_id:
                  type: integer
                project_id:
                  type: integer
                date:
                  type: string
                hours_spent:
                  type: string
                description:
                  type: string
      '422':
        description: Unprocessable entity
    requestBody:
      content:
```

```
        application/json:
          schema:
            type: object
            properties:
              user_id:
                type: integer
              project_id:
                type: integer
              date:
                type: string
              hours_spent:
                type: string
              description:
                type: string
      description: Create a time entries
"/api/time_entries/{id}":
  get:
    summary: Show a time entry
    tags:
    - Time Entries
    description: Show a time entry
    parameters:
    - name: id
      in: path
      required: true
      description: Time entry identification
      schema:
        type: integer
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: integer
                user_id:
                  type: integer
                project_id:
                  type: integer
                date:
                  type: string
                hours_spent:
                  type: string
                description:
                  type: string
      '401':
        description: Unauthorized
      '404':
        description: Not Found
  put:
    summary: Updates a time entry
    tags:
    - Time Entries
    description: Updates a time entry
    parameters:
    - name: id
      in: path
      required: true
      description: Time entry identification
```

```
      schema:
        type: integer
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: integer
                user_id:
                  type: integer
                project_id:
                  type: integer
                date:
                  type: string
                hours_spent:
                  type: string
                description:
                  type: string
      '404':
        description: Not Found
      '422':
        description: Unprocessable entity
    requestBody:
      content:
        application/json:
          schema:
            type: object
            properties:
              user_id:
                type: integer
              project_id:
                type: integer
              date:
                type: string
              hours_spent:
                type: string
              description:
                type: string
      description: Updates a time entry
  delete:
    summary: Delete a time entry
    tags:
    - Time Entries
    description: Delete a time entry
    parameters:
    - name: id
      in: path
      required: true
      description: Time entry identification
      schema:
        type: integer
    responses:
      '204':
        description: OK
      '404':
        description: Not Found
"/api/users/sign_up":
  post:
```

```
    summary: Create a user
    tags:
    - Users
    parameters: []
    responses:
      '201':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: integer
                email:
                  type: string
                role:
                  type: string
                profile:
                  type: object
                  properties:
                    id:
                      type: integer
                    first_name:
                      type: string
                    last_name:
                      type: string
                    photo_url:
                      type: string
                    contact_number:
                      type: string
                    status:
                      type: string
                    education:
                      type: string
                    experience:
                      type: string
      '401':
        description: Failed
        content:
          application/json:
            schema:
              type: object
              properties:
                error:
                  type: string
    requestBody:
      content:
        application/json:
          schema:
            type: object
            properties:
              email:
                type: string
              password:
                type: string
              password_confirmation:
                type: string
            required:
            - email
            - password
            - password_confirmation
```

```
      description: Create a user
"/api/users/sign_in":
  post:
    summary: Sign in as user
    tags:
    - Users
    parameters: []
    responses:
      '201':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: integer
                email:
                  type: string
                role:
                  type: string
                profile:
                  type: object
                  properties:
                    id:
                      type: integer
                    first_name:
                      type: string
                    last_name:
                      type: string
                    photo_url:
                      type: string
                    contact_number:
                      type: string
                    status:
                      type: string
                    education:
                      type: string
                    experience:
                      type: string
      '401':
        description: Failed
        content:
          application/json:
            schema:
              type: object
              properties:
                error:
                  type: string
    requestBody:
      content:
        application/json:
          schema:
            type: object
            properties:
              email:
                type: string
              password:
                type: string
            required:
            - email
            - password
```

```
      description: Sign in as user
"/api/users/recoveries":
  post:
    summary: Recovery password for user
    tags:
    - Users
    parameters: []
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                status:
                  type: string
                message:
                  type: string
      '422':
        description: Not Found
        content:
          application/json:
            schema:
              type: object
              properties:
                error:
                  type: string
    requestBody:
      content:
        application/json:
          schema:
            type: object
            properties:
              email:
                type: string
              captcha_token:
                type: string
            required:
            - email
            - captcha_token
      description: Recovery password for user
"/api/users/passwords":
  post:
    summary: Set new password for user
    tags:
    - Users
    parameters: []
    responses:
      '201':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: integer
                email:
                  type: string
                role:
                  type: string
```

```
                profile:
                  type: object
                  properties:
                    id:
                      type: integer
                    first_name:
                      type: string
                    last_name:
                      type: string
                    photo_url:
                      type: string
                    contact_number:
                      type: string
                    status:
                      type: string
                    education:
                      type: string
                    experience:
                      type: string
      '404':
        description: Not Found
      '422':
        description: Unprocessable entity
    requestBody:
      content:
        application/json:
          schema:
            type: object
            properties:
              change_password_token:
                type: string
              password:
                type: string
              password_confirmation:
                type: string
            required:
            - change_password_token
            - password
            - password_confirmation
      description: Set new password for user
"/api/users/":
  get:
    summary: Show info about user
    tags:
    - Users
    description: Delete a user
    parameters:
    - name: id
      in: path
      required: true
      description: User identification
      schema:
        type: integer
    responses:
      '201':
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
```

```
                  type: integer
                email:
                  type: string
                role:
                  type: string
                profile:
                  type: object
                  properties:
                    id:
                      type: integer
                    first_name:
                      type: string
                    last_name:
                      type: string
                    photo_url:
                      type: string
                    contact_number:
                      type: string
                    status:
                      type: string
                    education:
                      type: string
                    experience:
                      type: string
        '404':
          description: Not Found
        '422':
          description: Unprocessable entity
servers:
- url: https://{defaultHost}
  variables:
    defaultHost:
      default: www.example.com
```

ДОДАТОК Є

Файл маршрутів – routes.rb

```ruby
# frozen_string_literal: true

Rails.application.routes.draw do
  mount Rswag::Api::Engine => '/api-docs'
  mount Rswag::Ui::Engine => '/api-docs'
  namespace :api do
    devise_for :users,
               path_names: { registration: 'sign_up', sign_in: 'sign_in',
sign_out: 'sign_out' },
               controllers: { sessions: 'api/users/sessions', registrations:
'api/users/registrations' }

    namespace :users do
      resources :recoveries, only: %i[create]
      resources :passwords, only: %i[create]
    end
    resource :users, only: %i[show update]
    resources :projects
    resources :reports
    resources :time_entries
    resources :assignments
  end
end
```

ДОДАТОК Ж

Контролери, серіалайзери, моделі та сервіси

```ruby
# frozen_string_literal: true

module Api
  module Users
    class PasswordsController < ApiController
      def create
        return render_bad_captcha unless verify_captcha

        recovery_user = find_recovery_user
        return render_invalid_token unless recovery_user

        return render_expired_token if
expired_token?(recovery_user.reset_password_sent_at)

        recovery_user.update(password_params.except(:reset_password_token,
:captcha_token))
        head :no_content
      end

      private

      def render_bad_captcha
        render json: { error: 'Invalid captcha. Try reload page' }, status:
:not_found
      end

      def verify_captcha
        ::Recaptcha::VerifyService.call(password_params[:captcha_token],
'change_password')
      end

      def find_recovery_user
        User.find_by(reset_password_token:
password_params[:reset_password_token])
      end

      def render_invalid_token
        render json: { error: 'Invalid recovery token' }
      end

      def render_expired_token
        render json: { error: 'Recovery token has been expired' }
      end

      def expired_token?(token)
        return false if token.nil?

        expiration_threshold = 1.hour
        Time.zone.now > token + expiration_threshold
      end

      def password_params
        params.permit(:reset_password_token, :password, :password_confirmation,
:captcha_token)
      end
    end
```

```
  end
end
# frozen_string_literal: true

module Api
  module Users
    class RecoveriesController < ApiController
      def create
        unless ::Recaptcha::VerifyService.call(recovery_params[:captcha_token],
'recovery')
          return render json: { error: 'Invalid captcha. Try reload page' },
status: :not_found
        end

        @recovery_user = User.find_by(email: recovery_params[:email])
        return head :no_content unless @recovery_user

        @recovery_user.send_reset_password_instructions
        head :no_content
      end

      private

      def recovery_params
        params.permit(:captcha_token, :email)
      end
    end
  end
end
# frozen_string_literal: true

module Api
  module Users
    class RegistrationsController < Devise::RegistrationsController
      include EasyResponse

      def create
        unless ::Recaptcha::VerifyService.call(params[:captcha_token],
'register')
          return render json: { error: 'Invalid captcha. Try reload page' },
status: :not_found
        end

        super do |resource|
          unless exist?(resource)
            return render json: { error: 'Oops! It seems like you already have
an account with us' },
                          status: :conflict
          end

          update_referrer(resource) if resource.persisted? &&
params[:referral_token].present?

          create_subscription(resource)
          create_referrer_balance(resource)

          begin
            UserMailer.welcome_email(resource).deliver!
          rescue StandardError => e
            Rails.logger.error "An error occurred while sending welcome email:
#{e.message}"
          end
```

```
        end
      end

      private

      def exist?(user)
        user.id.present?
      end

      def update_referrer(user)
        referrer = User.find_by(referral_token: params[:referral_token])
        user.update(referrer: referrer) if referrer
      end

      def create_subscription(user)
        Subscription.create(user: user, subscription_plan:
SubscriptionPlan.find_by(code: 'free'))
      end

      def create_referrer_balance(user)
        ReferrerBalance.create(user: user)
      end

      def respond_with(resource, _opts = {})
        if resource.persisted?
          render_data(UserSerializer.new(resource), status: :ok)
        else
          render_validation_errors(resource)
        end
      end
    end
  end
end
# frozen_string_literal: true

class Api::Users::SessionsController < Devise::SessionsController
  include EasyResponse

  def create
    unless ::Recaptcha::VerifyService.call(params[:captcha_token], 'login')
      return render json: { error: 'Invalid captcha. Try reload page' }, status:
:not_found
    end

    super
  end

  private

  def respond_with(resource, _opts = {})
    return render_data(UserSerializer.new(resource), status: :ok) if
resource.persisted?

    render_validation_errors(resource)
  end
end
# frozen_string_literal: true

module Api
  class AssignmentsController < ApiController
    before_action :authenticate_api_user!
    before_action :set_assignments, only: %i[show update destroy]
```

```
    def index
      @assignments = resource_class.all
      render json: @assignments
    end

    def show
      render json: @assignment
    end

    def create
      @assignment = resource_class.new(assignments_params)

      if @assignment.save
        render json: @assignment, status: :created
      else
        render json: @assignment.errors, status: :unprocessable_entity
      end
    end

    def update
      if @assignment.update(assignments_params)
        render json: @assignment
      else
        render json: @assignment.errors, status: :unprocessable_entity
      end
    end

    def destroy
      @assignment.destroy
    end

    private

    def resource_class
      Assignment
    end

    def custom_serializer_class
      AssignmentsSerializer
    end

    def assignments_params
      params.permit(
        :id, :user_id, :project_id, :role, :assigned_date, :release_date
      )
    end

    def set_assignments
      @assignment = resource_class.find(params[:id])
    end
  end
end
# frozen_string_literal: true

module Api
  class ProjectsController < ApiController
    before_action :authenticate_api_user!
    before_action :set_projects, only: %i[show update destroy]

    def index
      @projects = resource_class.all
```

```
      render json: @projects
    end

    def show
      render json: @project
    end

    def create
      @project = resource_class.new(project_params)

      if @project.save
        render json: @project, status: :created
      else
        render json: @project.errors, status: :unprocessable_entity
      end
    end

    def update
      if @project.update(project_params)
        render json: @project
      else
        render json: @project.errors, status: :unprocessable_entity
      end
    end

    def destroy
      @project.destroy
    end

    private

    def resource_class
      Project
    end

    def custom_serializer_class
      ProjectSerializer
    end

    def project_params
      params.permit(
        :id, :name, :description, :start_date, :end_date, :status
      )
    end

    def set_projects
      @project = resource_class.find(params[:id])
    end
  end
end
# frozen_string_literal: true

module Api
  class ReportsController < ApiController
    before_action :authenticate_api_user!
    before_action :set_report, only: %i[show update destroy]

    def index
      @reports = resource_class.all
      render json: @reports
    end
```

```
    def show
      render json: @report
    end

    def create
      @report = resource_class.new(report_params)

      if @report.save
        render json: @report, status: :created
      else
        render json: @report.errors, status: :unprocessable_entity
      end
    end

    def update
      if @report.update(report_params)
        render json: @report
      else
        render json: @report.errors, status: :unprocessable_entity
      end
    end

    def destroy
      @report.destroy
    end

    private

    def resource_class
      Report
    end

    def entries
      resource_class.includes(associations_to_include)
                    .where(user: current_api_user)
                    .search(params[:search])
                    .order(order_clause)
    end

    def custom_serializer_class
      ReportSerializer
    end

    def associations_to_include
      %i[user]
    end

    def report_params
      params.permit(
        :id, :title, :content, :generated_date, :user_id
      )
    end

    def set_report
      @report = resource_class.find(params[:id])
    end
  end
end
# frozen_string_literal: true

module Api
  class TimeEntriesController < ApiController
```

```
    before_action :authenticate_api_user!
    before_action :set_time_entries, only: %i[show update destroy]

    def index
      @time_entries = resource_class.all
      render json: @time_entries
    end

    def show
      render json: @time_entry
    end

    def create
      @time_entry = resource_class.new(time_entries_params)

      if @time_entry.save
        render json: @time_entry, status: :created
      else
        render json: @time_entry.errors, status: :unprocessable_entity
      end
    end

    def update
      if @time_entry.update(time_entries_params)
        render json: @time_entry
      else
        render json: @time_entry.errors, status: :unprocessable_entity
      end
    end

    def destroy
      @time_entry.destroy
    end

    private

    def resource_class
      TimeEntry
    end

    def custom_serializer_class
      TimeEntrySerializer
    end

    def time_entries_params
      params.permit(
        :id, :user_id, :project_id, :role, :assigned_date, :release_date
      )
    end

    def set_time_entries
      @time_entry = resource_class.find(params[:id])
    end
  end
end
# frozen_string_literal: true

module Api
  class UsersController < ApiController
    before_action :authenticate_api_user!

    def show
```

```
      render json: current_api_user, status: :ok
    end

    def update
      current_api_user
        .update(user_params)

      render json: current_api_user, status: :ok
    end

    private

    def user_params
      params.permit(
        :first_name,
        :last_name,
        :language
      )
    end
  end
end
# frozen_string_literal: true

class ApiController < ActionController::API
  include Pundit::Authorization
  include Pagy::Backend

  rescue_from Pundit::NotAuthorizedError, with: :user_not_authorized
  def index
    pagination, records = pagy(entries, items: per_page, page: page)

    render json: { records: serialized_records(records), per_page: per_page,
                   count: pagination.count }
  end

  private

  def order_clause
    column = params[:column].presence || 'created_at'
    direction = params[:direction].presence || 'desc'
    "#{column} #{direction}"
  end

  def entries
    resource_class.includes(associations_to_include)
                  .search(params[:search]).order(order_clause)
  end

  def resource_class
    raise NotImplementedError, 'Resource class not implemented in
ApplicationController'
  end

  def associations_to_include
    []
  end

  def custom_serializer_class
    nil
  end

  def serialized_records(records)
```

```
    serializer_class = custom_serializer_class
    if serializer_class
      ActiveModelSerializers::SerializableResource.new(records,
                                                       each_serializer:
serializer_class)
    else
      records
    end
  end

  def page
    params[:page] || 1
  end

  def per_page
    params[:page_size] || 10
  end

  def user_not_authorized
    render json: { error: 'You are not authorized to perform this action.' },
status: :forbidden
  end

  def pundit_user
    current_api_user
  end
end

# frozen_string_literal: true

class ApplicationController < ActionController::Base
  respond_to :json, :html

  protect_from_forgery with: :null_session
  before_action :configure_permitted_parameters, if: :devise_controller?

  protected

  def configure_permitted_parameters
    devise_parameter_sanitizer.permit(:sign_up) do |user|
      user.permit(:email, :password, :password_confirmation)
    end
    devise_parameter_sanitizer.permit(:sign_in) do |u|
      u.permit(:otp_attempt, :email, :password, :authenticity_token)
    end
  end
end
```

```
# frozen_string_literal: true

class Assignment < ApplicationRecord
  enum :role, %i[developer quality_assurance project_manager bisnes_analitic
designer product_owner ceo hr],
       prefix: true, scopes: false

  belongs_to :project
  belongs_to :user
end
```

```ruby
# frozen_string_literal: true

class Profile < ApplicationRecord
  belongs_to :user
end
```

```ruby
# frozen_string_literal: true

class Project < ApplicationRecord
  validates :name, presence: true
  validates :description, presence: true
  validates :start_date, presence: false
  validates :end_date, presence: false

  has_many :time_entries, :dependent => :destroy
  has_many :assignments, :dependent => :destroy
end
```

```ruby
# frozen_string_literal: true

class Report < ApplicationRecord
  belongs_to :user
end
```

```ruby
# frozen_string_literal: true

class TimeEntry < ApplicationRecord
  belongs_to :user
  belongs_to :project
end
```

```ruby
# frozen_string_literal: true

class User < ApplicationRecord
  include Devise::JWT::RevocationStrategies::JTIMatcher
  include PgSearch::Model

  devise :database_authenticatable, :registerable, :recoverable, :validatable,
         :jwt_authenticatable, jwt_revocation_strategy: self

  enum :role, %i[user project_manage employee], prefix: true, scopes: false

  has_one :profile
  has_many :assignments
  has_many :time_entries
end
```

```ruby
# frozen_string_literal: true

class AssignmentsSerializer < ActiveModel::Serializer
  attributes :id, :user_id, :project_id
end
```

```ruby
# frozen_string_literal: true

class ProfileSerializer < ActiveModel::Serializer
  attributes :id, :first_name, :last_name, :photo_url, :contact_number, :status,
:education,
             :experience
end
```

```ruby
# frozen_string_literal: true

class ProjectSerializer < ActiveModel::Serializer
  attributes :id, :name, :description, :start_date, :end_date, :status
end
```

```ruby
# frozen_string_literal: true

class ReportSerializer < ActiveModel::Serializer
  attributes :id, :user_id, :title, :content, :generated_date
end
```

```ruby
# frozen_string_literal: true

class TimeEntrySerializer < ActiveModel::Serializer
  attributes :id, :user_id, :project_id, :date, :hours_spent, :description
end
```

```ruby
# frozen_string_literal: true

class UserSerializer < ActiveModel::Serializer
  attributes :id, :email, :role

  has_one :profile, serializer: ::ProfileSerializer
end
```

```ruby
# frozen_string_literal: true

class ApplicationService
  def self.call(*args, &block)
    new(*args, &block).call
  end

  def call
    raise "No implemented method #{self.__method__}"
  end
end
```

```ruby
# frozen_string_literal: true

require 'net/http'

module Recaptcha
  class VerifyService < ApplicationService
    RECAPTCHA_MINIMUM_SCORE = 0.5

    attr_reader :token, :action

    def initialize(token, action)
      @token = token
      @action = action
    end

    def call
      response = verify_recaptcha

      success = response['success']
      score = response['score']
      recaptcha_action = response['action']

      success && score > RECAPTCHA_MINIMUM_SCORE && recaptcha_action == action
    end

    private

    def verify_recaptcha
      uri = recaptcha_api_uri
      response = Net::HTTP.get_response(uri)
      JSON.parse(response.body)
    end

    def recaptcha_api_uri
      secret_key = ENV['RECAPTCHA_SECRET']
      base_url = 'https://www.google.com/recaptcha/api/siteverify'
      URI("#{base_url}?secret=#{secret_key}&response=#{token}")
    end
  end
end
```

```ruby
# frozen_string_literal: true

class CreateUsers < ActiveRecord::Migration[7.0]
  def change
    create_table :users do |t|
      t.string :email,              null: false, default: ""
      t.string :encrypted_password, null: false, default: ""
      t.integer :role, null: false, default: 0
      t.string :jti

      t.timestamps null: false
    end

    add_index :users, :email, unique: true
  end
end
```

```ruby
# frozen_string_literal: true

class CreateProfiles < ActiveRecord::Migration[7.0]
  def change
    create_table :profiles do |t|
      t.references :user, foreign_key: { to_table: :users }
      t.string :first_name
      t.string :last_name
      t.string :photo_url
      t.string :contact_number
      t.integer :status, default: 0
      t.string :education
      t.string :experience
      t.timestamps
    end
  end
end
```

```ruby
# frozen_string_literal: true

class CreateProjects < ActiveRecord::Migration[7.0]
  def change
    create_table :projects do |t|
      t.string :name
      t.string :description
      t.datetime :start_date
      t.datetime :end_date
      t.integer :status
      t.timestamps
    end
  end
end
```

```ruby
# frozen_string_literal: true

class CreateAssignments < ActiveRecord::Migration[7.0]
  def change
    create_table :assignments do |t|
      t.references :user, foreign_key: { to_table: :users }
      t.references :project, foreign_key: { to_table: :projects }
      t.string :role
      t.datetime :assigned_date
      t.datetime :release_date
      t.timestamps
    end
  end
end
# frozen_string_literal: true

class CreateTimeEntries < ActiveRecord::Migration[7.0]
  def change
    create_table :time_entries do |t|
      t.references :user, foreign_key: { to_table: :users }
      t.references :project, foreign_key: { to_table: :projects }
      t.date :date
      t.decimal :hours_spent
      t.string :description
      t.timestamps
```

```
    end
  end
end
```

```
# frozen_string_literal: true

class CreateReports < ActiveRecord::Migration[7.0]
  def change
    create_table :reports do |t|
      t.references :user, foreign_key: { to_table: :users }
      t.string :title
      t.string :content
      t.datetime :generated_date
      t.timestamps
    end
  end
end
```