

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Веб-додаток для генерації коду на основі синтаксису PlantUML»

Виконав студент групи ПЗ-21-5
ХРОБУСТ Антон Олексійович

Керівник роботи:
МОРОЗОВ Андрій Васильович

Рецензент:
КОНЦИДАЙЛО Андрій Михайлович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ХРОБУСТ Антон Олексійович**

Керівник роботи: **МОРОЗОВ Андрій Васильович**

Тема роботи: **«Веб-додаток для генерації коду на основі синтаксису PlantUML»**,

затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: об'єктом дослідження є система автоматизованого перетворення UML діаграм класів у програмний код різними мовами програмування. Предметом дослідження є принципи та алгоритми парсингу PlantUML нотації, методи генерації коду для множинних мов програмування та архітектурні рішення веб-платформи з інтернаціоналізацією.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Морозов А.В.	27.02.2025	27.02.2025
2	Морозов А.В.	27.02.2025	27.02.2025
3	Морозов А.В.	27.02.2025	27.02.2025

РЕФЕРАТ

Кваліфікаційна робота бакалавра присвячена розробці та дослідженню системи автоматичної генерації програмного коду з діаграм UML на базі інструменту UML2Code. Пояснювальна записка до роботи містить 59 сторінок, 30 ілюстрацій та 30 використаних джерел.

Метою роботи є спрощення процесу розробки програмного забезпечення шляхом створення ефективної системи, що дозволяє автоматично перетворювати UML-діаграми в код різних мов програмування, забезпечуючи зменшення кількості помилок і прискорення розробки.

У ході роботи було детально досліджено та проаналізовано сучасні підходи до автоматичної генерації коду з використанням UML-діаграм, існуючі рішення на ринку та їхні недоліки. На основі отриманих результатів було розроблено власний інструмент UML2Code, який вирішує завдання генерації високоякісного та підтримуваного коду. Система підтримує генерацію коду кількома популярними мовами програмування, включаючи Java, C#, Python та інші, та дозволяє легко налаштовувати шаблони генерації.

Ефективність розробленого інструменту було перевірено практично. Результати продемонстрували суттєве скорочення часу на розробку програмного забезпечення, зменшення кількості помилок на етапі кодування та покращення загальної якості створюваних програмних продуктів.

Розроблена система є готовим до використання інструментом, який рекомендований для інтеграції в процеси розробки програмного забезпечення, що базується на UML-діаграмах.

КЛЮЧОВІ СЛОВА: UML, ГЕНЕРАЦІЯ КОДУ, UML2CODE, ПРОГРАМНА ІНЖЕНЕРІЯ, JAVA, C#, PYTHON, АВТОМАТИЗАЦІЯ.

					ІПЗ.КР.Б – 121 – 25 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Веб-додаток для генерації коду на основі синтаксису PlantUML	Літ.	Арк.	Аркушів
Розроб.		Хробуст А.О.						
Керівник		Морозов А.В.					3	59
Рецензент		Концидайло А.М.				ФІКТ Гр. ІПЗ-21-5		
Зав. каф.		Вакалюк Т. А.						

ABSTRACT

The bachelor's thesis is dedicated to the development and research of an automated code generation system from UML diagrams based on the UML2Code tool. The explanatory note to the work contains 59 pages, 29 illustrations, and 30 references.

The main objective of the work is to simplify the software development process by creating an efficient system capable of automatically converting UML diagrams into code for various programming languages, thus reducing errors and accelerating development.

The work provides an in-depth analysis of current approaches to automatic code generation using UML diagrams, existing market solutions, and their limitations. Based on this analysis, the UML2Code tool was developed, solving the task of generating high-quality, maintainable code. The system supports code generation in several popular programming languages, including Java, C#, Python, among others, and allows easy customization of generation templates.

The effectiveness of the developed tool was confirmed through practical experiments. Results demonstrated significant reductions in software development time, decreased error rates during coding phases, and overall improvement in software quality.

The developed system is a ready-to-use tool recommended for integration into UML-based software development processes.

KEYWORDS: UML, CODE GENERATION, UML2CODE, SOFTWARE ENGINEERING, JAVA, C#, PYTHON, AUTOMATION.

ЗМІСТ

ВСТУП.....	6
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ: PLANTUML ТА АЛГОРИТМИ ГЕНЕРАЦІЇ КОДУ	8
1.1. Постановка задачі.....	8
1.2. Аналіз аналогів програмного продукту	9
1.3. Вибір архітектури веб-додату для генерації коду на основі синтаксису PlantUML.....	11
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	16
Висновки до першого розділу	19
РОЗДІЛ 2. РОЗРОБКА ТА ОПТИМІЗАЦІЯ ВЕБ-ДОДАТКУ ДЛЯ ГЕНЕРАЦІЇ КОДУ НА ОСНОВІ СИНТАКСИСУ PLANTUML.....	20
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	20
2.2. Розробка бази даних веб-додатку	23
2.3. Проектування та реалізація алгоритмів роботи системи.....	25
2.4. Реалізація системи конвертації UML діаграм у програмний код	29
Висновки до другого розділу	33
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ПЛАТФОРМОЮ ДЛЯ ГЕНЕРАЦІЇ КОДУ НА ОСНОВІ СИНТАКСИСУ PLANTUML	36
3.1. Порядок встановлення та налаштування параметрів системи.....	36
3.2. Структура інтерфейсу веб-додатку	38
Висновки до третього розділу.....	48
ВИСНОВКИ	50
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	52
ДОДАТКИ.....	55

ВСТУП

Цифрова епоха характеризується постійним зростанням складності програмних продуктів, що зумовлює потребу у застосуванні ефективних методів управління процесами їх розробки, тестування, впровадження та подальшого супроводу. З розвитком технологій підвищуються вимоги до якості, швидкості та структурованості програмних рішень, тому особливої актуальності набувають інструменти, які дозволяють автоматизувати окремі етапи життєвого циклу програмного забезпечення.

Одним із провідних підходів до проєктування складних програмних систем є застосування уніфікованої мови моделювання (UML), яка дозволяє формалізовано описувати архітектуру, поведінку та взаємозв'язки між компонентами програмної системи. UML-діаграми значно полегшують візуальне представлення структури системи, однак процес перетворення моделей у робочий програмний код часто виконується вручну, що призводить до зростання витрат часу, ймовірності помилок і дублювання зусиль.

Саме тому останніми роками активно розвивається напрямок автоматизації генерації коду на основі UML-діаграм. Такий підхід дозволяє суттєво скоротити тривалість розробки програмного забезпечення, зменшити кількість помилок при кодуванні, а також забезпечити кращу відповідність кінцевого продукту проєктній документації. Інструменти генерації коду з UML слугують важливою ланкою між етапами моделювання та реалізації програмних систем, дозволяючи автоматично трансформувати концептуальні моделі у вихідний код на конкретних мовах програмування.

Одним із перспективних інструментів у цій сфері є UML2Code — програмна система, що дозволяє ефективно автоматизувати трансляцію UML-діаграм у програмний код мовами Java, C#, Python, PHP тощо. Цей інструмент підтримує налаштовувані шаблони генерації, що робить його адаптивним до вимог конкретних проєктів. Завдяки можливості інтеграції з сучасними середовищами розробки та використанню сучасної веб-архітектури, UML2Code

може бути корисним як для індивідуальних розробників, так і для командної роботи.

Актуальність теми полягає в необхідності дослідження методів і засобів, які забезпечують автоматизацію процесу генерації якісного, структурованого програмного коду з UML-діаграм. Це дозволяє не лише підвищити продуктивність праці розробників, але й покращити підтримуваність програмних рішень, скоротити терміни виводу продукту на ринок та забезпечити відповідність програмного коду його архітектурному опису.

Метою випускної кваліфікаційної роботи бакалавра є розробка ефективної веб-системи автоматичної генерації програмного коду на основі UML-діаграм із використанням інструменту UML2Code, дослідження її функціональних можливостей та оцінка ефективності роботи на прикладах реальних програмних проєктів.

Завдання на кваліфікаційну роботу становлять дослідження синтаксису мови PlantUML з метою виявлення ключових конструкцій для генерації коду; аналіз наявних рішень автоматичного перетворення UML-діаграм у програмний код; розробка архітектури веб-додатку з використанням фреймворку Symfony; створення алгоритмів парсингу та генерації коду для мов Java, PHP, Python і C#; реалізація RESTful API та інтерактивного веб-інтерфейсу на базі Twig і Monaco Editor; впровадження засобів автентифікації, авторизації, логування, зберігання історії операцій та інтеграції з OAuth; а також контейнеризація системи за допомогою Docker і тестування її ефективності на практичних прикладах.

Об'єктом дослідження виступають системи автоматизованого перетворення UML-діаграм у програмний код для різних мов програмування.

Предметом дослідження є алгоритми та методи автоматичної генерації коду, архітектурні рішення при створенні генераторів коду, а також підходи до інтеграції генератора у середовище розробки програмного забезпечення.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ: PLANTUML ТА АЛГОРИТМИ ГЕНЕРАЦІЇ КОДУ

1.1. Постановка задачі

Метою даної роботи є розробка інтегрованої веб-системи, яка автоматизує процес генерації коду за текстовими описами, написаними у синтаксисі PlantUML. Сучасні засоби програмної інженерії та моделювання активно впроваджуються у розробку програмного забезпечення, проте ручний перехід від UML-діаграм до коду залишається трудомістким і схильним до помилок. Саме тому існує необхідність створення системи, яка дозволить не лише перетворювати опис архітектури у готовий код, а й забезпечить додаткові можливості для конвертації у різні мови програмування.

У межах цієї роботи передбачено кілька основних завдань:

По-перше, необхідно провести аналіз синтаксису PlantUML, зосередившись на вивченні основних конструкцій та принципів опису UML-діаграм, щоб визначити ключові елементи, які слід враховувати при генерації коду.

Далі потрібно розробити ефективні алгоритми, які дозволятимуть перетворювати текстові описи, створені за допомогою PlantUML, у вихідний код. Особлива увага приділяється тому, щоб забезпечити підтримку генерації коду для різних мов програмування, таких як PHP, Python, Java тощо, що підвищить універсальність та адаптивність системи до потреб різних проєктів.

Наступним кроком є створення інтерактивного веб-інтерфейсу з використанням технологій Symfony та Twig, який забезпечить зручне редагування текстових описів, миттєве відображення згенерованого коду та можливість вибору мови програмування для конвертації.

Останнім завданням є контейнеризація додатку за допомогою Docker, що дозволить створити портативне середовище розгортання з простим встановленням, масштабованістю та стабільною роботою в різних операційних середовищах.

Таким чином, завданням даної роботи є створення комплексного веб-додатку, який автоматизує процес генерації коду з описів на основі PlantUML, забезпечує можливість конвертації у різні мови програмування, а також підтримує селф-хостинг через використання контейнеризації. Це дозволить оптимізувати процес розробки програмних продуктів, знизити ризик помилок і підвищити ефективність роботи розробників.

1.2. Аналіз аналогів програмного продукту

Аналіз аналогів:

1. Yeschat.ai PlantUML Code Generator (<https://www.yeschat.ai/gpts-2OTogs7cyh-PlantUML-GPT>):

Цей сервіс є веб-інструментом, який використовує сучасні підходи, зокрема, алгоритми на основі моделей штучного інтелекту для генерації коду з описів, написаних за допомогою синтаксису PlantUML. Основними перевагами даного рішення є інтуїтивно зрозумілий інтерфейс та швидкість отримання результатів. Проте детальна інформація щодо підтримуваних мов програмування, ліцензійних умов та можливостей інтеграції може бути обмеженою або не розкритою для користувача.

2. puml2code (GitHub – <https://github.com/jupe/puml2code>):

Це open-source проект, розміщений на GitHub, який реалізує алгоритмічний підхід до конвертації UML-описів у код. Основною перевагою є відкритість коду, що дозволяє користувачам вносити власні зміни та адаптувати продукт під конкретні потреби. Однак, інтерфейс користувача може бути мінімалістичним (іноді навіть обмежуватися CLI або простим веб-інтерфейсом), а підтримка сучасних веб-технологій може відставати від комерційних рішень.

Таблиця 1.1. Порівняння аналогів

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						9

Характеристика	Yeschat.ai PlantUML Code Generator	puml2code (GitHub)	Наш продукт
Тип продукту	Веб-сервіс / Комерційний інструмент	Open-source проект	Веб-додаток (Symfony + Twig, Docker)
Ліцензія	Невідома (ймовірно комерційна)	Відкрита MIT	Формування ліцензійної політики залежно від вибору
Інтерфейс користувача	Інтерактивний веб- інтерфейс, сучасний дизайн	Простий, можливо CLI або мінімалістичний веб- інтерфейс	Інтерактивний, зручний веб- інтерфейс з можливістю розширення
Підтримка мов програмування	Конвертація у декілька мов (деталі можуть бути обмеженими)	Підтримка основних мов, визначених розробником	Планується багатомовна підтримка (PHP, Python, Java та ін.)
Алгоритми генерації коду	Використання ШІ (GPT-подібні моделі) для генерації коду	Алгоритмічний підхід, базований на аналізі тексту	Розробка власних алгоритмів з можливістю розширення функціоналу
Контейнеризація / Селф-хостинг	Найчастіше орієнтований на хмарне розгортання	Можливість розгортання за допомогою Docker (за необхідності)	Вбудована підтримка Docker для селф- хостингу та масштабування
Документація та підтримка	Документація може бути обмеженою	Документація завжди доступна через GitHub та спільноту	Планується створення повної документації з прикладми використання
Інтеграція та розширюваність	Обмежені можливості інтеграції з іншими сервісами	Можливості інтеграції через API або розширення за рахунок відкритого коду	Проект розробляється з орієнтацією на високу інтеграцію та можливість розширення функціоналу

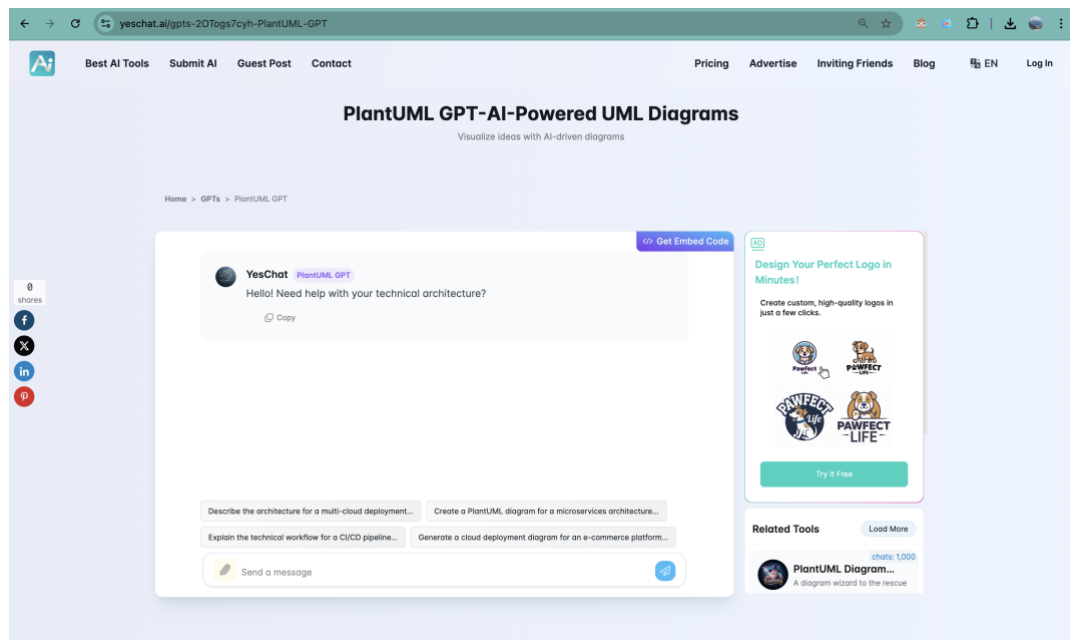


Рис. 1.1 Скріншот інтерфейсу Yeschat.ai PlantUML Code Generator

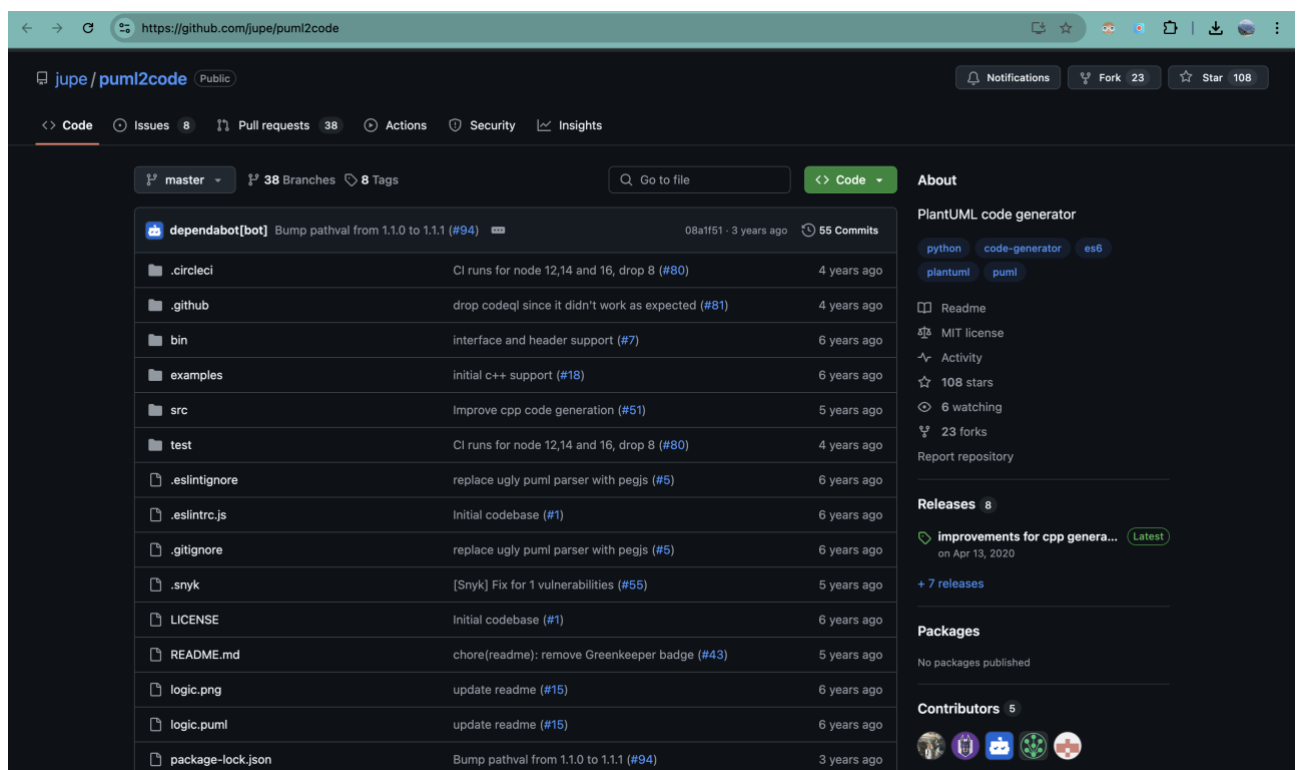


Рис. 1.2 Скріншот репозиторію puml2code

1.3. Вибір архітектури веб-додату для генерації коду на основі синтаксису PlantUML

У цьому розділі розглядається архітектурне рішення для розробки веб-

додатку, який забезпечує автоматизацію конвертації UML діаграм класів у програмний код. Основна мета – створення системи, здатної приймати діаграми класів у синтаксисі PlantUML, аналізувати їх структуру та конвертувати у робочий код для декількох мов програмування (Java, C#, PHP, Python), забезпечуючи при цьому підтримку кешування та розгортання за допомогою Docker. У якості бази даних обрано PostgreSQL завдяки її високій надійності, масштабованості та потужним можливостям для зберігання історії конвертацій.

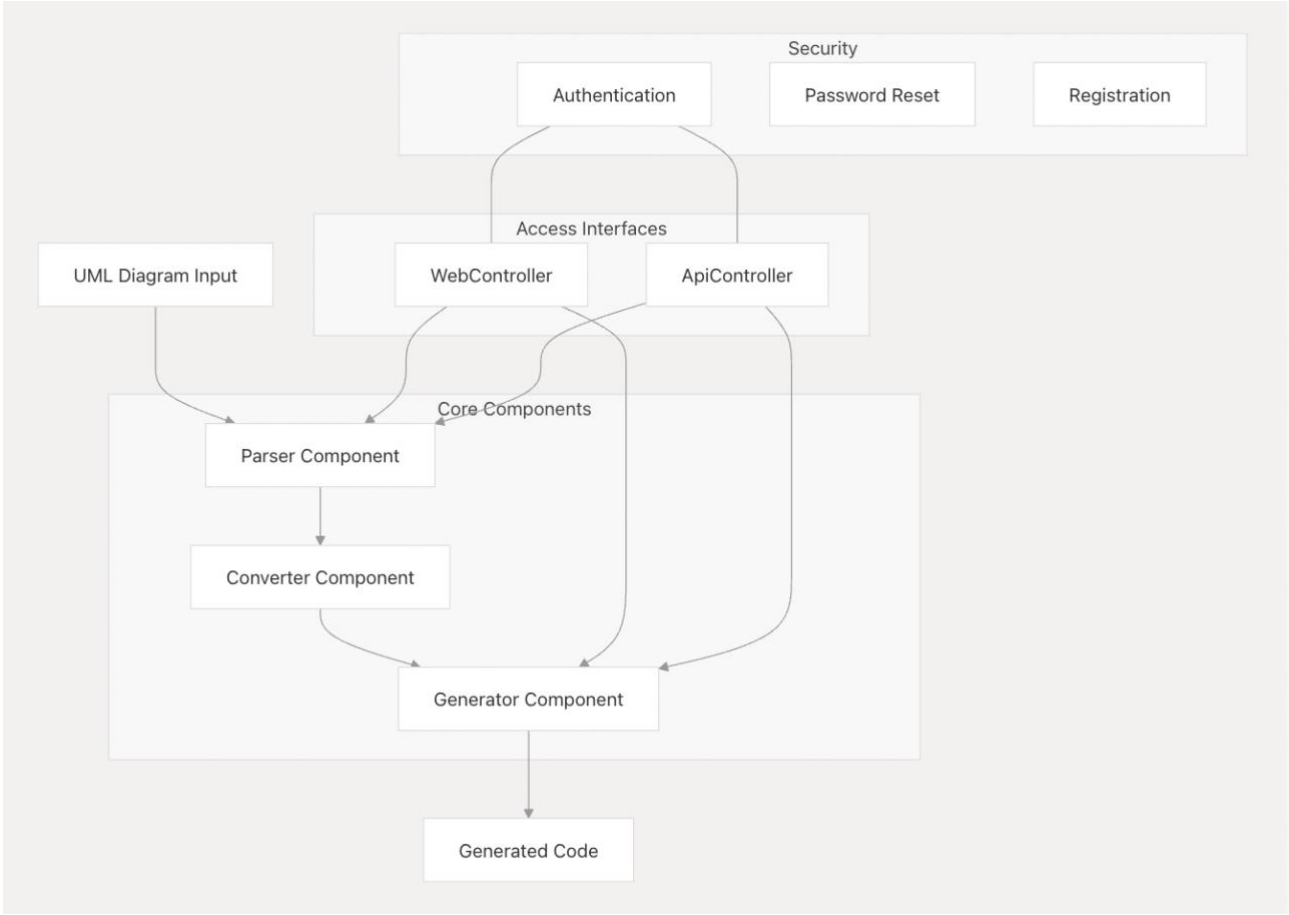


Рис. 1.3 Архітектура веб-додатку для генерації коду

Архітектура системи побудована на сучасному фреймворку Symfony з використанням принципів чистої архітектури та розподілом на шари відповідно до патерну MVC. Основні компоненти системи можна розділити на кілька рівнів:

1. Клієнтський рівень (Frontend)

Інтерфейс користувача реалізований за допомогою шаблонізатора Twig, який забезпечує рендеринг HTML-сторінок з підтримкою двох мов (української та англійської). Інтерактивність додатку посилюється за рахунок інтеграції Monaco Editor для редагування UML-діаграм та перегляду згенерованого коду. Система підтримує світлу та темну теми оформлення, які автоматично синхронізуються з налаштуваннями редактора. Взаємодія через HTTP реалізована за допомогою RESTful API, що дозволяє клієнту надсилати запити до сервера у форматі JSON. Це забезпечує гнучкий інтерфейс для всіх основних операцій: парсингу UML, генерації коду та прямої конвертації.

2. Рівень контролерів

Symfony контролери обробляють вхідні HTTP-запити та розподіляють їх між трьома основними модулями системи: Parser (для аналізу UML), Generator (для генерації коду) та Converter (для прямої конвертації). Контролери виконують валідацію вхідних даних, делегують виконання бізнес-логіки відповідним сервісам та формують відповіді. Завдяки модульному підходу та чистій архітектурі, кожен компонент залишається незалежним та легко розширюваним.

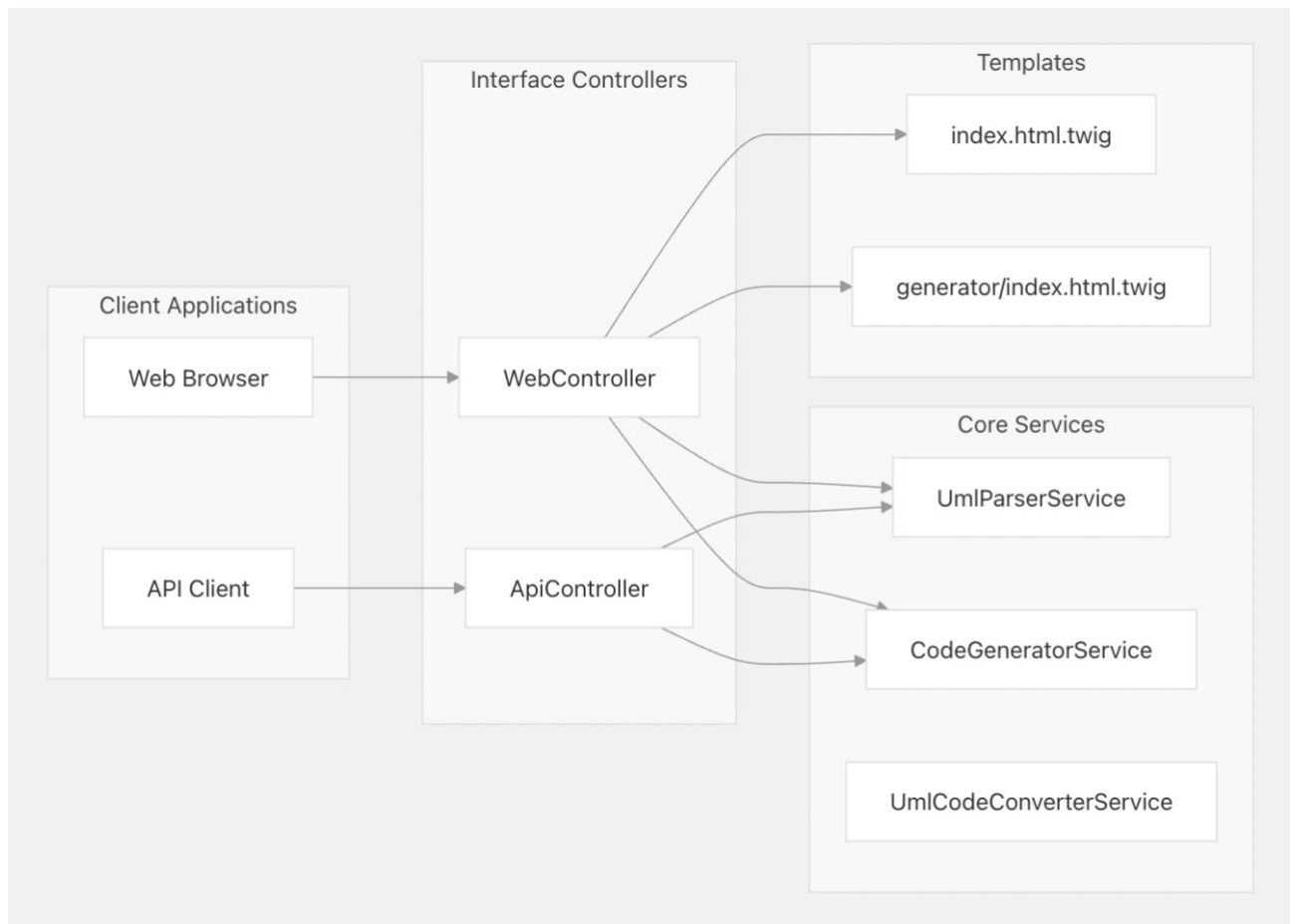


Рис. 1.4. Діаграма взаємодії основних компонентів системи

3. Рівень сервісів (Service Layer)

В основі бізнес-логіки системи лежать три ключові сервісні модулі, кожен з яких відповідає за окремий аспект обробки UML діаграм. Сервіси реалізують чисту архітектуру з чітким розподілом на доменну логіку та інфраструктурні компоненти. Parser Service відповідає за синтаксичний аналіз PlantUML діаграм та їх перетворення у структурований JSON формат. Цей сервіс використовує спеціалізований парсер, який розпізнає елементи діаграми класів: класи, інтерфейси, відношення між ними, атрибути та методи. Generator Service приймає JSON-представлення діаграми та генерує програмний код відповідно до обраної мови програмування. Система підтримує генерацію коду для Java, C#, PHP та Python, при цьому архітектура дозволяє легко додавати підтримку нових мов завдяки використанню патерну Strategy для різних генераторів коду.

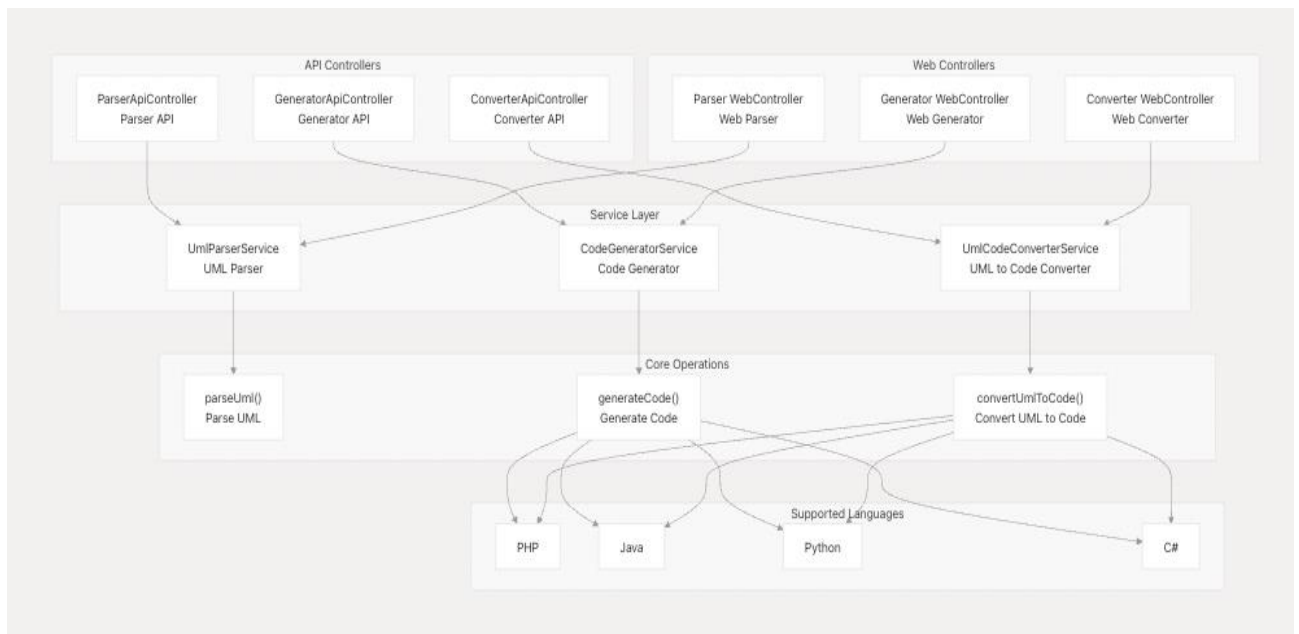


Рис.1.5. Діаграма структури сервісного шару

4. Інфраструктурний рівень

Система розгортається у контейнеризованому середовищі за допомогою Docker, що забезпечує ізоляцію компонентів та спрощує процес розгортання. Docker-композиція включає контейнери для веб-сервера Nginx, PHP-FPM та PostgreSQL бази даних. База даних PostgreSQL використовується для зберігання:

- Історії конвертацій з можливістю відстеження змін
- Користувацьких налаштувань та пререференцій
- Даних автентифікації та авторизації
- Статистики користувачів
- Скидування паролів
- Системних логів

5. Система безпеки та авторизації

Реалізовано багаторівневу систему безпеки, що включає:

- Автентифікацію користувачів з підтримкою ролей (ROLE_USER, ROLE_ADMIN)
- Захист API-endpoints через систему токенів
- Валідацію вхідних даних на рівні контролерів
- Логування всіх критичних операцій

- Механізм відновлення паролю та управління сесіями

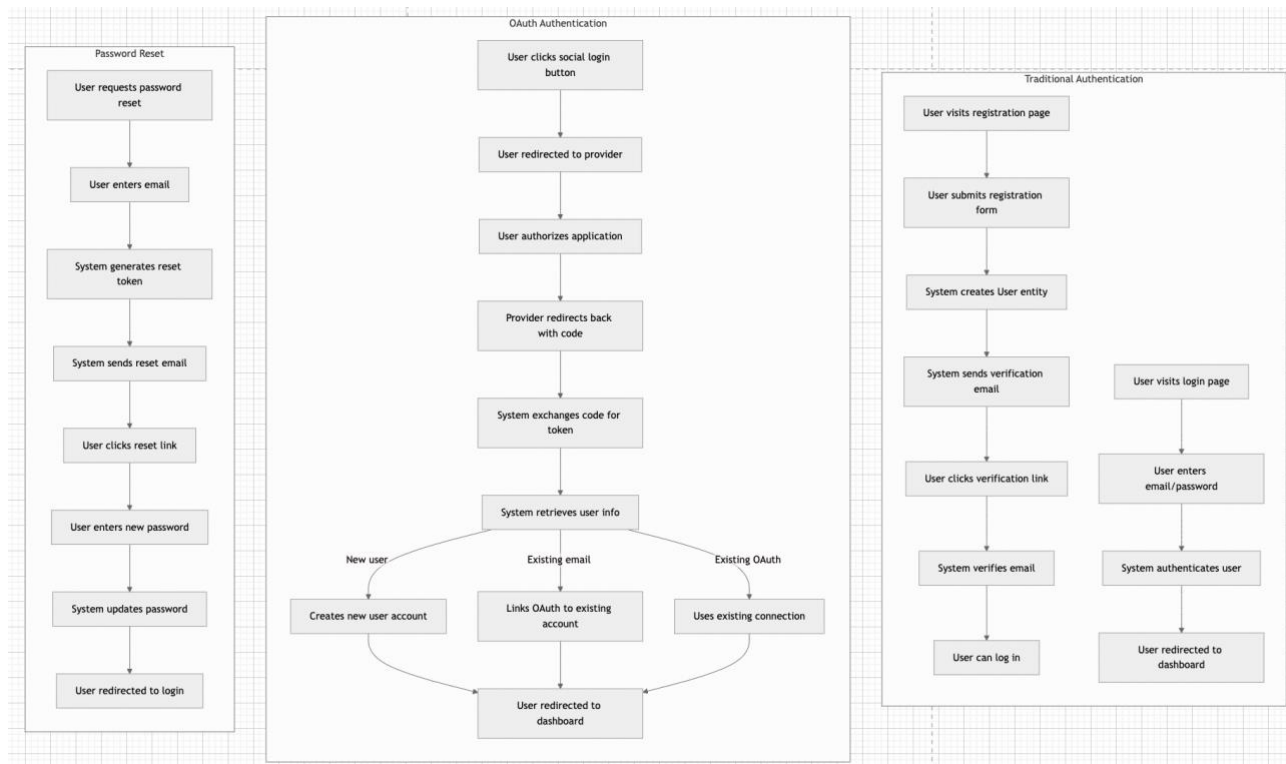


Рис.1.6. Діаграма безпеки та авторизації

Така архітектура забезпечує високу масштабованість системи, можливість легкого розширення функціоналу та додавання підтримки нових мов програмування. Використання сучасних технологій та практик розробки, таких як контейнеризація, кешування та чиста архітектура, дозволяє досягти високої продуктивності та надійності системи при збереженні простоти її підтримки та розвитку.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

При розробці системи конвертації UML діаграм у програмний код було обрано сучасний стек технологій, який забезпечує надійність, масштабованість та зручність розробки. Основним фреймворком для серверної частини було обрано Symfony 7.2, який надає потужну екосистему для створення високопродуктивних веб-додатків на PHP.

Symfony вирізняється серед інших PHP фреймворків, таких як Laravel чи Yii, своєю архітектурною гнучкістю та потужними можливостями для реалізації складних бізнес-процесів. Фреймворк забезпечує вбудовану підтримку інтернаціоналізації, що дозволило реалізувати двомовний інтерфейс (українська та англійська мови) з використанням YAML-файлів для зберігання перекладів. Важливою перевагою Symfony є його модульна архітектура та потужна система компонентів, що дозволило ефективно організувати код для парсингу UML діаграм, генерації коду та конвертації між різними мовами програмування.

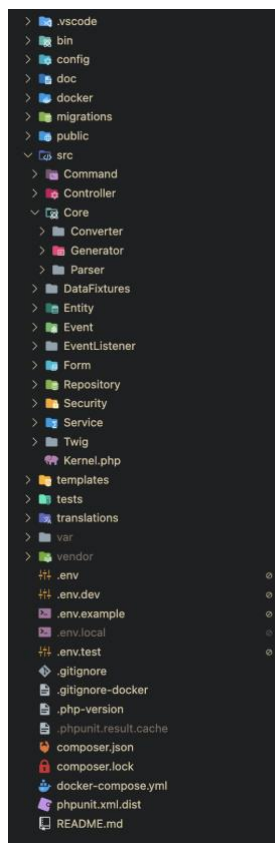


Рис.1.7. Файлова структура проекту

Для клієнтської частини було обрано Monaco Editor - потужний веб-редактор коду, який використовується в Visual Studio Code. На відміну від простих текстових редакторів, Monaco Editor надає розширені можливості для роботи з кодом, включаючи підсвічування синтаксису PlantUML, автодоповнення та валідацію синтаксису. Інтеграція Monaco Editor дозволила створити професійне середовище для редагування UML діаграм та перегляду згенерованого коду з підтримкою світлої та темної тем оформлення.

Для забезпечення надійного зберігання даних було обрано PostgreSQL - об'єктно-реляційну систему управління базами даних. PostgreSQL, на відміну від MySQL, надає кращу підтримку для складних запитів та транзакцій, що важливо для зберігання історії конвертацій та користувацьких налаштувань. База даних забезпечує високу продуктивність при одночасній роботі багатьох користувачів та надійне зберігання даних.

Для розгортання системи використовується Docker, що дозволяє створити ізольовані контейнери для кожного компонента системи: веб-сервера Nginx, PHP-FPM, PostgreSQL. Порівняно з традиційним розгортанням або альтернативними системами контейнеризації (Podman, LXD), Docker забезпечує кращу стандартизацію середовища розробки та експлуатації, спрощує процес масштабування та оновлення компонентів системи. Мінімальні вимоги до апаратного забезпечення для розгортання системи:

- Процесор: 2 ядра, 2.0 GHz
- Оперативна пам'ять: 4 GB RAM
- Дисковий простір: 20 GB
- Операційна система: Linux/Unix-подібна система з підтримкою Docker

Рекомендовані вимоги для комфортної роботи з системою:

- Процесор: 4 ядра, 2.5 GHz
- Оперативна пам'ять: 8 GB RAM
- Дисковий простір: 40 GB SSD
- Мережеве з'єднання: 10 Mbps

Обраний стек технологій дозволяє створити надійну, масштабовану систему для конвертації UML діаграм у програмний код з підтримкою декількох мов програмування. Використання сучасних інструментів розробки забезпечує високу продуктивність роботи, зручний користувацький інтерфейс та можливість подальшого розширення функціоналу системи.

Висновки до першого розділу

Аналіз предметної області та вимог до системи конвертації UML діаграм у програмний код дозволив визначити ключові аспекти, які необхідно врахувати при розробці. В результаті дослідження існуючих рішень було виявлено, що критично важливими характеристиками системи є підтримка різних мов програмування, точність конвертації UML-нотацій та зручність користувацького інтерфейсу.

Для реалізації системи було обрано оптимальну архітектуру, що базується на принципах чистої архітектури з розподілом на шари та використанням RESTful API для комунікації між компонентами. В якості основного фреймворку було обрано Symfony 7.2, який забезпечує потужну екосистему для розробки серверної частини, вбудовану підтримку інтернаціоналізації та гнучку систему маршрутизації. Для створення інтерактивного середовища редагування UML діаграм було інтегровано Monaco Editor, який надає професійні можливості для роботи з кодом. Порівняно з альтернативними рішеннями, такими як простий текстовий редактор чи ACE Editor, Monaco Editor забезпечує кращу підтримку синтаксису PlantUML та більш зручний користувацький досвід.

В якості системи управління базами даних було обрано PostgreSQL, що вирізняється надійністю та підтримкою складних запитів, необхідних для роботи з історією конвертацій. Для забезпечення ізоляції компонентів та спрощення розгортання системи використовується Docker, який, порівняно з Podman та LXD, має кращу підтримку оркестрації контейнерів та більш розвинену екосистему. Обраний стек технологій є оптимальним для реалізації поставлених завдань та забезпечує необхідну гнучкість для подальшого розвитку системи.

РОЗДІЛ 2. РОЗРОБКА ТА ОПТИМІЗАЦІЯ ВЕБ-ДОДАТКУ ДЛЯ ГЕНЕРАЦІЇ КОДУ НА ОСНОВІ СИНТАКСИСУ PLANTUML

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Бізнес-вимоги

Основною метою проекту є створення веб-системи для автоматизованої конвертації UML діаграм класів у програмний код різними мовами програмування. Система має забезпечувати повний цикл роботи з UML діаграмами: від їх створення через парсинг до генерації готового коду, підтримуючи при цьому різні мови програмування та зберігаючи історію конвертацій.

Система надає розширені можливості для роботи з UML діаграмами, включаючи інтерактивний редактор з підсвічуванням синтаксису PlantUML, валідацією введених даних та автодоповненням коду. Користувачі можуть працювати з діаграмами в зручному веб-інтерфейсі з підтримкою світлої та темної тем оформлення.

Важливою складовою є можливість генерації коду для різних мов програмування (Java, C#, PHP, Python) з підтримкою різних версій мов та можливістю розширення для додавання нових мов. Система зберігає історію всіх конвертацій, що дозволяє користувачам відстежувати зміни та повертатися до попередніх версій згенерованого коду.

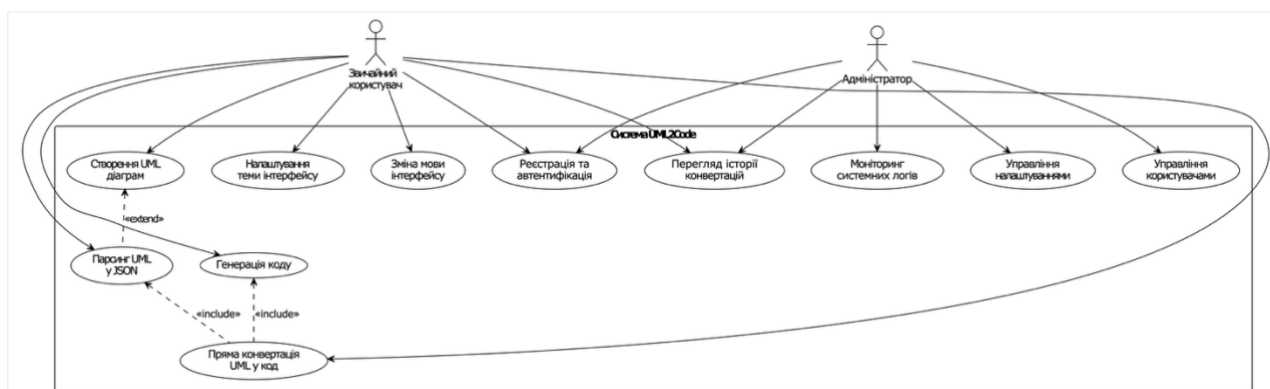


Рис.2.1. Діаграма використання системи

Вимоги користувачів

Звичайний користувач (ROLE_USER):

- Створення та редагування UML діаграм класів у PlantUML форматі
- Конвертація UML діаграм у програмний код обраною мовою програмування
- Перегляд історії конвертацій та завантаження згенерованих файлів
- Налаштування персональних преференцій (мова інтерфейсу, тема оформлення)
- Доступ до API для автоматизації процесу конвертації

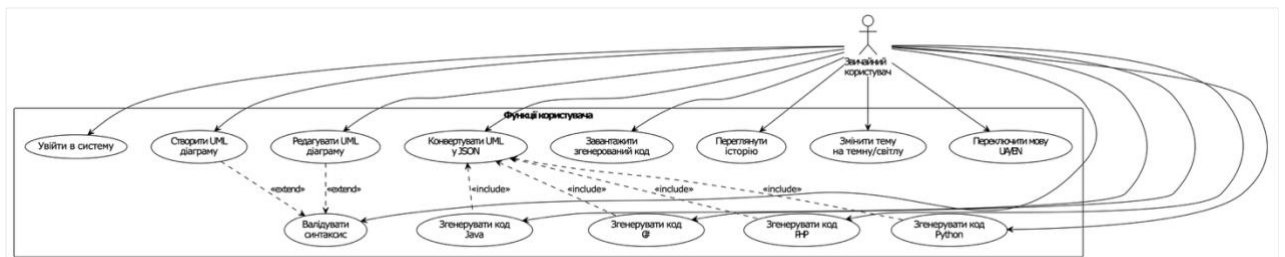


Рис.2.2. Діаграма використання звичайного користувача

Адміністратор системи (ROLE_ADMIN):

- Управління користувачами системи
- Моніторинг системних логів та історії дій користувачів
- Налаштування параметрів системи та підтримуваних мов програмування
- Доступ до розширеної статистики використання системи

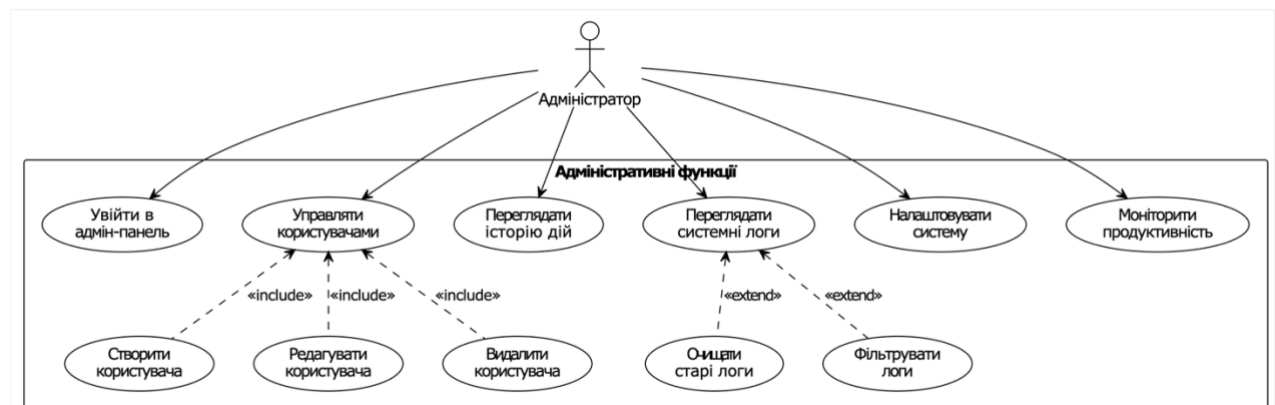


Рис.2.3. Діаграма використання адміністратора

Функціональні вимоги

Система автентифікації та авторизації забезпечує розмежування доступу

до функціоналу системи. Неавторизовані користувачі мають доступ лише до загальної інформації та сторінки входу. Авторизовані користувачі отримують доступ до всіх функцій конвертації та історії своїх операцій. Адміністратори мають повний доступ до управління системою через спеціальний адміністративний інтерфейс.

Система надає RESTful API для програмного доступу до функцій конвертації, що дозволяє інтегрувати її з іншими системами та автоматизувати процеси генерації коду.

Нефункціональні вимоги

Час навчання роботи з системою для нових користувачів не повинен перевищувати 1 годину завдяки інтуїтивному інтерфейсу та детальній документації.

Швидкодія системи забезпечується оптимізованою архітектурою:

- Час відповіді на запити конвертації - не більше 5 секунд
- Час завантаження інтерфейсу - не більше 2 секунд
- Підтримка одночасної роботи до 100 користувачів

Доступність системи має становити 99.9% часу, за винятком планових технічних робіт. Система повинна коректно обробляти помилки та надавати зрозумілі повідомлення користувачам.

Бізнес-логіка

Система надає безкоштовний доступ до базового функціоналу для всіх зареєстрованих користувачів, включаючи можливість створення UML діаграм та їх конвертації у код. Історія конвертацій зберігається для кожного користувача, дозволяючи відстежувати зміни. Адміністративний функціонал доступний лише для користувачів з роллю ROLE_ADMIN та включає можливості управління користувачами, моніторингу системи та налаштування параметрів конвертації для різних мов програмування.

Архітектура бази даних побудована з урахуванням специфіки системи конвертації UML та включає сім основних таблиць, які забезпечують повний функціонал системи.

Таблиця *users* є центральною сутністю системи та зберігає повну інформацію про зареєстрованих користувачів. Окрім стандартних полів ідентифікації, вона містить поля для імені та прізвища користувача, статус верифікації електронної пошти, час останнього входу в систему та статус підписки. Електронна пошта виступає як основний засіб ідентифікації з обмеженням унікальності, а система ролей у JSON форматі дозволяє гнучко розділяти права доступу між звичайними користувачами та адміністраторами.

Таблиця *oauth_connections* забезпечує інтеграцію з зовнішніми OAuth провайдерами, такими як Google та GitHub. Вона зберігає інформацію про провайдера, унікальний ідентифікатор користувача у зовнішній системі, токени доступу та оновлення, час закінчення дії токенів. Зовнішній ключ з каскадним видаленням забезпечує цілісність даних при видаленні користувача.

Для безпечного відновлення паролів використовується таблиця *reset_password_request*, яка зберігає тимчасові селектори та хешовані токени. Кожен запис має обмежений час дії, що унеможливорює зловживання системою відновлення паролів.

Таблиця *action_history* є критично важливою для функціонування системи, оскільки зберігає детальну історію всіх операцій конвертації. Вона фіксує тип операції, вхідні UML дані, тип діаграми, згенеровані файли у JSON форматі, обрану мову програмування, версію генератора, загальну кількість рядків коду, розмір діаграми та час виконання. Ця інформація дозволяє користувачам переглядати історію своїх конвертацій та завантажувати попередні результати.

Системні події та помилки логуються у таблиці *system_logs*, яка містить рівень логування, канал, повідомлення, контекстну інформацію у JSON форматі, додаткові дані, IP-адресу користувача, інформацію про браузер та URL запиту. Ця таблиця забезпечує повний аудит системи та діагностику проблем.

Таблиця *user_statistics* відстежує детальну статистику активності кожного користувача, включаючи загальну кількість операцій парсингу, конвертації та генерації коду, кількість згенерованих файлів, рядків коду та статистику по мовах програмування у JSON форматі. Ці дані використовуються для аналітики та оптимізації роботи системи.

Технічна таблиця *doctrine_migration_versions* управляє версіонуванням структури бази даних та відстежує виконані міграції Doctrine ORM, що забезпечує коректне оновлення схеми бази даних при розгортанні нових версій системи.

2.3. Проектування та реалізація алгоритмів роботи системи

Основою функціонування системи UML2Code є складний комплекс алгоритмів, що забезпечують повний цикл конвертації від PlantUML діаграм класів до готового програмного коду. Розробка цих алгоритмів базувалася на принципах чистої архітектури з чітким розподілом відповідальності між компонентами та забезпеченням максимальної точності перетворення UML-нотацій у синтаксично коректний код.

Архітектура алгоритмів конвертації побудована навколо трьох основних етапів обробки даних: лексичного аналізу та парсингу PlantUML тексту, побудови внутрішнього представлення діаграми класів у вигляді доменної моделі та генерації цільового коду з урахуванням специфіки обраної мови програмування. Кожен етап реалізований як незалежний модуль з власними алгоритмами та може функціонувати окремо, що забезпечує гнучкість системи та можливість легкого розширення функціоналу.

Алгоритм парсингу PlantUML діаграм

Процес парсингу PlantUML діаграм реалізовано через багатоетапний алгоритм, що включає попередню обробку тексту, розпізнавання синтаксичних конструкцій та побудову структурованого представлення. Основний парсер PlantUmlParser(додаток А) використовує регулярні вирази для ідентифікації

ключових елементів діаграми: класів, інтерфейсів, абстрактних класів, енумерацій та зв'язків між ними.

Алгоритм парсингу працює за принципом послідовного аналізу рядків вхідного тексту з накопиченням інформації про структуру діаграми. Спочатку відбувається нормалізація тексту з видаленням коментарів та порожніх рядків. Далі кожен рядок аналізується на предмет відповідності одному з шаблонів: оголошення класу, визначення атрибуту, оголошення методу або встановлення зв'язку між класами.

Таблиця 2.1. - Регулярні вирази для розпізнавання PlantUML конструкцій

Конструкція	Регулярний вираз	Приклад
Клас	<code>class\s+(\w+)(?:\s*<([^\>]+)>)?</code>	<code>class User<T></code>
Атрибут	<code>([+-]?)\s*(\w+)\s*:\s*(.+)</code>	<code>+name: String</code>
Метод	<code>([+-]?)\s*(\w+)\s*\(((\^[^\)]*)\)\s*:\s*(.+)</code>	<code>+getName(): String</code>
Зв'язок	<code>(\w+)\s*(--> --\ > \. . >)\s*(\w+)</code>	<code>User --> Order</code>

Особливу увагу в алгоритмі парсингу приділено обробці генеричних типів, які можуть мати складну вкладену структуру типу `Map<String, List<User>>`. Для цього розроблено спеціальний алгоритм рекурсивного парсингу типів (додаток Б), що коректно обробляє дужки та кома-розділені параметри типів. Результатом роботи парсера є доменна модель `ClassDiagram`, що містить колекції об'єктів `ClassElement`, `Attribute`, `Method` та `Relationship`. Ця модель служить проміжним представленням, незалежним від синтаксису конкретної мови програмування.

Алгоритм генерації програмного коду

Система генерації коду реалізована через патерн `Strategy` з використанням абстрактного класу `AbstractLanguageCodeGenerator`, що визначає загальний інтерфейс для всіх мовно-специфічних генераторів. Кожна підтримувана мова програмування має власну реалізацію генератора з урахуванням її синтаксичних особливостей та конвенцій найменування.

Алгоритм генерації коду працює у декілька етапів. Спочатку відбувається аналіз доменної моделі для визначення залежностей між класами та необхідних імпортів. Далі для кожного класу генерується окремий файл з дотриманням

структури: імпорти, оголошення класу, атрибути, конструктор, методи доступу та бізнес-методи.



Рис.2.6. Схема алгоритму генерації коду

Генератор PHP враховує особливості мови, включаючи типізацію параметрів, nullable типи та namespace структуру. Для PHP 8.0+ використовуються constructor property promotion та union types, тоді як для PHP 7.4 генерується більш вербозний код з явним оголошенням властивостей.

Генератор Java забезпечує коректну обробку generic типів, автоматичне додавання необхідних імпортів та генерацію стандартних методів toString(),

`equals()` та `hashCode()`. Особливу увагу приділено правильному мапінгу типів PlantUML на Java типи з урахуванням примітивних та об'єктних типів.

Python генератор створює код з використанням `type hints` згідно з PEP 484 та `dataclasses` для простих моделей даних. Алгоритм автоматично додає необхідні імпорти `typing` модуля та генерує `docstrings` у Google Style.

Таблиця 2.2. – Мапінг типів PlantUML на мови програмування

PlantUML тип	Java	PHP	Python	C#
String	String	string	str	string
int	int	int	int	int
List<T>	List<T>	array	List[T]	List<T>
Map<K,V>	Map<K,V>	array	Dict[K,V]	Dictionary<K,V>

Система прямої конвертації та оптимізація

Система прямої конвертації об'єднує алгоритми парсингу та генерації в єдиний `workflow` через `ConverterService`. Цей сервіс координує весь процес трансформації, включаючи валідацію вхідних даних, очистку проміжного представлення та оптимізацію згенерованого коду.

Алгоритм очистки діаграми `cleanDiagramArray()` виконує постобробку парсингу для усунення дублікатів класів, виправлення некоректних зв'язків та нормалізації імен методів і атрибутів. Цей етап критично важливий для забезпечення коректності генерованого коду.

Оптимізація генерованого коду включає мінімізацію кількості імпортів, групування пов'язаних методів та автоматичне додавання `utility` методів там, де це доцільно. Алгоритм аналізує структуру класів та автоматично визначає, чи потрібно генерувати конструктори, геттери, сеттери та методи порівняння.

Обробка помилок та валідація

Кожен етап алгоритму конвертації включає комплексну систему валідації та обробки помилок. Парсер перевіряє синтаксичну коректність PlantUML коду та генерує детальні повідомлення про помилки з указанням рядка та характеру проблеми.

Генератори коду виконують семантичну валідацію доменної моделі, перевіряючи існування класів для зв'язків, коректність типів атрибутів

та методів, відсутність циклічних залежностей. У разі виявлення проблем система генерує попередження але продовжує роботу, створюючи максимально коректний код.

Метрики продуктивності та тестування

Алгоритми конвертації оптимізовано для роботи з діаграмами середнього розміру до 50 класів. Час парсингу типової діаграми з 10-15 класами не перевищує 100 мілісекунд, а генерація коду для всіх підтримуваних мов займає менше 500 мілісекунд.

Система включає комплексні unit та integration тести, що покривають різні сценарії використання від простих класів до складних ієрархій з generic типами та множинними зв'язками. Тести забезпечують покриття коду понад 85% та автоматично запускаються при кожному оновленні кодової бази.

Результатом реалізації описаних алгоритмів є надійна система конвертації, що забезпечує високу точність перетворення UML діаграм у синтаксично коректний та структурно організований програмний код для чотирьох основних мов програмування з можливістю легкого розширення для додавання підтримки нових мов.

2.4. Реалізація системи конвертації UML діаграм у програмний код

Технічна реалізація системи UML2Code базується на модульній архітектурі з чітким розподілом відповідальності між компонентами. Основою системи є три ключові модулі Core, розташовані у директорії src/Core: Parser для аналізу PlantUML діаграм, Generator для генерації програмного коду та Converter для координації повного циклу конвертації.

Модуль Parser реалізований через сервіс UmlParserService, який служить точкою входу для всіх операцій парсингу. Цей сервіс делегує безпосередню обробку PlantUML тексту класу PlantUmlParser, що містить складну логіку розбору синтаксичних конструкцій. Парсер використовує регулярні вирази для ідентифікації класів, методів, атрибутів та зв'язків, створюючи внутрішнє

представлення у вигляді доменних об'єктів `ClassDiagram`, `ClassElement`, `Attribute` та `Relationship`.

Особливу увагу приділено обробці складних типів даних та генеричних конструкцій. Клас `Type` забезпечує розпізнавання та коректне оброблення вкладених генеричних типів типу `Map<String, List<User>>`, що критично важливо для генерації синтаксично коректного коду у сучасних мовах програмування.

Система генерації коду побудована на патерні `Strategy` через абстрактний клас `AbstractLanguageCodeGenerator` (додаток Г), що дозволяє легко додавати підтримку нових мов програмування. Кожна мова має власну ієрархію генераторів з урахуванням версійних особливостей. Наприклад, для PHP реалізовано окремі генератори `Php74CodeGenerator` та `Php80CodeGenerator`, що враховують нововведення PHP 8.0 такі як `constructor property promotion` та `union types`.

Генератор Java включає розширену підтримку `generic` типів та автоматичну генерацію стандартних методів `equals()`, `hashCode()` та `toString()`. Python генератор створює код з використанням `type hints` згідно з PEP 484 та автоматично додає необхідні імпорти з `typing` модуля. Веб-інтерфейс системи реалізовано через контролери, що обслуговують як браузерні запити через `WebController`, так і API викликів через `ApiController`. Система підтримує повну інтернаціоналізацію з використанням `Symfony Translation Component` та `YAML` файлів перекладів для української та англійської мов.

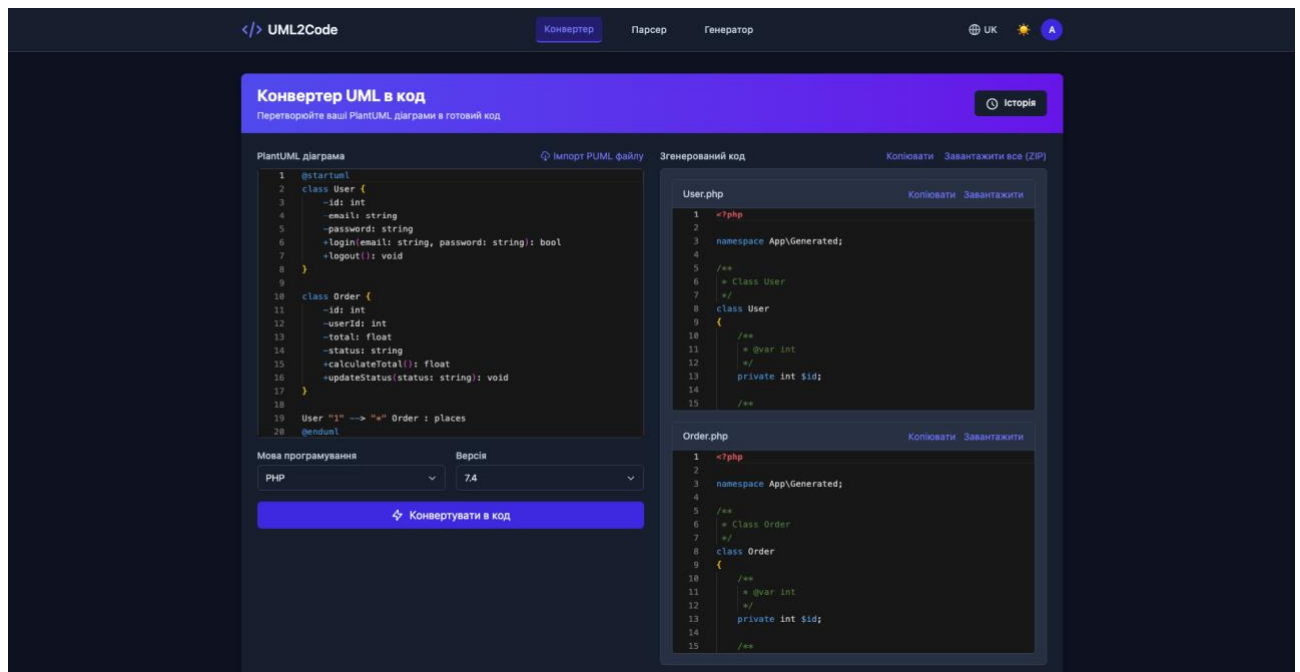


Рис.2.7. Скріншот головного інтерфейсу системи

Центральним елементом користувацького інтерфейсу є інтеграція з Monaco Editor, що забезпечує професійне середовище для редагування UML діаграм. Редактор налаштовано з підсвічуванням синтаксису PlantUML, автодоповненням та валідацією коду в реальному часі. Система тем автоматично синхронізується між веб-інтерфейсом та Monaco Editor, дозволяючи користувачам переключатися між світлою та темною темами.

Архітектура RESTful API забезпечує програмний доступ до всіх функцій системи. Ключові endpoints включають `/api/parser/parse` для парсингу UML діаграм, `/api/generator/generate` для генерації коду та `/api/converter/convert` для прямої конвертації. Всі API endpoints підтримують аутентифікацію через систему ролей Symfony Security Component. Система аутентифікації реалізована через багат шарову архітектуру з підтримкою як традиційної форми входу, так і OAuth інтеграції з Google та GitHub. Клас OAuthAuthenticator забезпечує безпечну інтеграцію з зовнішніми провайдерами, а EmailVerifier координує процес верифікації електронної пошти нових користувачів.

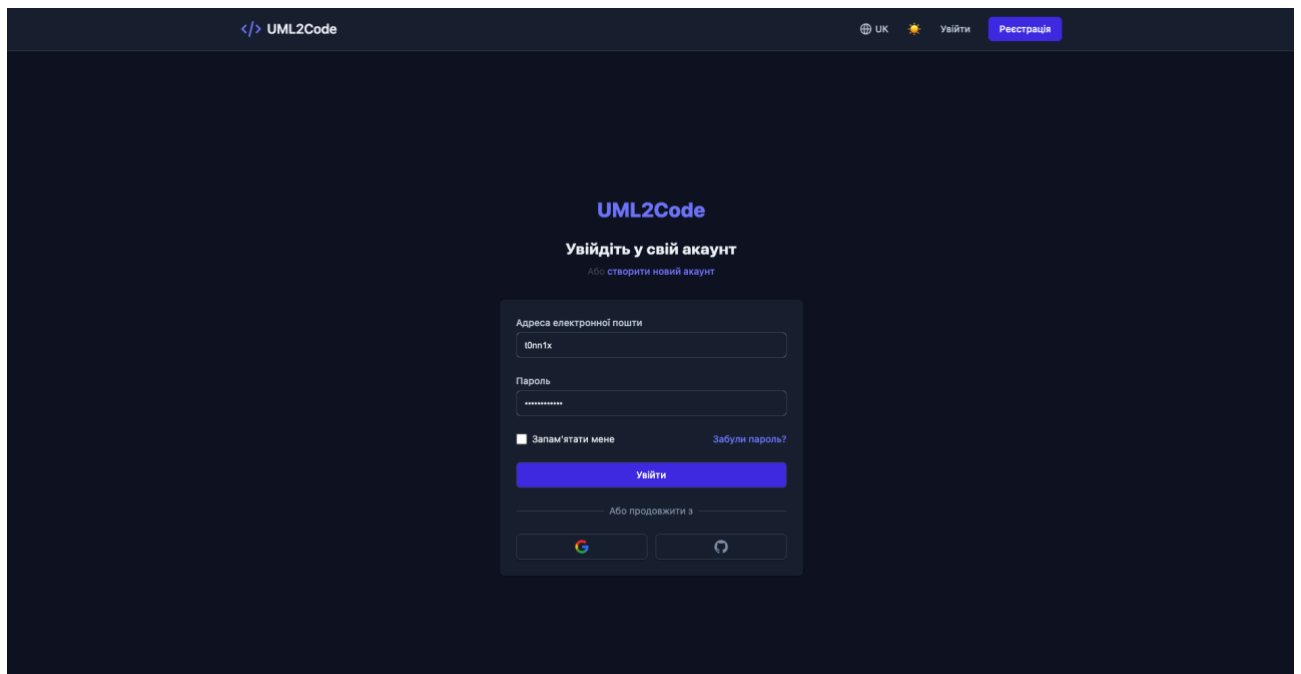


Рис.2.8. Скріншот сторінки входу та реєстрації

Адміністративна панель побудована на EasyAdmin Bundle та надає повний контроль над системою. Адміністратори можуть управляти користувачами через UserCrudController, переглядати історію дій через ActionHistoryCrudController та аналізувати системні логи через SystemLogCrudController. Кожен контролер має власні фільтри та можливості пошуку для ефективного управління даними.

Система логування реалізована через Monolog з автоматичним записом всіх критичних операцій у базу даних PostgreSQL. Логи включають детальну інформацію про помилки, час виконання операцій та контекстну інформацію для діагностики проблем. Реалізовано автоматичне очищення старих логів через cron завдання для оптимізації використання дискового простору.

Кешування результатів конвертації реалізовано на рівні сервісів з використанням Symfony Cache Component. Часто використовувані шаблони коду та результати парсингу зберігаються у кеші для зменшення часу відповіді системи. Invalidation кешу відбувається автоматично при зміні налаштувань генераторів коду.

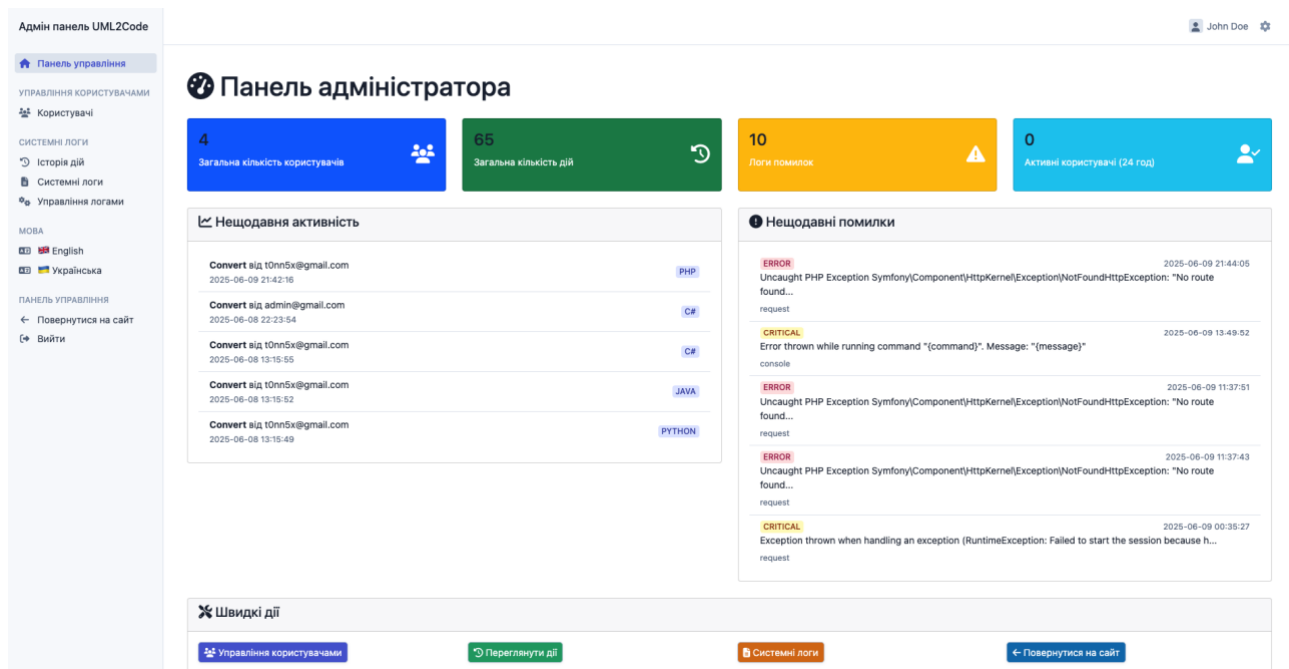


Рис.2.9. Скріншот головної сторінки панелі адміністратора

Тестування системи охоплює всі критичні компоненти через комбінацію unit, integration та functional тестів. Тести парсера перевіряють коректність обробки різних UML конструкцій, тести генераторів валідують синтаксичну коректність згенерованого коду, а API тести забезпечують стабільність веб-інтерфейсів.

Розгортання системи автоматизовано через Docker Compose з окремими контейнерами для Nginx, PHP-FPM та PostgreSQL.

Моніторинг продуктивності реалізовано через збір метрик часу виконання операцій, використання пам'яті та кількості активних користувачів. Дані зберігаються у таблиці user_statistics та використовуються для оптимізації алгоритмів конвертації і планування ресурсів системи.

Результатом технічної реалізації є стабільна, масштабована система, що забезпечує високу якість конвертації UML діаграм у програмний код з підтримкою сучасних веб-стандартів, інтернаціоналізації та комплексної системи безпеки.

Висновки до другого розділу

У результаті проведеного аналізу та проектування було створено повноцінну архітектуру системи конвертації UML діаграм у програмний код, що

забезпечує надійне та ефективне перетворення PlantUML нотацій у синтаксично коректний код чотирьох основних мов програмування.

Розроблена система варіантів використання чітко визначила функціональні вимоги для двох основних типів користувачів: звичайних користувачів з правами конвертації та адміністраторів з повним доступом до управління системою. Виділено ключові сценарії роботи, включаючи створення та редагування UML діаграм, їх валідацію, парсинг у структуроване представлення, генерацію коду різними мовами та збереження історії конвертацій. Нефункціональні вимоги забезпечують високу продуктивність системи з часом відповіді не більше 5 секунд для операцій конвертації та підтримкою до 100 одночасних користувачів.

Архітектура бази даних на основі PostgreSQL оптимізована для зберігання всіх необхідних даних системи через сім взаємопов'язаних таблиць. Центральна таблиця users забезпечує управління користувачами з підтримкою ролей та OAuth автентифікації. Таблиця action_history зберігає детальну інформацію про всі операції конвертації, включаючи вхідні UML дані, згенеровані файли та метрики продуктивності. Система логування через system_logs надає повний аудит роботи системи, а user_statistics відстежує статистику активності для оптимізації продуктивності.

Розроблені алгоритми конвертації реалізують повний цикл обробки від PlantUML тексту до готового програмного коду через трьохетапний процес. Алгоритм парсингу використовує регулярні вирази для розпізнавання синтаксичних конструкцій та побудови внутрішнього представлення діаграми класів. Система генерації коду реалізована через патерн Strategy з окремими генераторами для кожної мови, що забезпечує коректне мапінг типів даних та дотримання мовних конвенцій. Алгоритм прямої конвертації координує весь процес з валідацією, очисткою даних та обробкою помилок.

Технічна реалізація системи базується на модульній архітектурі Symfony з чітким розподілом на три основні модулі: Parser, Generator та Converter. Веб-інтерфейс інтегрує Monaco Editor для професійного редагування UML діаграм

з підсвічуванням синтаксису та підтримкою тем оформлення. RESTful API забезпечує програмний доступ до всіх функцій системи з автентифікацією та детальним логуванням операцій. Адміністративна панель на базі EasyAdmin Bundle надає повний контроль над користувачами, історією дій та системними логами.

Результатом другого розділу є повністю спроектована та реалізована система, що забезпечує високоякісну конвертацію UML діаграм класів у програмний код з підтримкою сучасних веб-стандартів, інтернаціоналізації та масштабованої архітектури для подальшого розвитку функціоналу.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ПЛАТФОРМОЮ ДЛЯ ГЕНЕРАЦІЇ КОДУ НА ОСНОВІ СИНТАКСИСУ PLANTUML

3.1. Порядок встановлення та налаштування параметрів системи

Мінімальні вимоги

Вимоги до апаратного забезпечення для стабільної роботи системи UML2Code включають сервер з наступними характеристиками:

Оперативна пам'ять обсягом 4ГБ для забезпечення одночасної обробки множинних запитів конвертації UML діаграм, процесор з частотою не менше 2.0 ГГц для ефективного парсингу та генерації коду, дисковий простір 20ГБ для зберігання системних файлів та логів операцій. Операційна система має бути Linux дистрибутив Ubuntu 20.04 або вище, що забезпечує сумісність з усіма використовуваними технологіями та контейнеризацією.

Вимоги до системи управління базами даних передбачають наявність PostgreSQL версії 13 або вище як основної реляційної СУБД. PostgreSQL обрано через її надійність у роботі з JSON структурами для зберігання історії конвертацій та розширені можливості для індексації даних, що критично важливо для швидкого пошуку у великих обсягах логів операцій.

Додаткові програмні вимоги включають встановлену платформу контейнеризації Docker версії 20.10 або вище та інструмент Docker Compose версії 2.0 для оркестрації багатоконтейнерних додатків. Система також потребує доступу до мережі Інтернет для завантаження залежностей та інтеграції з OAuth провайдерами Google та GitHub.

Порядок встановлення:

Перший етап передбачає підготовку серверного середовища через встановлення платформи контейнеризації Docker та інструмента Docker Compose. Встановлення здійснюється через офіційні репозиторії Ubuntu з подальшою перевіркою коректності інсталяції командами `docker --version` та `docker-compose --version`.

Другий етап включає клонування проекту з Git репозиторію за допомогою команди `git clone` та перенесення до кореневої директорії проекту файлу з `environment` конфігурацією `.env`. У цьому файлі зосереджені критично важливі параметри системи: ключі для SMTP сервера електронної пошти, OAuth ключі для інтеграції з Google та GitHub, секретний ключ додатку `APP_SECRET` для криптографічних операцій, параметри підключення до бази даних PostgreSQL та налаштування системи логування. Конфігураційний файл `.env` містить наступні основні параметри: `DATABASE_URL` для підключення до PostgreSQL, `MAILER_DSN` для налаштування відправки електронних листів, `OAUTH_GOOGLE_CLIENT_ID` та `OAUTH_GOOGLE_CLIENT_SECRET` для інтеграції з Google OAuth, `OAUTH_GITHUB_CLIENT_ID` та `OAUTH_GITHUB_CLIENT_SECRET` для автентифікації через GitHub.

Третій етап передбачає ініціалізацію системи через виконання команди `docker-compose up -d`, що запускає всі необхідні контейнери у фоновому режимі. Система автоматично створює чотири основні контейнери: `nginx` для веб-сервера, `php` для обробки PHP коду, `postgres` для бази даних та `redis` для кешування сесій користувачів.

Четвертий етап включає виконання міграцій бази даних командою `docker-compose exec php bin/console doctrine:migrations:migrate` для створення необхідних таблиць та початкових даних. Система автоматично створює структуру бази даних згідно з розробленою схемою, включаючи таблиці користувачів, історії дій, системних логів та статистики.

П'ятий етап передбачає перевірку коректності розгортання через команду `docker-compose ps` для відображення статусу всіх контейнерів та `docker-compose logs` для аналізу логів запуску. Всі контейнери мають мати статус "Up" для підтвердження успішного розгортання системи.

Заключний етап включає створення адміністративного користувача через команду `docker-compose exec php bin/console app:create-admin` для забезпечення доступу до адміністративної панелі системи. Адміністратор отримує повні права

для управління користувачами, перегляду логів та налаштування параметрів конвертації.

Після завершення встановлення система буде доступна за адресою <http://localhost:8080> для веб-інтерфейсу та <http://localhost:8080/api> для програмного доступу через RESTful API. Адміністративна панель розташована за адресою <http://localhost:8080/uk/admin> та потребує автентифікації з правами ROLE_ADMIN для доступу до функцій управління системою.

3.2. Структура інтерфейсу веб-додатку

Архітектура користувацького інтерфейсу системи UML2Code побудована за принципом поступового надання доступу до функціоналу залежно від рівня автентифікації користувача. Інтерфейс реалізовано з використанням responsive дизайну та підтримує повну інтернаціоналізацію між українською та англійською мовами.

Сторінка автентифікації та реєстрації

Початковою точкою взаємодії з системою є сторінка входу, розташована за маршрутом `/locale/login`. Інтерфейс автентифікації реалізований через SecurityController та включає три основні способи входу в систему: традиційну форму з електронною поштою та паролем, OAuth автентифікацію через Google та OAuth автентифікацію через GitHub.

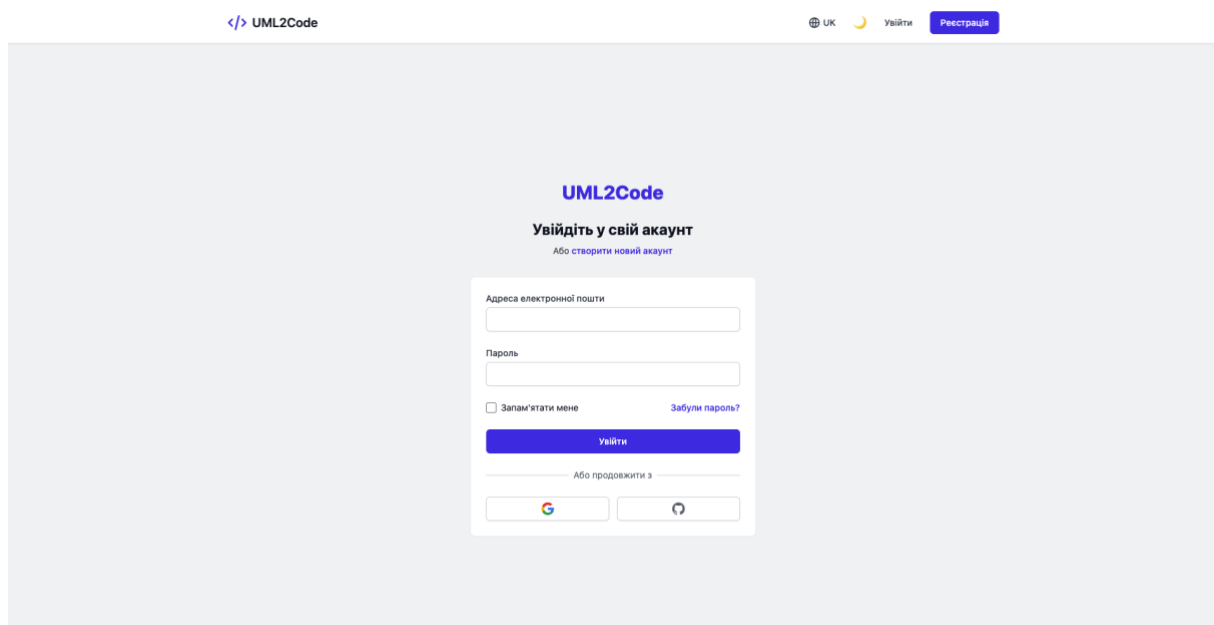


Рис.3.1. Сторінка входу

Форма входу містить поля для введення електронної пошти та пароля з клієнтською валідацією JavaScript та серверною валідацією через Symfony Validator. Під формою розташовані кнопки для входу через зовнішні провайдери OAuth з відповідними іконками та кольоровою схемою брендів. Внизу сторінки знаходяться посилання на реєстрацію нового користувача та відновлення забутого пароля.

Сторінка реєстрації за маршрутом `/locale/register` включає розширену форму з полями для імені, прізвища, електронної пошти, пароля та підтвердження пароля. Система автоматично перевіряє унікальність електронної пошти та силу пароля з відображенням рекомендацій у реальному часі. Після успішної реєстрації користувач отримує лист підтвердження на зазначену електронну адресу.

Рис.3.2. Сторінка реєстрації

Процес відновлення пароля реалізований через окрему сторінку `/locale/reset-password` з поетапним workflow: введення електронної пошти, отримання листа з посиланням, встановлення нового пароля через захищену форму з токеном.

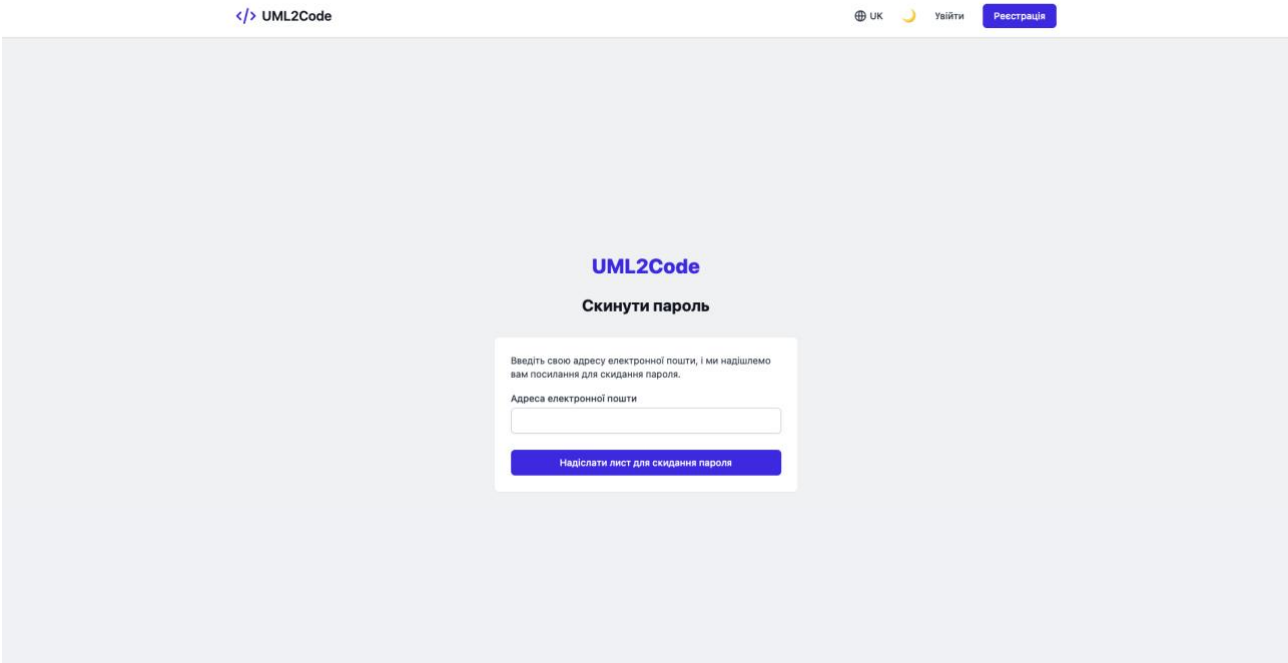


Рис.3.3. Сторінка скидання пароля

Головна сторінка системи

Після успішної автентифікації користувач потрапляє на головну сторінку за маршрутом `/locale/dashboard`, що служить панеллю керування системою. Верхня частина містить навігаційну панель з логотипом системи, меню переключення мови (UA/EN), селектор теми оформлення (світла/темна) та інформацію про поточного користувача разом зі статистикою його виконаних дій, діаграмою використаних мов, графіком активності та історією.

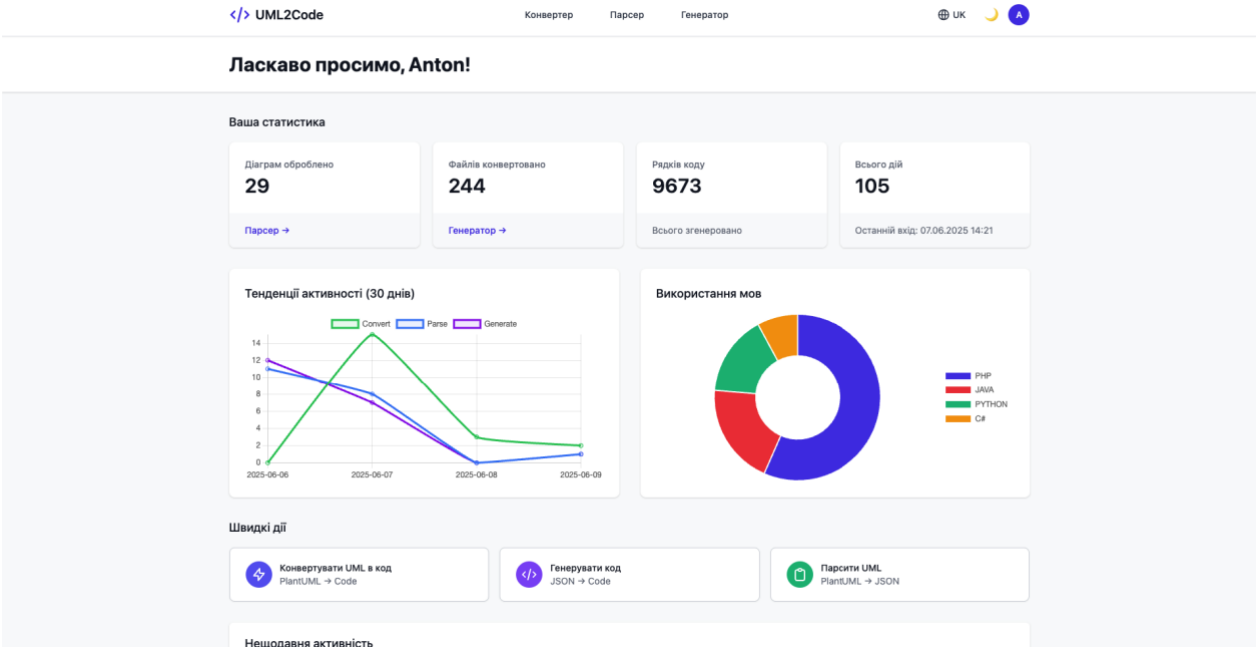


Рис.3.4. Панель управління

Центральна частина головної сторінки представляє три основні функціональні блоки системи у вигляді карток з описами та кнопками переходу: UML Parser для конвертації PlantUML діаграм у JSON структуру, Code Generator для генерації програмного коду з JSON представлення та Direct Converter для прямої конвертації UML у код без проміжних кроків.

Інтерфейс UML Parser

Сторінка парсера за маршрутом `/locale/parser` призначена для конвертації PlantUML діаграм у структурований JSON формат. Інтерфейс розділено на дві основні панелі: ліва панель для введення PlantUML коду та права панель для відображення результату парсингу.

Ліва панель містить Monaco Editor з налаштованим підсвічуванням синтаксису PlantUML, автодоповненням ключових слів та валідацією коду в реальному часі. Над редактором розташована панель інструментів з кнопками для завантаження файлу, збереження поточного коду, очищення редактора та швидкої вставки шаблонів класів.

Права панель відображає результат парсингу у форматованому JSON вигляді з підсвічуванням синтаксису та можливістю скопіювати результат.

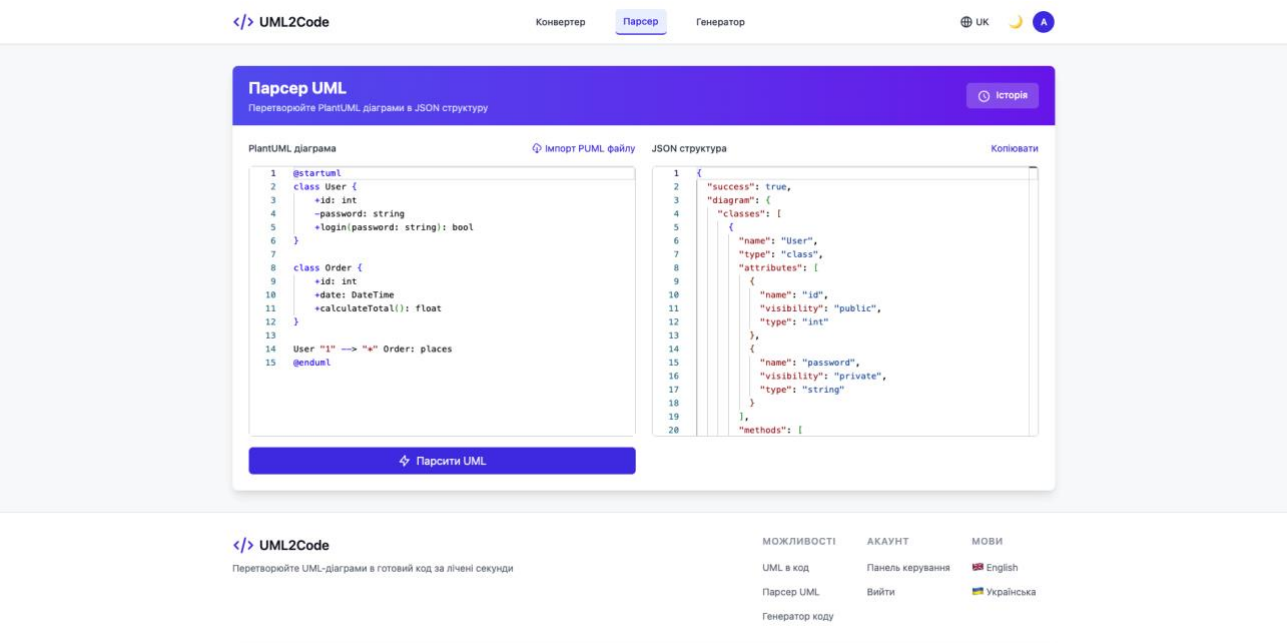


Рис.3.5. Інтерфейс парсера

Інтерфейс Code Generator

Сторінка генератора коду за маршрутом `/locale/generator` дозволяє створювати програмний код з JSON представлення UML діаграми. Інтерфейс організовано у три основні секції: налаштування генерації, попередній перегляд та результати генерації.

Секція налаштувань містить випадаючий список для вибору мови програмування (Java, C#, PHP, Python) та селектор версії мови. Над налаштуваннями знаходиться текстове поле для введення або завантаження JSON структури діаграми.

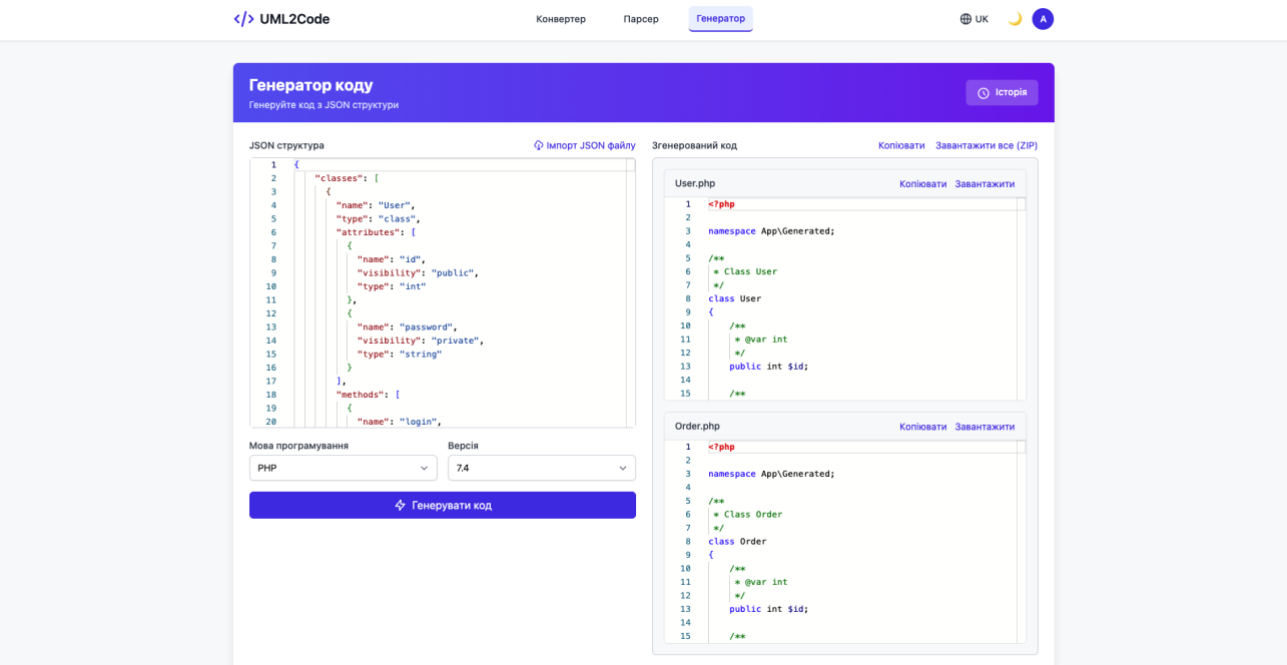


Рис.3.6. Інтерфейс генератора

Інтерфейс Direct Converter

Сторінка прямої конвертації за маршрутом `/locale/converter` об'єднує функціонал парсингу та генерації коду в єдиному інтерфейсі. Дизайн представляє собою двохпанельний layout з PlantUML редактором зліва, налаштуваннями під ним та результатами справа.

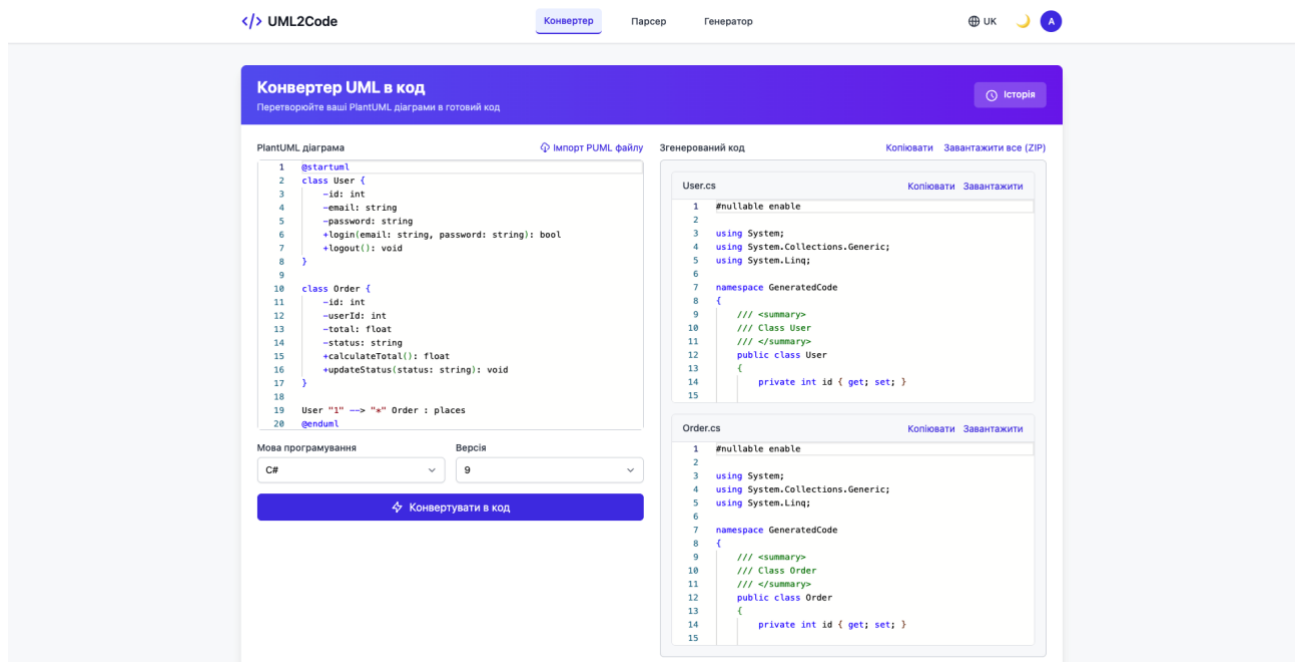


Рис.3.7. Інтерфейс конвертера

Інтерфейс модального вікна перегляду історії

На кожній зі сторінок для конвертації/парсингу/генерації наявна кнопка для перегляду історії. За замовчуванням вона відкриє історію того типу дії, на сторінці якого ми наразі знаходимося.

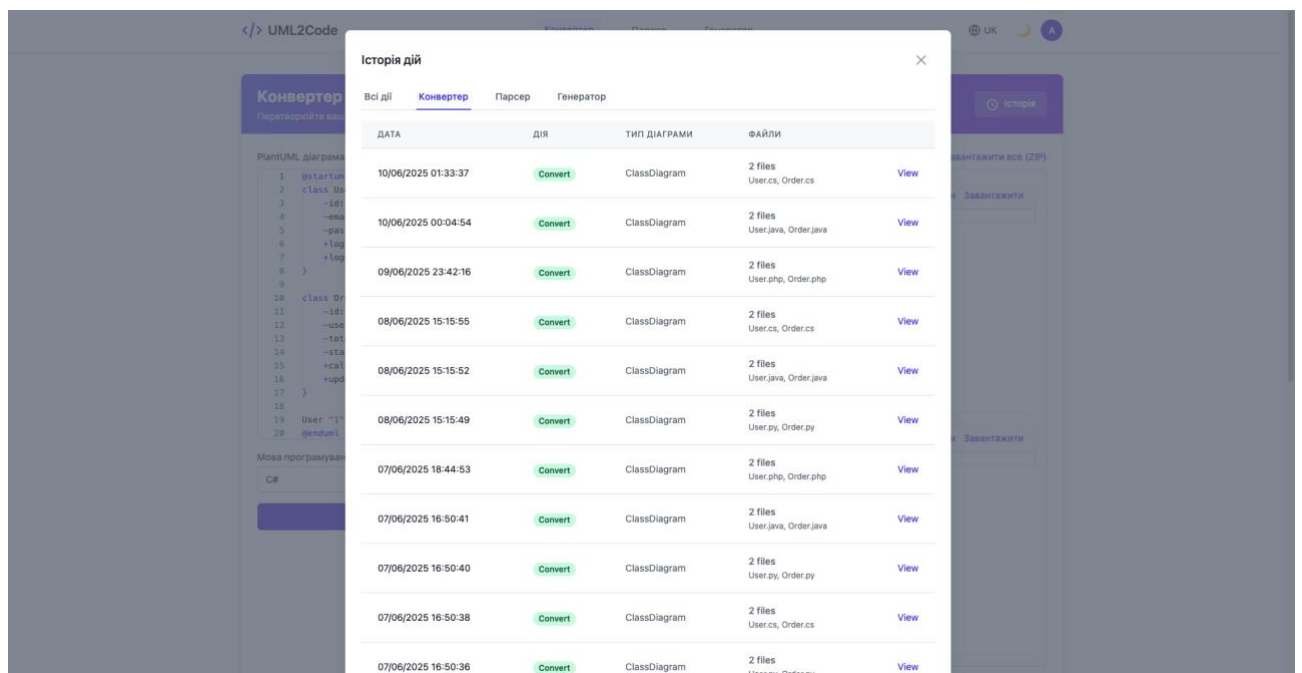


Рис.3.8. Сторінка перегляду історії конкретного інструменту

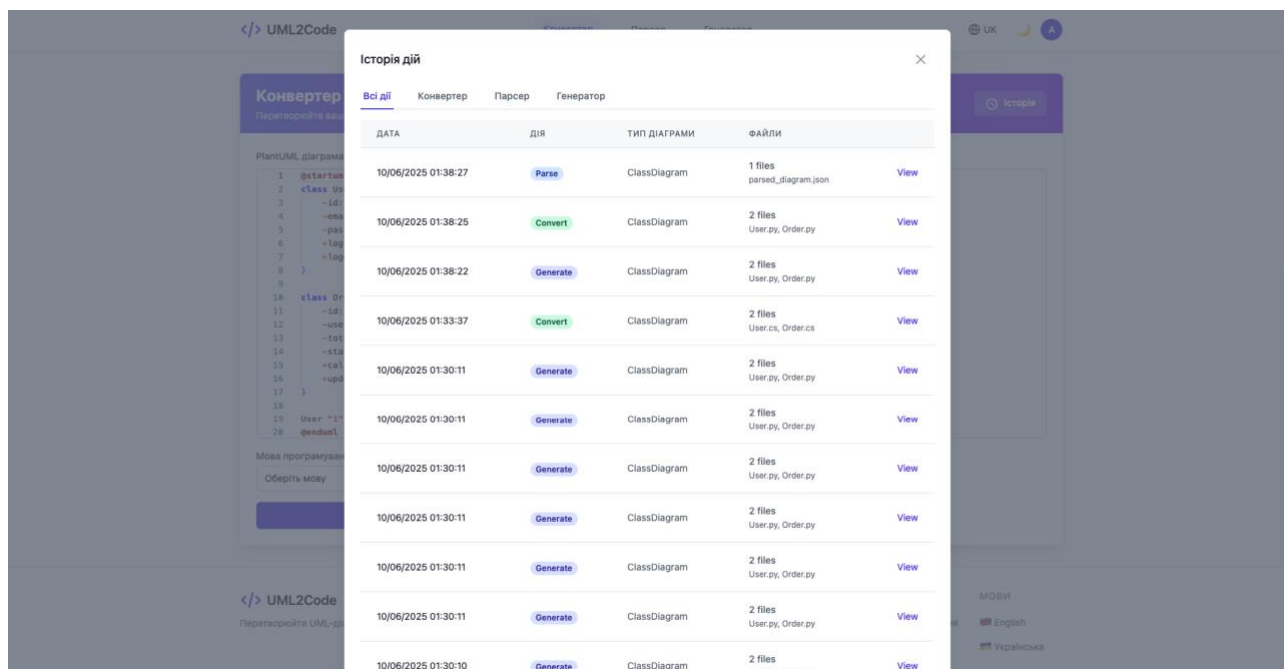


Рис.3.9. Сторінка перегляду усіх останніх дій

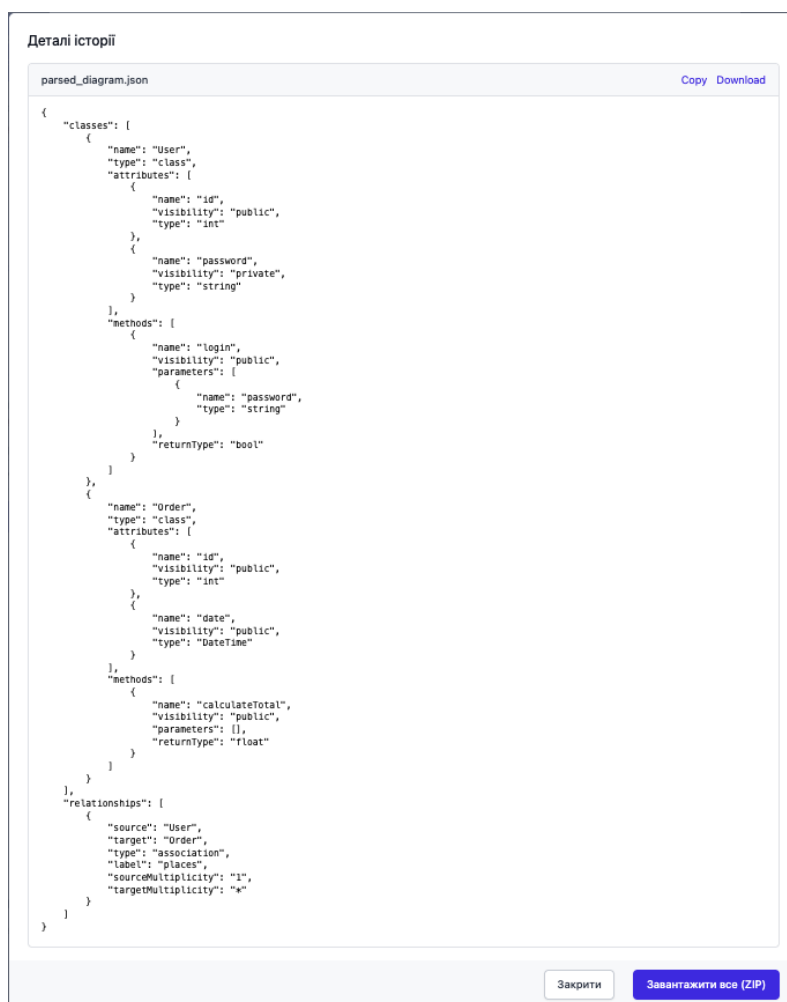


Рис.3.10. Сторінка перегляду історії конкретної дії

Адміністративна панель

Адміністративний інтерфейс за маршрутом `/locale/admin` доступний лише користувачам з роллю `ROLE_ADMIN` та реалізований через `EasyAdmin Bundle`. Головна сторінка адмін панелі містить статистичні віджети з ключовими метриками системи: загальна кількість користувачів, кількість конвертацій за різні періоди, статистика помилок та використання ресурсів. Ліва навігаційна панель включає секції для управління користувачами, перегляду історії дій, аналізу системних логів та налаштування параметрів системи. Кожна секція має власні фільтри, пошук та можливості експорту даних.

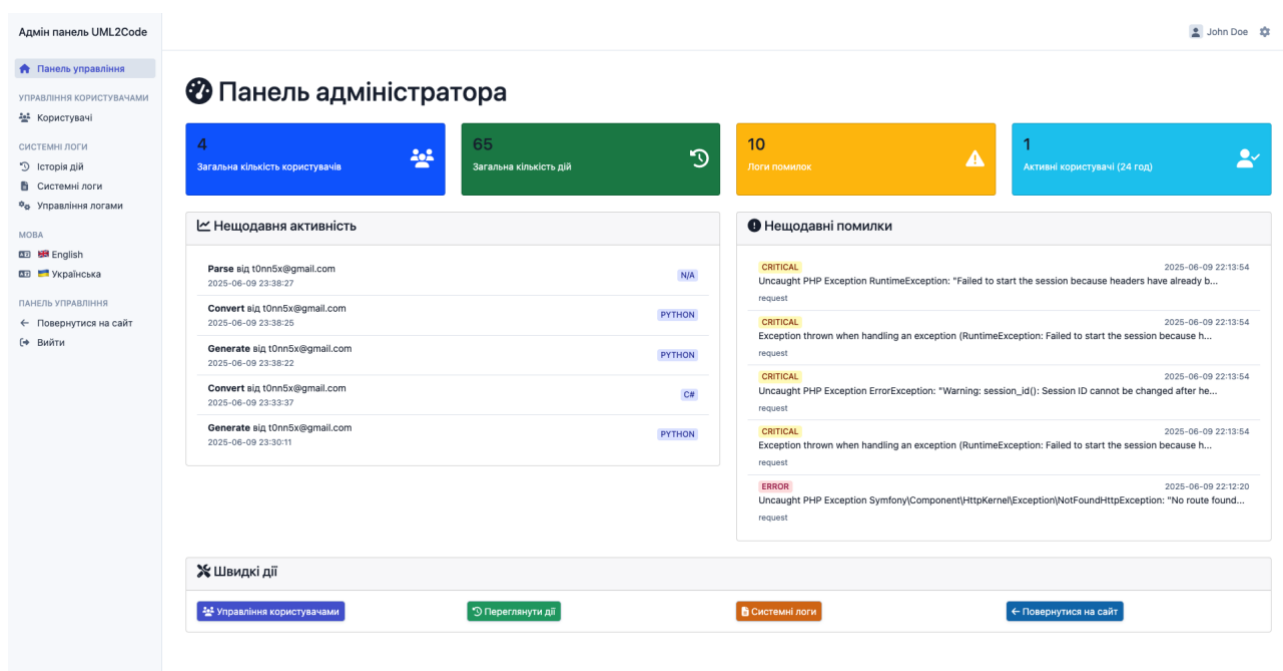


Рис.3.11. Інтерфейс адмін-панелі

Сторінка управління користувачами дозволяє переглядати, редагувати та видаляти облікові записи з детальною інформацією про активність кожного користувача. Інтерфейс включає розширені фільтри по ролях, датах реєстрації та статусу активності.

Системні логи дозволяють переглядати усі логи, зафіксовані системою, їх типи, повідомлення, користувача, рівень, канал, тощо:

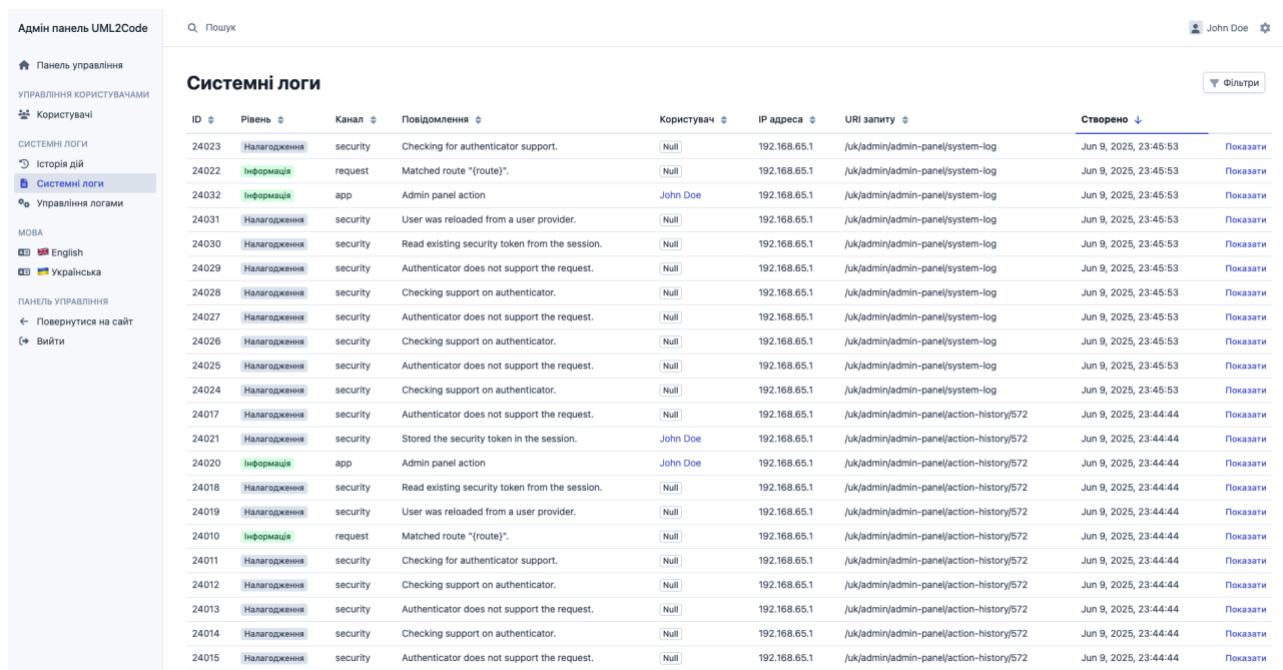


Рис.3.13. Інтерфейс перегляду системних логів адмін-панелі

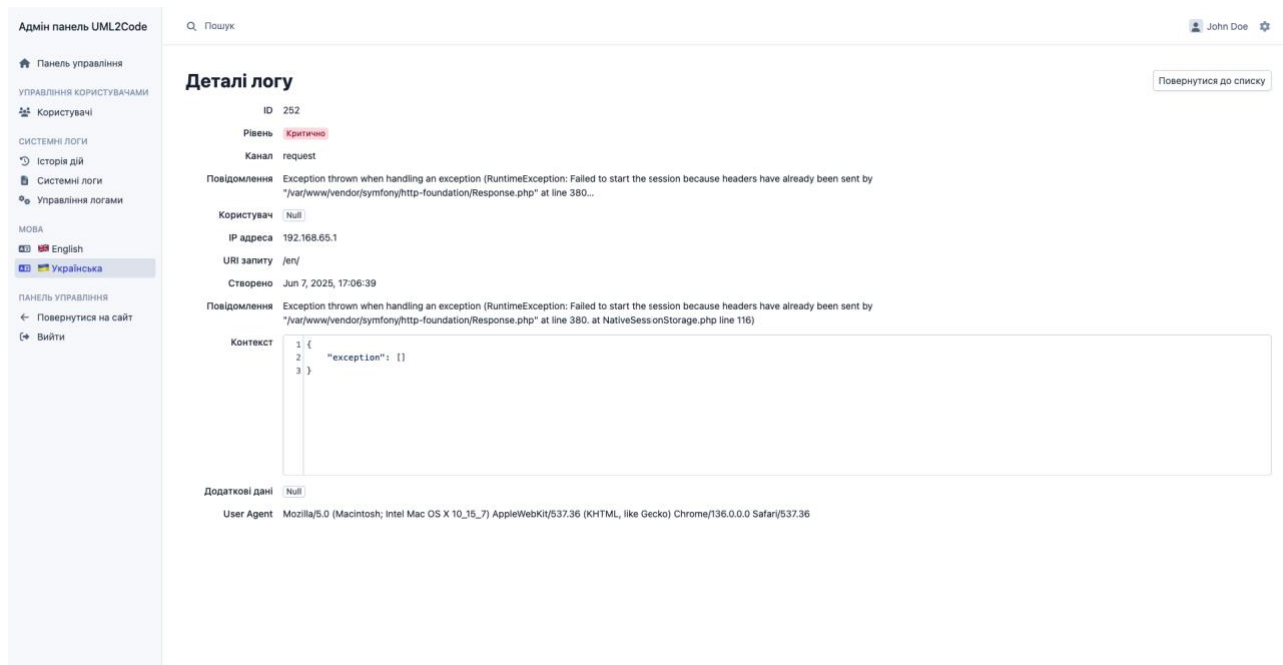


Рис.3.14. Інтерфейс перегляду конкретного логу

Управління системними логами включає інструменти для фільтрації по рівнях логування, пошуку по повідомленнях та автоматичного очищення старих записів. Система відображає статистику трендів помилок та

продуктивності для моніторингу стану системи.

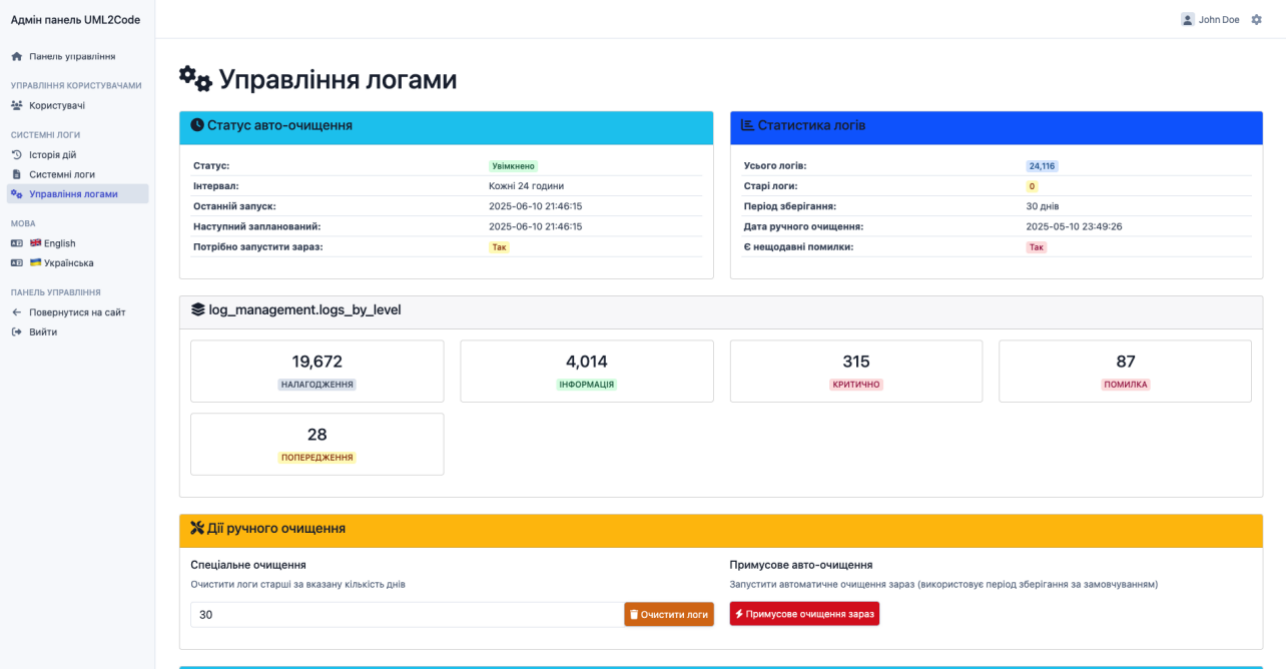


Рис.3.15. Інтерфейс сторінки управління логами

Висновки до третього розділу

В даному розділі було детально описано процес встановлення та налаштування системи UML2Code, а також визначено необхідні технічні вимоги та конфігураційні параметри для забезпечення стабільної роботи додатку. Встановлено мінімальні апаратні вимоги: 4ГБ оперативної пам'яті, процесор 2.0 ГГц та операційна система Ubuntu 20.04+, що забезпечують ефективну обробку UML діаграм та генерацію програмного коду.

Розкрито покрокову процедуру розгортання системи через контейнеризацію Docker, що включає налаштування PostgreSQL бази даних, конфігурацію веб-сервера Nginx, інтеграцію з OAuth провайдерами та ініціалізацію системи логування. Детально описано структуру конфігураційного файлу .env з критично важливими параметрами безпеки та інтеграції з зовнішніми сервісами.

Представлено комплексну архітектуру користувацького інтерфейсу з поетапним описом всіх ключових сторінок системи: від автентифікації користувачів до адміністративної панелі управління. Розкрито взаємодію між

компонентами через аналіз інтерфейсів парсингу UML діаграм, генерації програмного коду та прямої конвертації з використанням Monaco Editor для професійного редагування коду.

Висвітлено архітектуру програмного комплексу з підтримкою інтернаціоналізації, системи тем оформлення та розширеної функціональності адміністративної панелі на базі EasyAdmin Bundle.

Надано конкретні приклади навігації по системі та взаємодії з основними функціональними модулями, включаючи детальні інструкції щодо використання інтерфейсів парсингу, генерації коду та адміністрування системи. Розглянуто систему безпеки з багаторівневою автентифікацією через форми та OAuth провайдерів.

У даному розділі основну увагу приділено практичним аспектам експлуатації розробленого програмного продукту, що дозволяє забезпечити його успішне впровадження та ефективне використання кінцевими користувачами. Представлена архітектура інтерфейсу демонструє повноту функціональних можливостей системи та її готовність до промислової експлуатації.

ВИСНОВКИ

Під час виконання випускної роботи бакалавра було спроектовано та реалізовано веб-систему конвертації UML діаграм класів у програмний код. Для успішної реалізації даної задачі було необхідно провести ретельний аналіз існуючих рішень, а також розглянути різні методи та засоби автоматизації процесу генерації коду з UML нотацій.

В першому розділі аналіз предметної області дозволив виявити основні аспекти, які необхідно врахувати при розробці системи конвертації UML діаграм. Під час вивчення існуючих аналогів було виявлено що найважливішими характеристиками майбутньої системи є точність конвертації PlantUML нотацій, підтримка множинних мов програмування та зручність користувацького інтерфейсу.

Після вибору оптимальної архітектури на основі принципів чистої архітектури з використанням RESTful API, були обрані підходящі технології для її реалізації, а саме Symfony 7.2, Monaco Editor, PostgreSQL та Docker. В якості аналогів було розглянуто такі технології як Laravel, простий текстовий редактор, MySQL, Podman та LXD. Проте обрані технології є найбільш підходящими та сучасними для успішної реалізації завдань автоматизованої конвертації UML діаграм.

В другому розділі було детально спроектовано систему, що реалізує повний цикл конвертації від PlantUML діаграм класів до готового програмного коду чотирма мовами програмування: Java, C#, PHP та Python. Для оптимального функціонування додатку було розроблено детальну структуру бази даних PostgreSQL з можливістю зберігання історії конвертацій, користувацьких налаштувань та системної аналітики.

Детально вивчалися функціональні та нефункціональні вимоги до системи, завдяки чому було розроблено алгоритми парсингу PlantUML діаграм, генерації програмного коду та прямої конвертації з візуалізацією у вигляді UML діаграм та схем потоків даних. В останньому пункті другого розділу було розглянуто

аспекти технічної реалізації трьох ключових модулів системи: Parser, Generator та Converter.

В третьому розділі було детально описано процес встановлення, налаштування та розгортання системи за допомогою Docker Compose, а також були визначені мінімальні технічні вимоги до серверного обладнання. Розкрито структуру користувацького інтерфейсу від сторінки автентифікації до адміністративної панелі з детальним описом функціональних можливостей кожного компонента.

Додатково, викладено конкретні приклади використання системи та надано інструкції щодо роботи з інтерфейсами парсингу UML діаграм, генерації коду та прямої конвертації. У цьому розділі основна увага приділена практичним аспектам експлуатації розробленого програмного продукту з урахуванням вимог безпеки та масштабованості.

В результаті розробки було створено повнофункціональну систему конвертації UML діаграм класів у програмний код, яка повністю задовольняє поставлені вимоги щодо точності парсингу PlantUML нотацій, якості генерованого коду та зручності користувацького інтерфейсу.

Завдяки використанню контейнеризації Docker, проект з легкістю можна розвернути на будь-якій UNIX-подібній операційній системі з мінімальними вимогами до конфігурації. Модульна архітектура системи дозволяє легко додавати підтримку нових мов програмування та розширювати функціонал парсингу UML конструкцій.

Цей проект може бути використаний як основа для розробки більш масштабних систем автоматизації процесів програмування, що відрізнятимуться надійним рівнем безпеки через систему ролей та OAuth автентифікацію, високою продуктивністю завдяки оптимізованим алгоритмам конвертації, зручністю використання через інтеграцію з Monaco Editor та можливістю масштабування через контейнеризовану архітектуру.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. PlantUML [Електронний ресурс] – Режим доступу до ресурсу: <https://plantuml.com/>
2. StarUML [Електронний ресурс] – Режим доступу до ресурсу: <https://staruml.io/>
3. RESTful API Design [Електронний ресурс] – Режим доступу до ресурсу: <https://restfulapi.net/>
4. Документація Symfony [Електронний ресурс] – Режим доступу до ресурсу: <https://symfony.com/doc/current/index.html>
5. Документація Doctrine ORM [Електронний ресурс] – Режим доступу до ресурсу: <https://www.doctrine-project.org/projects/doctrine-orm/en/3.3/index.html>
6. Документація PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/docs/>
7. Monaco Editor [Електронний ресурс] – Режим доступу до ресурсу: <https://microsoft.github.io/monaco-editor/>
8. EasyAdmin Bundle [Електронний ресурс] – Режим доступу до ресурсу: <https://symfony.com/bundles/EasyAdminBundle/current/index.html>
9. Twig Template Engine [Електронний ресурс] – Режим доступу до ресурсу: <https://twig.symfony.com/doc/3.x/>
10. Документація Docker [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com/>
11. Docker Compose [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com/compose/>
12. NGINX [Електронний ресурс] – Режим доступу до ресурсу: <https://nginx.org/en/docs/>
13. OAuth 2.0 Authorization Framework [Електронний ресурс] – Режим доступу до ресурсу: <https://oauth.net/2/>
14. GitHub OAuth Apps [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.github.com/en/apps/oauth-apps>

15. Google OAuth 2.0 [Електронний ресурс] – Режим доступу до ресурсу: <https://developers.google.com/identity/protocols/oauth2>
16. UML Class Diagrams [Електронний ресурс] – Режим доступу до ресурсу: <https://www.uml-diagrams.org/class-diagrams-overview.html>
17. Clean Architecture [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>
18. Monolog PHP Logging Library [Електронний ресурс] – Режим доступу до ресурсу: <https://seldaek.github.io/monolog/>
19. Symfony Security Component [Електронний ресурс] – Режим доступу до ресурсу: <https://symfony.com/doc/current/security.html>
20. PHP: Офіційна документація [Електронний ресурс] – Режим доступу до ресурсу: <https://www.php.net/manual/en/>
21. Python: Офіційна документація [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.python.org/3/>
22. Java SE Documentation [Електронний ресурс] – Режим доступу: <https://docs.oracle.com/javase/8/docs/>
23. Visual Studio Code [Електронний ресурс] – Режим доступу: <https://code.visualstudio.com/>
24. Git: Документація [Електронний ресурс] – Режим доступу: <https://git-scm.com/doc>
25. Composer – PHP Dependency Manager [Електронний ресурс] – Режим доступу: <https://getcomposer.org/>
26. OpenAPI Specification [Електронний ресурс] – Режим доступу: <https://swagger.io/specification/>
27. API Platform [Електронний ресурс] – Режим доступу: <https://api-platform.com/docs/>
28. Webpack Encore (Symfony) [Електронний ресурс] – Режим доступу: <https://symfony.com/doc/current/frontend/encore/installation.html>

29. JWT Authentication for Symfony [Электронный ресурс] – Режим доступа: <https://github.com/lexik/LexikJWTAuthenticationBundle>

30. HTML Living Standard [Электронный ресурс] – Режим доступа: <https://html.spec.whatwg.org/>

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						55

Основний парсер PlantUmlParser (метод parse)

```

public function parse(string $content): ClassDiagram
{
    $this->initialize($content);
    $this->validateStartAndEnd();

    while ($this->currentLine < count($this->lines)) {
        $line = trim($this->lines[$this->currentLine]);

        if (empty($line) || $this->isComment($line) || $this->
            >isDirective($line)) {
            $this->currentLine++;
            continue;
        }

        if
        (preg_match('/^(class|abstract\s+class|interface|enum)\s+(\w+
        )/', $line, $matches)) {
            $this->parseClassElement($matches[1],
            $matches[2]);
        } elseif ($this->isRelationship($line)) {
            $this->relationships[] = $line;
            $this->currentLine++;
        } else {
            $this->currentLine++;
        }
    }

    $this->processRelationships();
    return $this->diagram;
}

```

Рекурсивний парсинг генеричних типів

```

private function splitGenericParameters(string $paramsStr):
array
{
    $result = [];
    $current = '';
    $depth = 0;

    for ($i = 0; $i < strlen($paramsStr); $i++) {
        $char = $paramsStr[$i];

        if ($char === '<') {
            $depth++;
            $current .= $char;
        } elseif ($char === '>') {
            $depth--;
            $current .= $char;
        } elseif ($char === ',' && $depth === 0) {
            $result[] = $current;
            $current = '';
        } else {
            $current .= $char;
        }
    }

    if ($current !== '') {
        $result[] = $current;
    }

    return array_map('trim', $result);
}

```

Додавання класу до діаграми

```

public function addClass(ClassElement $class): self
{
    $this->classes[] = $class;
    return $this;
}

public function findClass(string $name): ?ClassElement
{
    foreach ($this->classes as $class) {
        if ($class->getName() === $name) {
            return $class;
        }
    }
    return null;
}

public function toArray(): array
{
    $result = [];
    if ($this->title) {
        $result['title'] = $this->title;
    }

    $classes = [];
    foreach ($this->classes as $class) {
        $classes[] = $class->toArray();
    }
    $result['classes'] = $classes;

    return $result;
}

```

Абстрактний генератор коду

```
abstract class AbstractLanguageCodeGenerator extends
AbstractCodeGenerator implements LanguageCodeGenerator
{
    protected string $outputDirectory = '';

    public function setOutputDirectory(string
$outputDirectory): self
    {
        $this->outputDirectory = rtrim($outputDirectory,
'/');
        return $this;
    }

    public function getLanguage(): string
    {
        return $this->language;
    }

    public function getVersion(): string
    {
        return $this->version;
    }
}
```