Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ”

Кафедра інформатики факультету інформатики

Курсова робота на тему:
Новий підхід до client-side розробки в Rails 7 - Hotwire, Turbo та Stimulus

Керівник курсової роботи:
Захоженко П.О.
(прізвище та ініціали)

_____
(підпис)
“_____”_____ 2022 р.

Виконав студент 4-го року навчання
спеціальності “Комп’ютерні науки”
Декрет Владислав Володимирович
(ПІБ)

Київ – 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ”

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри математики, проф.
д,ф-м.н.
_____ Б.В. Олійник
(підпис)
“_____”_____ 2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студенту 4-го курсу факультету інформатики
Декрет Владиславу Володимировичу

Тема: Новий підхід до client-side розробки в Rails 7 - Hotwire, Turbo та Stimulus.

Вихідні дані: Створено веб-чат на основі Rails 7.

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

1 Анотація
2 Вступ
3 Основні означення та попередні результати
4 Основні результати
5 Висновки та література

Дата видачі “ _____”_____ 2022 р. Керівник _____
(підпис)

Завдання отримав _____
(підпис)

Тема: Новий підхід до client-side розробки в Rails 7 - Hotwire, Turbo та Stimulus.

Календарний план виконання роботи:

Номер	Назва етапу курсової	Термін виконання етапу	Примітка
1.	Отримання теми курсової роботи.	04.10.2021	
2.	Ознайомлення з темою курсової.	11.10.2021	
3.	Розробка плану та структури роботи.	11.10.2021	
4.	Робота з науковою літературою	.10.21-.02.22	
5.	Написання робочої програми	05.05.2022	
6.	Робота над текстовим оформленням результатів.	10.05.2022	
7.	Попередній аналіз курсової. Виправлення помилок.	18.05.2022	

Зміст

1 Анотація 5

2 Вступ 6

3 Основні означення та попередні результати 8

3.1 Означення 8

3.2 Gemfile 9

3.3 База даних 10

4 Основні результати 13

4.1 Функціональність HotwireChat для неавторизованого користувача . . 13

4.2 Функціональність HotwireChat для авторизованого користувача 15

4.3 Hotwire, Turbo та Stimulus технології 16

4.4 Використання Turbo в додатку 18

4.5 Використання Stimulus в додатку 19

4.6 Реалізація декоратору за допомогою Draper 21

4.7 Реалізація автентифікації та реєстрації за допомогою Devise 22

4.8 Десктопна версія додатку 23

4.9 Мобільна версія додатку 24

5 Висновки 25

Література 26

1 Анотація

У даній курсовій роботі розглядаються й аналізуються глобальні нововведення Rails 7.

Написано веб-чат, який тестує його front-end нововведення відносно шостої версії: Hotwire, Turbo, Stimulus. Також тестується їх взаємодія з back-end гемами, а саме: Devise, Draper, Active Record тощо.

Використано для декорації лайку гем draper.

Використовуємо esbuild замість webpack.

Замість bootstrap – tailwindcss.

Реляційна база даних - PostgreSQL.

Реалізований інтерфейс під десктопні та мобільні девайси окремо.

2 Вступ

У 2021-ому році випустили 7-ьому версію Rails, де роблять основний акцент на single-page application підхід client-side розробки, використовуючи Hotwire, Turbo та Stimulus.

Hotwire підтримує візуалізацію на стороні сервера та забезпечує більш простий і продуктивний досвід розробки, який Rails завжди втілював у своїх технологіях, не жертвуючи при цьому швидкістю чи реагуванням пов'язаним з традиційним single-page додатком.

Зміни у front-end:

1) JavaScript в Rails 7 більше не потребує NodeJS чи Webpack, але за потреби без них все ще можна використовувати пакети npm.
2) У цій версії дефолтно встановлюється importmaps-rails гем. Замість того, щоб писати в package.json та встановлювати залежності за допомогою npm або yarn, ми використовуємо ./bin/importmap CLI для роботи з залежностями.
3) Отримують Turbo і Stimulus (від Hotwire) за замовчуванням, замість Turbolinks і UJS.

Зміни в back-end:

1) У Rails 7 певні поля бази даних можуть бути зашифровані за допомогою encrypts методу. Розшифрування і шифрування проводиться автоматично між базою даних та додатком.
2) Додали асинхронний запит load_async. Це особливо важливо, коли потрібно завантажити кілька не пов'язаних запитів з дії контролера.
3) Усі додатки Rails 7 будуть запускатись через Zeitwerk, а не класичний завантажувач.

В цілому оновлення досить масштабне і потребує адаптації та дослідження в нових гемах та методах.

Rails 7 вже має високу популярність серед Ruby програмістів, які вже півроку тестують цю версію та в цілому задоволені нею. Проте цього не робили ми, тому приступимо.

3 Основні означення та попередні результати

3.1 Означення

Ruby on Rails (скорочено rails або RoR) – фреймворк для ruby, для створення веб додатків на базі MVC.

MVC (Model View Controller) – патерн, суть якого в розподіленні задач веб-додатку між моделлю, контролером та вьюшкою. Контролер відповідає за основну логіку додатку, модель за базу даних та зв’язки у ній, вьюшка – front-end.

Gem (в подальшому в роботі гем) – пакет (файл) з бібліотекою чи додатком. Має стандартизований вигляд і зберігається в мережі.

Active Record – відповідає за модель в MVC Rails, є шаром у системі, відповідальним за представлення бізнес-логіки та даних. Active Record спрощує створення та використання бізнес-об'єктів, переварюючи їх в зручні для ООП об’єкти. Сама собою ця реалізація патерна Active Record є описом системи ORM.

ORM (Object-Relational Maping) – технологія програмування, яка представляє реляційну базу даних у вигляді об’єктів ООП і зв’язків між ними.

Автентифікація – перевірка на ідентичність користувача. Зазвичай запитується логін та пароль (sign in).

Авторизація – перевірка на роль та доступ користувача після автентифікації.

Реєстрація – створення акаунту в базі даних (sign up).

ESM – стандартна модульна система Javascript.

WebSocket – протокол для обміну інформацією між браузером та вебсервером наживо.

Single Page Application (SPA) – це тип веб-застосунків, в яких завантаження необхідного коду відбувається на одну сторінку. Це дозволяє заощадити час на повторне завантаження тих самих елементів.

Search Engine Optimization (SEO) – пошукова оптимізація сайту.

3.2 Gemfile

У цьому додатку я використовув ruby версії 3.0.1. працюючи у WSL.

Геми:

rails 7.0.1 – Ruby on Rails 7-ої версії, оскільки саме ця версія нам потрібна для дефолтного Hotwire;

sprockets-rails – це гем для компіляції та обслуговування web-assets. Він дозволяє організувати файли JavaScript та CSS на менші, більш керовані фрагменти, які можна розподілити по ряду каталогів і файлів. Sprockets надає структуру та методи включення assets у проект;

pg – надає ruby інтерфейс для використовування PostgreSQL;

puma – створює високопродуктивний сервер для програм Ruby, заснований на веб-сервері Mongrel, дозволяє встановити мінімальну і максимальну кількість потоків, а також підтримує кластерний режим, в якому ви можете використовувати розгалужені процеси для одночасної обробки запитів;

importmap-rails – дає можливість використовувати ESM, щоб керувати сучасним Javascript у Rails без транспіляції або бандлінгу;

turbo-rails – реалізує на фронті single-page веб-додаток зменшуючи до мінімума використання Javascript у ньому;

stimulus-rails – Javascript фреймворк для HTML;

tailwindcss-rails – інтегрує Tailwind CSS у Rails;

redis – для зберігання деяких тимчасових даних під час перегляду WebSocket;

devise – гнучкий шаблон для аутентифікації в Rails;

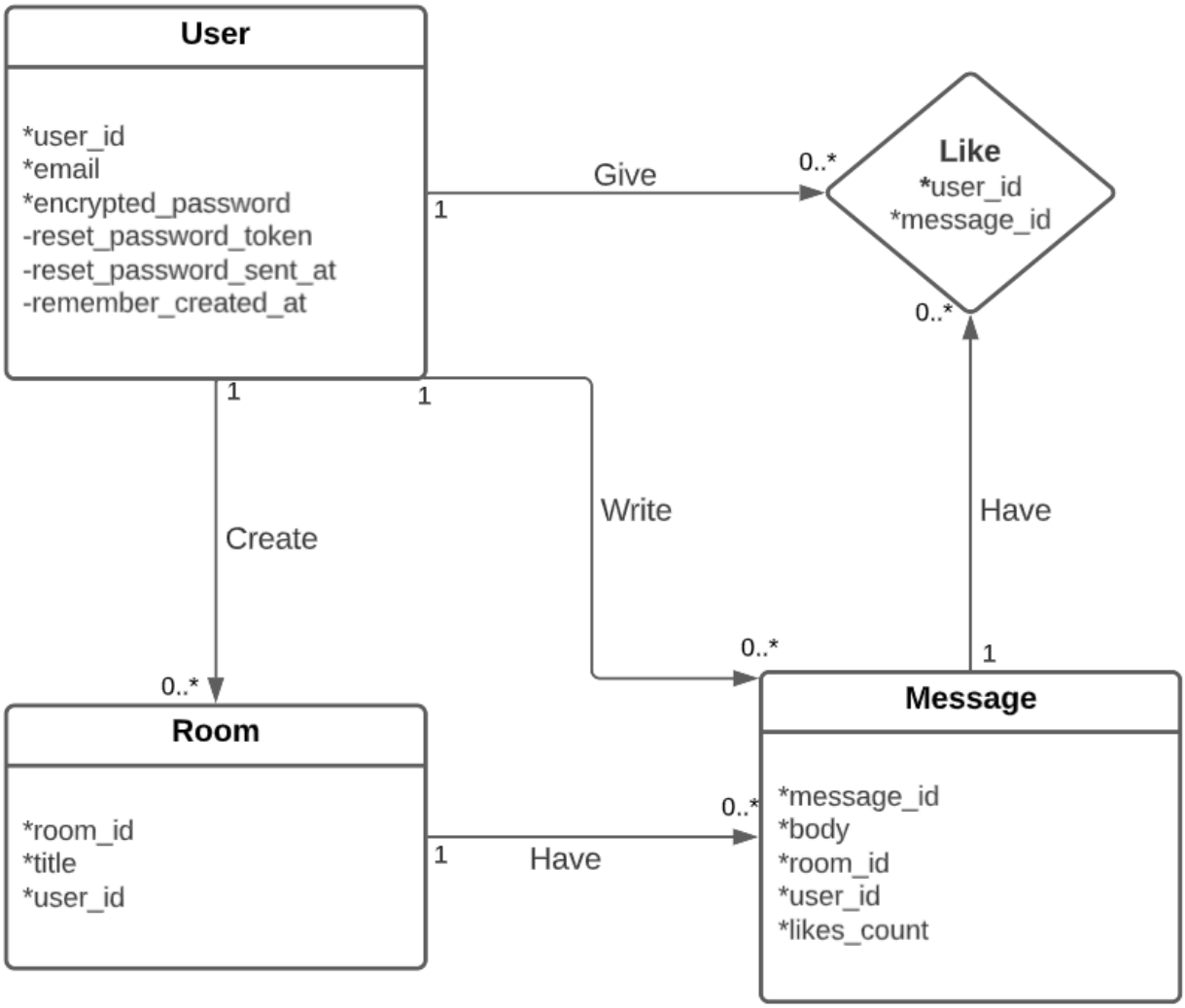
draper – надає можливість створювати декоратори для View.

3.3 База даних

Реляційну базу даних реалізовано в PostgreSQL за допомогою гему Active Record.

База даних мала ім’я hotwirechat_development.

ER-модель:



Усього було чотири таблички: **User**, **Room**, **Message**, **Like**.

Таблиця **User** має зв’язок багато до багатьох з табличкою **Message** через табличку **Like**, цей зв’язок втілює лайкання повідомлення юзером.

Таблиця **User** має зв’язок один до багатьох з табличкою **Message**, цей зв’язок втілює написання повідомлення юзером.

Таблиця **User** має зв’язок один до багатьох з табличкою **Room**, цей зв’язок втілює створення нової кімнати юзером.

Таблиця **Room** має зв’язок один до багатьох з табличкою **Message**, цей зв’язок показує, що кімната має багато повідомлень.

User							
№	Ключ	Ім’я атрибуту	Тип	null / not null	on delete	on update	Пояснення
1	PK	user_id	bigint	nn			первинний ключ юзера
2		email	string	nn			email юзера
3		encrypted_ password	string	nn			зашифрований devise пароль
4		reset_ password_ token	string	n			токен необхідний devise для reset password
5		reset_ password_ sent_at	datetime	n			атрибут необхідний devise для reset password
6		remember_ created_at	datetime	n			атрибут необхідний devise

Room							
№	Ключ	Ім’я атрибуту	Тип	null / not null	on delete	on update	Пояснення
1	PK	room_id	bigint	nn			первинний ключ кімнати

2	FK	user_id	bigint	nn	cascade	cascade	foreign key юзера, реалізує зв’язок з таблицею User
3		title	string	nn			заголовок

Message							
№	Ключ	Ім’я атрибуту	Тип	null / not null	on delete	on update	Пояснення
1	PK	message_id	bigint	nn			первинний ключ повідомлення
2	FK	room_id	bigint	nn	cascade	cascade	foreign key кімнати, реалізує зв’язок з таблицею Room
3	FK	user_id	bigint	nn	cascade	cascade	foreign key юзера, реалізує зв’язок з таблицею User
4		body	string	nn			текст повідомлення
5		likes_count	bigint	nn			кількість лайків

Like							
№	Ключ	Ім’я атрибуту	Тип	null / not null	on delete	on update	Пояснення
1	PK/FK	message_id	bigint	nn	cascade	cascade	foreign key повідомлення, реалізує зв’язок з таблицею Message
2	PK/FK	user_id	bigint	nn	cascade	cascade	foreign key юзера, реалізує зв’язок з таблицею User

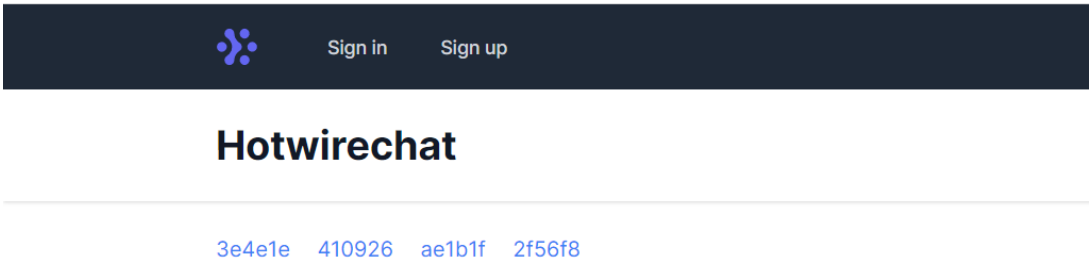
4 Основні результати

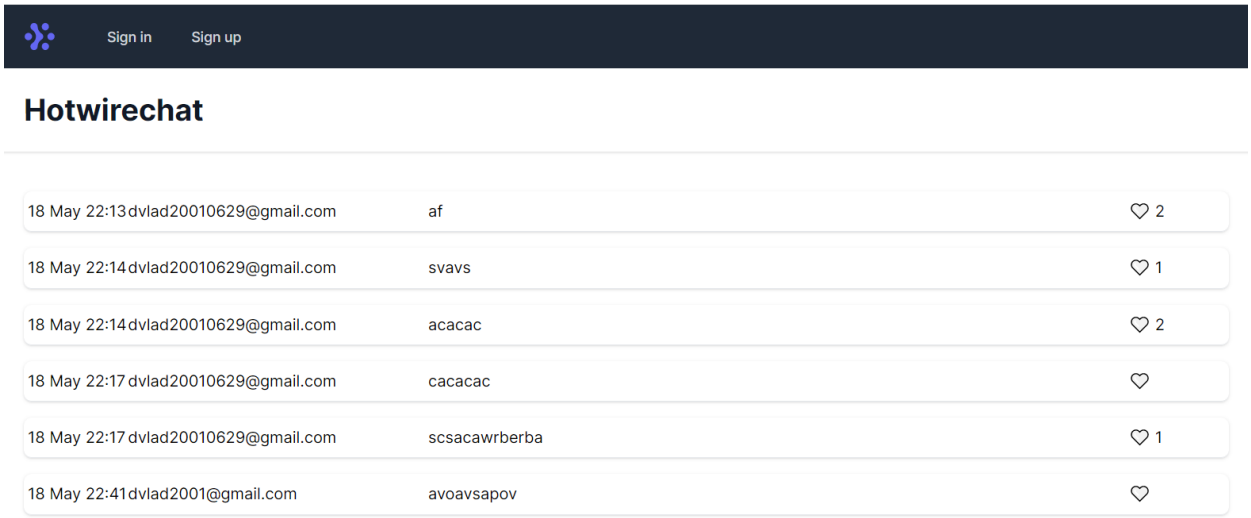
4.1 Функціональність HotwireChat для неавторизованого користувача

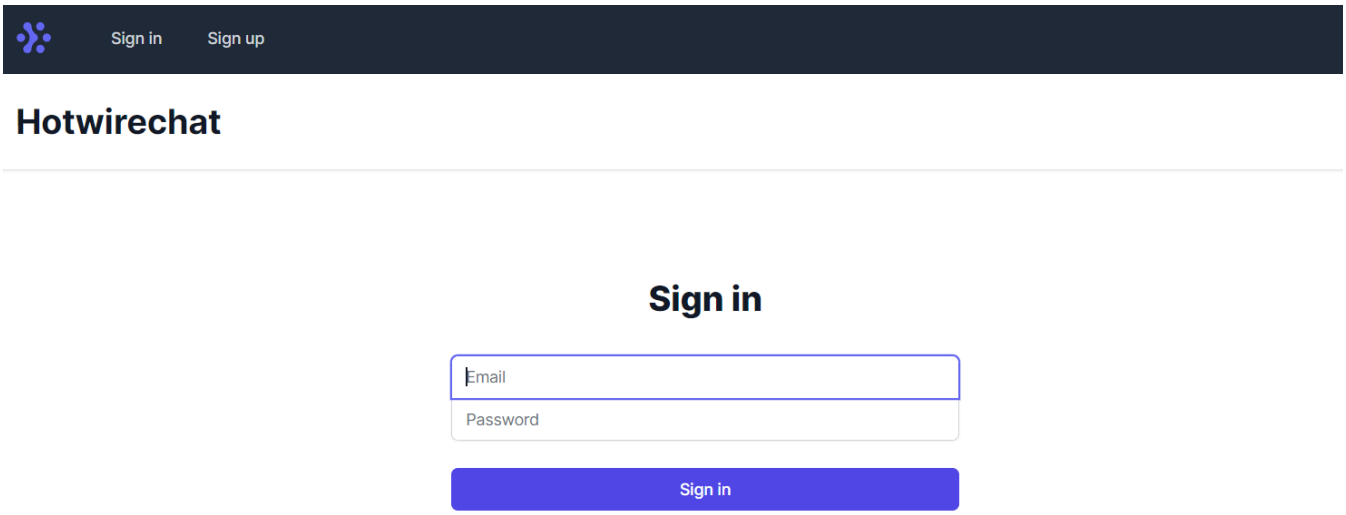
Не авторизований юзер може зайти у будь-яку кімнату та лише переглядати повідомлення, бачити автора і лайки в кожній з них.

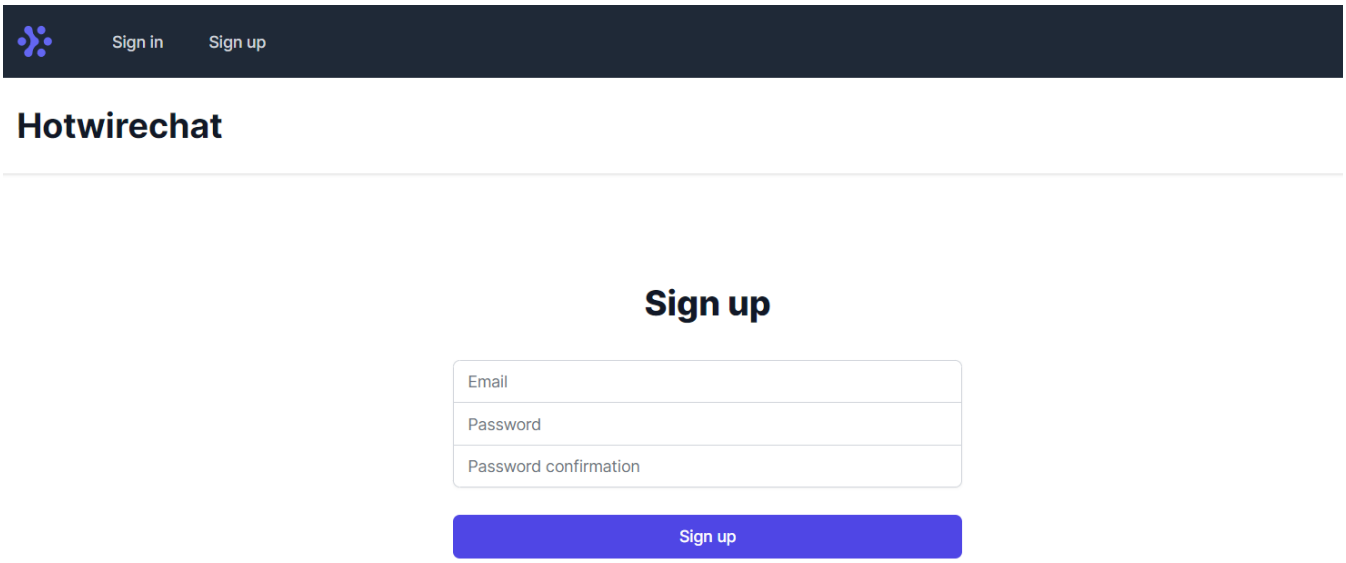
Звісно у нього є опція Sign in та Sign up, щоб автентифікуватись або зареєструватись.

Якщо потрібно повернутись до головної сторінки, то потрібно натиснути на емблему додатку.

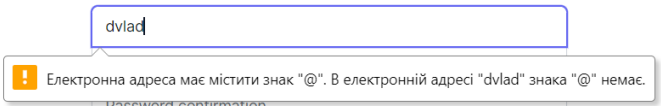


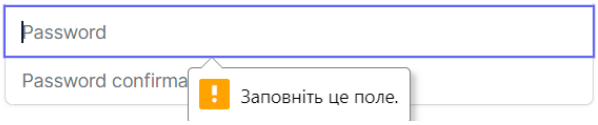






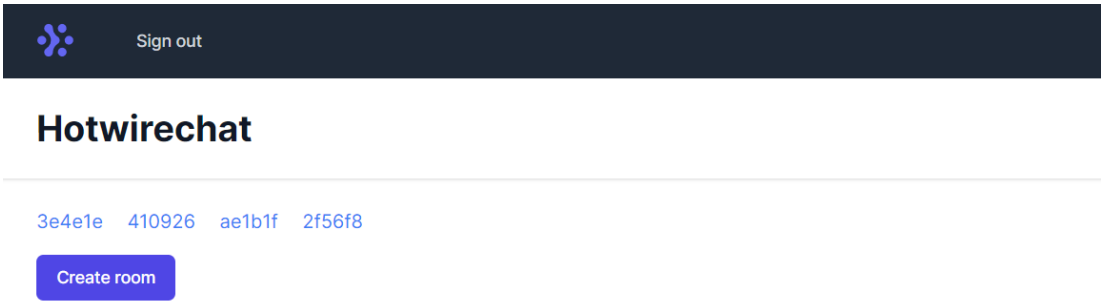
Для реєстрації необхідний валідний email та наявність паролю, який займає більше, ніж 6 символів. Зрозуміло, що пароль перевірки має співпадати з паролем вище.

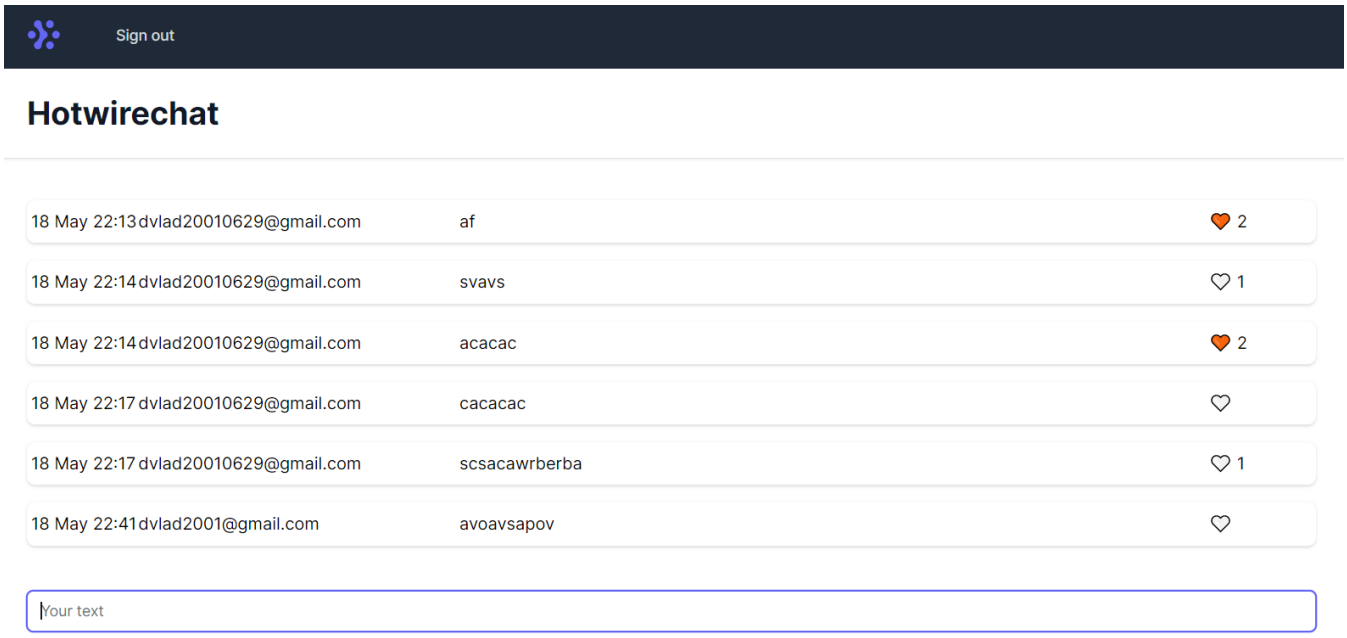




4.2 Функціональність HotwireChat для авторизованого користувача

Авторизований юзер може створити нову кімнату, чи зайти в уже створені і залишити там повідомлення, поставити лайк тощо. Також він може закінчити сесію за допомогою Sign out.





Якщо сердечко червоне, то це означає, що лайк ставили саме ви.

4.3 Hotwire, Turbo та Stimulus технології

За допомогою **Hotwire** ми можемо створювати веб-додатки без використання JavaScript надсилаючи лише HTML по дроту замість звичного в **Rails 6** JSON, що значно прискорює завантаження сторінок та подання форм.

Він підтримує візуалізацію на стороні сервера та забезпечує більш простий і продуктивний досвід розробки, який **Rails** завжди втілював у своїх технологіях, не жертвуючи при цьому швидкістю чи реагуванням пов'язаним з традиційним single-page додатком.

Основним двигуном **Hotwire** є гем **Turbo**. Це набір додаткових методів, які прискорюють навігацію сторінками та подання форм, поділяють складні сторінки на компоненти та передають часткові оновлення сторінок через **WebSocket** (який складається з **ActionCable**, каналів і потокових даних).

Hotwire використовує server-side rendering для вирішення деяких проблем, пов'язаних зі SPA, зберігаючи його основні переваги. SSR повертає процес візуалізації, переносячи частину зусиль рендерингу SPA на сервер, що схоже на традиційне завантаження. SSR може забезпечити користувачам більш ефективне завантаження програми, оскільки частина візуалізації виконується на сервері. Окрім можливості покращення продуктивності, це допомагає впоратися з деякими проблемами SEO, такими як індексація.

Hotwire досить простий у використанні, для нього потрібен стандартний пакет проекту **Rails** (**Ruby**, **RoR**, **ActionCable** та **WebSocket**), гем **Turbo**, який завантажує залежності JS, і **Redis** для зберігання деяких тимчасових даних під час перегляду **WebSocket**.

Завдяки **Hotwire** не потрібно вивчати іншу мову (JavaScript), щоб мати швидкість односторінкового веб додатку, оскільки **Turbo** доповнюється **Turbo Drive**, **Frames**, **Streams** і **Native**, де:

Turbo Drive - прискорює надсилання посилань і форм, усуваючи необхідність повного перезавантаження сторінки.

Turbo Frames - розкладає сторінки на незалежні контексти, що включають у собі навігацію та можуть ліниво завантажуватись.

Turbo Streams - забезпечують зміни сторінок через **WebSocket**, SSE або у відповідь на подання форми використовувати лише HTML та схожих на CRUD операцій.

Turbo Native – дає можливість писати гібридні (з елементами web і native) веб-додатки в оболонці iOS та Android.

Все це робиться шляхом надсилання HTML по дроту. А в тих випадках, коли необхідне щось, що є лише в JavaScript, можемо довершити роботу за допомогою **Stimulus**.

Stimulus фреймворк JavaScript для HTML, призначений для покращення сучасної програми **Rails**, працюючи з HTML, що згенерований на стороні сервера. Стан живе в Document Object Model (DOM), а фреймворк пропонує стандартні способи взаємодії з елементами та подіями в DOM. Він працює пліч-о-пліч з **Turbolinks**, щоб покращити продуктивність і час завантаження за допомогою коду, який обмежений і призначений для чітко визначеної мети.

4.4 Використання Turbo в додатку

Приклади використання Turbo у веб-додатку:

```
<%= turbo_stream_from :rooms %>

<%= turbo_frame_tag :rooms do %>
  <%= render @rooms %>
<% end %>
```

Код належить файлу views/rooms/index.html.erb

Тут *<%= turbo_stream_from :rooms %>* забезпечує показ усіх доступних кімнат на головній сторінці. При створенні нової або видаленні він зробить аналогічні дії на сторінці.

А *<%= turbo_frame_tag :rooms %>* прив'язує до кожного title, які показуються на основній сторінці, лінку, яка веде до особистої сторінки відповідної кімнати.

```
<%= turbo_stream_from @room %>
```

Код належить файлу views/rooms/show.html.erb

Тут *<%= turbo_stream_from @room %>* забезпечує показ коректної сторінки користувачу, тобто сторінки, яка відповідає title.

```
<%= link_to "Sign out", destroy_user_session_path, data: { "turbo-method": :delete },
  class: "text-gray-300 hover:bg-gray-700 hover:text-white px-3 py-2 rounded-md text-sm font-medium"
%>
```

Код належить файлу views/layouts/application.html.erb

Тут присутній *{ "turbo-method": :delete }*, який реалізує схожу на delete з CRUD операцію над сесією юзера, тобто забезпечує Sign out.

4.5 Використання Stimulus в додатку

Приклад використання Stimulus в додатку:

```
import { Controller } from "@hotwired/stimulus"
import { cable } from "@hotwired/turbo-rails"

You, yesterday | 1 author (You)
export default class extends Controller {
  connect() {
    this.subscribe()
    this.scrollMessages()
  }

  subscribe() {      You, yesterday • add first logic code …
    const turboStreamFromTag = document.querySelector("turbo-cable-stream-source")
    const channelName = turboStreamFromTag.getAttribute("channel")
    const signedStreamName = turboStreamFromTag.getAttribute("signed-stream-name")

    const scrollMessages = this.scrollMessages.bind(this)

    this.channel = cable.subscribeTo({ channel: channelName, signed_stream_name: signedStreamName }, {
      received(data) {
        setTimeout(scrollMessages, 100)
      }
    })
  }

  clearInput() {
    this.element.reset()
  }
```

```
  clearInput() {
    this.element.reset()
  }      You, yesterday • add first logic code …

  scrollMessages() {
    const chatContainer = document.getElementById("chat-container")
    if (chatContainer) chatContainer.scrollTop = chatContainer.scrollHeight
  }
}
```

Код належить файлу javascript/controllers/chat_refresh_controller.js

Клас реалізовує scroll і оновлення повідомлень чату.

Де turbo-cable-stream-source відображаємо забезпечує отримання title, за яким він розуміє, що це за кімната.

```
import ChatRefreshController from "./chat_refresh_controller"
application.register("chat_refresh", ChatRefreshController)
```

Код належить файлу javascript/controllers/index.js

Під’єднаємо клас ChatRefreshController, що ми написали за допомогою Stimulus, в index.js

```
<%= form_with model: @new_message, data: { controller: "chat_refresh", action: "turbo:submit-end->chat_refresh#clearInput"
```

Код належить файлу views/rooms/show.html.erb

У show.html.erb вже реалізовуємо даний контролер.

Який і буде забезпечувати нам scroll та оновлення сторінки за допомогою даного контролера написаного на Stimulus.

4.6 Реалізація декоратору за допомогою Draper

За допомогою цього гему я створював декоратор для лайків:

```
class MessageDecorator < ApplicationDecorator
  delegate_all

  def heart(user)
    if object.likes.find_by(user: user).present?
      "❤️"
    else
      "🤍"
    end
  end

  def likes_count
    object.likes_count if object.likes_count.positive?
  end
end      You, yesterday • add first logic code …
```

Код належить файлу decorators/message_decorator.rb

Реалізовуємо логіку наявності лайка в даному класі та вставляємо потрібне емоджі.

```
def show
  @room = Room.find_by!(title: params[:title])
  @messages = MessageDecorator.decorate_collection(@room.messages.includes(:user))
  @new_message = current_user&.messages&.build(room: @room)
end
```

Код належить файлу controllers/rooms_controller.rb

Декоруємо колекцію меседжів даної кімнати в методі show.

Гем draper дає можливість вивести емоджі правильно.

4.7 Реалізація автентифікації та реєстрації за допомогою Devise

Реалізація реєстрації:

```
<%= form_for(resource, as: resource_name, url: registration_path(resource_name), html: { class: "mt-8 space-y-6" }) do |f| %>
  <div class="rounded-md shadow-sm -space-y-px">
    <div>
      <%= f.label :email, class: "sr-only" %>
      <%= f.email_field :email, autofocus: true, autocomplete: "email", required: true, placeholder: "Email", class: "appearance-none
    </div>
    <div>
      <%= f.label :password, class: "sr-only" %>      You, yesterday • add first logic code …
      <%= f.password_field :password, autocomplete: "new-password", required: true, placeholder: "Password", class: "appearance-none
    </div>
    <div>
      <%= f.label :password_confirmation, class: "sr-only" %>
      <%= f.password_field :password_confirmation, autocomplete: "new-password", required: true, placeholder: "Password confirmation"
    </div>
  </div>

  <div>
    <%= f.submit "Sign up", class: "group relative w-full flex justify-center py-2 px-4 border border-transparent text-sm font-medium
  </div>
<% end %>
```

Код належить файлу views/devise/registrations/new.html.erb

Усі контролери з гему devise, що значно пришвидшує програмування реєстрації.

Реалізація автентифікації:

```
<%= form_for(resource, as: resource_name, url: session_path(resource_name), html: { class: "mt-8 space-y-6" }) do |f| %>
  <div class="rounded-md shadow-sm -space-y-px">
    <div>
      <%= f.label :email, class: "sr-only" %>
      <%= f.email_field :email, autofocus: true, autocomplete: "email", required: true, placeholder: "Email", class: "appearance-none
    </div>
    <div>
      <%= f.label :password, class: "sr-only" %>
      <%= f.password_field :password, autocomplete: "current-password", required: true, placeholder: "Password", class: "appearance-n
    </div>
  </div>
  You, yesterday • add first logic code …
  <div>
    <%= f.submit "Sign in", class: "group relative w-full flex justify-center py-2 px-4 border border-transparent text-sm font-medium
  </div>
<% end %>
```

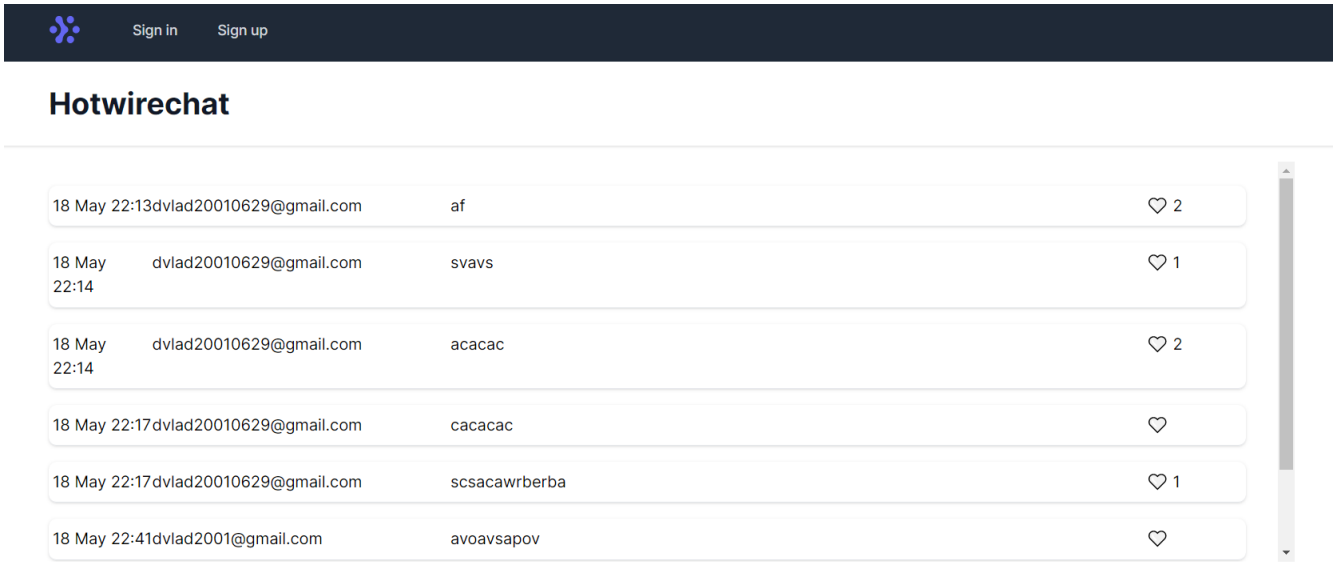
Код належить файлу views/devise/sessions/new.html.erb

Усі контролери з гему devise, що значно пришвидшує програмування автентифікації.

4.8 Десктопна версія додатку

Реалізований інтерфейс для десктопних девайсів.

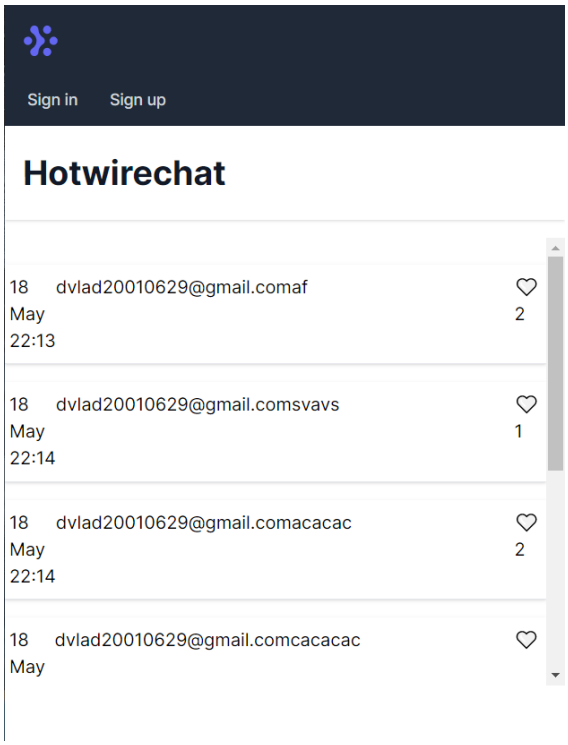
```
<div class="min-h-full">
  <nav class="bg-gray-800">
    <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
      <div class="flex items-center justify-between h-16">
        <div class="flex items-center">
          <div class="flex-shrink-0">
            <a href="/">
              <img class="h-8 w-8" src="https://tailwindui.com/img/logos/workflow-mark-indigo-500.svg" alt="Workflow">
            </a>
          </div>
          <div class="hidden md:block">
            <div class="ml-10 flex items-baseline space-x-4">
              <% if current_user.present? %>
                <%= link_to "Sign out", destroy_user_session_path, data: { "turbo-method": :delete },
                  class: "text-gray-300 hover:bg-gray-700 hover:text-white px-3 py-2 rounded-md text-sm font-medium"
                %>
              <% else %>
                <%= link_to "Sign in", new_user_session_path, class: "text-gray-300
                  hover:bg-gray-700 hover:text-white px-3 py-2 rounded-md text-sm font-medium
                  "%>        You, yesterday • add first logic code …
                <%= link_to "Sign up", new_user_registration_path, class: "text-gray-300
                  hover:bg-gray-700 hover:text-white px-3 py-2 rounded-md text-sm font-medium
                  "%>
              <% end %>
            </div>
          </div>
        </div>
      </div>
    </div>
```



4.9 Мобільна версія додатку

Реалізований інтерфейс для мобільних девайсів.

```
    <!-- Mobile menu, show/hide based on menu state. -->
    <div class="md:hidden" id="mobile-menu">
      <div class="px-2 pt-2 pb-3 space-y-1 sm:px-3">
        <% if current_user.present? %>
          <%= link_to "Sign out", destroy_user_session_path, data: { "turbo-method": :delete },
            class: "text-gray-300 hover:bg-gray-700 hover:text-white px-3 py-2 rounded-md text-sm f
          %>
        <% else %>
          <%= link_to "Sign in", new_user_session_path, class: "text-gray-300
            hover:bg-gray-700 hover:text-white px-3 py-2 rounded-md text-sm font-medium
            "%>
          <%= link_to "Sign up", new_user_registration_path, class: "text-gray-300
            hover:bg-gray-700 hover:text-white px-3 py-2 rounded-md text-sm font-medium
            "%>
        <% end %>
      </div>
    </div>
  </nav>
```



5 Висновки

У даній курсовій роботі я дослідив та аналізував нововведення у front-end-і **Rails 7**, а саме: **Hotwire**, **Turbo** та **Stimulus**. Також дослідили їх взаємодію з гемами: **Devise**, **Draper**, **Active Record**, **TailwindCSS**.

Написав веб-чат з автентифікацією, яка була реалізована за допомогою **Devise**, з красивим інтерфейсом використовуючи **TailwindCSS** та **Draper**, з PostgreSQL базою даних через гем **Pg**. Реалізував scroll та оновлення сторінки повідомлень в кімнаті за допомогою гему **Stimulus**. Навігацію між сторінками і пов'язання контенту я реалізував за допомогою гему **Turbo**.

Single-page application підхід був реалізований **Hotwire**.

Порівнюючи SPA, як основний підхід для client-side розробки в **Rails 7**, з SPA+API бачимо, що швидкість майже однакова, що досягається технологією **HTML-over-the-wire** та гемом **Turbo**. Проте перевага у простоті програмування і розумінні коду буде за **Hotwire**, оскільки не потрібно дублювати дозволи, інформацію та будувати зв'язки між двома або більше серверами. Також не буде ніякої проблеми з додаванням front-end-у, що має лише Javascript, оскільки ми маємо gem **Stimulus**.

Я вважаю, що це все є рішучими перевагами для нових проектів і значно прискорить їх програмування в **Rails 7**.

У цілому мені дуже зручно працювати з гемом **Stimulus** та **Turbo**, вони хоч і потребують певного часу на опанування, але є значно зручнішими для використання на мій погляд. Сподобався **Stimulus** з його зручним інтерфейсом користування. Також **Turbo** дуже добре себе показав як і в **Turbo Frame**, так і в **Turbo Stream**, так і в **Turbo Drive**.

Посилання на GitHub: https://github.com/VladyslavDekret/HotwireChat

Література

[1] https://rubygems.org/gems
[2] https://hotwired.dev/#screencast
[3] https://turbo.hotwired.dev/
[4] https://stimulus.hotwired.dev/
[5] https://tailwindui.com/