МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Секція інформаційно-комінікаційних технологій

ВИПУСКНА РОБОТА

на тему:

«REST-сервер торгівельного майданчику за допомогою Ruby on Rails»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Шутилєва О.В.**

Студента групи ІН – 62 **Махнівський Р.В.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую _____

Зав. кафедрою Довбиш А.С.

“_____” _____2020 р.

Завдання
до випускної роботи

Студента четвертого курсу, ІН – 62 спеціальності “Комп’ютерні науки” денної форми навчання Махнівського Руслана Вікторовича.

Тема: “**REST-сервер торгівельного майданчику за допомогою Ruby on Rails**”

Затверджена наказом по СумДУ

№_____ від _____2020 р.

Зміст пояснювальної записки: 1) огляд проблеми та існуючих рішень; 2) аналіз популярних фреймворків; 3) визначення переліку технологій для реалізації системи; 4) проектування REST-серверу; 5) реалізація системи за допомогою вибраних технології; 6) тестування API.

Дата видачі завдання “_____” _____2020 р.

Керівник випускної роботи _____ Шутилєва О.В.

Завдання прийняв до виконання _____ Махнівський Р.В.

РЕФЕРАТ

Записка: 97 стор., 8 рис., 1 додаток, 19 джерел.

Об’єкт дослідження – процес проектування, розробки та тестування REST-серверу за допомогою Ruby on Rails.

Мета роботи – реалізація REST-серверу торгівельного майданчику.

Результати – розроблено REST-серверу торгівельного майданчику за допомогою фреймворку Ruby on Rails. Проведено порівняння популярних технологій реалізації.

REST API, ВЕБ-ФРЕЙМВОРК, ІНТЕРНЕТ-АУКЦІОН,
СХОВИЩЕ ДАНИХ, ТЕСТУВАННЯ

ЗМІСТ

ВСТУП

Стандартний спосіб купівлі та продажу – покупець відвідує магазин, шукає потрібний товар, шукає цінник і оплачує товар. У деяких випадках покупець також може піти на переговори, що є абсолютно необов'язковим. Але саме тут запускається ідея аукціону. На аукціоні зацікавлені покупці виставляють свої пропозиції, і той, хто має максимальну ставку, забирає товар.

Концепція аукціону зросла протягом останніх років, і причиною є інтернет-аукціон. Просто тому, що це кращий і задовольняючий спосіб купівлі-продажу як для продавців, так і для покупців. Пройшли дні, коли проведення аукціону вимагає людських ресурсів або фізичної присутності. Таким чином, вам не потрібно пропускати аукціон тільки тому, що ви не можете дійти до місця проведення. Це причина, чому на живих аукціонах завжди обмежена кількість товару та обмежена кількість покупців. Тепер все можна зробити лише в один клік. На аукціонах в інтернеті все, що вам потрібно – це підключення до інтернету, і ви готові робити ставки.

У наш час аукціони трансформуються і переходять в онлайн. За допомогою онлайн-аукціонів знайти вподобану річ набагато простіше, ніж шукати її в звичайних аукціонах. На звичайних аукціонах ставки підвищуються до тих пір, поки один з учасників не запропонує найбільшу, в інтернет-аукціоні час обмежує продавець. Можна вибрати і купити потрібний товар сидячи вдома, а сам процес займає мінімум часу. Сьогодні онлайн аукціони є невід'ємною частиною електронної комерції, яка відкриває широкі можливості для тих, хто зацікавлений в успішному розвитку бізнесу. Тому в перспективі очікується тільки зростання затребуваності подібних систем.

1 ЛІТЕРАТУРНИЙ ОГЛЯД

1.1 RESTful API (REST API)

RESTful API – це інтерфейс прикладної програми (API), який використовує HTTP-запити GET, PUT, POST та DELETE даних.

API для веб-сайту – це код, який дозволяє двом програмним програмам взаємодіяти між собою. API визначає належний спосіб розробнику написати програму, яка вимагає надання послуг з операційної системи або іншої програми.

Технологія REST, як правило, віддається перевазі більш надійній технології простого протоколу доступу до об'єктів (SOAP), оскільки REST використовує меншу пропускну здатність, що робить її більш придатною для ефективного використання інтернету.

API RESTful використовує існуючі методології HTTP, визначені протоколом RFC 2616. Вони використовують GET для отримання ресурсу; PUT для зміни стану або оновлення ресурсу, який може бути об'єктом, файлом або блоком; POST для створення цього ресурсу; та DELETE, щоб видалити його.

Ключовою абстракцією інформації в REST є ресурс. Будь-яка інформація, яку можна назвати, може бути ресурсом: документом або зображенням, тимчасовою службою, колекцією інших ресурсів тощо. REST використовує ідентифікатор ресурсу для ідентифікації конкретного ресурсу, що бере участь у взаємодії між компонентами. Стан ресурсу у будь-яку конкретну часову позначку називається представленням ресурсу.

1.2 Порівняння популярних веб-фреймворків

1.2.1 Laravel

Laravel – це фреймворк, що базується на PHP, яка характеризується своїм зручним синтаксисом, здатністю розміщувати великі команди та функціоналом свого сучасного інструментарію.

Laravel слідує архітектурній схемі MVC та пропонує власну систему міграції баз даних і має надійну екосистему. Особливості:

- простий і швидкий двигун маршрутизації;
- поставляється з власною CLI;
- потужна система шаблонів (Blade);
- хороша документація.

1.2.2 CakePHP

CakePHP – однин з перших фреймворків PHP, який був випущений ще в 2005 році. З тих пір він пройшов довгий шлях і в даний час відомий як сучасна структура PHP, що дозволяє розробникам швидко розробляти додатки.

CakePHP використовує чисті конвенції MVC і дуже розширюється, що робить його чудовим вибором для створення як великих, так і малих додатків.

Особливості:

- Швидка розробка;
- Поставляється з "batteries included";
- Побудований з урахуванням безпеки;
- Для початку роботи не потрібна складна конфігурація.

1.2.3 Django

Django – це фреймворк високого рівня Python, побудована з ідеєю "batteries included". Тому менше необхідності в сторонніх плагінах, і все в Django працює разом. Однак Django створений для більшого застосування. Тому, при розробці чогось невеликого, Django може виявитися не найкращим варіантом, оскільки він може зробити невеликий проект роздутим непотрібними функціями.

Кілька прикладів великих веб-сайтів, побудованих на Django, включають: Disqus, Mozilla, National Geographic, Pinterest. Особливості:

- легко налаштовується;
- масштабований;

- велика спільнота та документація.

1.2.4 Ruby on Rails

Ruby on Rails – серверний веб-фреймворк, написаний мовою програмування Ruby, яка заохочує використання моделей дизайну, таких як MVC та DRY. Кілька прикладів великих веб-сайтів, побудованих на Rails, включають: Shopify, SoundCloud, Basecamp, GitHub. Особливості:

- доступна велика бібліотека плагінів;
- дуже чіткий синтаксис;
- велика спільнота;
- невеликі проекти легко розробити та керувати.

1.2.5 Flask

Flask – це ще один фреймворк, створений на основі Python. Однак, на відміну від Django, він легкий і більше підходить для розробки менших проектів. Flask пропонує підтримку шаблонів Jinja2, захищені файли cookie, тестування блоку та відправлення RESTful запиту, велику документацію і є прекрасним рішенням для програмістів Python, яким не потрібні всі бібліотеки, з якими надходить Django. Особливості:

- гнучкі налаштування;
- більш легкий, ніж Django, чудово підходить для менших проектів;
- структурована документація;
- пропонує можливість швидкої побудови прототипів.

1.2.6 Express

Express – це швидкий, мінімалістичний фреймворк на Node.js. Він надає основні функції веб-додатків, не приховуючи функції Node.js. Крім того, легко створити надійний API. Кілька прикладів великих веб-сайтів, що використовують Express, включають: Uber, Accenture, IBM. Особливості:

- відмінний API маршрутизації;
- мінімалістичний;
- просте налаштування;
- велика кількість плагінів, доступних для використання.

1.3 Редактори коду

Вибір редактора тексту програмування на вибір сьогодні здається непосильним через численні захоплюючі варіанти. Незважаючи на те, що велика кількість програмістів може мати загальну перевагу перед тим чи іншим редактором, рішення часто зводиться до особистих уподобань та робочого процесу. Ось декілька найпопулярніших текстових редакторів:

— RubyMine допомагає збільшити продуктивність для всіх типів Ruby-проектів та передових технологій. Це потужний IDE з інтелектуальною допомогою кодування та розширеними функціями тестування та налагодження. Її смарт-редактор коду забезпечує найкращу підтримку Ruby on Rails, JavaScript і CoffeeScript, ERB, HAML, CSS, Sass і Less та інше. Він також містить особливості мовного заповнення коду, виявлення помилок, виправлення коду та багато іншого;

— Sublime Text вже деякий час існує з випуском ще в 2007 році. Це чудовий редактор завдяки своїй гнучкості та швидкості. Однак він коштує 70 доларів за ліцензійну плату з безкоштовною пробною версією та доступний на платформах Windows, Mac та Linux;

— Atom є відносно новачком у світі текстових редакторів, але він набув величезних обертів з моменту першого випуску в 2014 році. Він чудовий для тих, хто хоче легко налаштувати свій редактор. Він також безкоштовний і доступний на платформах Windows, Mac та Linux

— Visual Studio Code може бути відносно новачком також у світі текстових редакторів з його випуском у 2015 році, але він став одним із найулюбленіших редакторів тестів завдяки численним інтеграціям та

плагінам, доступним для нього. Він також безкоштовний і доступний на платформах Windows, Mac та Linux;

— TextMate дуже схожий на Sublime Text, але з додаванням керування джерелами та інших функцій навігації. Це було вже деякий час, але він доступний лише для Mac. Це також коштує декількох доларів у розмірі близько 54 доларів ліцензійної плати та безкоштовну пробну версію;

— Vim – це дуже потужний редактор тексту програмування з високою кривою навчання, який найкраще підходить професійним або дуже досвідченим програмістам, завдяки неймовірно швидкому та ефективному середовищу розробки. Він також безкоштовний і доступний на платформах Windows, Mac та Linux.

1.4 Постановка задачі

Користувач (User) повинен мати можливість зареєструватися в системі. Система повинна надіслати на електронну пошту повідомлення про підтвердження реєстрації з посиланням. Він повинен зайти за посиланням, вказаним в електронному листі. Коли користувач перейде за цим посиланням, система закінчить процес реєстрації.

Користувач може увійти в систему за допомогою своєї електронної пошти та пароля. Система повинна підтримувати здатність запам'ятовувати користувача в системі та відновлення пароля.

На сторінці всіх лотів(Lot) користувач повинен побачити всі лоти зі статусом in_process. Якщо користувач хоче переглянути повну інформацію про лот, він повинен перейти на сторінку цього лота.

Сторінка my lots містить усі лоти, створені користувачем як продавця або в яких користувач бере участь як замовник (запропоновано будь-яку ставку). користувач повинен мати можливість фільтрувати сторінку за наступними критеріями:

— всі лоти;

— лише лоти, які користувач створює для продажу;

— лоти, в яких користувая приймає участь.

Будь-який користувач повинен мати можливість створити лот. Після створення лот матиме статус pending. Це означає, що час початку ще не настав. Незважаючи на те, що лот має цей статус, продавець може видалити або оновити його. Лот зі статусом pending не повинний бути присутнім на сторінці всіх лотів. Коли настає час запуску, система повинна змінити статус лота з pending на in_process.

Якщо користувач хоче придбати будь-який лот, який він бачить на сторінці всіх лотів, він повинен створити ставку(Bid). Продавець та клієнти повинні бачити нові ставки в таблиці детальної сторінки лоту в режимі реального часу, не оновлюючи сторінку. Клієнти та продавець не повинні бачити жодних особистих даних один одного. Користувач виграє, якщо запропонував найвищу ставку. Після цього система повинна змінити статус лота з in_process на closed. Після цього клієнти не можуть робити ставки на цей лот.

Система надсилає електронний лист переможцю, що він виграв лот і може створити замовлення(Order). Система надсилає продавцю електронну пошту та повідомляє його про те, що лот закрито, та про кінцеву ціну лота. Замовлення - це суб'єкт, який представляє, яким чином товар повинен бути доставлений від продавця до замовника, який виграв лот. Система має надіслати повідомлення продавцю з вимогою виконати замовлення. Після того, як продавець надіслав замовнику товар, система змінює статус з pending на sent. Система має надіслати електронний лист клієнту про те, що продавець відправив товар. Коли користувач отримує, він повинен змінити статус замовлення на delivered. Це останній крок, і ніхто не може змінити лот, ставки та замовлення. Повідомлення електронною поштою про доставку товару має бути надіслано продавцю.

2 МЕТОДИ ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Об'єктно-орієнтована мова програмування Ruby

Об'єктно-орієнтована – це означає, що мова використовує об'єкти в своїх процесах, які дозволяють або окремі частини програми або всієї програми в цілому, для повторного використання в інших проектах. Крім того, об'єктно-орієнтоване програмування забезпечує чітку модульну структуру проектів програміста.

Ruby має репутацію дуже зручної для інновацій – вона не тільки має безліч функцій, які можна вибрати за замовчуванням, але також легко приймає більшість нових реалізацій та оновлень.

За допомогою менеджерів пакетів або сторонніх інструментів є безліч варіантів встановлення та управління Ruby.

Параметри установки Ruby включають наступне:

— Менеджер пакунків – у такій операційній системі, як UNIX, Linux або MacOS, найпростіше використовувати менеджер пакунків системи. Однак це не рекомендується, оскільки упакована версія Ruby може бути не новітньою;

— Інсталятори – вони можуть бути використані для встановлення певної або декількох версій Ruby. Установки Ruby: ruby-build, ruby-install, RubyInstaller (для windows) RailsInstaller та Ruby Stack;

— Менеджери – такі менеджери Ruby, як chruby, rbenv, RVM та uru, можуть використовуватися для встановлення та переключення між декількома версіями Ruby у системі. Це найбільш рекомендований варіант для установки Ruby.

Розширення, які поставляються з Ruby зазвичай іменовані стандартною бібліотекою. Стандартна бібліотека включає в себе розширення для найрізноманітніших проектів і завдань: управління базами даних, мережеві роботи, спеціалізована математика, обробка XML та багато іншого. Бібліотеки в

рубіні, які містять певний фрагмент функціональності, а також будь-які файли чи ресурси, пов'язані з цим функціоналом, часто упаковуються у дорогоцінні камені.

Ключовим фактором використання розширень та бібліотек є require та load методи. Ці методи дозволяють завантажувати розширення під час виконання, включаючи розширення, які ви пишете самостійно. Основна відмінність обох методів полягає в тому, що require буде завантажувати переданий файл лише один раз, а load фактично завантажуватиме переданий файл щоразу, коли він викликається. Ось що слід зазначити, що речі, які завантажуються у програму під час виконання, викликаються особливістю, розширенням або бібліотекою.

Особливість є найбільш абстрактною і її рідко чують поза спеціалізованим використанням, "requiring a feature" (тобто за потребою). Бібліотека більш конкретна та поширена. Він конотує власне код, а також основний факт, що набір програм існує і може завантажуватися. Розширення може стосуватися будь-якої завантажуваної бібліотеки додатків, але часто використовується для позначення бібліотеки для Ruby, написаної мовою програмування на C, а не на Ruby. Якщо ви скажете людям, що ви написали розширення Ruby, вони, ймовірно, припустять, що ви маєте на увазі, що це C.

Після встановлення Ruby, отримуємо кілька важливих інструментів командного рядка, які встановлені в будь-якій папці, налаштованій як bindir.

Це інструменти:

- ruby – Перекладач;
- irb – Інтерактивний перекладач Ruby;
- rake – Ruby make, утиліта управління завданнями;
- rdoc – Інструменти документації Ruby;
- erb – Шаблонна система.

Інтерпретатор Ruby – це програма, яка здатна інтерпретувати вихідний код, написаний мовою Ruby. Під час запуску програми Ruby інтерпретатор переводить код у набір інструкцій, який може працювати у віртуальній машині Ruby (VM).

Інтерактивний Ruby (irb) – це інструмент для інтерактивного виконання рубінових виразів, прочитаних з командного терміналу без створення файлу. Щоб

його використовувати, ви запускаєте irb виконуваний файл і вводите свій код Ruby під запитом, irb оцінює код, який ви вводите, і відображає результати.

Rake – це програмне забезпечення для управління завданнями та створення засобів автоматизації, написане мовою програмування Ruby. Це утиліта управління завданнями, яка дозволяє задавати завдання та виконувати завдання, визначені у файлі під назвою Rakefile.

rdoc – це вбудований генератор документації для мови програмування Ruby, він автоматично генерує документацію з коментарів та структури вашого коду. Хоча ri – версія рубінових сторінок man, що подає інформацію API з командного рядка, це довідки ruby в автономному режимі, які подаються через командний рядок.

erb – потужна шаблонна система для Ruby, яка дозволяє додавати фактичний код Ruby до будь-якої простої текстової інформації або HTML-коду. Це допомагає генерувати динамічні сторінки, що живляться з даних у вашої бази даних.

2.2 Ruby on Rails

Ruby on Rails, також відомий як RoR, або просто «Rails» – це структура з відкритим кодом, побудована на мові Ruby, яка може використовуватися з декількома іншими мовами, такими як XML та JavaScript. Фреймворк був випущений в грудні 2005 року. Хоча в галузі існує безліч інших досить популярних і широко використовуваних технологій, ROR залишається популярним серед розробників у всіх областях.

Це фреймворк з відкритим кодом із структурою об'єктно-орієнтованого програмування (ООР), відомий як ефективний інструмент веб-розробки. Він має багатошарову функціональну структуру з компонентами Model, View та Controller, схожими на компоненти багатьох інших веб-структур. Він добре відомий своєю швидкістю в процесі розробки. Крім того, чудові показники подачі можна досягти, використовуючи Ruby on Rails.

Rails ідеально підходить як для експертів, так і для новачків. Він оснащує розробників набором унікальних функцій, придатних для вирішення складних проблем експертів за допомогою спрощених рішень. Хоча RoR має круту криву навчання, освоїти розробку для новачків порівняно легко. Кожен, хто навчається та використовує RoR, може скористатися цим винахідливим та унікальним фреймворком. Ruby on Rails широко застосовується великими організаціями, такими як Scribd, GitHub і Hulu, а також стартапи та громадські проекти, які тільки починають свій шлях у своїй галузі.

Завдяки унікальним особливостям, ця технологія показала програмістам, як зробити кодування розумним. Він обробляє складні речі та робить програмування простим для всіх. RoR робить багато, щоб заощадити час та зусилля. RoR виконує все це за допомогою застосування функцій, таких як інтегроване автоматизоване тестування, методологія "convention over configuration", scaffolding та вдосконалені методи кешування.

Ось деякі функції Ruby on Rails, завдяки яким він виділяється:

— Конвенція щодо конфігурації допомагає створити відповідні вдосконалені компоненти за допомогою автоматичного зондування простих звичайних елементів;
— Автоматизоване тестування – RoR проводить власний набір тестів на написаний код, що може заощадити час та зусилля забезпечивши якість;
— Функція локалізації допомагає інтегрувати попередньо розроблений код у рамку RoR для більшого проекту;
— Scaffolding, функція яка дозволяє програмісту визначити, як повинна функціонувати база даних додатків. Після цього фреймворк автоматично генерує необхідний код відповідно до нього. Техніка scaffold створює інтерфейси автоматично;
— RoR має безліч обширних бібліотек для оснащення розробника всіма необхідними інструментами для створення високоякісного продукту. Бібліотеки AJAX, бібліотеки доступу до бази даних та бібліотеки загальних завдань - одні з небагатьох, які RoR містить у своїй колекції.

Швидкість кодування RoR пояснюється його унікальними особливостями, описаними вище. RoR швидкий, оскільки це економить багато часу, спрощуючи безліч завдань, тим самим швидше виконуючи цілі. Ось як це робить більше за менший час без шкоди для якості. Ця вигода важлива, коли мова йде про швидкий розвиток. Scaffolding, узгодження конфігурації та автоматичне тестування – деякі з цінних функцій RoR, які економлять багато часу та сил.

RoR добре впорається зі складністю логіки і представляє точність, тим самим допомагаючи генерувати відмінний код, орієнтований на точний результат. Він має методологію тестування завдяки інтегрованій функції автоматизованого тестування, яка допомагає досягти кращої узгодженості та меншої кількості розбіжностей. Таким чином, RoR дуже надійний і простий в обслуговуванні.

Масштабованість Ruby on Rails – це функція, яка допомагає розробникам впоратися з жорсткішими та більшими обов'язками. Навіть при роботі з меншим проектом, його можна легко збільшити, не погіршуючи якість. RoR призначений для високої якості за менших витрат.

Завжди є достатня кількість кваліфікованих та досвідчених розробників, які є фахівцями в Ruby on Rails і можуть допомогти вирішити деякі проблеми, проконсультуватися в проблемних сферах та вирішити проблеми.

Ще одна специфічна характеристика Rails –RESTful дизайн додатків. REST посилається на стиль архітектури програмного забезпечення, що базується на відносинах клієнт-сервер. Це сприяє логічній структурі в додатках, а значить, їх можна легко розгорнути як API.

Додатки Ruby on Rails є найпоширенішими у таких областях веб-розробки:

- Проекти, що передбачають широкий спектр складних функцій;
- Великі проекти, що потребують кардинальних перетворень;
- Довгострокові проекти, які проходять через постійні зміни параметрів;
- Проекти, що мають інтенсивний трафік;
- Маленькі, швидкі проекти з розробки прототипів та MVP.

Деякі розробники не рекомендують Ruby on Rails у таких випадках:

- Немає кардинальних перетворень у проекті;
- Проект з обмеженою функціональністю та рівномірними операціями;
- Немає потреби у швидких рішеннях;
- Ваш проект вимагає низького споживання ресурсів.

2.3 Система контролю версій Git

Git – це безкоштовний, відкритий код розповсюдженої системи контролю версій, призначений для швидкого та ефективного управління всіма проектами від маленьких до дуже великих проектів. Git має функціональність, продуктивність, безпеку та гнучкість, які потребують більшість команд та окремих розробників.

Git – це інструмент управління розподіленою версією, який підтримує розподілені нелінійні робочі процеси, надаючи гарантію даних для розробки програмного забезпечення якості. Git надає користувачеві усі засоби розповсюдження VCS. Git-сховища дуже легко знайти та отримати доступ.

Git випускається під ліцензією GPL (General Public License) з відкритим кодом. Вам не потрібно купувати Git. Це абсолютно безкоштовно. А оскільки він є відкритим кодом, ви можете змінювати вихідний код відповідно до ваших вимог. Оскільки для виконання всіх операцій не потрібно підключатися до будь-якої мережі, вона виконує всі завдання дуже швидко. Тести на ефективність, зроблені Mozilla, показали, що це на порядок швидше, ніж інші системи управління версіями. Доступ до історії версій з локально збереженого сховища може бути в сто разів швидшим, ніж його отримання з віддаленого сервера. Основна частина Git написана на C, що дозволяє уникнути накладних витрат, пов'язаних з іншими мовами високого рівня.

Git дуже масштабований. Отже, якщо в майбутньому кількість співробітників збільшиться, Git може легко впоратися з цією зміною. Хоча Git являє собою ціле сховище, даних, що зберігаються на стороні клієнта, дуже мало, оскільки Git стискає всі величезні дані завдяки техніці стиснення без втрат.

Оскільки кожен учасник має власне локальне сховище, у випадку збою системи втрачені дані можуть бути відновлені з будь-якого локального сховища. Ви завжди матимете резервну копію всіх своїх файлів.

Git використовує SHA1 (функцію захищеного хешу) для іменування та ідентифікації об'єктів у своєму сховищі. Історія Git зберігається таким чином, що ідентифікатор конкретної версії (фіксація в умовах Git) залежить від усієї історії розвитку. Після його публікації неможливо змінити старі версії, не помітивши їх.

Централізована система управління версіями (CVCS) використовує центральний сервер для зберігання всіх файлів і дозволяє командну співпрацю. Він працює в єдиному сховищі, до якого користувачі можуть безпосередньо отримати доступ до центрального сервера.

Розподіленій системі управління версіями (DVCS) не обов'язково покладаються на центральний сервер для зберігання всіх версій файлу проекту. У розподіленому VCS кожен дописувач має локальну копію або "клон" головного сховища, тобто кожен підтримує власне локальне сховище, яке містить усі файли та метадані, наявні в головному сховищі.

У випадку CVCS центральний сервер повинен бути достатньо потужним, щоб обслуговувати запити всієї команди. Для менших команд це не проблема, але в міру збільшення розміру команди обмеження обладнання на сервері можуть бути вузьким місцем продуктивності. У випадку DVCS, розробники не взаємодіють із сервером, якщо їм не потрібно відправляти чи стягувати зміни. Весь важкий підйом відбувається на стороні клієнта, тому апаратне забезпечення сервера може бути дуже простим.

Git підтримує швидке розгалуження та злиття та включає конкретні інструменти для візуалізації та навігації по нелінійній історії розвитку. Основне припущення Git полягає в тому, що зміни будуть об'єднуватися частіше, ніж це написано, оскільки вони передаються навколо різних рецензентів. Гілки у Git дуже легкі. Відділення в Git - це лише посилання на один комміт. Завдяки батьківським зобов'язанням може бути побудована повна галузева структура.

2.4 База даних SQLite

База даних є невід'ємною частиною побудови програмної системи, яка ефективно зберігає та читає дані. SQLite – це невеликий додаток до бази даних, який використовується у мільйонах програмного забезпечення та пристроїв. SQLite – це високопродуктивна, легка реляційна база даних з відкритим кодом з великою кількістю документації.

Збереження даних у flat файлі не так ефективно для зчитування та зберігання даних. Бази даних впорядковують дані в належному порядку таким чином, що читання та запис даних значно швидше. Дані можуть бути структурованими, напівструктурованими або неструктурованими. Бази даних в основному використовуються для зберігання структурованих та напівструктурованих даних. Реляційні бази даних містять лише один тип контейнера даних: таблиця. Окрім моделі реляційних даних, модель даних ключа/значення колись підтримувалася SQLite, але потім була видалена.

SQLite безсерверний, це означає, що немає окремого серверного процесу для планування читання/запису файлів бази даних. Це призводить до відносно низької продуктивності одночасності порівняно з іншими системами баз даних клієнт/сервер, такими як PostgreSQL, MySQL. Щодо моделі контролю сумісності, SQLite дотримується суворого двофазного блокування та реалізує дуже простий протокол блокування на рівні бази даних, який дозволяє користування декільком читачам, але лише одному автору в одній базі даних одночасно. Що стосується деталей реалізації, блокування та контроль одночасності керуються модулем пейджера. Завдання пейджера полягає в тому, щоб отримати певну сторінку в базі даних з диска в буфер пам'яті і запобігти запису від будь-яких інших транзакцій, використовуючи п'ять схем стану блокування, коли сторінка записується.

SQLite використовує B-дерево для структури даних індексації за замовчуванням. Користувачі можуть створювати індекси за допомогою синтаксису SQL з CREATE INDEX з наступним іменем нового індексу. Крім того, SQLite підтримує R-дерево для виконання запитів діапазону, якщо

DSQLITE_ENABLE_RTREE=1 прапор встановлений перед компіляцією. Модуль R-дерева SQLite реалізований у вигляді віртуальної таблиці. SQLite підтримує індексацію первинних ключів, індексацію вторинних ключів, індексацію похідних ключів та часткову індексацію. Будь-який індекс, що включає в себе пункт WHERE в кінці, вважається частковим індексом. Індекси, які опускають пункт WHERE, – це звичайні повні індекси.

SQL-компілятор SQLite складається з токенайзера, парсера та генератора коду. Після токенізації та розбору операторів SQL генератор коду переводить дерево розбору на якусь мову складання (байт-код), специфічну для SQLite. Ця мова складання складається з інструкцій, які виконуються її віртуальною машиною. Єдиним завданням генератора коду є перетворення дерева розбору в повну міні-програму, написану цією мовою складання, і передачу його на віртуальну машину для обробки.

Стандартні команди SQLite схожі на SQL. Він упускає деякі функції (RIGHT та FULL OUTER JOIN, GRANT і REVOKE тощо), в той же час додаючи декілька власних функцій (PRAGMA тощо). Команда SQLite спілкується з реквізитними базами даних в основному за допомогою CREATE, SELECT, INSERT, UPDATE, DELETE та DROP. Крім стандартної мови SQL, API SQLite забезпечує велику кількість прив'язок мови хоста (C, C ++, Perl, Python тощо). API SQLite можна розділити на дві загальні частини: основний API, який виконує команди SQL, і API розширення, який розширює/налаштовує SQLite, шляхом створення визначених користувачем функцій.

2.5 Менеджер версій Ruby (RVM)

RVM – це інструмент командного рядка, який дозволяє легко встановлювати, керувати та працювати з декількома середовищами в Ruby від інтерпретаторів до наборів дорогоцінних каменів.

RVM дозволяє розгорнути кожен проект із власним повністю автономним та виділеним середовищем. Наявність точного набору дорогоцінних каменів

також дозволяє уникнути конфлікту між версіями, проектами, що може спричинити важкі для відстеження помилки. З RVM не встановлено ніяких інших каменів, окрім потрібних. Це робить роботу з декількома складними додатками, де кожен має довгий список залежних від дорогоцінних каменів, значно ефективнішим. RVM дозволяє легко протестувати оновлення дорогоцінних каменів, перейшовши на новий чистий набір дорогоцінних каменів для тестування, залишаючи свій первісний набір недоторканим.

RVM дозволяє дуже легко та послідовно протестувати шляхи оновлення. За допомогою RVM можна запускати тестовий набір, завдання rake, gem команди для кількох версій Ruby одночасно.

RVM має надзвичайно гнучку систему управління дорогоцінними каменями, яку називають gemsets. Gemsets RVM роблять керування дорогоцінними каменями в декількох версіях Ruby доволі легким. RVM дозволяє вам додавати невеликий текстовий файл у сховище вашої програми, замість того, щоб перевіряти тонни дорогоцінних каменів, які б непотрібно збільшували розмір вашого сховища. Крім того, управління Gemset RVM використовує загальний каталог кешу, тому лише одна завантажена версія кожного дорогоцінного каміння знаходиться на диску, а не в декількох копіях. RVM допомагає забезпечити, що всі аспекти Ruby повністю містяться в просторі користувача, сильно заохочуючи безпечніше використання, не використовуючи root. Таким чином, використання RVM забезпечує більш високий рівень безпеки системи, а отже, зменшує ризик та скорочує загальний час простою системи. Крім того, оскільки всі процеси працюють на рівні користувача, компрометований процес Ruby не може поставити під загрозу всю систему.

2.6 Система управління базами даних Redis

Redis (розшифровується як Remote Dictionary Server) – це швидке сховище даних типу «ключ-значення» в пам'яті з відкритим вихідним кодом для використання в якості бази даних, кеша, брокера повідомлень або черги. Redis

забезпечує час відгуку на рівні часткою мілісекунди і дозволяє додаткам, що працюють в режимі реального часу, виконувати мільйони запитів в секунду. Такі додатки затребувані в сфері ігор, рекламних технологій, фінансових сервісів, охорони здоров'я і IoT. Redis широко застосовується для кешування, управління сеансами, розробки ігор, створення таблиць лідерів, аналітики в режимі реального часу, роботи з геопросторовими даними, підтримки служб таксі, чатів і сервісів обміну повідомленнями.

Всі дані в Redis зберігаються в пам'яті, а не на дисках або твердотільних накопичувачах, як в інших базах даних. Оскільки Redis, як і інші сховища даних в пам'яті, не потребує доступі до диска, це виключає затримки, пов'язані з пошуком, і забезпечує доступ до даних за мікросекунди. У число можливостей Redis входить підтримка різноманітних структур даних, забезпечення високої доступності, робота з геопросторовими даними, створення скриптів Lua, проведення транзакцій, постійне зберігання даних на диску і підтримка кластерів. Все це спрощує створення додатків, що працюють в режимі реального часу в масштабі всього Інтернету.

Всі дані Redis знаходяться в основній пам'яті сервера, на відміну від таких баз даних, як PostgreSQL, Cassandra, MongoDB та інших, які більшу частину даних зберігають на магнітних дисках або SSD-накопичувачах. У порівнянні з традиційними дисковими базами даних, які вимагають циклічного звернення до диска для більшості операцій, сховища даних в пам'яті, такі як Redis, вільні від цього обмеження. Завдяки цьому істотно зростає кількість виконуваних операцій і скорочується час відгуку. В результаті забезпечується надзвичайно висока продуктивність. Операції читання або запису в середньому займають менше мілісекунди, швидкість роботи досягає мільйонів операцій в секунду.

На відміну від спрощених сховищ на основі пар «ключ - значення», які підтримують обмежений набір структур даних, Redis підтримує величезну різноманітність структур даних, що дозволяє задовольнити потреби різноманітних додатків. Типи даних Redis включають:

- рядки – текстові або двійкові дані розміром до 512 МБ;

- списки – колекції рядків, впорядковані у порядку додавання;
- множини – невпорядковані колекції рядків з можливістю перетину, об'єднання і порівняння з іншими типами множин;
- сортовані множини – множини, впорядковані за значенням;
- хеш-таблиці – структури даних для зберігання списків полів і значень;
- бітові масиви – тип даних, який дає можливість виконувати операції на рівні бітів;
- структури HyperLogLog – імовірнісні структури даних, що служать для оцінки кількості унікальних елементів в наборі даних.

Redis спрощує код, дозволяючи писати менше рядків для зберігання, використання даних і організації доступу до даних в додатках. Наприклад, якщо додаток містить дані, що зберігаються в хеш-таблиці, і потрібно зберегти ці дані в сховище, можна просто використовувати структуру даних хеш-таблиці Redis. Рішення такого завдання з використанням сховища даних, що не підтримує структури хеш-таблиць, зажадає написання серйозного обсягу коду для перетворення даних з одного формату в інший. Redis вже оснащений вбудованими структурами даних і надає безліч можливостей їх комбінування і взаємодії з даними клієнта. Розробникам під Redis доступні більше ста клієнтів з відкритим вихідним кодом. Мови програмування включають Java, Python, PHP, C, C ++, C #, JavaScript, Node.js, Ruby, R, Go і багато інших.

У Redis застосовується архітектура вузлів "ведучий-підлеглий" і підтримується асинхронна реплікація, при якій дані можуть копіюватися на кілька підлеглих серверів. Це забезпечує як поліпшені характеристики читання (так як запити можуть бути розподілені між серверами), так і прискорене відновлення в разі збою основного сервера. Для забезпечення постійного зберігання Redis підтримує знімки стану на момент часу (копіювання наборів даних Redis на диск).

Redis пропонує архітектуру "ведучий-підлеглий" з одним ведучим вузлом або з кластерною топологією. Це дозволяє створювати високодоступні рішення, що забезпечують стабільну продуктивність і надійність. Якщо потрібно налаштувати розмір кластера, доступні різні варіанти вертикального і

горизонтального масштабування. В результаті можна нарощувати кластер відповідно до потреб.

2.7 Sidekiq

При розробці програми Ruby on Rails можуть виникнути такі завдання, які повинні виконуватися асинхронно. Обробка даних, масова розсилка електронної пошти, взаємодія з зовнішніми API і інші подібні завдання можуть виконуватися асинхронно в вигляді фонових завдань. Використання фонових завдань допоможе підвищити продуктивність вашої програми за рахунок розвантаження потенційно ресурсномістких завдань в фонову чергу зі звільненням початкового циклу запит / відповідь.

Sidekiq – одна з найбільш широко використовуваних інфраструктур фонових завдань, які можна реалізувати в додатку Rails. Вона заснована на системі зберігання пар ключ-значення в оперативній пам'яті Redis, що відрізняється гнучкістю і високою продуктивністю. Sidekiq використовує Redis як сховище для управління завданнями, щоб обробляти тисячі завдань в секунду.

Sidekiq став досить популярним у спільноті Ruby з моменту випуску, не малою частиною, завдяки високій продуктивності, простоті монтажу та простоті використання. Він також працює з комерційними хостинг-сервісами, такими як Heroku, припускаючи, що ви спочатку встановите екземпляр Redis.

Робота із затримкою дає розумння, що не все має відбуватися негайно. Швидше можна затримати певні завдання, відклавши їх на другий план, дозволяючи веб-серверу відповідати користувачам швидше, ніж це було б в іншому випадку. І коли швидкість є настільки важливим елементом успіху веб-додатків, розсудливе використання Sidekiq, ймовірно, призведе до великої зміни.

2.8 Factory Bot

Factory Bot створює тестові прилади, які можна використовувати як інструмент для автоматичного тестування. Factory Bot призначений бути більш динамічним, ніж засоби за замовчуванням, які мають Rails. Іншими словами – Factory Bot створює тестові пристосування, які є підробленими тестовими об'єктами, які можна повторно використовувати протягом тестування програми.

Тестові прилади Factory Bot можуть використовувати фіксовані дані, які можуть бути використані для створення фіксованого середовища тестування. Це також дозволяє перевірити свої моделі та код, не торкаючись вашої бази даних. Можливість ефективного створення об'єктів у контрольованому середовищі також може допомогти протестувати більш масштабовані програми. Крім того, factories також дозволяють протестувати конкретну інформацію про атрибути.

2.9 Devise

Devise використовується для автентифікації Ruby on Rails. Завдяки Devise можливо створити користувача, який може входити та виходити з програми, настільки просто, тому що Devise забезпечує всі контролери, необхідні для створення користувачів та для сеансів користувача. Gem заснований на Warden і обробляє автентифікацію за допомогою bcrypt. Реалізація автентифікації вручну вимагає декількох контроллерів та декількох годин налаштування, не включаючи тестування.

Devise використовує 10 модулів для налаштування автентифікації користувачів, які знаходяться всередині моделі користувача. Шість із цих модулів включені за замовчуванням (позначені символом *). Можна змінити існуючу конфігурацію у файлі config/initializers/devise.rb. Модуль *Database Authenticable отримує та зберігає пароль у базі даних. Автентифікація здійснюється за запитом POST. Необхідно зберегти користувальницький/хешований пароль у БД. Модуль Omniauthable додає підтримку постачальника Omniauth, що дозволяє входити

через сторонніх постачальників, таких як Facebook, Twitter тощо. Модуль Confirmable вимикає доступ до облікового запису користувача, якщо користувач не підтвердив свій обліковий запис електронною поштою. Модуль *Recoverable додає посилання "Забув свій пароль", що дозволяє користувачеві скинути свій пароль за допомогою електронної пошти. Модуль *Registerable створює процес реєстрації, тепер користувачі можуть редагувати та видаляти свій обліковий запис. Слід вимкнути для бета-тестування/сайтів, які мають лише запрошення. Модуль *Rememberable створює маркер і зберігає сеанс користувача із збереженим файлом cookie (додає прапорець "Запам’ятати мене"). Модуль *Trackable відстежує IP-адреси користувачів, кількість входів, останній вхід та часові позначки. Модуль Timeoutable завершує сеанс користувача через певний проміжок часу. Модуль *Validatable використовує вбудовані перевіроки для адреси електронної пошти та пароля (довжина, символи тощо). Змінення перевірок у config/inicijalizer/devise.rb. Модуль Lockable блокує обліковий запис через певний час або певну кількість спроб входу.

2.10 RSpec

RSpec – найпопулярніша структура тестування Ruby on Rails відповідно до спільноти Ruby on Rails. Він також чудово здатний перевірити будь-який написаний Ruby код. Репозиторій RSpec чітко організований у менші частини, такі як:

- rspec – основний пакет RSpec;
- rspec-core – структура, що надає дорогоцінний камінь, для написання виконуваних прикладів та настроюваної команди rspec;
- rspec-rails призначений для тестування додатків Ruby on Rails;
- rspec-mock –забезпечує підтримку stabbing та mocking запитів;
- rspec-expectations – пакет RSpec, який відкриває читабельний API для вираження прикладів коду очікуваних результатів.

Як було сказано, RSpec є найпопулярнішим вибором для тестування проектів Ruby on Rails. Однак налаштування програми з Ruby on Rails за замовчуванням не поставляються, її потрібно встановити вручну. Типовою основою для тестування додатків Ruby on Rails є MiniTest. Це вбудований механізм з офіційною документацією Ruby on Rails про те, як писати одиничні, функціональні, інтеграційні та системні тести, і пояснені всі його термінології. Ще однин інструмент тестування, який широко використовується в Ruby і Ruby on Rails, як у Java, Electron та багатьох інших - це Cucumber.

RSpec має найкращу документацію, яка містить багато корисних прикладів та сценаріїв реального життя, які зазвичай потребують розробники. Це має вирішальне значення для використання всього, що забезпечує свою власну доменну мову – DSL, як це робить RSpec. Крім того, RSpec має всі функції, які потрібні розробникам, і всі вони дуже доглянуті. RSpec – це додаток, тоді як MiniTest і Cucumber – це просто фреймворк. Для запуску тестового набору не потрібно нічого іншого, забезпечуючи всебічне рішення для тестування Ruby.

Використовуючи RSpec в Ruby on Rails, найпопулярнішим вибором для обчислення звітів про покриття коду проекту є SimpleCov. За замовчуванням він забезпечує простий загальний відсоток покриття кодом тестового набору проекту. Крім того, SimpleCov дозволяє групувати звіти за змістовними категоріями, такими як контролери, моделі, представлення тощо.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

REST-сервер для торгівельного майданчику, де люди (як продавці) можуть презентувати свою продукцію для продажу на лотах аукціонів з одного боку, а з іншого – приймати участь у торгах (як клієнти), щоб купувати продуктію. У базі даних аукціонного ринку є 4 типи об'єктів: користувачі, лоти, ставки, замовлення.

Користувачі –це сутності, які створюються при реєстрації в системі. Кожен користувач може бути продавцем або/і замовником продукції. Загалом, користувачі мають наступні атрибути (усі атрибути обов'язкові):

— email;

— phone;

— firstname;

— lastname;

— birth day.

Виконано валідацію для таких атрибутів: email, phone, birth_day (вік повинен бути більше 21 року). Поля email та phone повинні відповідати заданому у відповідних валідаціях формату.

Якщо користувач хоче продати будь-який товар, він повинен створити лот в системі. Усі користувачі повинні мати можливість бачити список усіх лотів, які мають статус "in_process". Лоти мають наступні атрибути (усі атрибути обов'язкові):

—title;

—image;

—description;

—status;

—created at;

—current price – відображає стартову ціну або максимальну пропозицію, якщо вони існують;

—estimated price – ціна, максимальна для поточного лота. Якщо користувач запропонує цю ціну, то він може купити лот негайно;

—lot start time - час, коли лот буде відкрито;

—lot end time - час, коли лот буде закрито.

Статуси:

— pending – за замовчуванням;

— in_process – змінюється, коли наступає час початку лота;

— closed – змінюється, коли настав час закінчення лота або будь-який клієнт пропонує максимальну орієнтовну ціну (estimated price).

Валідації:

—поля, пов’язані з ціною не можуть бути негативними;

—час закінчення не може бути меншим, ніж час початку, а час початку не може бути меншим за поточний час;

—максимальна ціна повинна бути більшою ніж початкова.

Коли статус лотів стає "in_process", інші користувачі(клієнти), які хочуть придбати цей лот, можуть запропонувати їх ціну. Учасники тендеру представляють ці пропозиції. Користувач може створювати ставки стільки разів, скільки хоче, поки партія перебуває в стані "in_process".

Атрибути ставок (усі атрибути обов'язкові):

—bid creation time;

—proposed price.

Валідації:

—запропонована ціна не може бути негативною та повинна бети більшою за поточну ціну лота.

—Коли статус лота закривається, і є одна або більше пропозицій, система автоматично надсилає повідомлення замовнику, який виграв цей лот.

—Якщо клієнт готовий, він створює запис про замовлення відповідної ставки, який представляє інформацію про доставку товару.

Атрибути замовлення (усі поля обов'язкові):

—arrival location – текстове поле, де користувач повинен ввести адресу доставки;

—arrival type (pickup, royal_mail, united_states_postal_service, dhl_express);

Статуси:

— pending – чекає, поки продавець прийме замовлення клієнта;

— sent – означає, що продавець приймає замовлення і процес доставки товару розпочато;

— delivered – користувач встановлює цей статус після отримання товару.

Проект складається з:

—app/controllers – підкаталог controllers, де Rails шукає класи контролерів. Контролер обробляє веб-запит від користувача;

—app/models – підкаталог models містить класи, які моделюють і упаковують дані, що зберігаються в базі даних нашого проекту;

—config – цей каталог містить невеликий обсяг коду конфігурації, який знадобиться додатку, включаючи конфігурацію бази даних (в файлі database.yml), структуру середовища Rails (environment.rb) і маршрутизацію веб-запитів (routes.rb);

—db – можна управляти реляційної базою даних за допомогою скриптів, які ви створюються і розміщуються в цьому каталозі;

—spec – файлів з тестами.

Для строрення проекту Rails API потрібно ввести наступну команду:

rails new auction_marketplace_api –api

Ця команда створює новий Rails-додаток, з назвою auction_marketplace_api. Підтримка опції --api була додана в Rails 5. Rails завантажує менший набір middleware. Після створення нового Rails-додатки, наступний етап - запуск Bundler, для установки і включення гемів, необхідних додатку. Bundler автоматично запускається (через bundle install) командою rails.

Файлова структура проекту API зображена на рис.3.1.

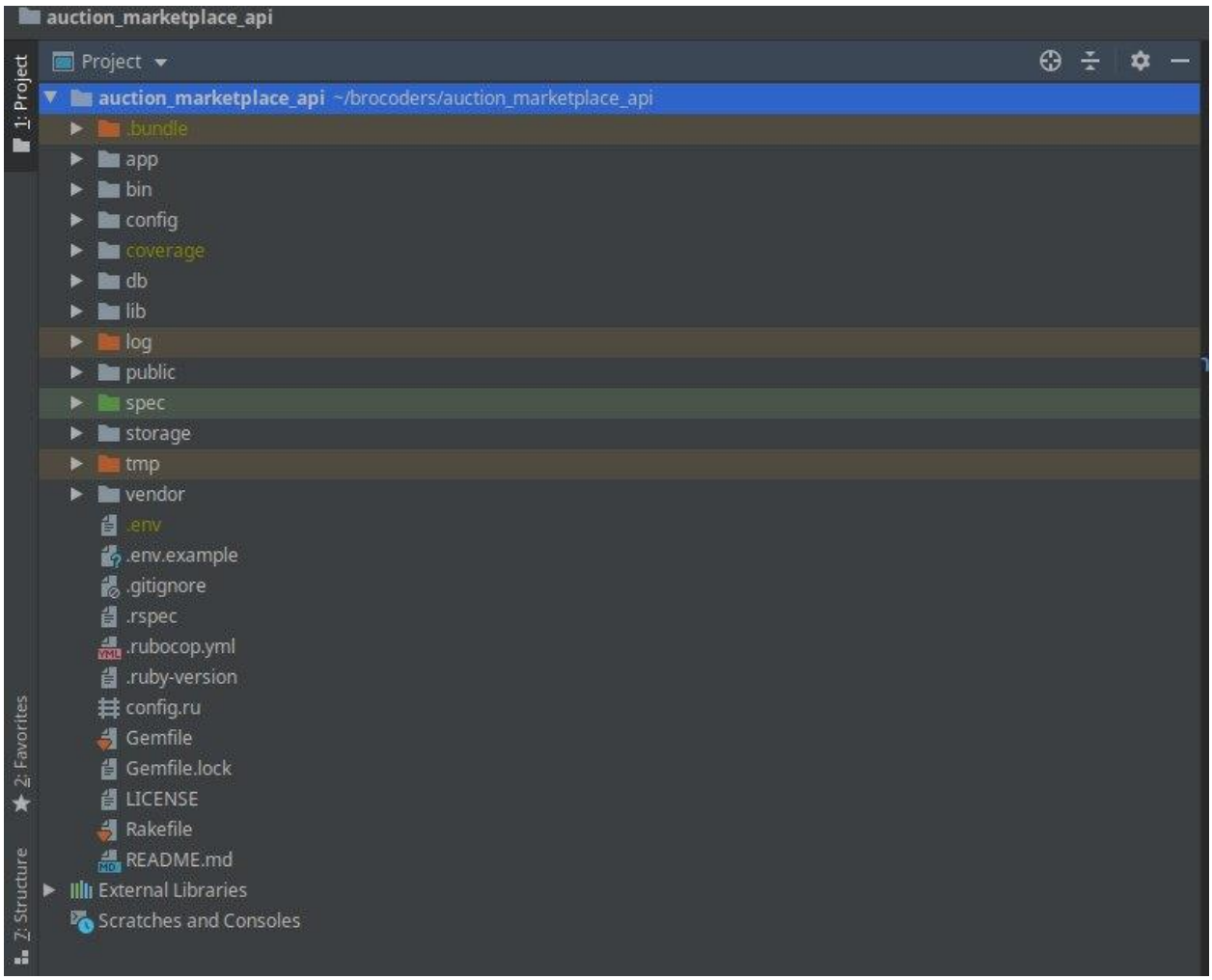

Рисунок 3.1 – Структура файлів API

Код основних файлів проекту розміщено в додатку А.

Перевірка роботи тестів здійснюється на тестовій базі даних, котра оновлюється перед початком кожного тестування. Вони забражені на рис. 3..

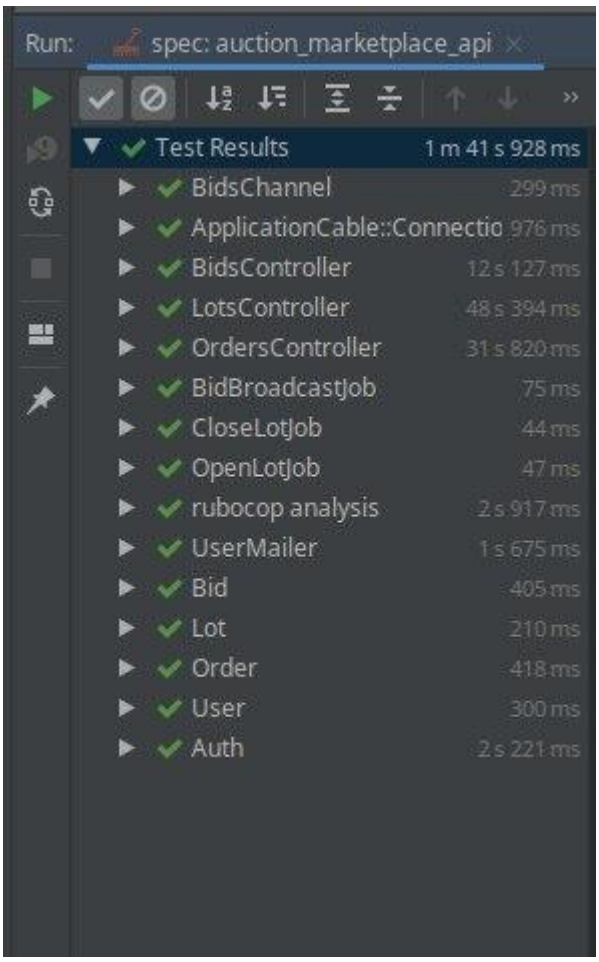

Рисунок 3.2 – Успішно виконані тести

Такі тести проводилися регулярно на кожному з етапів розробки проекту. Код файлів тестування розміщено в додатку А

ВИСНОВКИ

Під час виконання випускної роботи було реалізовано REST-сервер торгівельного майданчику. Платформою для створення інтернет-сервісу було обрано фреймворк Ruby on Rails, який дозволяє ефективно створювати проекти високої складності з використанням різних модулів.

Rails – сучасний стандарт майже усіх веб-додатків для бізнесу, який дозволяє швидко створювати прототипи веб-додатків. Для написання тестів використовувалася бібліотека RSpec. RSpec дуже корисний на рівні модульного тестування, тестуючи деталі і бізнес-логіку додатку. RSpec робить наголос не на тому, як працює додаток, а на тому, як він поводиться, іншими словами, що робить додаток.

У перспективах розвитку планується збільшити функціонал системи та розробити веб та мобільний додатки.

СПИСОК ЛІТЕРАТУРИ

1. Д. Томас, Д. Х. Хэнссон - Гибкая разработка веб-приложений в среде Rails – 342 с.
2. Оби Фернандес - Путь Rails. Подробное руководство по созданию приложений в среде Ruby on Rails – 2009 – «Символ-Плюс» - 74 с.
3. Ruby on Rails – Framework – Режим доступу: http://www.tutorialspoint.com/ruby-on-rails/rails-framework.htm
4. Ruby on Rails Tutorial by Michael Hurtl – Режим доступу: http://railstutorial.ru
5. Devise WIKI – Режим доступу: https://github.com/plataformatec/devise/wiki
6. Аутентификация в Rails-приложениях с помощью Devise. Часть 1: базовая настройка – Режим доступу: http://habrahabr.ru/post/208056/
7. Ruby on Rails Tutorial by Michael Hurtl – Режим доступу: http://railstutorial.ru Ruby on Rails Tutorial by Michael Hurtl – Режим доступу: http://railstutorial.ru
8. Д. Флэнаган, Ю. Мацумото - Язык программирования Ruby – 2011 - ООО Издательство «Питер» - 121 с.
9. Сэм Руби, Дейв Томас, Дэвид Хэннсон - Rails 4. Гибкая разработка вебприложений – 2014 - ООО Издательство «Питер» - 231 с.
10. Майкл Фитцджеральд - Язык программирования Ruby – 2008 – «БХВПетербург» - 324 с.
11. RSpec. Часть #1: создаем тесты для модели – Режим доступу: http://habrahabr.ru/post/53264/
12. Pro Git Book – Режим доступу: http://gitscm.com/book/ru/v1
13. Роутинг в Rails – Режим доступу: http://rusrails.ru/rails-routing
14. Factory Bot: Getting Started – Режим доступу:

https://github.com/thoughtbot/factory_bot/blob/master/GETTING_STARTED.md

15. Redis для начинающих – Режим доступу: https://webdevblog.ru/redis-dlya-nachinajushhij/
16. How to Use The Redis Database in Ruby – Режим доступу: https://www.rubyguides.com/2019/04/ruby-redis/
17. SimpleCov. Code coverage for Ruby – Режим доступу: https://rubydoc.info/gems/simplecov/frames
18. SQLite Ruby tutorial – Режим доступу: http://zetcode.com/db/sqliteruby/
19. Online auction – Режим доступу: https://en.wikipedia.org/wiki/Online_auction

ДОДАТОК А

Програмний код проекту.

Папка app/channels/application_cable:

Файл connection.rb

```
module ApplicationCable
  class Connection < ActionCable::Connection::Base
    identified_by :current_user

    def connect
      params = request.headers

      access_token = params["access-token"]
      uid = params["uid"]
      client = params["client"]

      self.current_user = find_verified_user access_token, uid, client
    end

    private
      def find_verified_user(token, uid, client_id)
        user = User.find_by uid: uid
        if user && user.valid_token?(token, client_id)
          user
        else
          reject_unauthorized_connection
        end
      end
  end
end
```

Папка app/channels

Файл bids_channel.rb

```
class BidsChannel < ApplicationCable::Channel
  def subscribed
    reject unless Lot.find_by(id: params[:lot_id])
    stream_from "bids_for_lot_#{params[:lot_id]}"
  end
end
```

Папка app/controllers/concerns

Файл devise_whitelist.rb

```
module DeviseWhitelist
  extend ActiveSupport::Concern

  included do
    before_action :configure_permitted_parameters, if: :devise_controller?
  end

  protected
    def configure_permitted_parameters
      devise_parameter_sanitizer.permit(:sign_up, keys: [:firstname, :lastname, :phone, :birth_day])
      devise_parameter_sanitizer.permit(:account_update, keys: [:firstname, :lastname, :phone])
    end
end
```

Папка app/controllers

Файл application_controller.rb

```
class ApplicationController < ActionController::API
```

```
  include DeviseTokenAuth::Concerns::SetUserByToken
  include DeviseWhitelist
  include Pundit
  include ActionController::Helpers

  before_action :authenticate_user!, unless: :devise_controller?

  rescue_from ActiveRecord::RecordNotFound do |exception|
    render json: { error: exception.message }, status: :not_found
  end

  rescue_from ActiveRecord::RecordInvalid do |exception|
    render json: { error: exception.message }, status: :unprocessable_entity
  end

  rescue_from Pundit::NotAuthorizedError do
    render json: { error: "You are not have permission for this action" }, status:
:unauthorized
  end
end
```

Файл bids_controller.rb

```
class BidsController < ApplicationController
  expose :bid
  expose :lot

  def index
    bids = Bid.where(lot_id: params[:lot_id]).order(proposed_price: :desc)
    render json: bids, status: :ok
  end

  def create
    bid = lot.bids.new(bid_params.merge(user: current_user))
    authorize bid
    bid.save!
```

```
  BidBroadcastJob.perform_later bid.id
  render json: bid, status: :created
 end

 private
  def bid_params
   params.require(:bid).permit(:proposed_price)
  end
end
```

Файл lots_controller.rb

```
class LotsController < ApplicationController
 expose :lot

 def index
  lots = Lot.in_process.order(created_at: :desc).page(params[:page])
  render json: lots, status: :ok
 end

 def my_lots
  lots = filtered_lot.order(created_at: :desc).page(params[:page])
  check_my_lot(lots)
  render json: lots, check_my_lot: true, status: :ok
 end

 def current_user_info
  render json: { code: Digest::SHA1.hexdigest([current_user.id, lot.id].join)[0...10] }
 end

 def show
  check_win(lot) if lot.closed? && lot.bids.present?
  render json: lot, check_my_win: lot.closed?, status: :ok
 end
```

```
def create
  lot = current_user.lots.new(lot_params)
  lot.save!
  render json: lot, status: :created
end

def update
  authorize lot
  lot.update!(lot_params)
  render json: lot
end

def destroy
  authorize lot
  lot.destroy
end

private
  def filtered_lot
    if params[:filter] == "created"
      current_user.lots
    elsif  params[:filter] == "participation"
      participation_lot
    else
      participation_lot.or(Lot.where(user_id: current_user.lots.pluck(:user_id)))
    end
  end

  def participation_lot
```

```
    Lot.where(id: current_user.bids.pluck(:lot_id))
  end

  def check_my_lot(lots)
    lots.map do |lot|
      lot.my_lot = (lot.user_id == current_user.id)
    end
  end

  def check_win(lot)
    lot.my_win = lot.winner_bid.user == current_user
  end

  def lot_params
    params.require(:lot).permit(:title,
                                :description,
                                :image,
                                :current_price,
                                :estimated_price,
                                :lot_start_time,
                                :lot_end_time)
  end
end
```

Файл orders_controller.rb

```
class OrdersController < ApplicationController
  expose :order, -> { Order.find_by!(lot_id: params[:lot_id]) }
  expose :lot

  def show
```

```
    authorize order
    render json: order, status: :ok
  end

  def create
    order = Order.new(order_params.merge(lot: lot, bid: lot.winner_bid))
    authorize order
    order.save!
    render json: order, status: :created
  end

  def update
    authorize order
    if order.pending? && order.lot.user == current_user
      order.send_lot
    elsif order.sent? && order.bid.user == current_user
      order.confirm_delivery
    else
      order.update!(order_params)
    end
    render json: order, status: :ok
  end

  private
    def order_params
      params.require(:order).permit(:arrival_location, :arrival_type)
    end
end
```

Папка app/jobs

Файл bid_broadcast_job.rb

```
class BidBroadcastJob < ApplicationJob
  queue_as :default

  def perform(id)
    bid = Bid.find(id)
    ActionCable.server.broadcast "bids_for_lot_#{bid.lot_id}",
BidSerializer.new(bid).as_json
  end
end
```

Файл close_lot_job.rb

```
class CloseLotJob < ApplicationJob
  queue_as :default

  def perform(id)
    lot = Lot.find(id)
    lot.close! if lot.in_process?
  end
end
```

Файл open_lot_job.rb

```
class OpenLotJob < ApplicationJob
  queue_as :default

  def perform(id)
    lot = Lot.find(id)
    lot.in_process! if lot.pending?
  end
end
```

Папка app/mailers

Файл user_mailer.rb

```
class UserMailer < ApplicationMailer
  def email_for_winner(lot)
    user = lot.winner_bid.user
    @data = { title: lot.title, firstname: user.firstname }
    mail(to: user.email, subject: "You won a lot #{lot.title}")
  end

  def email_for_owner(lot)
    user = lot.user
    @data = { firstname: user.firstname, current_price: lot.current_price, bids_present: lot.bids.present? }
    mail(to: user.email, subject: "Your lot #{lot.title} is closed")
  end

  def email_for_seller(order)
    user = order.lot.user
    @data = { title: order.lot.title, firstname: user.firstname }
    mail(to: user.email, subject: "An order has been created for your lot #{@data[:title]}")
  end

  def email_about_sending(order)
    user = order.bid.user
    @data = { title: order.lot.title, firstname: user.firstname }
    mail(to: user.email, subject: "Your order #{@data[:title]} has been sent successfully")
  end
```

```
  def email_about_delivery(order)
    user = order.lot.user
    @data = { title: order.lot.title, firstname: user.firstname }
    mail(to: user.email, subject: "Your lot #{@data[:title]} has been successfully received by the buyer")
  end
end
```

Папка app/models

Файл bid.rb

```
class Bid < ApplicationRecord
  belongs_to :user
  belongs_to :lot
  has_one :order

  attr_accessor :customer

  after_create :lot_current_price_update, :check_estimated_price_lot

  validates :proposed_price, presence: true
  validates_numericality_of :proposed_price, greater_than: 0.0
  validate :proposed_great_current, :lot_in_process

  private
    def proposed_great_current
      return if proposed_price.blank? || lot.nil?

      errors.add(:proposed_price, "must be greater than current price") if proposed_price <= lot.current_price
```

```
      end

    def lot_in_process
      return if lot.nil?

      errors.add(:lot, "lot status must be in_process") unless lot.in_process?
    end

    def lot_current_price_update
      lot.current_price = proposed_price
      lot.save!
    end

    def check_estimated_price_lot
      if proposed_price >= lot.estimated_price
        lot.close!
        Sidekiq::ScheduledSet.new.select { |job| job.display_args.first == lot.id
}.map(&:delete)
      end
    end
  end
```

Файл lot.rb

```
  class Lot < ApplicationRecord
    mount_uploader :image, ImageUploader

    belongs_to :user
    has_many :bids, dependent: :destroy
    has_one :order, dependent: :destroy
    attr_accessor :my_lot, :my_win
```

```
    after_create_commit :jobs_add
    after_update_commit :set_new_jobs
    after_destroy_commit :jobs_delete

    enum status: [:pending, :in_process, :closed]
    validates :title, :current_price, :estimated_price, :status, :lot_start_time, :lot_end_time, presence: true
    validates_numericality_of :current_price, :estimated_price, greater_than: 0.0
    validate :est_price_greater_current, :end_after_start, :start_after_current_time

    def close!
      closed!
      UserMailer.email_for_winner(self).deliver_later if bids.present?
      UserMailer.email_for_owner(self).deliver_later
    end

    def winner_bid
      bids.order(proposed_price: :desc).first
    end

    private
      def est_price_greater_current
        return if estimated_price.blank? || current_price.blank?

        if estimated_price < current_price && pending?
          errors.add(:estimated_price, "must be greater than current price")
        end
      end
```

```
    def start_after_current_time
      return if lot_start_time.blank?

      if lot_start_time <= DateTime.current
        errors.add(:lot_start_time, "must be after the current time")
      end
    end

    def end_after_start
      return if lot_end_time.blank? || lot_start_time.blank?

      if lot_end_time < lot_start_time
        errors.add(:lot_end_time, "must be after the start time")
      end
    end

    def set_new_jobs
      jobs_delete
      jobs_add if pending?
    end

    def jobs_add
      OpenLotJob.set(wait_until: lot_start_time).perform_later(id)
      CloseLotJob.set(wait_until: lot_end_time).perform_later(id)
    end

    def jobs_delete
      Sidekiq::ScheduledSet.new.select { |job| job.display_args.first == id }.map(&:delete)
    end
```

```
end
```

Файл order.rb

```
class Order < ApplicationRecord
  belongs_to :bid
  belongs_to :lot

  after_create :send_mail_to_seller

  enum arrival_type: [:pickup, :royal_mail, :united_states_postal_service, :dhl_express]
  enum status: [:pending, :sent, :delivered]
  validates :arrival_location, :arrival_type, :status, presence: true
  validate :lot_closed, :bid_belong_lot

  def send_lot
    sent!
    UserMailer.email_about_sending(self).deliver_later
  end

  def confirm_delivery
    delivered!
    UserMailer.email_about_delivery(self).deliver_later
  end

  private
    def send_mail_to_seller
      UserMailer.email_for_seller(self).deliver_later
    end
```

```
    def lot_closed
      return if lot.nil?

      errors.add(:lot, "lot status must be closed") unless lot.closed?
    end

    def bid_belong_lot
      return if lot.nil? || bid.nil?

      errors.add(:bid, "bid must be winner and belong to the lot") unless lot.winner_bid == bid
    end
  end
```

Файл user.rb

```
  class User < ActiveRecord::Base
    # Include default devise modules. Others available are:
    # :lockable, :timeoutable and :omniauthable, :trackable
    devise :database_authenticatable, :registerable,
           :recoverable, :rememberable, :validatable,
           :confirmable
    include DeviseTokenAuth::Concerns::User

    with_options dependent: :destroy do |user|
      user.has_many :lots
      user.has_many :bids
    end

    validates :firstname, :lastname, :phone, :birth_day, presence: true
    validates :email, :phone, uniqueness: true
```

```
    validates_format_of :phone, with: /\A(?:\+?\d{1,3}\s*-?)?\(?(?:\d{3})?\)?[-.]?\d{3}[- .]?\d{4}\z/,
                        message: "is not a phone"
    validate :validate_age

    private
      def validate_age
        if birth_day.present? && birth_day > 21.years.ago
          errors.add(:birth_day, "You must be 21 years or older")
        end
      end
    end
```

Папка app/policies

Файл application_policy.rb

```
class ApplicationPolicy
  attr_reader :user, :record

  def initialize(user, record)
    @user = user
    @record = record
  end

  def index?
    false
  end

  def show?
    false
  end
```

```
def create?
  false
end

def new?
  create?
end

def update?
  false
end

def edit?
  update?
end

def destroy?
  false
end

class Scope
  attr_reader :user, :scope

  def initialize(user, scope)
    @user = user
    @scope = scope
  end

  def resolve
```

```
      scope.all
    end
  end
end
```

Файл bid_policy.rb

```
class BidPolicy < ApplicationPolicy
  def create?
    user != record.lot.user
  end
end
```

Файл lot_policy.rb

```
class LotPolicy < ApplicationPolicy
  def update?
    pending_lot_owner?
  end

  def destroy?
    pending_lot_owner?
  end

  private
    def pending_lot_owner?
      user == record.user && record.status == "pending"
    end
end
```

Файл order_policy.rb

```
class OrderPolicy < ApplicationPolicy
```

```
  def show?
    user == bid.user || user == lot.user
  end

  def create?
    lot.winner_bid.user == user
  end

  def update?
    (record.sent? && bid.user == user) || (record.pending? && (bid.user == user ||
lot.user == user))
  end

  private
    def lot
      record.lot
    end

    def bid
      record.bid
    end
  end
```

Папка app/serializers

Файл bid_serializer.rb

```
class BidSerializer < ActiveModel::Serializer
  attributes :id, :proposed_price, :created_at, :customer

  def customer
    Digest::SHA1.hexdigest([object.user_id, object.lot_id].join)[0...10]
```

```
  end
end
```

Файл lot_serializer.rb

```
class LotSerializer < ActiveModel::Serializer
  attributes :id, :user_id, :title, :description, :current_price, :estimated_price,
          :image, :lot_start_time, :lot_end_time, :status
  attribute :my_lot, if: -> { instance_options[:check_my_lot] }
  attribute :my_win, if: -> { instance_options[:check_my_win] }
end
```

Файл order_serializer.rb

```
class OrderSerializer < ActiveModel::Serializer
  attributes :id, :arrival_location, :arrival_type, :status, :created_at, :updated_at
end
```

Папка config

Файл routes.rb

```
Rails.application.routes.draw do
  mount_devise_token_auth_for 'User', at: 'auth'
  # For details on the DSL available within this file, see
http://guides.rubyonrails.org/routing.html
  mount ActionCable.server => "/cable"
  resources :lots, except: [:new, :edit] do
    resources :bids, only: [:index, :create]
    resource :order, only: [:show, :create, :update]
  end
  get 'lots/my_lots' => 'lots#my_lots', as: 'my_lots'
  get 'current_user_info' => 'lots#current_user_info', as: 'current_user_info'
```

```
  require 'sidekiq/web'
  mount Sidekiq::Web, at: "/sidekiq"
end
```

Папка spec/channels

Файл bids_channel_spec.rb

```
require "rails_helper"

RSpec.describe BidsChannel, type: :channel do
  let(:lot) { create :lot }
  let(:bid) { create :bid }

  it "rejects when no lot id" do
    subscribe
    expect(subscription).to be_rejected
  end

  it "subscribes to a stream when lot id is provided" do
    subscribe(lot_id: lot.id)

    expect(subscription).to be_confirmed
    expect(subscription).to have_stream_from "bids_for_lot_#{lot.id}"
  end
end
```

Файл connection_spec.rb

```
require "rails_helper"

RSpec.describe ApplicationCable::Connection, type: :channel do
  let(:user) { create :user }
```

```
  let(:headers) { user.create_new_auth_token }

  it "rejects connection" do
    expect { connect "/cable" }.to have_rejected_connection
  end

  it "successfully connects" do
    connect "/cable", headers: headers
    expect(connection.current_user).to eq user
  end
end
```

Папка spec/controllers

Файл bids_controller_spec.rb

```
require "rails_helper"

RSpec.describe BidsController, type: :controller do
  let(:fields) do
    %i[id proposed_price created_at]
  end

  describe "GET #index" do
    let(:lot) { create :lot, status: :in_process }
    let(:params) { { lot_id: lot.id } }
    subject { get :index, params: params }

    context "when user authorized" do
      login(:user)

      it "should status code 200" do
```

```
    subject
    expect(response).to have_http_status(200)
  end

  context "should return proper json" do
    let(:fields) { super() + [:customer] }
    let!(:bid1) { create :bid, lot: lot, proposed_price: 15.0 }
    let!(:bid2) { create :bid, lot: lot, proposed_price: 20.0 }

    it "should return 2 bids" do
      subject
      expect(json.length).to eq(2)
    end

    it "should return bids in the proper order" do
      subject
      expect(json.pluck(:id)).to eq [bid2.id, bid1.id]
    end

    it "should use serializer" do
      subject
      expect(json).to all include(*fields)
    end
  end
end

context "when user not authorized" do
  it "should status code 401" do
    subject
    expect(response).to have_http_status(401)
```

```
    end

    it "should contain right message" do
      subject
      expect(json).to eq errors: ["You need to sign in or sign up before continuing."]
    end
  end
end

describe "POST #create" do
  login(:user)
  let(:lot) { create :lot, status: :in_process }
  let(:proposed_price) { lot.current_price + 1.0 }
  subject { post :create, params: { bid: { proposed_price: proposed_price }, lot_id: lot.id } }

  context "with valid attributes" do
    it "creates a new bid" do
      expect { subject }.to change(Bid, :count).by(1)
    end

    it "should status code 201" do
      is_expected.to have_http_status(201)
    end

    it "should queues the job" do
      expect(BidBroadcastJob).to receive(:perform_later)
      subject
    end
```

```
      it "should use serializer" do
        subject
        expect(json).to include(*fields)
      end
    end

    context "with invalid attributes" do
      let(:proposed_price) { lot.current_price - 1.0 }

      it "does not save the new bid" do
        expect { subject }.to_not change(Bid, :count)
      end

      it "should status code 422" do
        is_expected.to have_http_status(422)
      end

      it "should contain right message" do
        subject
        expect(json).to eq error: "Validation failed: Proposed price must be greater than current price"
      end
    end

    context "when user is creator" do
      before(:each) do
        lot.update! user: @user
      end
```

```
      it "should error for lot creator" do
        subject
        expect(json).to eq error: "You are not have permission for this action"
      end
    end
  end
end
```

Файл lots_controller_spec.rb

```
require "rails_helper"

RSpec.describe LotsController, type: :controller do
  let(:fields) do
    %i[id user_id title description current_price estimated_price
         image lot_start_time lot_end_time status]
  end

  describe "GET #index" do
    let(:params) { {} }
    subject { get :index, params: params }

    context "when user authorized" do
      login(:user)

      it "should status code 200" do
        subject
        expect(response).to have_http_status(200)
      end

      context "should return proper json" do
```

```
    let!(:progress_lots) { create_list :lot, 13, user: @user, status: :in_process }
    let!(:pending_lots) { create_list :lot, 5, user: @user }

    it "should return 10 articles without parameters" do
      subject
      expect(json.length).to eq(10)
    end

    it "should use serializer" do
      subject
      expect(json).to all include(*fields)
    end

    context "page 2" do
      let(:params) { { page: 2 } }

      it "should return 3 articles for page 2" do
        subject
        expect(json.length).to eq(3)
      end
    end
  end

  context "order" do
    let!(:old_lot) { create :lot, status: :in_process, created_at: 1.day.ago }
    let!(:newer_lot) { create :lot, status: :in_process, created_at:
DateTime.current }

    it "should return lots in the proper order" do
      subject
```

```
          expect(json.pluck(:id)).to eq [newer_lot.id, old_lot.id]
        end
      end
    end

    context "when user not authorized" do
      it "should status code 401" do
        subject
        expect(response).to have_http_status(401)
      end

      it "should contain right message" do
        subject
        expect(json).to eq errors: ["You need to sign in or sign up before continuing."]
      end
    end
  end

  describe "GET #my_lots" do
    login(:user)
    let(:params) { {} }
    subject { get :my_lots, params: params }

    it "should status code 200" do
      subject
      expect(response).to have_http_status(200)
    end

    context "should return proper json" do
```

```
let(:no_user_lot) { create :lot, status: :in_process }
let(:no_user_lot2) { create :lot, status: :in_process }
let!(:no_user_lot3) { create :lot }
let!(:bid) { create :bid, lot: no_user_lot, user: @user }
let!(:bid2) { create :bid, lot: no_user_lot2, user: @user }
let!(:user_lots) { create_list :lot, 9, user: @user }
let(:fields) { super() + [:my_lot] }

it "should return 10 articles without parameters" do
  subject
  expect(json.length).to eq(10)
end

it "should return :my_lot 10 true" do
  subject
  my_lot_array = json.pluck(:my_lot)
  expect(my_lot_array.select { |my_lot|  my_lot }.count).to eq 9
end

it "should use serializer" do
  subject
  expect(json).to all include(*fields)
end

context "page 2" do
  let(:params) { { page: 2 } }

  it "should return 1 article for page 2" do
    subject
    expect(json.length).to eq(1)
```

```
    end
  end

  context "with created filter" do
    let(:params) { { filter: :created } }

    it "should return 9 articles" do
      subject
      expect(json.length).to eq(9)
    end
  end

  context "with participation filter" do
    let(:params) { { filter: :participation } }

    it "should return 2 articles" do
      subject
      expect(json.length).to eq(2)
    end
  end
end

context "order" do
  let!(:old_lot) { create :lot, user: @user, created_at: 1.day.ago }
  let!(:newer_lot) { create :lot, user: @user, created_at: DateTime.current }

  it "should return lots in the proper order" do
    subject
    expect(json.pluck(:id)).to eq [newer_lot.id, old_lot.id]
  end
```

```
    end
  end

  describe "GET #current_user_info" do
    login(:user)
    let(:lot) { create :lot }
    let(:params) { { id: lot.id } }
    subject { get :current_user_info, params: params }

    it "should status code 200" do
      subject
      expect(response).to have_http_status(200)
    end

    it "should return proper json" do
      subject
      expect(json[:code]).to eq(Digest::SHA1.hexdigest([@user.id, lot.id].join)[0...10])
    end
  end

  describe "GET #show" do
    login(:user)
    let(:lot) { create :lot, status: :in_process }
    let(:params) { { id: lot.id } }
    subject { get :show, params: params }

    context "when the record exists" do
      it "should status code 200" do
        subject
```

```
    expect(response).to have_http_status(200)
  end

  it "should return the lot" do
    subject
    expect(json).not_to be_empty
    expect(json[:id]).to eq(lot.id)
  end
end

context "when the record does not exist" do
  let(:params) { { id: 100 } }

  it "should return status code 404" do
    subject
    expect(response).to have_http_status(404)
  end

  it "should return a not found message" do
    subject
    expect(json).to eq error: "Couldn't find Lot with 'id'=100"
  end
end

context "should return proper json" do
  let(:fields) { super() + [:my_win] }
  before(:each) do
    lot.closed!
  end
```

```
  it "should use serializer" do
    subject
    expect(json).to include(*fields)
  end
end

context "when current user is winner" do
  before(:each) do
    create(:bid, lot: lot, proposed_price: lot.current_price + 1.0)
    create(:bid, lot: lot, user: @user, proposed_price: lot.current_price + 2.0)
    lot.closed!
  end

  it "should my_win is true" do
    subject
    expect(json[:my_win]).to eq true
  end
end

context "when current user user is not winner" do
  before(:each) do
    create(:bid, lot: lot, user: @user, proposed_price: lot.current_price + 1.0)
    create(:bid, lot: lot, proposed_price: lot.current_price + 2.0)
    lot.closed!
  end

  it "should my_win is false" do
    subject
    expect(json[:my_win]).to eq false
  end
```

```
  end
end

describe "POST #create" do
  login(:user)
  let(:title) { "Valid title" }
  subject { post :create, params: { lot: attributes_for(:lot, title: title) } }

  context "with valid attributes" do
    it "creates a new lot" do
      expect { subject }.to change(Lot, :count).by(1)
    end

    it "should status code 201" do
      is_expected.to have_http_status(201)
    end

    context "should return proper json" do
      it "should use serializer" do
        subject
        expect(json).to include(*fields)
      end
    end
  end

  context "with invalid attributes" do
    let(:title) { "" }

    it "does not save the new lot" do
      expect { subject }.to_not change(Lot, :count)
```

```
    end

    it "should status code 422" do
      is_expected.to have_http_status(422)
    end

    it "should contain right message" do
      subject
      expect(json).to eq error: "Validation failed: Title can't be blank"
    end
  end
end

describe "PUT #update" do
  let(:user) { create(:user) }
  let(:lot) { create :lot, user: user }
  let(:params) { { id: lot.id, lot: { title: "Title2" } } }
  before(:each) do
    login_by user
  end
  subject { put :update, params: params }

  context "with valid attributes" do
    it "changes lot attributes" do
      expect { subject }.to change { lot.reload.title }.to("Title2")
    end
  end

  context "with invalid attributes" do
    let(:params) { { id: lot.id, lot: { title: "" } } }
```

```
  it "does not change lot attributes" do
    expect { subject }.to_not change { lot.reload.title }
  end

  it "should status code 422" do
    is_expected.to have_http_status(422)
  end

  it "should contain right message" do
    subject
    expect(json).to eq error: "Validation failed: Title can't be blank"
  end
end

context "should return proper json" do
  it "should use serializer" do
    subject
    expect(json).to include(*fields)
  end
end

context "with status :in_process" do
  let(:lot) { create :lot, user: user, status: :in_process }

  it "does not change lot attributes" do
    subject
    lot.reload
    expect(lot.title).to_not eq("Title2")
  end
```

```
end

context "with status :closed" do
  let(:lot) { create :lot, user: user, status: :closed }

  it "does not change lot attributes" do
    subject
    lot.reload
    expect(lot.title).to_not eq("Title2")
  end
end

context "with not valid user" do
  let!(:user2) { create :user }
  let!(:user2_login) { login_by user2 }

  it "do not change with no owner" do
    subject
    lot.reload
    expect(lot.title).to_not eq("Title2")
  end

  it "should status code 401" do
    subject
    expect(response).to have_http_status(401)
  end

  it "should error message" do
    subject
    expect(json).to eq error: "You are not have permission for this action"
```

```
    end
  end
end

describe "DELETE #destroy" do
  let(:user) { create(:user) }
  let!(:lot) { create :lot, user: user }
  before(:each) do
    login_by user
  end
  subject { delete :destroy, params: { id: lot.id } }

  it "deletes the lot" do
    expect { subject }.to change(Lot, :count).by(-1)
  end

  it "should return status code 204" do
    subject
    expect(response).to have_http_status(204)
  end

  context "with not valid user" do
    let!(:user2) { create :user }
    let!(:user2_login) { login_by user2 }

    it "do not change with no owner" do
      expect { subject }.to_not change(Lot, :count)
    end

    it "should status code 401" do
```

```
        subject
        expect(response).to have_http_status(401)
      end

      it "should error message" do
        subject
        expect(json).to eq error: "You are not have permission for this action"
      end
    end

    context "with status :in_process" do
      let(:lot) { create :lot, status: :in_process, user: user }

      it "do not delete with :in_progress status" do
        subject
        expect(response).to have_http_status(401)
      end
    end

    context "with status :closed" do
      let(:lot) { create :lot, status: :closed, user: user }

      it "do not delete with :closed status" do
        subject
        expect(response).to have_http_status(401)
      end
    end
  end
end
```

Файл orders_controller_spec.rb

```
require "rails_helper"

RSpec.describe OrdersController, type: :controller do
  let(:fields) do
    %i[id arrival_location arrival_type status created_at updated_at]
  end

  describe "#show" do
    login(:user)

    let(:lot) { create :lot, status: :in_process, user: @user }
    let!(:bid) { create :bid, lot: lot }
    let(:params) { { lot_id: lot.id } }

    before(:each) do
      lot.closed!
    end

    let!(:order) { create :order, lot: lot, bid: bid }

    subject { get :show, params: params }

    it "should use serializer" do
      subject
      expect(json).to include(*fields)
    end

    context "when the record exists" do
      it "should status code 200" do
```

```
    subject
    expect(response).to have_http_status(200)
  end

  it "should return the order" do
    subject
    expect(json).not_to be_empty
    expect(json[:id]).to eq(order.id)
  end
end

context "when the record does not exist" do
  let(:params) { { lot_id: 100 } }

  it "should return status code 404" do
    subject
    expect(response).to have_http_status(404)
  end

  it "should return a not found message" do
    subject
    expect(json).to eq error: "Couldn't find Order"
  end
end

context "when current user is not lot or order creator" do
  before(:each) do
    lot.update! user: create(:user)
  end
```

```
        it "should contain right message" do
          subject
          expect(json).to eq error: "You are not have permission for this action"
        end
      end
    end

    describe "#create" do
      let(:user) { create :user }
      let(:lot) { create :lot, status: :in_process }
      let(:location) { "location" }

      before(:each) do
        create(:bid, lot: lot, user: user)
        lot.closed!
      end

      subject { post :create, params: { order: { arrival_location: location, arrival_type:
:royal_mail },
                                        lot_id: lot.id } }

      context "when user authorized" do
        before(:each) do
          login_by user
        end

        context "with valid attributes" do
          it "creates a new order" do
            expect { subject }.to change(Order, :count).by(1)
          end
```

```
  it "should status code 201" do
    is_expected.to have_http_status(201)
  end

  it "should use serializer" do
    subject
    expect(json).to include(*fields)
  end

  it "should return proper json" do
    subject
    expect(json[:arrival_location]).to eq("location")
    expect(json[:arrival_type]).to eq("royal_mail")
  end
end

context "with invalid attributes" do
  let(:location) { "" }

  it "does not save the new order" do
    expect { subject }.to_not change(Order, :count)
  end

  it "should status code 422" do
    is_expected.to have_http_status(422)
  end

  it "should contain right message" do
    subject
```

```
          expect(json).to eq error: "Validation failed: Arrival location can't be blank"
        end
      end
    end

    context "when user not authorized" do
      it "should status code 401" do
        subject
        expect(response).to have_http_status(401)
      end

      it "should contain right message" do
        subject
        expect(json).to eq errors: ["You need to sign in or sign up before continuing."]
      end
    end
  end

  describe "#update" do
    login :user

    let(:bid_creator) { create :user }
    let(:lot) { create :lot, status: :in_process, user: @user }
    let!(:bid) { create :bid, lot: lot, user: bid_creator }
    let(:params) { { lot_id: lot.id } }
    let(:message_delivery) { instance_double(ActionMailer::MessageDelivery) }

    before(:each) do
      lot.closed!
```

```
        end

        let!(:order) { create :order, lot: lot, bid: bid }

        subject { put :update, params: params }

        context "when order status :pending and current user is order creator" do
          let(:params) { { lot_id: lot.id, order: { arrival_location: "new_location" } } }
          before(:each) do
            login_by bid_creator
          end

          it "should use serializer" do
            subject
            expect(json).to include(*fields)
          end

          context "with valid attributes" do
            it "changes order attributes" do
              expect { subject }.to change { order.reload.arrival_location }.to("new_location")
            end
          end

          context "with invalid attributes" do
            let(:params) { { lot_id: lot.id, order: { arrival_location: "" } } }

            it "does not change lot attributes" do
              expect { subject }.to_not change { order.reload.arrival_location }
            end
```

```
      it "should status code 422" do
        is_expected.to have_http_status(422)
      end

      it "should contain right message" do
        subject
        expect(json).to eq error: "Validation failed: Arrival location can't be blank"
      end
    end
  end

  context "when order status :pending and current user is lot owner" do
    it "should status code 200" do
      is_expected.to have_http_status(200)
    end

    it "changes order status to :sent" do
      expect { subject }.to change { order.reload.status }.to("sent")
    end

    it "should email queues the job" do
      expect(UserMailer).to
receive(:email_about_sending).and_return(message_delivery)
      allow(message_delivery).to receive(:deliver_later)
      subject
    end
  end

  context "when order status :sent and current user is order creator" do
```

```
        before(:each) do
          order.sent!
          login_by bid_creator
        end

        it "should status code 200" do
          is_expected.to have_http_status(200)
        end

        it "changes order status to :sent" do
          expect { subject }.to change { order.reload.status }.to("delivered")
        end

        it "should email queues the job" do
          expect(UserMailer).to
receive(:email_about_delivery).and_return(message_delivery)
          allow(message_delivery).to receive(:deliver_later)
          subject
        end
      end

      context "when current user is not lot owner or order creator" do
        let(:new_user) { create(:user) }

        before(:each) do
          login_by new_user
        end

        it "should status code 401" do
          is_expected.to have_http_status(401)
```

```
      end

      it "should contain right message" do
        subject
        expect(json).to eq error: "You are not have permission for this action"
      end
    end
  end
end
```

Папка spec/factories

Файл bids.rb

```
FactoryBot.define do
  factory :bid do
    proposed_price { Faker::Number.between(10.5, 30.4) }
    user
    association :lot, status: :in_process
  end
end
```

Файл lots.rb

```
FactoryBot.define do
  factory :lot do
    title { Faker::Food.fruits }
    current_price { Faker::Number.between(2.1, 10.5) }
    estimated_price { Faker::Number.between(30.5, 50.0) }
    lot_start_time { Faker::Date.forward(10) }
    lot_end_time { Faker::Date.between(11.days.from_now, 20.days.from_now) }
    user
  end
```

```
end
```

Файл orders.rb

```
FactoryBot.define do
  factory :order do
    arrival_location { Faker::Address.full_address }
    arrival_type { %i[pickup royal_mail united_states_postal_service dhl_express].sample }
    association :lot, status: :closed
  end
end
```

Файл users.rb

```
FactoryBot.define do
  factory :user do
    firstname { Faker::Name.first_name }
    lastname { Faker::Name.last_name }
    email { Faker::Internet.unique.email }
    phone { Faker::PhoneNumber.unique.cell_phone }
    birth_day { Faker::Date.birthday(21, 65) }
    password { "123456" }
    confirmed_at { Date.today }
  end
end
```

Папка spec/jobs

Файл bid_broadcast_job_spec.rb

```
require "rails_helper"

RSpec.describe BidBroadcastJob, type: :job do
```

```
  let(:param) { 1 }
  subject(:job) { described_class.perform_later param }

  it "queues the job" do
    expect { job }.to change(ActiveJob::Base.queue_adapter.enqueued_jobs, :size).by(1)
  end

  it "is in default queue" do
    expect(described_class.new.queue_name).to eq("default")
  end

  describe "#perform" do
    let(:bid) { create :bid }
    let(:bid_job) { described_class.new }

    it "should broadcasts bid" do
      expect { bid_job.perform(bid.id) }
        .to have_broadcasted_to("bids_for_lot_#{bid.lot_id}").from_channel(BidsChannel)
          .with(hash_excluding(BidSerializer.new(bid).to_json))
    end
  end
end
```

Файл close_lot_job_spec.rb

```
require "rails_helper"

RSpec.describe CloseLotJob, type: :job do
  let(:param) { 1 }
```

```
    subject(:job) { described_class.perform_later param }

    it "queues the job" do
      expect { job }.to change(ActiveJob::Base.queue_adapter.enqueued_jobs, :size).by(1)
    end

    it "matches with enqueued job" do
      expect {
        described_class.set(wait_until: Date.tomorrow.noon).perform_later(param)
      }.to have_enqueued_job.at(Date.tomorrow.noon)
    end

    it "is in default queue" do
      expect(described_class.new.queue_name).to eq("default")
    end

    describe "#perform" do
      let(:lot_status) { :in_process }
      let(:lot) { create :lot, status: lot_status }
      subject { described_class.new.perform lot.id }

      it "submit service was launched" do
        expect(Lot).to receive(:find).with(lot.id).and_return(lot)
        expect(lot).to receive(:close!)
        subject
      end

      context "with pending status" do
        let(:lot_status) { :pending }
```

```
        subject { described_class.new.perform lot.id }

        it "submit service was launched" do
          expect(Lot).to receive(:find).with(lot.id).and_return(lot)
          expect(lot).to_not receive(:close!)
          subject
        end
      end
    end
  end
```

Файл open_lot_job_spec.rb

```
require "rails_helper"

RSpec.describe OpenLotJob, type: :job do
  let(:param) { 1 }
  subject(:job) { described_class.perform_later param }

  it "queues the job" do
    expect { job }.to change(ActiveJob::Base.queue_adapter.enqueued_jobs, :size).by(1)
  end

  it "matches with enqueued job" do
    expect {
      described_class.set(wait_until: Date.tomorrow.noon).perform_later(param)
    }.to have_enqueued_job.at(Date.tomorrow.noon)
  end

  it "is in default queue" do
```

```
    expect(described_class.new.queue_name).to eq("default")
  end

  describe "#perform" do
    let(:lot_status) { :pending }
    let(:lot) { create :lot, status: lot_status }
    subject { described_class.new.perform lot.id }

    it "submit service was launched" do
      expect(Lot).to receive(:find).with(lot.id).and_return(lot)
      expect(lot).to receive(:in_process!)
      subject
    end

    context "with closed status" do
      let(:lot_status) { :closed }
      subject { described_class.new.perform lot.id }

      it "submit service was launched" do
        expect(Lot).to receive(:find).with(lot.id).and_return(lot)
        expect(lot).to_not receive(:in_process!)
        subject
      end
    end
  end
end
```

Папка spec/requests

Файл auth_spec.rb

```
require "rails_helper"
```

```
RSpec.describe "Auth", type: :request do
  describe "POST /auth" do
    let(:params) { attributes_for(:user) }
    subject { post "/auth", params: params }

    context "with correct attributes" do
      it "should status code 200" do
        subject
        expect(response).to have_http_status(200)
      end

      it "should create a new user" do
        expect { subject }.to change(User, :count).by(1)
      end

      it "should send a confirmation email" do
        expect { subject }.to change(Devise.mailer.deliveries, :count).by(1)
      end
    end

    context "with not correct attributes" do
      let(:params) { attributes_for(:user, password: "123") }

      it "should status code 422" do
        subject
        expect(response).to have_http_status(422)
      end

      it "should not create a new user" do
```

```
      expect { subject }.to_not change(User, :count)
    end

    it "should not send a confirmation email" do
      expect { subject }.to_not change(Devise.mailer.deliveries, :count)
    end
  end
end

describe "POST /auth/sign_in" do
  let(:user) { create(:user) }
  let(:params) { { email: user.email, password: "123456" } }
  subject { post "/auth/sign_in", params: params }

  context "with correct attributes" do
    it "should status code 200" do
      subject
      expect(response).to have_http_status(200)
    end
  end

  context "with not correct password" do
    let(:params) { { email: user.email, password: "123123" } }

    it "should status code 401" do
      subject
      expect(response).to have_http_status(401)
    end
  end
```

```
  context "with not correct email" do
    let(:params) { { email: "email@gmal.com", password: "123456" } }

    it "should status code 401" do
      subject
      expect(response).to have_http_status(401)
    end
  end
end

describe "POST /auth/sign_out" do
  let(:headers) { create(:user).create_new_auth_token }
  subject { delete "/auth/sign_out", headers: headers }

  context "when headers present" do
    it "should status code 200" do
      subject
      expect(response).to have_http_status(200)
    end
  end

  context "when headers are not present" do
    let(:headers) { {} }

    it "should not be successful" do
      subject
      expect(response).to_not be_successful
    end
  end
end
```

```
end
```

Папка spec/linters

Файл rubocop_spec.rb

```
require "spec_helper"

RSpec.describe "rubocop analysis" do
  subject(:report) { `rubocop` }

  it "has no offenses" do
    expect(report).to match(/no\ offenses\ detected/)
  end
end
```

Папка db/migrate

Файл devise_token_auth_create_users.rb_spec.rb

```
class DeviseTokenAuthCreateUsers < ActiveRecord::Migration[5.2]
  def change

    create_table(:users) do |t|
      ## Required
      t.string :provider, :null => false, :default => "email"
      t.string :uid, :null => false, :default => ""

      ## Database authenticatable
      t.string :encrypted_password, :null => false, :default => ""

      ## Recoverable
      t.string   :reset_password_token
      t.datetime :reset_password_sent_at
      t.boolean  :allow_password_change, :default => false
```

```
      ## Rememberable
      t.datetime :remember_created_at

      ## Confirmable
      t.string   :confirmation_token
      t.datetime :confirmed_at
      t.datetime :confirmation_sent_at
      t.string   :unconfirmed_email # Only if using reconfirmable

      ## Lockable
      # t.integer  :failed_attempts, :default => 0, :null => false # Only if lock strategy
is :failed_attempts
      # t.string   :unlock_token # Only if unlock strategy is :email or :both
      # t.datetime :locked_at

      ## User Info
      t.string :firstname
      t.string :lastname
      t.string :phone
      t.date :birth_day
      t.string :email

      ## Tokens
      t.text :tokens

      t.timestamps
    end

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

```
    add_index :users, :phone,                unique: true
    add_index :users, [:uid, :provider],     unique: true
    add_index :users, :reset_password_token, unique: true
    add_index :users, :confirmation_token,   unique: true
    # add_index :users, :unlock_token,       unique: true
  end
end
```

Файл create_lots.rb_spec.rb

```
class CreateLots < ActiveRecord::Migration[5.2]
  def change
    create_table :lots do |t|
      t.references :user, foreign_key: true
      t.string :title
      t.string :image
      t.text :description
      t.integer :status, default: 0, null: false
      t.decimal :current_price
      t.decimal :estimated_price
      t.datetime :lot_start_time
      t.datetime :lot_end_time

      t.timestamps
    end
  end
end
```

Файл create_bids.rb_spec.rb

```
class CreateBids < ActiveRecord::Migration[5.2]
  def change
```

```
    create_table :bids do |t|
      t.references :user, foreign_key: true
      t.references :lot, foreign_key: true
      t.datetime :bid_creation_time
      t.decimal :proposed_price

      t.timestamps
    end
  end
end
```

Файл create_orders.rb_spec.rb

```
class CreateOrders < ActiveRecord::Migration[5.2]
  def change
    create_table :orders do |t|
      t.references :user, foreign_key: true
      t.references :lot, foreign_key: true
      t.text :arrival_location
      t.integer :arrival_type, default: 0, null: false
      t.integer :status, default: 0, null: false

      t.timestamps
    end
  end
end
```

Файл remove_bid_creation_time_from_bids.rb

```
class RemoveBidCreationTimeFromBids < ActiveRecord::Migration[5.2]
  def change
    remove_column :bids, :bid_creation_time, :datetime
```

```
  end
end
```

Файл change_orders_ref.rb

```
class ChangeOrdersRef < ActiveRecord::Migration[5.2]
  def change
    remove_reference :orders, :user, foreign_key: true
    add_reference :orders, :bid, foreign_key: true
  end
end
```

Файл remove_default_for_arrival_type.rb

```
class RemoveDefaultForArrivalType < ActiveRecord::Migration[5.2]
  def change
    change_column_default(:orders, :arrival_type, from: 0, to: nil)
  end
end
```

Папка db

Файл seeds.rb

This file should contain all the record creation needed to seed the database with its default values.

The data can then be loaded with the rails db:seed command (or created alongside the database with db:setup).

```
require 'factory_bot_rails'
require 'database_cleaner'

DatabaseCleaner.strategy = :truncation
```

```
DatabaseCleaner.clean

quantity = 10

#User
FactoryBot.create_list(:user, quantity)
puts "Created #{quantity} users"

#Lot
FactoryBot.create_list(:lot, quantity, user: User.first, status: :in_process)
puts "Created #{quantity} lots"

#Bid
Lot.all.each do |lot|
  FactoryBot.create(:bid, lot: lot, user: User.last)
end
puts "Created #{quantity} bids"

#Order
Lot.all.each do |lot|
  lot.closed!
  FactoryBot.create(:order, lot: lot, bid: lot.bids.last)
end
puts "Created #{quantity} orders"
```