МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 Комп'ютерні науки,
освітньо-професійної програми «Інформатика»
на тему: «Інтернет платформа для створення матеріалів навчання штучного інтелекту в автомобільно-дорожній сфері»
здобувача групи ІН – 01 Мєдвєдєва Романа Ігоровича

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

_____ Роман МЄДВЄДЄВ
(підпис)

Керівник, Надія ТИРКУСОВА
доцент кафедри комп'ютерних наук,
кандидат фізико-математичних наук

_____
(підпис)

Суми – 2024

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

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

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 Комп'ютерні науки, освітньо-професійної програми «Інформатика»
здобувача групи ІН-01 Мєдвєдєва Романа Ігоровича

1. Тема роботи: «Інтернет платформа для створення матеріалів навчання штучного інтелекту в автомобільно-дорожній сфері»
затверджую наказом по СумДУ від *«01» червня 2024_ р. № 0414-IV*
2. Термін здачі здобувачем кваліфікаційної роботи *до 01 червня 2024_ року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для створення інтернет-платформ. 3) Розробка інтелектуальної системи анотації файлів. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» травня 2024 р.

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

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	06.05.24-10.05.24	
2	*Огляд технологій, що використовуються для розробки платформи*	11.05.24-15.05.24	
3	*Розробка платформи для анотації зображень*	16.05.24-20.05.24	
4	*Аналіз отриманих результатів*	21.05.24-25.05.24	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	26.05.24-31.05.24	

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

АНОТАЦІЯ

Записка: 53 стр., 21 рис., 1 додаток, 21 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки завдяки стрімкому розвитку програм на базі штучного інтелекту створюється потреба в забезпеченні його якісними матеріалами для підвищення точності роботи.

Об’єкт дослідження — створення матеріалів для навчання штучного інтелекту.

Мета роботи — розробка інтернет-платформи для анотації файлів з метою створення якісних матеріалів для покращення точності роботи програм на базі алгоритмів штучного інтелекту в автодорожній сфері.

Методи дослідження — використання інтернет ресурсів для розробки сайту.

Результати — розроблено інтернет-платформу, яка дозволяє завантажувати фото матеріали та за допомогою вбудованих інструментів анотувати їх.

ІНТЕРНЕТ ПЛАТФОРМА ДЛЯ СТВОРЕННЯ МАТЕРІАЛІВ НАВЧАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В АВТОМОБІЛЬНО-ДОРОЖНІЙ СФЕРІ

Зміст

ВСТУП

Актуальність: Культурний світ не стоїть на одному місці та постійно розвивається, відкриваючи нові горизонти прогресу. Сучасні компанії інвестують велику кількість ресурсів у створення та вдосконалення складних програм, які базуються на роботі алгоритмів штучного інтелекту. Такі проєкти потребують великі об’єми різноманітних даних. Одним з важливих елементів у ланцюжку створення точних систем є якісні матеріали, на яких штучний інтелект має навчатись. Створення необхідних даних має важливу роль для покращення роботи алгоритмів, підвищення їхньої точності та ускладненням задач, які виконує штучний інтелект. Завдяки розвитку та популяризації машинного навчання, програми на його базі становляться доступними для багатьох компаній майже у всіх сферах діяльності людини. Медицина, автомобільна сфера, промислова, сфера безпеки та інші повлягаються на роботу алгоритмів комп’ютерного зору, а саме обробка зображень та відео.

Об’єкт дослідження: створення матеріалів для вдосконалення роботи алгоритмів штучного інтелекту в сфері комп’ютерного зору та машинного навчання. Маркування фото та відео зображень за допомогою інструментів анотації.

Предмет дослідження: інтернет-платформа для анотації фото та відео матеріалів.

Гіпотеза: створення інструментів для анотації зображень з урахуванням потреб користувача.

Новизна: інтернет-платформа дозволяє завантажувати та оброблювати вхідні файли, а саме фото та відео матеріали для маркування об’єктів на них. Таким сином забезпечується навчання програм на базі алгоритмів штучного інтелекту в сферах де застосовуються прилади які працюють за рахунок обробки зображень механізмами роботи ком’ютерного зору для підвищення точності та якості цих програм.

Структура. Дане робота складається зі вступу, аналітичного огляду,

постановки задачі, вибір методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Сучасний стан

В автомобільній промисловості, що стрімко розвивається, інтеграція передових технологій стала основним фактором підвищення безпеки, продуктивності та автономності транспортних засобів. Однією з таких трансформаційних технологій є комп'ютерний зір, який відіграє вирішальну роль у забезпеченні таких функцій, як автономне водіння, вдосконалені системи допомоги водієві (ADAS) та покращений моніторинг транспортного засобу. Для повного використання потенціалу комп'ютерного зору для навчання і перевірки моделей машинного навчання потрібні високоякісні анотовані дані.

Цей проєкт спрямований на створення комплексної веб-платформи в автодорожній сфері, яка сприятиме ефективному використанню інструментів анотування комп'ютерного зору. Ця платформа слугуватиме інструментом для фахівців з аналізу даних. Використовуючи передові інструменти анотування та спрощені робочі процеси, платформа значно прискорить розробку надійних моделей комп'ютерного зору, що в кінцевому підсумку сприятиме інноваціям та безпеці в автомобільній сфері.

1.2 Аналіз аналогічних проєктів

У швидкозмінному ландшафті автомобільної промисловості анотовані дані відіграють ключову роль, особливо в розробці передових систем допомоги водієві (ADAS) та автономних транспортних засобів.

1. CVAT (Computer Vision Annotation Tool) - це інструмент з відкритим вихідним кодом, розроблений компанією Intel. Він широко використовується для анотування цифрових зображень і відео для досліджень комп'ютерного зору. Хоча це не комерційна компанія, багато організацій використовують CVAT як основний інструмент для завдань анотування. Особливості

платформи:

Інтерфейс користувача:

- Інтуїтивно зрозумілий і зручний інтерфейс, розроблений для легкої навігації та ефективної анотації.

Гнучкість:

- Підтримує різні типи анотацій, включаючи обмежувальні рамки, полігони, полілінії та точки.

Спільна робота:

- Дозволяє створювати спільні анотації, що дуже важливо для великих проектів.

Інтеграція:

- Легко інтегрується з іншими інструментами та робочими процесами за допомогою REST API.

Переваги:

- Завдяки відкритому програмному коду, даний додаток можна використовувати навіть з обмеженими ресурсами.
- Легко налаштовується для задоволення конкретних вимог проекту.
- Потужна підтримка спільноти та регулярні оновлення.

Недоліки:

- Може вимагати додаткових ресурсів та налаштування для роботи з великими проектами.

2. Keymakr - це комерційний постачальник послуг, що спеціалізується на послугах з анотування та маркування даних, що обслуговує різні галузі, включаючи автомобільну промисловість, роздрібну торгівлю та охорону здоров'я. Особливості платформи:

Комплексні послуги:

- Пропонує комплексні послуги з анотування даних, включаючи

анотування зображень, відео та сенсорних даних.

Забезпечення якості:

- Впроваджує суворі заходи контролю якості для забезпечення високої точності анотацій.

Масштабованість:

- Здатність ефективно обробляти великі обсяги даних.

Спеціалізація:

- Експертиза в анотуванні складних наборів даних, необхідних для систем автономного водіння.

Переваги:

- Висока точність анотацій завдяки суворій перевірці якості.
- Великий досвід роботи в автомобільному секторі, що забезпечує експертизу в конкретній галузі.
- Швидкі терміни виконання великих проектів.

Недоліки:

- Висока вартість в порівнянні з альтернативами з відкритим кодом.

3. Scale AI - надає набір інструментів і сервісів для анотування даних, призначених для навчання і перевірки моделей машинного навчання в різних галузях, включаючи автомобільну промисловість. Особливості платформи:

Автоматизація:

- Використовує штучний інтелект для допомоги в процесі анотування, підвищуючи ефективність і узгодженість.

Різноманітні типи анотацій:

- Підтримує широкий спектр типів анотацій, включаючи 2D- і 3D-обмежувальні рамки, сегментацію та відстеження.

Управління якістю:

- Вдосконалені протоколи управління якістю для забезпечення точності.

Інформаційна панель:

- Надає комплексну інформаційну панель для відстеження та управління проектами.

Переваги:

- Високий рівень автоматизації зменшує ручну працю та пришвидшує процес анотування.
- Здатність ефективно керувати надзвичайно великими наборами даних.
- Легко інтегрується з робочими процесами та інструментами машинного навчання.

Недоліки

- Висока вартість через розширені функції та автоматизацію.
- Може вимагати технічних навичок для повного використання всіх функцій.

1.3 Інструменти анотації в автодорожній сфері

Інструменти для анотації зображень для навчання алгоритмів роботи комп’ютерного зору – це маркувальні засоби, які використовуються для виділення та опису об’єктів на фото та відео матеріалах. Створення таких продуктів вкрай необхідне для тренування моделей машинного зору. Головною метою цих матеріалів для навчання є розпізнавання та інтерпретування візуальної інформації, яка формує основу будь-якої моделі, яка базується на алгоритмах роботи штучного інтелекту.

В своїй суті інструменти для анотації дозволяють користувачам маркувати об’єкти на фото та відео та надаючи метадані які алгоритми використовують для вивчення закономірностей та прогнозування. Такі готові дані слугують підґрунтям, необхідними для навчання моделей, які контролюються користувачами. Серед найпоширеніших інструментів анотації є обмежувальні коробки, полігони, сегментації, точкового маркування та скелети. Окрім функціональних відмінностей такі види анотації також

відрізняються точністю, оскільки для різних сценаріїв потрібні різні інструменти. Виявлення об’єктів, класифікація зображень та розуміння сцени.

Всі ці інструменти широко застосовуються в різних сучасних галузях. В автомобільній сфері такі інструменти допомагають створювати алгоритми для пілотування транспорту без втручання людини.

Платформа для анотації дозволяє завантажувати фото матеріали для маркування необхідних об’єктів та за допомогою функціональних інструментів обробляти елементи на зображеннях. Використовуючи відповідні функції в додатку, необхідні об’єкти на фото можна виділяти та систематизувати за номером.

Інструменти для анотації відрізняються за принципом роботи та за характеристиками точності. В залежності від потреб користувача існують наступні види інструментів.

Обмежувальна коробка (англ. Bounding box) - це паралелепіпед, який обмежує об’єкти всередині свого простору. Такий інструмент використовується для виділення будь-яких елементів фото або відео без урахування їхніх меж. Завдяки своїй простій формі широко застосовується в галузях, де непотрібна висока точність, проте необхідна швидкість анотації(рис 1.1).

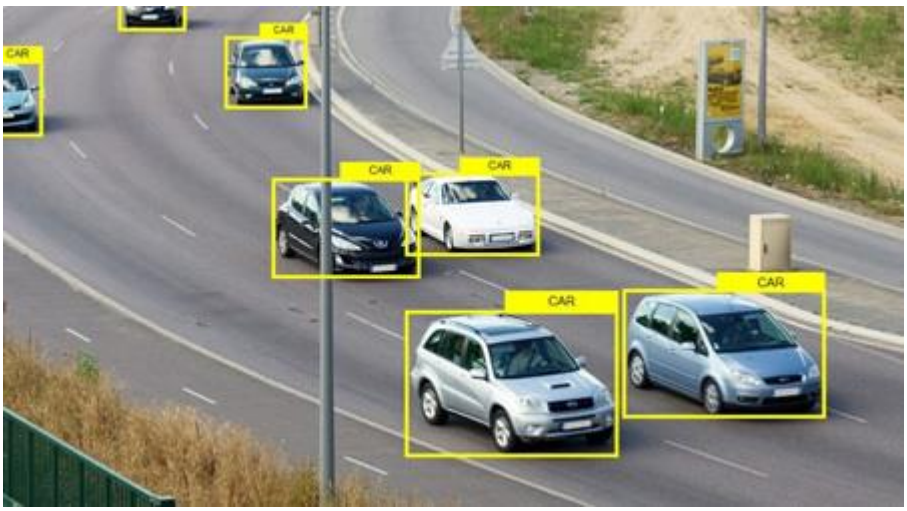


Рисунок 1.1 — Приклад анотації об’єктів за допомогою обмежувальної коробки

Полігон (англ. Polygon) - це інструмент який використовується для обведення об’єктів навколо їхніх меж за допомогою точок, які зв’язуються одна за одною. Перевагами такої анотації є висока точність та можливість виділяти об’єкти складної форми(рис 1.2).

Рисунок.1.2 — Приклад анотації об’єктів за допомогою полігону

Семантична сегментація (англ. Semantic Segmentation) – щільне маркування всього зображення в межах одного класу або об’єкта. Такий тип анотації забезпечує детальну сегментацію об’єктів з урахуванням їхніх меж,

проте не розрізняє окремі об’єкти в межах одного класу. Також дає можливість отримати загальний опис зображення, оскільки фон також враховується(рис 1.3).

Рисунок 1.3 — Приклад анотації об’єктів за допомогою сематичної сегментації

Сегментація екземплярів (англ. Instance Segmentation) – відповідно до семантичної сегментації забезпечує щільне маркування всього зображення, проте головної відмінністю є окреме виділення різних об’єктів в межах одного класу. Використовується під час необхідності виокремлення об’єктів із натовпу(рис 1.4).

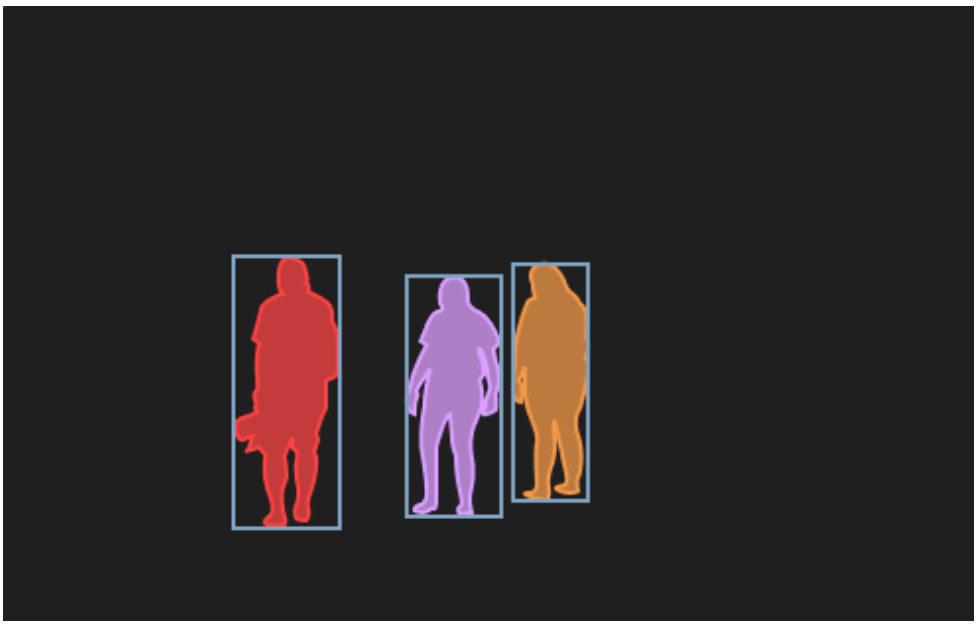


Рисунок 1.4 — Приклад анотації об’єктів за допомогою сегментації екземплярів

Паноптична сегментація (англ. Panoptic Segmentation) – поєднує сегментацію екземплярів та сематичну сегментацію, завдяки чому забезпечує повне розуміння сцени з урахуванням ідентичності окремих об’єктів, в залежності від класу(рис 1.5).

Рисунок 1.5 — Приклад анотації об’єктів за допомогою паноптичної сегментації

Лінії та сплайни (англ. Lines and Splines) – позначають зображення прямими або кривими лініями. Це важливо для розпізнавання меж, щоб позначити тротуари, дорожні знаки та інші виміри меж. Цей тип анотацій створюється за допомогою ліній і сплайнів. Він зазвичай використовується в автономних транспортних засобах для виявлення та розпізнавання смуг руху(рис 1.6).

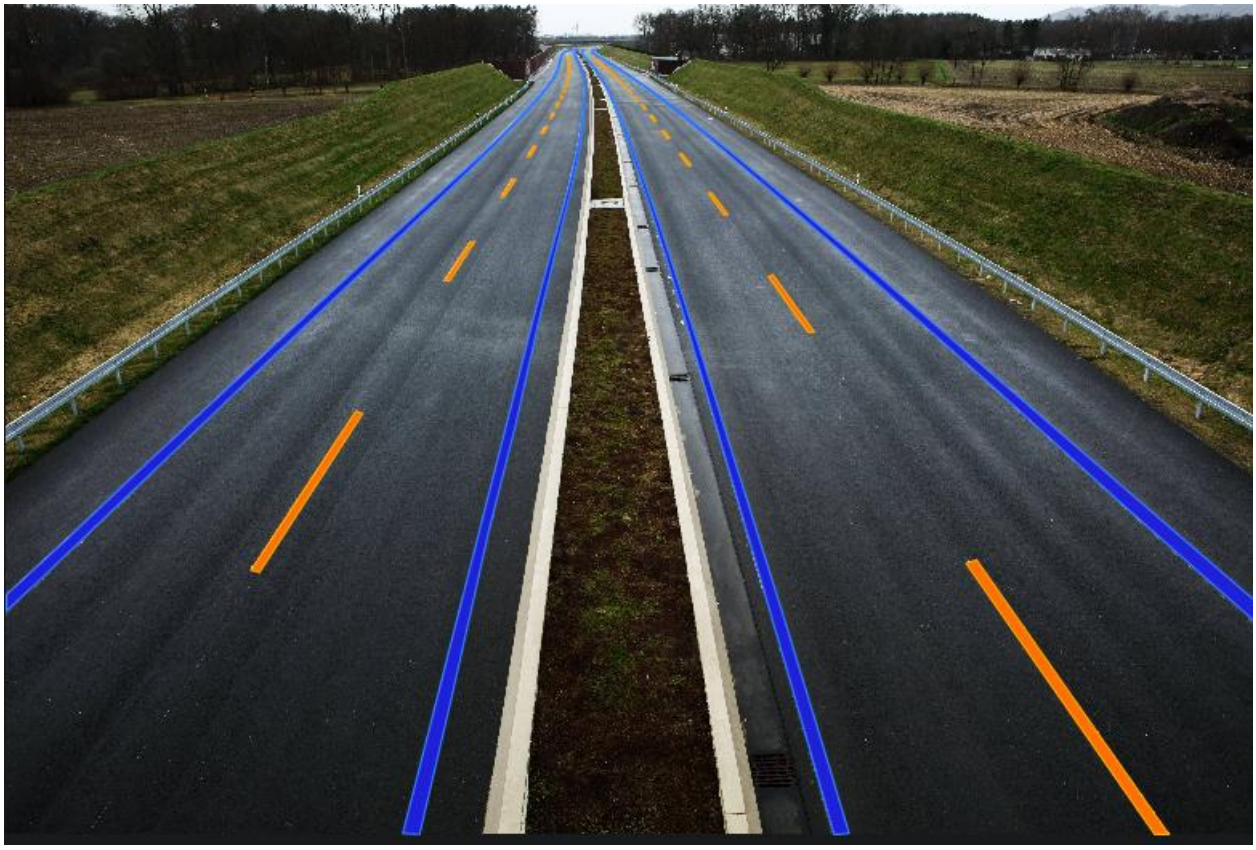


Рисунок 1.6 — Приклад анотації об’єктів за допомогою ліній та сплайнів

1.4 Постановка задачі

Головна мета цієї роботи полягає у створенні функціональної веб-платформи для обробки зображень. Для роботи користувачів на платформі, проєкт має відповідний функціонал. Для цього програма матиме робочу панель, до якої користувач матиме доступ. Для реалізації цих процесів будуть створені відповідні елементи інтерфейсу.

Етапи розробки проєкту:

1. Аналіз ринку:
    - Дослідити сучасний ринок зі створення інтернет-платформ для анотації зображень.
2. Визначення вимог, пов'язаних із функціональністю інтернет-платформи:
    - Встановити головні функції інтернет-платформи.
3. Проєктування дизайну та робочої зони:
    - Створити зручний та зрозумілий користувачеві інтерфейс платформи.
    - Забезпечити розробку дизайну платформи з урахуванням сучасних тенденцій зі створення дружнього до користувача інтерфейсу.
4. Створення наступного функціоналу:

    4.1 Реєстрація та авторизація:
    - Користувачі матимуть можливість створити власний акаунт, використовуючи свої дані.
    - Зареєстровані користувачі матимуть можливість авторизуватися на свій акаунт, використовуючи власні дані, а саме логін та пароль.

    4.2 Після успішної авторизації, користувачі мають можливість створити робочу частину з коротким описом необхідних задач.

    4.3 Для обробки матеріалів, користувач має завантажити зображення на носій, з якого вони будуть використовуватися під час анотації.

    4.4 Під час роботи на платформі з вивантаженими матеріалами, користувач має можливість використовувати інструменти для анотації зображень.

    4.5 Після закінчення маркування матеріалів користувач має можливість вивантажити їх з усіма метаданими.
5. Вдосконалення та технічна підтримка:
    - Забезпечення технічної підтримки платформи.
    - Вдосконалення функціоналу платформи

Цей план, з конкретно розписаними задачами, забезпечує чіткі кроки до розробки інтернет-платформи.

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Інформаційна модель додатку

Для забезпечення якісного досвіду використання інтернет-платформою, користувач має отримати відповідний функціонал.

1. Створення аккаунту.
    - Користувач має можливість створити власний акаунт, вказавши логін та пароль.
    - Користувач має можливість авторизуватися в свій обліковий запис, використовуючи власні дані.

Таким чином забезпечується безпека користувача, тому що лише автентифіковані користувачі можуть отримати доступ до певних частин платформи або виконувати певні дії. Це запобігає несанкціонованому доступу, захищає конфіденційні дані та зменшує ризик зловмисних дій.

Також зі збільшенням кількості користувачів і функцій веб-платформи, управління контролем доступу стає все складнішим. Фреймворки авторизації надають масштабовані рішення для управління правами доступу, ролями та дозволами користувачів, дозволяючи платформі рости без шкоди для безпеки та зручності використання.

2. Створення робочого проєкту.

Можливість створення окремого проєкту забезпечує виділеного робочого простору, що спрощує процес анотування, централізуючи всі необхідні інструменти та функції в одній платформі. Це економить час і зусилля користувачів, яким інакше довелося б перемикатися між кількома програмами або середовищами.

3. Використання інструментів анотації.

В робочому просторі користувач має можливість завантажувати зображення та, використовуючи інструменти платформи, анотувати їх. Після чого вивантажувати на власний носій інформації.

2.2 Вибір архітектурних рішень

1. Інформаційні системи.

При створенні веб-сайту вибір правильних архітектурних рішень має вирішальне значення для забезпечення масштабованості, продуктивності та зручності обслуговування. Найпоширенішими архітектурними рішеннями є:

- Монолітна архітектура - єдина уніфікована кодова база, де всі компоненти взаємопов'язані та взаємозалежні. Простота в розробці та розгортанні, легкість налагодження, проте існують проблеми з масштабуванням, складність у підтримці та оновленні великих кодових баз.
- Архітектура мікросервісів - набір невеликих сервісів, кожен з яких працює у власному процесі і взаємодіє через API. Масштабованість, гнучкість, незалежне розгортання. складність в управлінні, потенційна затримка у взаємодії між сервісами.
- Безсерверна архітектура - використовує хмарні сервіси для виконання коду у відповідь на події без управління серверами. Зниження операційних витрат, автоматичне масштабування, фокус на розробці. Обмежений контроль над середовищем, потенційна прив'язка до постачальника.
- Односторінковий додаток (SPA) - веб-додаток, який взаємодіє з користувачем шляхом динамічного переписування поточної сторінки, а не завантаження цілих нових сторінок з сервера. Покращений користувацький досвід, швидша взаємодія, зменшення навантаження на сервер. Проблеми з пошуковою оптимізацією, початковий час завантаження може бути високим.
- Прогресивний веб-додаток (PWA) - тип прикладного програмного забезпечення, що постачається через Інтернет, створений за допомогою поширених веб-технологій, включаючи HTML, CSS та JavaScript.

Офлайн-можливості, покращена продуктивність, краще залучення користувачів. Обмежена підтримка в деяких старих браузерах, складність у розробці.

- Архітектура на основі системи управління контентом (CMS) - використовує такі платформи, як WordPress, Joomla або Drupal для управління створенням і модифікацією цифрового контенту. Легке управління контентом, зручне для користувача, швидке розгортання. Потенційні проблеми з продуктивністю, обмежені можливості кастомізації.

Для даного проєкту я обрав архітектуру Model-View-Controller (MVC), яка є загальним шаблоном проектування, що використовується у веб-додатках для відокремлення проблем і організації коду в модульний спосіб. Використовуючи архітектуру MVC, проєкт підтримує чіткий розподіл завдань, роблячи кодову базу більш організованою, підтримуваною та масштабованою.

2. Клієнтська частина.

Фреймворки відіграють ключову роль у спрощенні та оптимізації розробки клієнтського/користувацького інтерфейсу при створенні веб-сайту. Існують наступні фреймворки:

- React - розроблений Facebook, React - це JavaScript бібліотека для створення користувацьких інтерфейсів. Вона дозволяє розробникам створювати багаторазові компоненти інтерфейсу та ефективно оновлювати вигляд при зміні даних. Ключові особливості: віртуальний DOM для ефективного рендерингу, компонентна архітектура, синтаксис JSX для написання HTML на JavaScript, управління станами за допомогою хуків React або Redux.
- Angular – розроблений Google, Angular - це фреймворк на основі TypeScript для створення веб-додатків. Він надає комплексне рішення з вбудованими інструментами для маршрутизації, формами, HTTP-

клієнтом тощо. Двостороння прив'язка даних, ін'єкція залежностей, модульна архітектура з NgModules, вбудована підтримка TypeScript, потужний CLI для скаффолдингових проектів.

- Vue.js - прогресивний JavaScript-фреймворк для створення користувацьких інтерфейсів. Він відомий своєю простотою та гнучкістю, що дозволяє розробникам поступово впроваджувати його функції в міру необхідності. Реактивне зв'язування даних, компонентна архітектура, віртуальний DOM з ефективним рендерингом, легка інтеграція з існуючими проектами, Vue CLI для роботи над проектами.

Для клієнтської частини цього проєкту я використовував комбінації шаблонів HTML, CSS, JavaScript та EJS (Embedded JavaScript). Завдяки поєднанню цих технологій клієнтська частина проекту забезпечує динамічний, чуйний і зручний інтерфейс, який безперешкодно взаємодіє з сервером для управління автентифікацією, даними проекту та анотаціями до зображень.

3. Серверна частина.

Серверна частина забезпечує структурований та ефективний спосіб обробки серверної логіки, управління взаємодією з базами даних та маршрутизації вхідних запитів. Найпопулярнішими серверними фреймворками є:

- Express.js - мінімалістичний фреймворк веб-додатків для Node.js. Він спрощує процес створення веб-серверів та API, надаючи надійний набір функцій, включаючи маршрутизацію, підтримку проміжного програмного забезпечення та інтеграцію з механізмом шаблонів.
- ASP.NET Core - це кросплатформенний веб-фреймворк з відкритим вихідним кодом для створення сучасних хмарних веб-додатків на базі .NET. Він надає уніфіковану основу для створення веб-інтерфейсів API, веб-додатків MVC та веб-додатків у режимі реального часу.
- Spring Boot - це фреймворк для створення веб-додатків на основі Java.

Він спрощує налаштування та конфігурацію додатків на основі Spring, надаючи значення за замовчуванням для різних компонентів і пропонуючи ряд функцій для створення готових до виробництва додатків.

Серверна частина цього проекту побудована з використанням комбінації технологій, зосереджених на Node.js та Express.js. Додаткові бібліотеки, такі як Mongoose, bcrypt, express-session, connect-mongo, dotenv використовуються для управління моделями даних, безпеки, управління сесіями, аналізу тіл запитів, управління конфігурацією та завантаженням файлів.

4. База даних.

Бази даних забезпечують зберігання та керування даними, на які покладається додаток. Існують наступні бази даних:

- Apache Cassandra - стовпчикова база даних, розроблена для масштабованості, відмовостійкості та високої доступності. Зазвичай використовується для часових рядів даних та аналітики в реальному часі.
- MongoDB - документно-орієнтована база даних NoSQL, яка зберігає дані у гнучких документах у форматі JSON. Зазвичай використовується для додатків, що вимагають високої масштабованості, таких як системи управління контентом та аналітика в реальному часі.
- Redis - сховище структур даних у пам'яті, відоме своєю високою продуктивністю та універсальністю. Часто використовується як рівень кешування, брокер повідомлень або сховище даних у реальному часі у веб-додатках.

5. Формати анотації зображень

Анотування зображень передбачає позначення зображень метаданими, які допомагають моделям навчатися і робити прогнози. Використовуються різні формати анотацій, кожен з яких підходить для різних завдань та інструментів. Найпоширеніші формати:

1. Pascal VOC (візуальні класи об'єктів) - це широко використовуваний формат анотацій, який підтримує виявлення об'єктів, класифікацію зображень і завдання сегментації. Формат базується на XML і містить таку інформацію

Ім'я файлу: Ім'я файлу зображення.

Розмір: Розміри зображення (ширина, висота, глибина).

Об'єкт: Деталі про кожен анотований об'єкт, включно з назвою класу, координатами обмежувальної рамки (xmin, ymin, xmax, ymax) та складністю (показник складності анотації).

2. COCO (Common Objects in Context)

COCO - це популярний формат, який використовується у великомасштабних завданнях виявлення об'єктів, сегментації та визначення ключових точок. Він базується на JSON і включає в себе:

Інформацію: Опис набору даних.

Зображення: Список зображень з такими атрибутами, як ім'я файлу, ширина та висота.

Анотації: Список анотацій з такими полями, як ідентифікатор зображення, ідентифікатор категорії, обмежувальна рамка і сегментація.

Категорії: Список категорій об'єктів з ідентифікаторами та назвами.

3. YOLO (You Only Look Only Once)

Формат YOLO призначений для алгоритму виявлення об'єктів YOLO. Анотації зберігаються у звичайних текстових файлах, кожен рядок яких представляє один об'єкт. Кожен рядок містить

Ідентифікатор класу: Ідентифікатор класу об'єкта.

Координати обмежувальної рамки: Центр x, центр y, ширина та висота, нормалізовані до ширини та висоти зображення.

4. TFRecord (TensorFlow Record)

TFRecord - це двійковий формат файлів, який використовується для зберігання послідовностей двійкових даних, придатний для моделей на основі TensorFlow. Анотації зберігаються у форматі протокольних буферів (protobufs), який включає:

Дані зображення: Закодовані дані зображення.

Анотації об'єктів: Включаючи мітки класів та координати обмежувальних рамок.

5. LabelMe

LabelMe - це формат анотацій, який використовується інструментом LabelMe, головним чином для сегментації зображень. Анотації зберігаються у форматі JSON і включають:

Фігури: Список фігур з такими атрибутами, як мітка, точки (вершини багатокутника) і тип фігури.

Шлях до зображення: Шлях до файлу зображення.

Дані зображення: Закодовані дані зображення (необов'язково).

6. Міські пейзажі

Формат Cityscapes призначений для розуміння міських пейзажів і включає в себе анотації для прикладу та семантичну сегментацію. Анотації зберігаються у форматі JSON і містять:

Інформацію про зображення: Включає розміри зображення та ім'я файлу.

Об'єкти: Список об'єктів з координатами полігональної сегментації, мітками класів та ідентифікаторами екземплярів.

7. KITTI

KITTI - це формат, що використовується для наборів даних автономного водіння. Він включає різні типи даних, такі як виявлення 3D-об'єктів, відстеження та виявлення доріг/смуг руху. Анотації зберігаються

у звичайних текстових файлах з:

Тип об'єкта: Мітка класу.

2D обмежувальна рамка: Координати (xmin, ymin, xmax, ymax).

3D Bounding Box: Тривимірні координати та розміри.

Рівні усікання та оклюзії: Індикатори усічення та оклюзії об'єкта.

Результатом роботи в проєкті є анотовані зображення, формат яких відповідає критеріям LabelMe.

2.3 Опис середовища розробки проєкту

IntelliJ IDEA - це потужне, багатофункціональне інтегроване середовище розробки (IDE), в першу чергу призначене для розробки Java, але воно також підтримує широкий спектр інших мов і технологій програмування. Розроблене компанією JetBrains, IntelliJ IDEA високо цінується за свій передовий аналіз коду, можливості рефакторингу та інструменти для підвищення продуктивності. IntelliJ IDEA підтримується багатьма сучасними операційними системами, такими як: Windows, Linux та macOS.

Дане середовище розробки програмних проєктів має велику кількість переваг для роботи користувачів. До головних переваг IntelliJ IDEA відносяться:

1. Розумна підтримка коду: IntelliJ IDEA пропонує надійне завершення коду, глибокий аналіз команд та функції інтелектуального рефакторингу. Він глибоко розуміє користувацький код і надає відповідні пропозиції та виправлення коду «на льоту».

2. Інтегровані інструменти та плагіни: IDE постачається з повним набором інструментів та багатою екосистемою плагінів, що дозволяє розробникам розширювати її функціональність. Інструменти для контролю версій, автоматизації збірки та серверів додатків легко інтегруються.

3. Зручний інтерфейс: IntelliJ IDEA може похвалитися інтуїтивно зрозумілим користувальницьким інтерфейсом, який легко налаштовується та покращує роботу розробника. Макет розроблений таким чином, щоб мінімізувати відволікаючі фактори та впорядкувати робочі процеси.

4. Багатомовна підтримка: Окрім Java, IntelliJ IDEA підтримує різні мови, такі як Kotlin, Scala, Groovy, Python, JavaScript, TypeScript та інші, що робить її універсальним інструментом для багатомовних проектів.

5. Продуктивність та швидкість реагування: Незважаючи на свої широкі можливості, IntelliJ IDEA оптимізована для продуктивності, гарантуючи, що навіть великі проекти працюють безперебійно і без значних затримок.

6. Інтеграція контролю версій: IDE підтримує всі основні системи контролю версій (VCS), такі як Git, Mercurial і SVN, з єдиним для всіх користувальницьким інтерфейсом. Це спрощує такі завдання, як розгалуження, злиття та перегляд історії.

7. Налагодження та тестування: IntelliJ IDEA надає потужні засоби налагодження, включаючи візуальний відладчик, та інтегрується з такими тестовими середовищами, як JUnit та TestNG. Це дозволяє легко виявляти та виправляти проблеми у вашому коді.

8. Спільнота та підтримка: Навколо IntelliJ IDEA існує велика і активна спільнота, а також вичерпна документація і чуйна підтримка від JetBrains, що гарантує, що розробники можуть знайти допомогу і

ресурси, коли це необхідно.

Завдяки встановленим розширенням, користувацький досвід покращився та написання програми стало більш комфортним(рис 2.1 та рис 2.2).

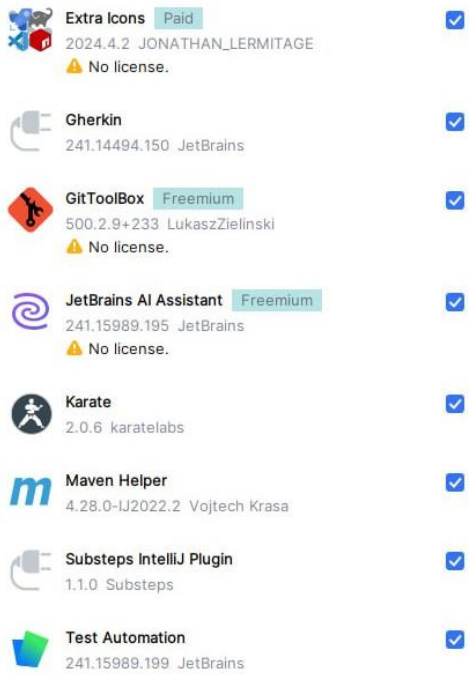


Рисунок 2.1 — Використані плагіни

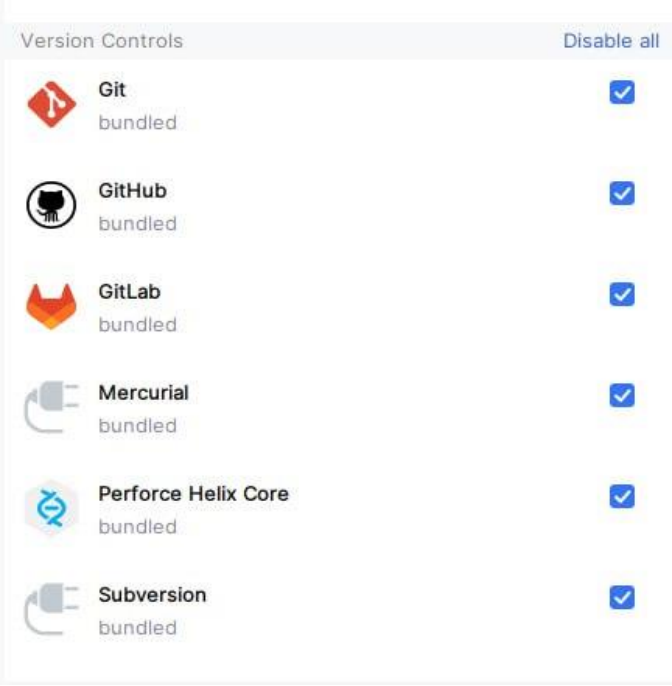


Рисунок 2.2 — Використані плагіни

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Структура проєкту

В даному проєкті реалізовані наступні структурні елементи(рис 3.1)

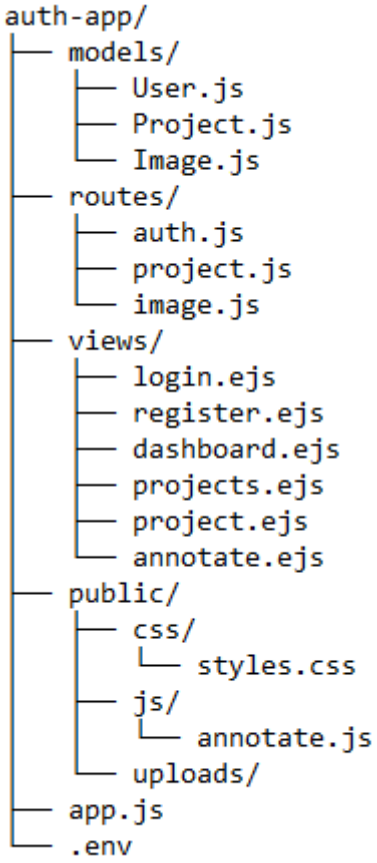


Рисунок 3.1 – Структура проєкту

3.2 Реалізація користувацької частини

В даному проєкті клієнтська частина була побудована використовуючи мову JavaScript. Завдяки цій мові я реалізував можливість автентифікації на проєкті, створення робочих задач та робочу область.

Папка проєкту містить необхідні для роботи додатку файли, які містять в собі реалізацію навігації по застосунку та опис зовнішнього вигляду інтерфейсу.

- Models

Містить моделі Mongoose, які визначають структуру даних, що зберігаються в MongoDB.

- Routes

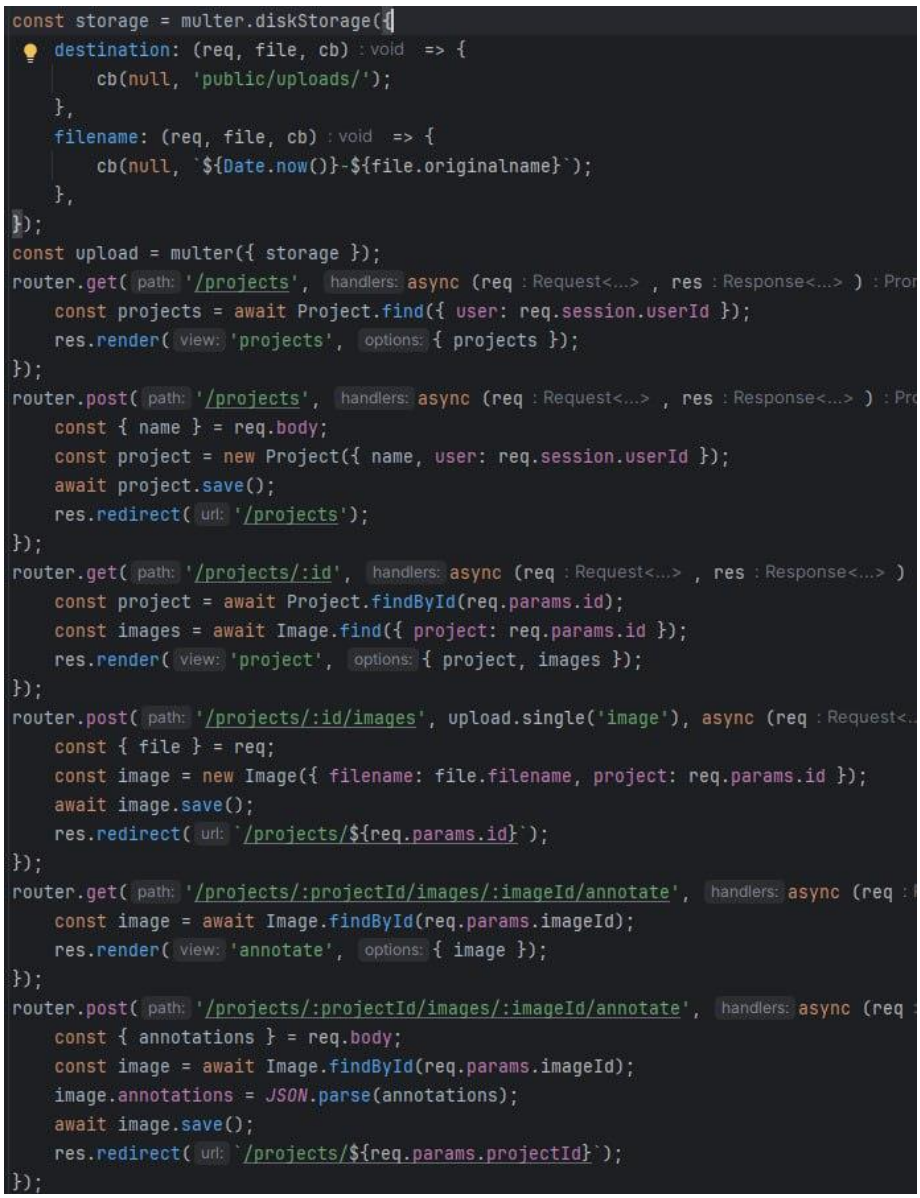
Містить обробники маршрутів для різних HTTP-запитів, згруповані за функціональністю. Цей каталог забезпечує обробку маршрутів пов’язаних з автентифікацією в системі, створення нових задач та анотацію зображень.

- Views

Містить шаблони EJS (Embedded JavaScript), які використовуються для відображення HTML на стороні сервера.

- Public

Містить усі статичні файли CSS, JavaScript які надаються користувачеві. Забезпечує візуальний вигляд елементів додатку(рис 3.2).



```
const storage = multer.diskStorage({
    destination: (req, file, cb) : void  => {
        cb(null, 'public/uploads/');
    },
    filename: (req, file, cb) : void  => {
        cb(null, `${Date.now()}-${file.originalname}`);
    },
});
const upload = multer({ storage });
router.get( path: '/projects', handlers: async (req : Request<...> , res : Response<...> ) : Pro
    const projects = await Project.find({ user: req.session.userId });
    res.render( view: 'projects', options: { projects });
});
router.post( path: '/projects', handlers: async (req : Request<...> , res : Response<...> ) : Pr
    const { name } = req.body;
    const project = new Project({ name, user: req.session.userId });
    await project.save();
    res.redirect( url: '/projects');
});
router.get( path: '/projects/:id', handlers: async (req : Request<...> , res : Response<...> )
    const project = await Project.findById(req.params.id);
    const images = await Image.find({ project: req.params.id });
    res.render( view: 'project', options: { project, images });
});
router.post( path: '/projects/:id/images', upload.single('image'), async (req : Request<..
    const { file } = req;
    const image = new Image({ filename: file.filename, project: req.params.id });
    await image.save();
    res.redirect( url: `/projects/${req.params.id}`);
});
router.get( path: '/projects/:projectId/images/:imageId/annotate', handlers: async (req :
    const image = await Image.findById(req.params.imageId);
    res.render( view: 'annotate', options: { image });
});
router.post( path: '/projects/:projectId/images/:imageId/annotate', handlers: async (req
    const { annotations } = req.body;
    const image = await Image.findById(req.params.imageId);
    image.annotations = JSON.parse(annotations);
    await image.save();
    res.redirect( url: `/projects/${req.params.projectId}`);
});
```

Рисунок 3.2 – Реалізація навігації

3.3 Реалізація серверної частини

Цей проєкт використовує різноманітні сучасні технології для створення безпечного, масштабованого та ефективного веб-додатку для автентифікації користувачів, управління проектами та анотацій до зображень.

Технології бекенду включають у себе наступні елементи:

Node.js - середовище виконання JavaScript, побудоване на JavaScript-движку Chrome V8, що використовується для створення швидких і масштабованих серверних додатків. Забезпечує роботу серверної логіки,

обробку HTTP-запитів, маршрутизацію та виконання проміжного програмного забезпечення.

Express.js - гнучкий фреймворк веб-додатків Node.js, який надає надійний набір функцій для веб додатку. Використовується для створення сервера, визначення маршрутів для автентифікації, управління проектами та зображеннями, а також для роботи проміжного програмного забезпечення.

MongoDB - база даних NoSQL, відома своєю масштабованістю та гнучкістю, що використовує JSON-подібні документи зі схемою. Зберігає дані користувача та дані проєкту. Надає гнучку модель даних для роботи з різними структурами даних.

Mongoose - серверна частина цього проекту побудована з використанням Node.js та Express.js, з базою даних MongoDB. Сервер відповідає за автентифікацію користувачів, управління задачами та анотації зображень.

Bcrypt - бібліотека для хешування паролів, що забезпечує надійні алгоритми хешування та засолювання для підвищення безпеки. Хешує паролі користувачів перед збереженням в базі даних, забезпечуючи їх безпечне зберігання.

express-session - Проміжне програмне забезпечення для керування сеансами в експрес-додатках. Дозволяє створювати, зберігати та керувати сеансами, дозволяючи користувачам залишатися автентифікованими під час різних запитів.

connect-mongo - Сховище сесій MongoDB для Express та Connect. Зберігає дані сесії в MongoDB, забезпечуючи стійкість і масштабованість сесій.

dotenv - Модуль, який завантажує змінні оточення з файлу .env в process.env. Керує налаштуваннями конфігурації та конфіденційною інформацією, такою як рядки з'єднання з базою даних та секрети сеансів.

Серверна частина цього проекту побудована з використанням Node.js та Express.js, з базою даних MongoDB. Сервер відповідає за автентифікацію

користувачів, управління задачами та анотації зображень. Кожний компонент реалізовано наступним чином:

1. Реєстрація користувачів.

Схема Mongoose визначає структуру документів користувача в MongoDB. Схема включає поля для імені користувача та пароля. Перед збереженням документа користувача, пароль хешується за допомогою bcrypt. Сервер перевіряє введені користувачем дані, щоб переконатися, що ім'я користувача є унікальним. Якщо користувач з таким нікнеймом вже існує, клієнту повертається повідомлення про помилку. Новий документ користувача створюється і зберігається в MongoDB. Після успішної реєстрації користувач буде перенаправлений на сторінку входу.

Вхід користувача

2. Алгоритм входу.

Сервер перевіряє, чи існує логін в базі даних. Якщо так, наданий пароль порівнюється з хешованим паролем, що зберігається в базі даних за допомогою bcrypt. Якщо облікові дані дійсні, для користувача створюється сесія за допомогою експрес-сесії. Ця сесія зберігається в MongoDB за допомогою connect-mongo, забезпечуючи стійкість при перезавантаженні сервера(рис 3.3).

```
const mongoose :   = require('mongoose');
const bcrypt : any | {...}  = require('bcryptjs');
const UserSchema : Schema  = new mongoose.Schema( definition: {
    username: {
        type: String,
        required: true,
        unique: true,
    },
    password: {
        type: String,
        required: true,
    },
});
UserSchema.pre( method: 'save',  fn: async function (next : CallbackWithoutResultAndO
    try {
        const salt = await bcrypt.genSalt( rounds: 10);
        this.password = await bcrypt.hash(this.password, salt);
        next();
    } catch (error) {
        next(error);
    }
});
UserSchema.methods.isValidPassword = async function (password) : Promise<...>  {
    try {
        return await bcrypt.compare(password, this.password);
    } catch (error) {
        throw error;
    }
};
module.exports = mongoose.model( name: 'User', UserSchema);
```

Рисунок 3.3 – Реалізація автентифікації

3. Створення задачі.

Схема Mongoose визначає структуру документів проекту в MongoDB. Кожен проект містить такі поля, як projectName, createdBy та createdAt. Проекти асоціюються з користувачами, що дозволяє кожному користувачеві керувати власними проектами. Схема Mongoose визначає структуру документів зображень у MongoDB. Кожне зображення містить такі поля, як imagePath, анотації та projectId. Анотації зберігаються як JSON-об'єкти, включаючи метадані, такі як координати рамки та підписи.

4. Анотація зображень.

Інтерфейс на стороні клієнта дозволяє користувачам завантажувати зображення, які потім зберігаються на сервері. Сервер зберігає ці зображення у визначеному каталозі завантажень і зберігає

шляхи до файлів у MongoDB. Завантажені зображення зберігаються у файловій системі, а шляхи до них вказуються в базі даних. Веб-інтерфейс, побудований на HTML, CSS і JavaScript, дозволяє користувачам створювати і налаштовувати обмежувальні рамки на завантажених зображеннях. Координати та підписи рамок фіксуються як анотації. Коли користувач зберігає анотації, дані надсилаються на сервер, який зберігає їх у MongoDB разом із метаданими зображення. Анотації зберігаються у стандартизованому форматі (наприклад, COCO або LabelMe) як JSON. Цей формат включає в себе дані про рамки, мітки та властивості зображень(рис 3.4, рис 3.5, рис 3.6).

```
document.addEventListener( type: 'DOMContentLoaded', listener: () : void => {
    const imageContainer : HTMLElement = document.getElementById( elementId: 'annotation-container');
    const image : HTMLElement = document.getElementById( elementId: 'image-to-annotate');
    const uploadButton : HTMLElement = document.getElementById( elementId: 'upload-image');
    const fileInput : HTMLElement = document.getElementById( elementId: 'image-upload');
    const saveButton : HTMLElement = document.getElementById( elementId: 'save-annotations');
    let annotations : any[] = [];
    fileInput.addEventListener( type: 'change', listener: (e : Event ) : void => {
        const file = e.target.files[0];
        if (file) {
            const reader : FileReader = new FileReader();
            reader.onload = (e : ProgressEvent<FileReader> ) : void => {
                image.src = e.target.result;
                image.style.display = 'block';
            };
            reader.readAsDataURL(file);
        }
    });
    uploadButton.addEventListener( type: 'click', listener: () : void => {
        fileInput.click();
    });
```

Рисунок 3.4 – Реалізація анотації

```
image.addEventListener( type: 'click', listener: (e : MouseEvent ) : void => {
    const rect : DOMRect = image.getBoundingClientRect();
    const x : number = e.clientX - rect.left;
    const y : number = e.clientY - rect.top;
    const boundingBox : HTMLDivElement = document.createElement( tagName: 'div');
    boundingBox.className = 'bounding-box';
    boundingBox.style.left = `${x}px`;
    boundingBox.style.top = `${y}px`;
    boundingBox.style.width = '100px';
    boundingBox.style.height = '100px';
    imageContainer.appendChild(boundingBox);
    let isResizing : boolean = false;
    let currentResizer;
    boundingBox.addEventListener( type: 'mousedown', listener: (e : MouseEvent ) : void => {
        if (e.target.classList.contains('resizer')) {
            isResizing = true;
            currentResizer = e.target;
        } else {
            boundingBox.dataset.dragging = true;
            boundingBox.dataset.offsetX = e.offsetX;
            boundingBox.dataset.offsetY = e.offsetY;
        }
    });
    document.addEventListener( type: 'mousemove', listener: (e : MouseEvent ) : void => {
        if (isResizing) {
            const rect : DOMRect = boundingBox.getBoundingClientRect();
            if (currentResizer.classList.contains('bottom-right')) {
                boundingBox.style.width = `${e.clientX - rect.left}px`;
                boundingBox.style.height = `${e.clientY - rect.top}px`;
            } else if (currentResizer.classList.contains('bottom-left')) {
                boundingBox.style.width = `${rect.width - (e.clientX - rect.left)}px`;
                boundingBox.style.height = `${e.clientY - rect.top}px`;
                boundingBox.style.left = `${e.clientX}px`;
            } else if (currentResizer.classList.contains('top-right')) {
                boundingBox.style.width = `${e.clientX - rect.left}px`;
                boundingBox.style.height = `${rect.height - (e.clientY - rect.top)}px`;
                boundingBox.style.top = `${e.clientY}px`;
            } else {
                boundingBox.style.width = `${rect.width - (e.clientX - rect.left)}px`;
                boundingBox.style.height = `${rect.height - (e.clientY - rect.top)}px`;
```

Рисунок 3.5 – Реалізація анотації

```
    document.addEventListener( type: 'mouseup', listener: () : void => {
        isResizing = false;
        delete boundingBox.dataset.dragging;
    });
    annotations.push({ x, y, width: 100, height: 100 });
    const resizers : string[] = ['top-left', 'top-right', 'bottom-left', 'bottom-right'];
    resizers.forEach((pos : string ) : void => {
        const resizer : HTMLDivElement = document.createElement( tagName: 'div');
        resizer.className = `resizer ${pos}`;
        boundingBox.appendChild(resizer);
    });
});
saveButton.addEventListener( type: 'click', listener: () : void => {
    console.log('Annotations:', annotations);
    alert('Annotations saved to console.');
});
});
```

Рисунок 3.6 – Реалізація анотації

3.4 Демонстрація використання застосунку

Спочатку користувач має створити новий профіль, ввівши свій логін та пароль, або ввійти в існуючий у вікні Login(рис 3.7).

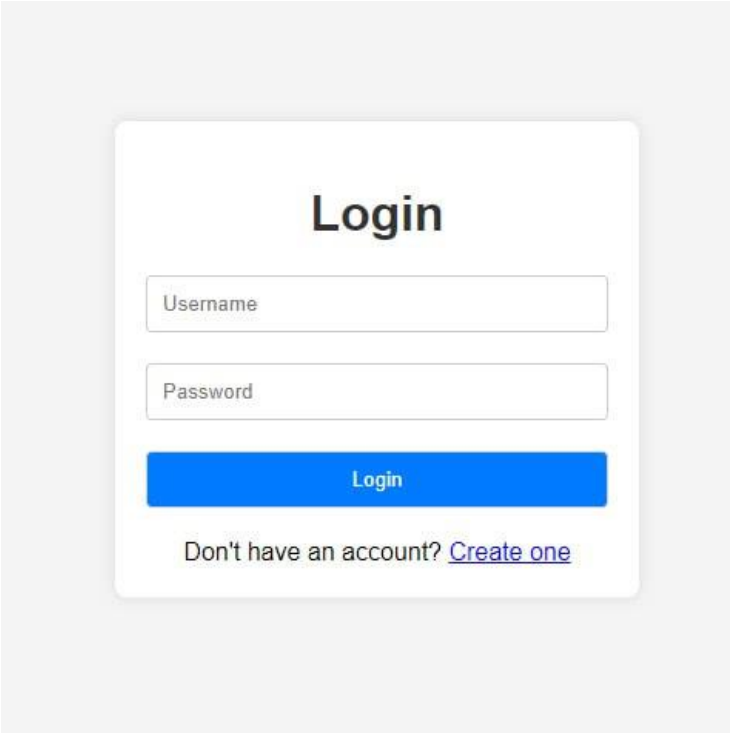


Рисунок 3.7 – Поле авторизації

Після авторизації, користувач перенаправляється на вікно з задачами, де він може створити новий проєкт або почати працювати в існуючому(рис 3.8).

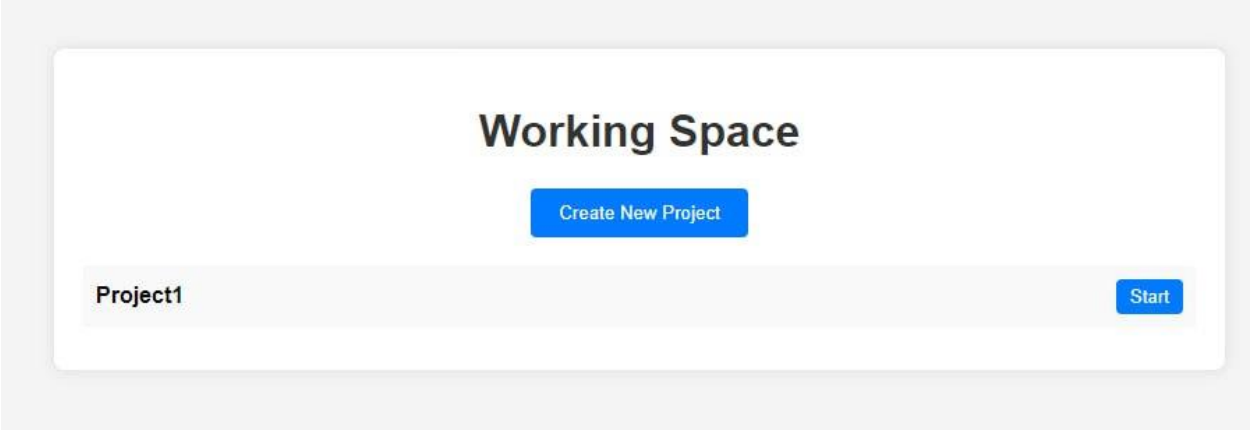


Рисунок 3.8 – Поле задач

Робоча область являє собою вікно, в якому користувач може завантажити зображення та зберегти після анотації(рис 3.9).

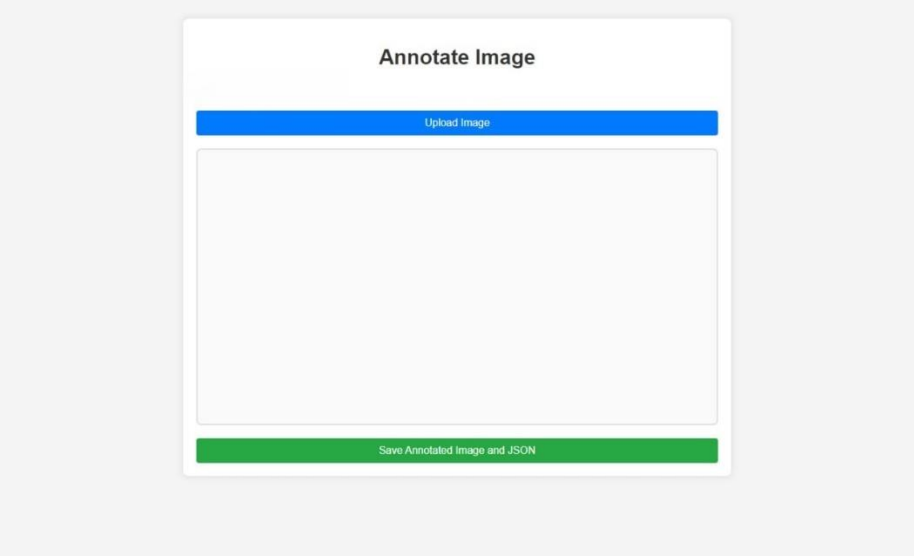


Рисунок 3.9 – Робочий простір

Анотація відбувається шляхом додавання обмежувальних коробок та обведенням ними об’єктів на зображенні. Обмежувальні коробки можна змінювати за розміром тягнучі їх за вершини квадрата(рис 3.10).

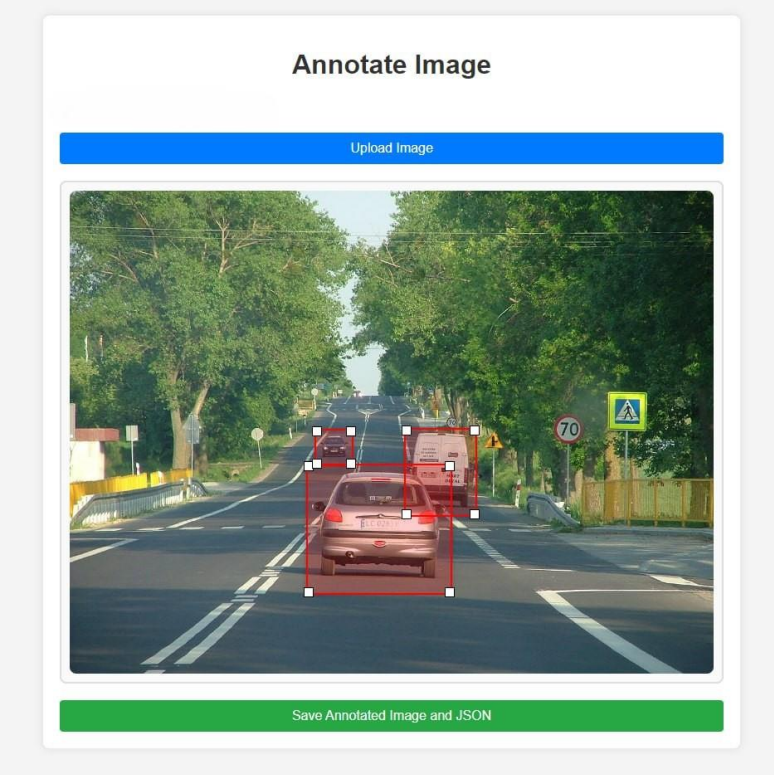
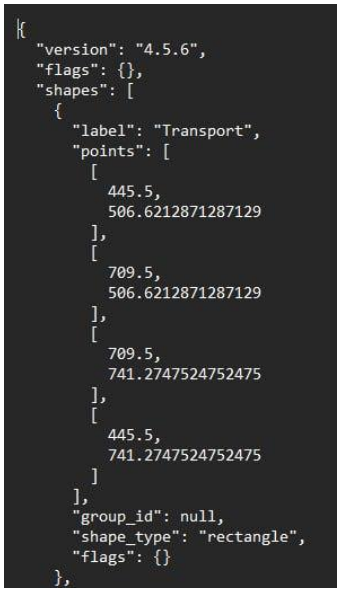


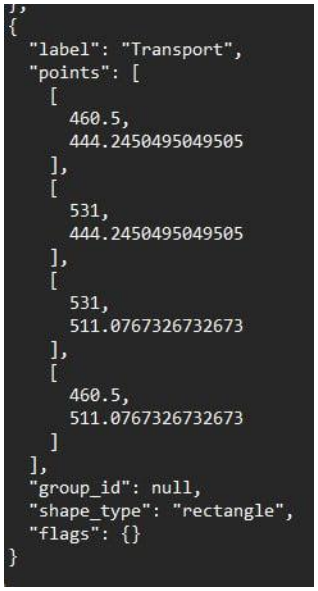
Рисунок 3.10 – Процес анотації зображення

Після закінчення анотації користувач має можливість вивантажити зображення зі збереженим положенням обмежувальних коробок та з JSON файлом в певному форматі, в якому вказані координати положення обмежувальної коробки та мітки з типом анотованого об’єкту (рис 3.11, рис 3.12, рис 3.13).



```
{
  "version": "4.5.6",
  "flags": {},
  "shapes": [
    {
      "label": "Transport",
      "points": [
        [
          445.5,
          506.6212871287129
        ],
        [
          709.5,
          506.6212871287129
        ],
        [
          709.5,
          741.2747524752475
        ],
        [
          445.5,
          741.2747524752475
        ]
      ],
      "group_id": null,
      "shape_type": "rectangle",
      "flags": {}
    },
```


Рисунок 3.11 – Результат анотації в форматі JSON



```
{
  "label": "Transport",
  "points": [
    [
      460.5,
      444.2450495049505
    ],
    [
      531,
      444.2450495049505
    ],
    [
      531,
      511.0767326732673
    ],
    [
      460.5,
      511.0767326732673
    ]
  ],
  "group_id": null,
  "shape_type": "rectangle",
  "flags": {}
}
```


Рисунок 3.12 – Результат анотації в форматі JSON

```
{
  "label": "Transport",
  "points": [
    [
      622.5,
      442.759900990099
    ],
    [
      754.5,
      442.759900990099
    ],
    [
      754.5,
      601.6707920792079
    ],
    [
      622.5,
      601.6707920792079
    ]
  ],
  "group_id": null,
  "shape_type": "rectangle",
  "flags": {}
},
{
  "label": "Transport",
```


Рисунок 3.13 – Результат анотації в форматі JSON

Також користувач отримує вивантаження анотованого зображення(рис 3.14).

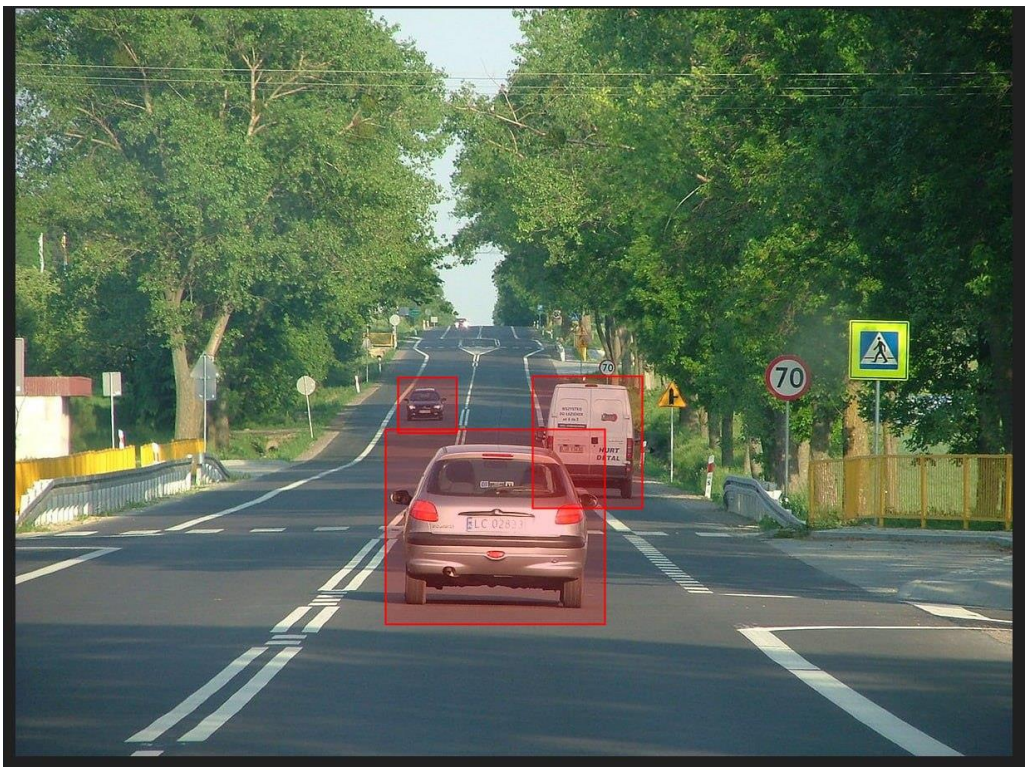


Рисунок 3.14 –Анотоване зображення

Таким чином маємо анотоване зображення та JSON файл, з необхідними даними про виділені об’єкти.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи був розроблений комплексний веб-додаток для анотування зображень, що поєднує надійні механізми автентифікації та зручний інтерфейс. Додаток використовує ряд сучасних веб-технологій і дотримується добре структурованої архітектури Model-View-Controller (MVC), що забезпечує зручність обслуговування і масштабованість. В даному проєкті використано наступні технології:

Node.js – забезпечує виконання JavaScript на стороні сервера.

Express.js - фреймворк веб-додатків для Node.js.

MongoDB – NoSQL база даних для зберігання даних.

Аналіз існуючих проєктів забезпечив створення чіткого плану роботи та очікування конкретної результату від додатку.

В даній роботі реалізовано систему автентифікації користувачів, можливість створювання задач та можливість використання інструменту анотації зображень обмежувальною коробкою, з подальшим вивантаженням і збереженням в форматі JSON.

Існує кілька напрямків для майбутніх удосконалень і покращень. Ці оновлення не лише покращать користувацький досвід, але й розширять функціональність та продуктивність програми, що зробить її більш надійним інструментом для різних завдань анотування та машинного навчання. Створення нових інструментів анотації, покращення безпеки даних користувача, можливість завантажувати та обробляти декілька зображень одночасно та інтегрування інструментів на базі штучного інтелекту для розширення можливостей платформи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Oliveira, G.L.; Burgard, W.; Brox, T. Efficient deep models for monocular road segmentation. In Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea, 9–14 October 2016; pp. 4885–4891.
2. Mendes, C.C.T.; Fremont, V.; Wolf, D.F. Exploiting Fully Convolutional Neural Networks for Fast Road Detection. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation, Stockholm, Sweden, 16–21 May 2016; Okamura, A., Menciassi, A., Ude, A., Burschka, D., Lee, D., Arrichiello, F., Liu, H., Eds.; IEEE: New York, NY, USA, 2016; pp. 3174–3179.
3. Zhang, X.; Chen, Z.; Wu, Q.M.J.; Cai, L.; Lu, D.; Li, X. Fast Semantic Segmentation for Scene Perception. *IEEE Trans. Ind. Inform.* **2019**, *15*, 1183–1192.
4. Khalilullah, K.M.I.; Jindai, M.; Ota, S.; Yasuda, T. Fast Road Detection Methods on a Large Scale Dataset for assisting robot navigation Using Kernel Principal Component Analysis and Deep Learning; IEEE: New York, NY, USA, 2018; pp. 798–803.
5. Annotation formats - https://xailient.com/blog/
6. Bounding boxes - https://www.ayadata.ai/
7. Lines and splines - https://sunix.in/lines-splines/
8. Node.js - https://developer.mozilla.org/en-US/docs/Glossary/Node.js
9. JavaScript - https://javascript.info/
10. MongoDB - https://www.mongodb.com/docs/
11. MVC - https://developer.mozilla.org/en-US/docs/Glossary/MVC
12. Data Labeling - https://xailient.com/blog/exploring-data-labeling-and-the-6-different-types-of-image-annotation/
13. Panoptic segmentation - https://www.v7labs.com/blog/coco-dataset-guide
14. Yolo - https://github.com/AlexeyAB/darknet
15. Cityscapes - https://www.cityscapes-dataset.com/

16. CVAT - https://github.com/cvat-ai/cvat
17. Keymakr - https://keymakr.com/\
18. Scale - https://scale.com/
19. Labelbox - https://labelbox.com/
20. VS - https://visualstudio.microsoft.com/
21. IntelliJ - https://www.jetbrains.com/idea/

ДОДАТОК

```
document.addEventListener('DOMContentLoaded', () => {
  const imageContainer = document.getElementById('annotation-container');
  const image = document.getElementById('image-to-annotate');
  const uploadButton = document.getElementById('upload-image');
  const fileInput = document.getElementById('image-upload');
  const saveAllButton = document.getElementById('save-all');
  let annotations = [];

  fileInput.addEventListener('change', (e) => {
    const file = e.target.files[0];
    if (file) {
      const reader = new FileReader();
      reader.onload = (e) => {
        image.src = e.target.result;
        image.style.display = 'block';
      };
      reader.readAsDataURL(file);
    }
  });

  uploadButton.addEventListener('click', () => {
    fileInput.click();
  });
```

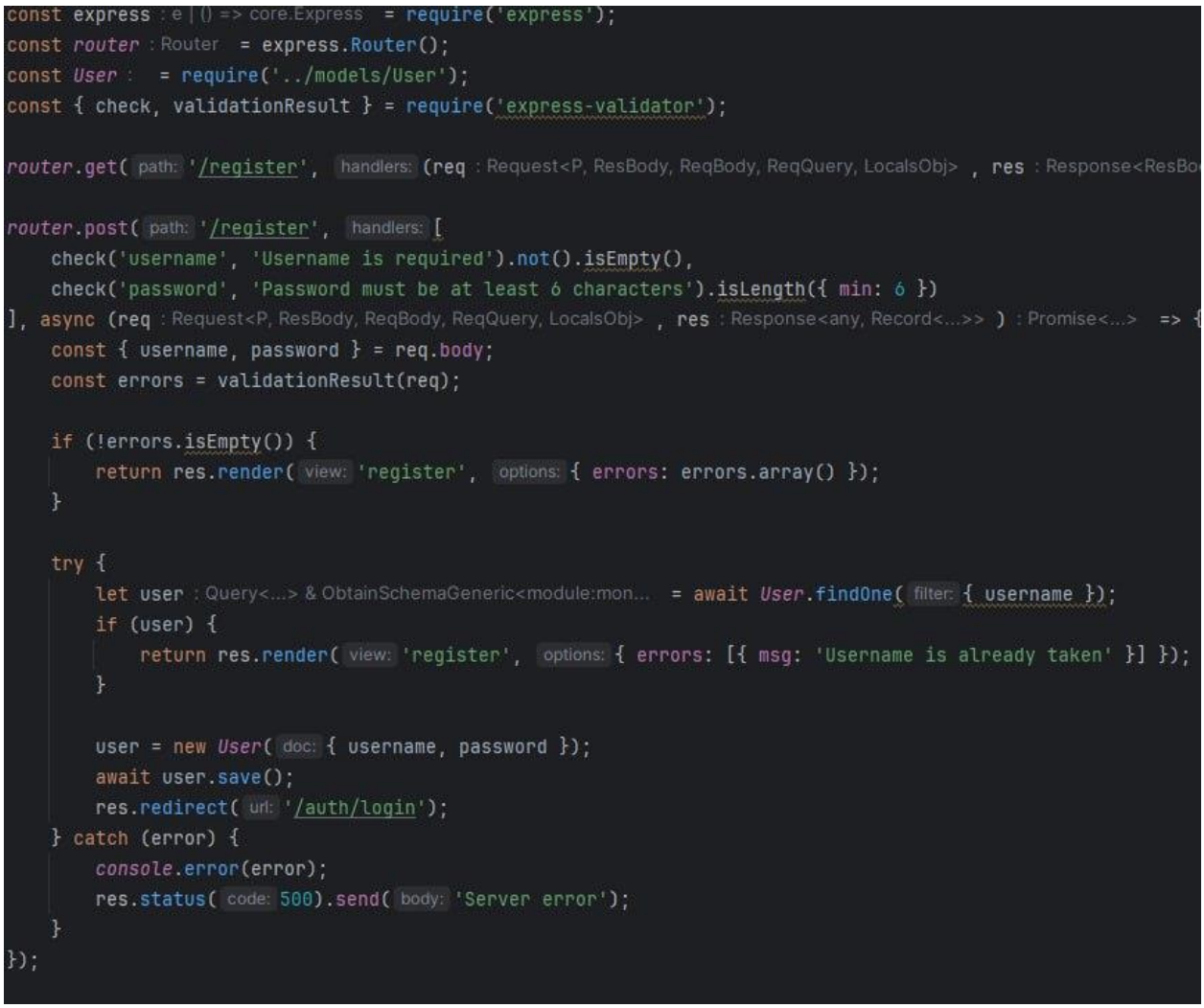
```
image.addEventListener('click', (e) => {
  const rect = image.getBoundingClientRect();
  const x = e.clientX - rect.left;
  const y = e.clientY - rect.top;

  const boundingBox = document.createElement('div');
  boundingBox.className = 'bounding-box';
  boundingBox.style.left = `${x}px`;
  boundingBox.style.top = `${y}px`;
  boundingBox.style.width = '100px';
  boundingBox.style.height = '100px';
  imageContainer.appendChild(boundingBox);

  let isResizing = false;
  let currentResizer;


  boundingBox.addEventListener('mousedown', (e) => {
    if (e.target.classList.contains('resizer')) {
      isResizing = true;
      currentResizer = e.target;
    } else {
      boundingBox.dataset.dragging = true;
      boundingBox.dataset.offsetX = e.offsetX;
      boundingBox.dataset.offsetY = e.offsetY;
    }
  });
```

```
document.addEventListener('mousemove', (e) => {
  if (isResizing) {
    const rect = boundingBox.getBoundingClientRect();
    if (currentResizer.classList.contains('bottom-right')) {
      boundingBox.style.width = `${e.clientX - rect.left}px`;
      boundingBox.style.height = `${e.clientY - rect.top}px`;
    } else if (currentResizer.classList.contains('bottom-left')) {
      boundingBox.style.width = `${rect.width - (e.clientX - rect.left)}px`;
      boundingBox.style.height = `${e.clientY - rect.top}px`;
      boundingBox.style.left = `${e.clientX}px`;
    } else if (currentResizer.classList.contains('top-right')) {
      boundingBox.style.width = `${e.clientX - rect.left}px`;
      boundingBox.style.height = `${rect.height - (e.clientY - rect.top)}px`;
      boundingBox.style.top = `${e.clientY}px`;
    } else {
      boundingBox.style.width = `${rect.width - (e.clientX - rect.left)}px`;
      boundingBox.style.height = `${rect.height - (e.clientY - rect.top)}px`;
      boundingBox.style.top = `${e.clientY}px`;
      boundingBox.style.left = `${e.clientX}px`;
    }
  } else if (boundingBox.dataset.dragging) {
    const rect = image.getBoundingClientRect();
    boundingBox.style.left = `${e.clientX - rect.left - boundingBox.dataset.offsetX}px`;
    boundingBox.style.top = `${e.clientY - rect.top - boundingBox.dataset.offsetY}px`;
  }
});

document.addEventListener('mouseup', () => {
  isResizing = false;
  delete boundingBox.dataset.dragging;
});
```

```
  const resizers = ['top-left', 'top-right', 'bottom-left', 'bottom-right'];
  resizers.forEach((pos) => {
    const resizer = document.createElement('div');
    resizer.className = `resizer ${pos}`;
    boundingBox.appendChild(resizer);
  });
});

function saveAnnotatedImageAndJSON() {  Show usages
  const canvas = document.createElement('canvas');
  const ctx = canvas.getContext('2d');
  canvas.width = image.naturalWidth;
  canvas.height = image.naturalHeight;

  ctx.drawImage(image, 0, 0, canvas.width, canvas.height);

  const scaledAnnotations = [];

  annotations.forEach((boundingBox) => {
    const rect = boundingBox.getBoundingClientRect();
    const containerRect = imageContainer.getBoundingClientRect();
    const scaleX = canvas.width / containerRect.width;
    const scaleY = canvas.height / containerRect.height;

    const x = (rect.left - containerRect.left) * scaleX;
    const y = (rect.top - containerRect.top) * scaleY;
    const width = rect.width * scaleX;
    const height = rect.height * scaleY;

    ctx.strokeStyle = 'red';
    ctx.lineWidth = 2;
    ctx.fillStyle = 'rgba(255, 0, 0, 0.2)';
    ctx.fillRect(x, y, width, height);
    ctx.strokeRect(x, y, width, height);
```

```
    scaledAnnotations.push({
      label: "Transport",
      points: [
        [x, y],
        [x + width, y],
        [x + width, y + height],
        [x, y + height]
      ],
      group_id: null,
      shape_type: "rectangle",
      flags: {}
    });
  });

  canvas.toBlob((blob) => {
    const link = document.createElement('a');
    link.href = URL.createObjectURL(blob);
    link.download = 'annotated_image.png';
    link.click();
  });

  const json = {
    version: "4.5.6",
    flags: {},
    shapes: scaledAnnotations,
    imagePath: image.src,
    imageData: null,
    imageHeight: canvas.height,
    imageWidth: canvas.width
  };

  const jsonBlob = new Blob([JSON.stringify(json, null, 2)], { type: 'application/json' });
  const jsonLink = document.createElement('a');
  jsonLink.href = URL.createObjectURL(jsonBlob);
  jsonLink.download = 'annotations.json';
  jsonLink.click();
}
```

```
const express : e | () => core.Express  = require('express');
const mongoose :   = require('mongoose');
const bodyParser : {json: Function, raw: Function, text: Function, urlencoded: Function} | {json?: any, raw?: any, text?: any, urle
const session : function({cookie?: Object, gen... | {} | {SessionData: SessionData}  = require('express-session');
const MongoStore = require('connect-mongo').default;
const authRoutes : Router | {...}  = require('./routes/auth');
const User :   = require('./models/User');
require('dotenv').config();

const app : any | Express  = express();

mongoose.connect(process.env.MONGO_URI, options: { useNewUrlParser: true, useUnifiedTopology: true }) Pron
    .then(() : any | void  => console.log('MongoDB connected')) Promise<void>
    .catch(err => console.log(err));

app.use(bodyParser.urlencoded({ extended: false }));
app.use(session( options: {
    secret: 'secret',
    resave: false,
    saveUninitialized: false,
    store: new MongoStore({ mongooseConnection: mongoose.connection })
}));
app.use(express.static( root: 'public'));
app.set('view engine', 'ejs');

app.use('/auth', authRoutes);

app.get('/dashboard', async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, Loc
    if (!req.session.userId) {
        return res.redirect( url: '/auth/login');
    }

    const user : Query<...> & ObtainSchemaGeneric<module:mon...  = await User.findById(req.session.userId);
    res.render( view: 'dashboard',  options: { username: user.username });
});

const PORT : string | number  = process.env.PORT || 5000;
app.listen(PORT,  callback: () : any | void  => console.log(`Server started on port ${PORT}`));
```



```
const express : e | () => core.Express  = require('express');
const router : Router  = express.Router();
const User :  = require('../models/User');
const { check, validationResult } = require('express-validator');

router.get( path: '/register',  handlers: (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBo

router.post( path: '/register',  handlers: [
    check('username', 'Username is required').not().isEmpty(),
    check('password', 'Password must be at least 6 characters').isLength({ min: 6 })
], async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<any, Record<...>> ) : Promise<...>  => {
    const { username, password } = req.body;
    const errors = validationResult(req);

    if (!errors.isEmpty()) {
        return res.render( view: 'register',  options: { errors: errors.array() });
    }

    try {
        let user : Query<...> & ObtainSchemaGeneric<module:mon...  = await User.findOne( filter: { username });
        if (user) {
            return res.render( view: 'register',  options: { errors: [{ msg: 'Username is already taken' }] });
        }

        user = new User( doc: { username, password });
        await user.save();
        res.redirect( url: '/auth/login');
    } catch (error) {
        console.error(error);
        res.status( code: 500).send( body: 'Server error');
    }
});
```

```
router.get( path: '/login', handlers: (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<Res

router.post( path: '/login', handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Respo
    const { username, password } = req.body;

    try {
        const user : Query<...> & ObtainSchemaGeneric<module:mon...  = await User.findOne( filter: { username });
        if (!user) {
            return res.render( view: 'login', options: { errors: [{ msg: 'Invalid credentials' }] });
        }

        const isMatch = await user.isValidPassword(password);
        if (!isMatch) {
            return res.render( view: 'login', options: { errors: [{ msg: 'Invalid credentials' }] });
        }

        req.session.userId = user.id;
        res.redirect( url: '/dashboard');
    } catch (error) {
        console.error(error);
        res.status( code: 500).send( body: 'Server error');
    }
});

router.get( path: '/logout', handlers: (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<Re
    req.session.destroy((err) : any | undefined  => {
        if (err) {
            return res.redirect( url: '/dashboard');
        }

        res.redirect( url: '/auth/login');
    });
});

module.exports = router;
```

```
const express : e | () => core.Express  = require('express');
const router : Router  = express.Router();
const Project = require('../models/Project');
const Image = require('../models/Image');
const multer = require('multer');
const path : PlatformPath | path  = require('path');
const storage = multer.diskStorage({
    destination: (req, file, cb) : void  => {
        cb(null, 'public/uploads/');
    },
    filename: (req, file, cb) : void  => {
        cb(null, `${Date.now()}-${file.originalname}`);
    },
});
```

```
const upload = multer({ storage });
router.get( path: '/projects', handlers: async (req : Request<...> , res : Response<...> ) : Promise<...> => {
    const projects = await Project.find({ user: req.session.userId });
    res.render( view: 'projects', options: { projects });
});
router.post( path: '/projects', handlers: async (req : Request<...> , res : Response<...> ) : Promise<...> => {
    const { name } = req.body;
    const project = new Project({ name, user: req.session.userId });
    await project.save();
    res.redirect( url: '/projects');
});
router.get( path: '/projects/:id', handlers: async (req : Request<...> , res : Response<...> ) : Promise<...> => {
    const project = await Project.findById(req.params.id);
    const images = await Image.find({ project: req.params.id });
    res.render( view: 'project', options: { project, images });
});
router.post( path: '/projects/:id/images', upload.single('image'), async (req : Request<...> , res : Response<...
    const { file } = req;
    const image = new Image({ filename: file.filename, project: req.params.id });
    await image.save();
    res.redirect( url: `/projects/${req.params.id}`);
});
router.get( path: '/projects/:projectId/images/:imageId/annotate', handlers: async (req : Request<...> , res : Re
    const image = await Image.findById(req.params.imageId);
    res.render( view: 'annotate', options: { image });
});
router.post( path: '/projects/:projectId/images/:imageId/annotate', handlers: async (req : Request<...> , res : R
    const { annotations } = req.body;
    const image = await Image.findById(req.params.imageId);
    image.annotations = JSON.parse(annotations);
    await image.save();
    res.redirect( url: `/projects/${req.params.projectId}`);
});
module.exports = router;
```