Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

УДК 004.912

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: "Розробка макету програмного забезпечення організації роботи студентів на кафедрі за умов дистанційного навчання. Серверне програмне забезпечення"

Спеціальність – 121 "Інженерія програмного забезпечення"

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ - ___.___.___._____ ІПЗ

Студент
ІПЗ-44 _____ /Олександр СВЯТЕЦЬКИЙ/

Науковий керівник
к.ф.-м.н., доц. _____ /Ірина ЮРЧУК/

Консультант
з питань нормоконтролю
фахівець_____ /Тамара ЧАПОВСЬКА/

Допускається до захисту

Завідувач кафедри
д.т.н., професор _____ /Олексій БИЧКОВ/

Київ - 2021

Рішенням Екзаменаційної комісії

Випускна кваліфікаційна робота студента

_____

Захищена з оцінкою

_____

Голова Екзаменаційної комісії

Професор, доктор технічних наук Віктор ВИШНІВСЬКИЙ

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Освітньо-кваліфікаційний рівень бакалавр
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО
Зав. кафедри програмних систем і технологій
_____ (Олексій БИЧКОВ)
“_____” _____ 2021 р.

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Святецькому Олександру Сергійовичу

(прізвище, ім’я, по-батькові)

1. **Тема випускної кваліфікаційної бакалаврської роботи <u>“Розробка макету програмного забезпечення організації роботи студентів на кафедрі за умов дистанційного навчання. Серверне програмне забезпечення”</u>**, керівник роботи к.ф.-м.н., доцент Юрчук Ірина Аркадіївна затверджені на засіданні кафедри програмних систем і технологій, протокол №6 від «11» листопада 2020р.
2. **Строк подання студентом закінченої роботи** _____
3. **Вихідні дані до роботи:** монографії, підручники, навчальні посібники, статті та тези конференцій вітчизняних і зарубіжних авторів, Інтернет–ресурси з питань дистанційного навчання та систем керування навчанням.
4. **Зміст розрахунково-пояснювальної записки**
   1. Огляд концепції дистанційного навчання та характеристика систем керування навчанням
   2. Розробка архітектури системи, опис ролей користувачів та моделей даних
   3. Розробка програмної реалізації системи
5. **Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)**

   1. Приклад сторінки у системі WebCT (рис. 1.1, ст. 16)

   2. Приклад сторінки у системі Moodle (рис. 1.2, ст. 17)

   3. Основні категорії функцій типової LMS (рис. 1.3, ст. 20)

   3. Порівняння функціональності популярних систем керування навчанням (табл. 1, ст. 22)

   4. Діаграма компонентів системи (рис. 2.1, ст. 24)

5. Структура шарів фреймворку Django (рис. 2.2, ст. 28)
6. Приклад роботи з PostgreSQL в середовищі pgAdmin (рис. 2.3, ст. 31)
7. Структура проекта розробленого веб-застосунка (рис. 2.4, ст. 32)
8. Схема основних сутностей в системі (рис. 2.5, ст. 33)
9. Клас спеціального поля з UUID (рис. 2.6, ст. 34)
10. Моделі для представлення контенту в системі (рис. 2.7, ст. 35)
11. Модель для представлення облікового запису користувача (рис. 2.8, ст. 37)
12. Клас-менеджер для моделі користувача (рис. 2.9, ст. 38)
13. Серіалізатор моделі для представлення дисциплін (рис. 2.10, ст. 39)
14. Клас для фільтрації дисциплін (рис. 2.11, ст. 40)
15. CRUD-контролер для роботи з моделлю дисциплін (рис. 2.12, ст. 41)
16. Запит на отримання кореневих точок API (рис. 2.13, ст. 43)
17. Приклад автентифікації користувача (рис. 2.14, ст. 44)
18. Створення нової дисципліни (рис. 2.15, ст. 44)
19. Перегляд списку дисциплін (рис. 2.16, ст. 45)
20. Приклад кінцевої точки в Swagger (рис. 2.17, ст. 46)
20. Приклад кінцевої точки в Redoc (рис. 2.18, ст. 46)
20. Приклад кінцевої точки в BrowsableAPI (рис. 2.19, ст. 47)

6. **Консультанти розділів проекту (роботи)**

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Концепція системи дистанційного навчання	І. А. Юрчук	І. А. Юрчук	
Реалізація серверного веб-застосунку для організації роботи студентів за умов дистанційного навчання	І. А. Юрчук	І. А. Юрчук	

7. **Дата видачі завдання** _____

Керівник _____ (Ірина ЮРЧУК)

Завдання прийняв до виконання _____ (Олександр СВЯТЕЦЬКИЙ)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Уточнення постановки задачі	29.11.2020-05.12.2020	виконано
2	Підбір та вивчення літератури	06.12.2020-04.01.2021	виконано
3	Аналіз існуючих рішень з організації дистанційного навчання	09.01.2021-16.01.2021	виконано
4	Огляд фреймворків для програмної реалізації серверної частини	28.01.2021-14.02.2021	виконано
5	Розробка архітектури серверного веб-застосунку	15.02.2021-20.03.2021	виконано
6	Розроблення серверного веб-застосунку у вигляді веб-API	21.03.2021-30.04.2021	виконано
7	Тестування розробленого програмного забезпечення	02.05.2021-15.05.2021	виконано
8	Оформлення і друк пояснювальної записки	16.05.2021-26.05.2021	виконано
9	Оформлення презентації	27.05.2021-03.06.2021	виконано
10	Отримання рецензії		
11	Затвердження пояснювальної записки роботи завідувачем кафедри		
12	Захист дипломної роботи		

Студент – бакалавр _____ (Олександр СВЯТЕЦЬКИЙ)
(підпис)

Керівник роботи _____ (Ірина ЮРЧУК)
(підпис)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 73 с., 20 рис., 1 табл., 5 додат., 11 джерел.

Тема: Розробка макету програмного забезпечення організації роботи студентів на кафедрі за умов дистанційного навчання. Серверне програмне забезпечення

Об'єкт дослідження: процес організації дистанційного навчання студентів.

Мета робота: вдосконалення процесу дистанційного навчання студентів за допомогою розробленого веб-застосунку.

Предмет дослідження: застосування програмних засобів для ефективної організації дистанційного навчання.

Результати дослідження: досліджено існуюче програмне забезпечення для організації дистанційного навчання. Створено клієнт-серверну систему організації роботи студентів за умов дистанційного навчання, з можливістю формування графіку навчання та створення й перевірки завдань для виконання.

Висновок

В результаті досліджень було розроблено серверний веб-API застосунок для організації роботи студентів, з можливістю формування навчального графіку, створення й перевірки завдань для виконання студентами.

СИСТЕМА УПРАВЛІННЯ НАВЧАННЯМ, НАВЧАЛЬНИЙ ГРАФІК, ВЕБ-API, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ПРЕДМЕТНА ОБЛАСТЬ ДИСТАНЦІЙНЕ НАВЧАННЯ

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 73 с., 20 рис., 1 табл., 5 доп., 11 источников.

Тема: Разработка макета программного обеспечения организации работы студентов на кафедре в условиях дистанционного обучения. Серверное программное обеспечение

Объект исследования: процесс организации дистанционного обучения студентов.

Цель работы: совершенствование процесса дистанционного обучения студентов с помощью разработанного веб-приложения.

Предмет исследования: применение программных средств для эффективной организации дистанционного обучения.

Результаты исследования: исследовано существующее программное обеспечение для организации дистанционного обучения. Создана система организации работы студентов в условиях дистанционного обучения, с возможностью формирования графика обучения, создания и проверки заданий для выполнения.

Вывод

В результате исследований было разработано серверное веб-API приложение для организации работы студентов, с возможностью формирования графика обучения, создания и проверки заданий для выполнения студентами.

СИСТЕМА УПРАВЛЕНИЯ ОБУЧЕНИЕМ, УЧЕБНЫЙ ГРАФИК, ВЕБ-API, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА, ПРЕДМЕТНАЯ ОБЛАСТЬ ДИСТАНЦИОННОЕ ОБУЧЕНИЕ

ABSTRACT

Graduation qualifying bachelor's thesis: 73 p., 20 figs., 1 table, 5 appendices, 11 sources.

Topic: Development of a software model for organizing the work of students at the academic department under conditions of distance learning. Server-side software

Object of research: the process of organizing distance learning for students.

Purpose: improving the process of distance learning of students with the help of a developed web application.

Subject of study: application of software for effective organization of distance learning.

Research results: the existing software for the organization of distance learning is investigated. A system of the organization of work of students under the conditions of distance learning has been created, with the abilities to form the training schedule and create/check course tasks.

Conclusion

As a result of the research, a server-side web API application was developed to organize the work of students, with the abilities to form the training schedule and create/check course tasks.

LEARNING MANAGEMENT SYSTEM, TRAINING SCHEDULE, WEB-API, CLIENT-SERVER ARCHITECTURE, SUBJECT AREA OF DISTANCE LEARNING

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

LMS – система управління навчанням (англ. Learning Management System)

API – прикладний програмний інтерфейс (англ. Application Programming Interface)

HTTP – протокол передачі гіпертексту (англ. HyperText Transfer Protocol)

URL – уніфікований локатор ресурсу (англ. Uniform Resource Locator)

URI – уніфікований ідентифікатор ресурсу (англ. Uniform Resource Identifier)

REST – передача репрезентативного стану (англ. Representational State Transfer)

ORM – об'єктно-реляційне відображення (англ. Object Relational Mapping)

СУБД – системи управління базами даних

UUID – універсальний унікальний ідентифікатор (англ. Universally Unique Identifier)

JSON – запис об’єктів JSON (англ. Javascript Object Notation)

ВСТУП

Сучасна ситуація з виникненням різних пандемій та відсутності можливості отримувати всі теоретичні та практичні навички під час навчання, які раніше були невід'ємною частиною традиційного навчання, змушує розробляти нові стратегії проведення та підтримання навчального процесу в дистанційних умовах. В наш час система дистанційного навчання часто є необхідністю, яка частково або повністю замінює традиційне навчання. Програмне забезпечення такого типу надає змогу всім користувачам цієї системи бути на зв'язку та завжди мати доступ до актуальної інформації щодо навчального процесу.

Актуальність теми: наявність системи дистанційного навчання дозволяє зберігати необхідну для підтримання навчального процесу інформацію в одному місці та надає можливість зручного доступу до неї. Система надаватиме користувачам змогу авторизуватись та використовувати її можливості залежно від їх ролей та прав доступу. Це спрощує процес навчання і вирішує сучасні проблеми з неможливістю проведення його в режимі оффлайн.

Метою роботи є вдосконалення процесу дистанційного навчання студентів за допомогою розробленого веб-застосунку.

Завдання:

1. Дослідити концепцію систем дистанційного навчання;
2. Провести аналіз типової функціональності систем дистанційного навчання;
3. Створити макет системи дистанційного навчання;
4. Провести тестування розробленого програмного забезпечення.

Об'єкт дослідження – процес організації дистанційного навчання студентів.

Предметом дослідження є застосування програмних засобів для ефективної організації дистанційного навчання.

Методи дослідження: мова програмування Python, редактор коду Visual Studio Code, фреймворки для створення веб-застосунків Django та Django REST, СУБД PostgreSQL, генератори API-документації Swagger та Redoc, застосунок для API-тестування Postman, засоби контейнеризації Docker.

Новизна одержаних результатів: розроблено макет системи дистанційного навчання з можливостями розширення функціональності та простого встановлення як на виділеному сервері, так і в хмарному сервісі.

Практичне значення одержаних результатів: практична цінність розробленого програмного забезпечення полягає в можливості організації дистанційного навчання в навчальному закладі без застосування комерційних рішень або складних систем з відкритим вихідним кодом.

РОЗДІЛ 1
КОНЦЕПЦІЯ СИСТЕМИ ДИСТАНЦІЙНОГО НАВЧАННЯ

1.1 Поняття дистанційного навчання

Для визначення поняття дистанційного навчання перш за все необхідно визначити протилежне поняття традиційної освіти. Традиційна освіта (також звичайна освіта) відноситься до встановлених суспільством норм та звичаїв у навчальних закладах, що традиційно визнаються доречними та/або ефективними в процесі навчання. Деякі форми освітньої реформи сприяють прийняттю прогресивної освітньої практики, більш цілісного підходу, який фокусується на потребах та самовираженні окремих учнів. На думку реформаторів в сфері освіти, навчальні заклади мають відмовитись від традиційного, орієнтованого на викладачів, підходу на користь підходів до навчання, орієнтованих на учнів та завдання. Залежно від контексту, протилежністю традиційній освіті може бути прогресивна освіта, сучасна освіта (освітні підходи, засновані на психології розвитку) або альтернативна освіта [1].

Традиційна освіта не є суто позитивним або негативним явищем і має певні переваги та недоліки. До переваг традиційної освіти можна віднести безпосередню присутність студентів на занятті та взаємодію з викладачем, чіткий графік занять та перерв, соціалізацію студентів та викладачів тощо. Однак, за певних умов деякі з переваг традиційної освіти перетворюються на недоліки, наприклад, складність безпосереднього перебування у навчальному закладі на заняттях під час обмеження руху громадського транспорту, застарілість засобів формування освітньої інформації (розклади занять, рейтинги студентів, списки оцінок з дисциплін), обмеженість засобів викладання навчальної інформації студентам на заняттях тощо. Деякі з цих проблем вирішує дистанційне навчання та інформаційні системи дистанційного навчання.

Для огляду призначення та функціональності типової системи дистанційного навчання необхідно визначити поняття дистанційного навчання.

Дистанційним навчанням називають взаємодію викладачів та студентів на відстані за допомогою Інтернет-технологій зі збереженням типових для навчального процесу компонентів. Дистанційне навчання відрізняється від так званої онлайн-освіти як можливістю комбінації дистанційного та очного навчання, так і застосуванням методів очного навчання з пристосуванням для дистанційної освіти. Традиційно дистанційне навчання передбачає проходження студентом дисциплін заочно та листування з навчальним закладом поштою, однак сьогодні також передбачає елементи онлайн-освіти. Програма дистанційного навчання може бути як повністю дистанційною, так і поєднанням дистанційного та традиційного навчання (так зване гібридне або змішане навчання) [2].

Дистанційне навчання пропонує освітні можливості, що відповідають мінливим потребам студентів та надають їм гнучкість навчання у будь-який час, в будь-якому місці та у темпі, що відповідає їхнім індивідуальним стилям навчання за допомогою Інтернет-технологій та/або відеоконференцій. Відповідно до означених понять, можна визначити такі відмінності дистанційної освіти від традиційної:

1. Гнучкість – як правило, дистанційна освіта передбачає асинхронну роботу студентів, тобто надає можливість виконувати завдання в будь-якому місці та в зручний для студента час, що може бути корисним для працюючих студентів.
2. Взаємодія з викладачем – сучасні платформи дистанційного навчання надають засоби для організації відеоконференцій та текстового спілкування, що, однак, не може замінити повноцінної взаємодії на очному занятті.
3. Навчальний матеріал – засоби дистанційного навчання дозволяють розширити можливості традиційного навчального матеріалу за допомогою розміщення графічних матеріалів, посилань на навчальну літературу, створення інтерактивних елементів (наприклад, простих середовищ програмування) тощо.

1.2 Поняття системи дистанційного навчання

Зростаюча популярність дистанційного навчання вводить нові терміни в освіту, такі як віртуальний клас – моделювання очного навчального процесу за допомогою Інтернет-технологій, що дозволяє викладачам та студентам взаємодіяти зручним чином та має на меті забезпечити подібний до очного навчання досвід; це мережева система, яка забезпечує розподілене, майже в режимі реального часу, електронне середовище спільної роботи, що дозволяє використовувати відео, аудіо та інші засоби для ефективного способу викладання матеріалу. Наприклад, викладач, який виступає з презентацією, може поєднувати усну лекцію зі статичними слайдами та одночасно застосовувати інструменти оцінки засвоєння матеріалу студентами [3].

Існує багато термінів, пов’язаних з онлайн-навчанням та технологіями, що розвинулися для його підтримки. Однією з постійних проблем є плутанина у визначенні скорочень: CMS та LMS. Термін CMS часто асоціюється з двома абсолютно різними типами програмних системи: "система управління вмістом" (англ. Content Management System) і "система управління курсами" (англ. Course Management System). Системи управління вмістом – це, по суті, програмні засоби, призначені для створення та управління цифровим вмістом у середовищі спільної роботи. З іншого боку, системи управління курсами використовуються в основному для навчання в Інтернеті або змішаного навчання, підтримки розміщення матеріалів курсу в режимі онлайн, асоціювання студентів з курсами, відстеження успішності студентів, зберігання заявок студентів та посередницького спілкування між студентами [4]. Більше того деякі вендори та науковці використовують абревіатуру LCMS, що означає "система управління змістом навчання" (англ. Learning Content Management System), маючи на увазі системи управління контентом. Різниця між LCMS та LMS полягає в тому, що остання є більш широкою за обсягом і включає можливість відстеження прогресу учнів через онлайн-курс. Надалі використовуватиметься абревіатура LMS як переклад понять “система

дистанційного навчання” та “система керування навчанням”, а самі ці поняття – як взаємозамінні.

Отже, система дистанційного навчання, або система керування навчанням, – це програмне забезпечення для адміністрування, документування, відстеження, звітності, автоматизації та проведення навчальних курсів і програм. Концепція системи управління навчанням виникла безпосередньо з електронного навчання, тобто принципів використання інформаційних технологій та засобів у навчанні. Перші системи управління навчанням з’явились в секторі вищої освіти, однак сьогодні більшість таких систем зосереджені в корпоративному сегменті та відіграють важливу роль в навчанні працівників [5]. Системи управління навчанням в першу чергу розроблялись для виявлення прогалин у навчанні за допомогою аналітики та орієнтовані на онлайн-навчання, але також можуть виступати загальною платформою для онлайн-контенту. Сучасні системи дистанційного навчання пропонують інтелектуальні алгоритми для формування рекомендацій щодо проходження курсів, базуючись на профілях користувачів та їх попередніх результатах.

Як і більшість програмного забезпечення, системи дистанційного навчання поділялись на дві категорії: пропрієтарні та з відкритим вихідним кодом. Однією з перших пропрієтарних систем стала WebCT, розроблена в Університеті Британської Колумбії в 1995 році та заснована на дослідженнях щодо покращення академічних показників за допомогою інформаційних технологій та веб-ресурсів. На піку свого використання WebCT (рис. 1.1) була найбільш широко використовуваною системою управління вмістом у всьому світі з понад 10 мільйонами користувачів у 80 країнах [6].

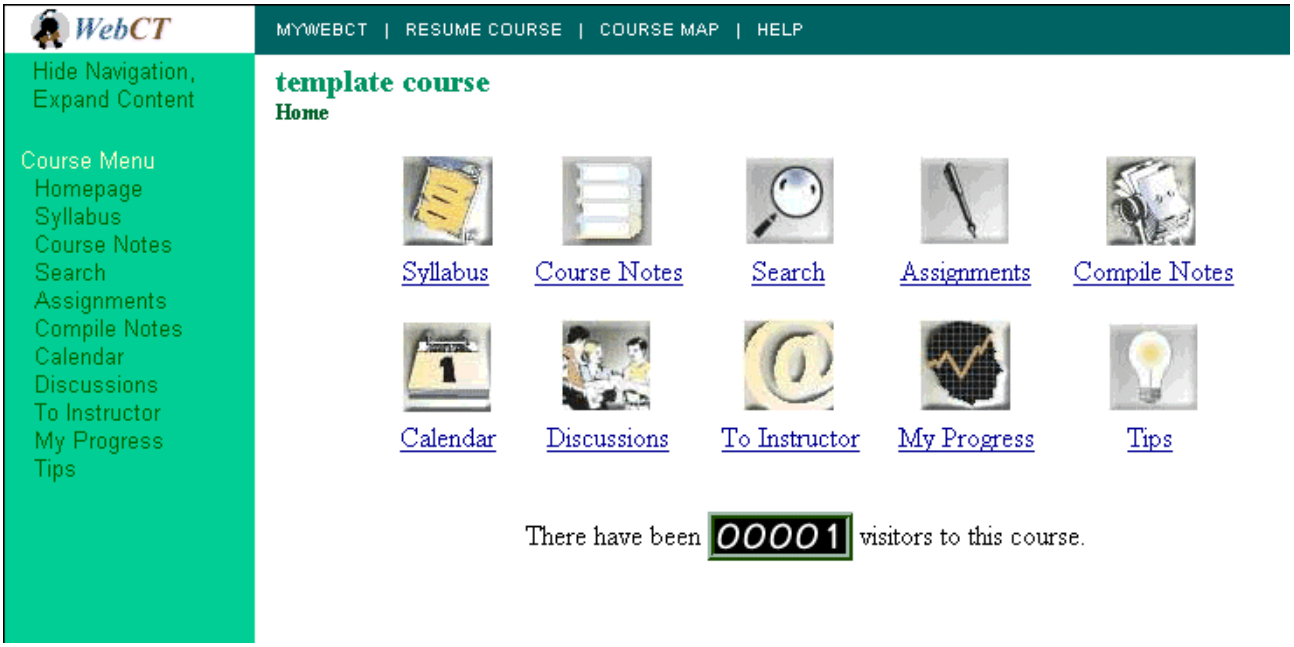


Рис. 1.1. Приклад сторінки у системі WebCT

На противагу пропрієтарним системам розроблялись також програмні рішення з відкритим вихідним кодом з метою зробити такі системи доступними для організацій та приватних осіб безкоштовно. Однією з найпопулярніших систем керування навчанням став Moodle (рис. 1.2), розроблений Мартіном Доугіамасом, що розшифровується як “Модульне об’єктно-орієнтоване динамічне навчальне середовище” (англ. Modular Object Oriented Dynamic Learning Environment). Дана система була розроблена на основі конструктивістської філософії, яка підкреслює роль учнів як творців змісту та орієнтована на організацію взаємодії між учнями й викладачем. Сьогодні зареєстровано близько 100 тисяч сайтів, що працюють на Moodle, у 229 країнах світу, а сама система перекладена на десятки мов [7].

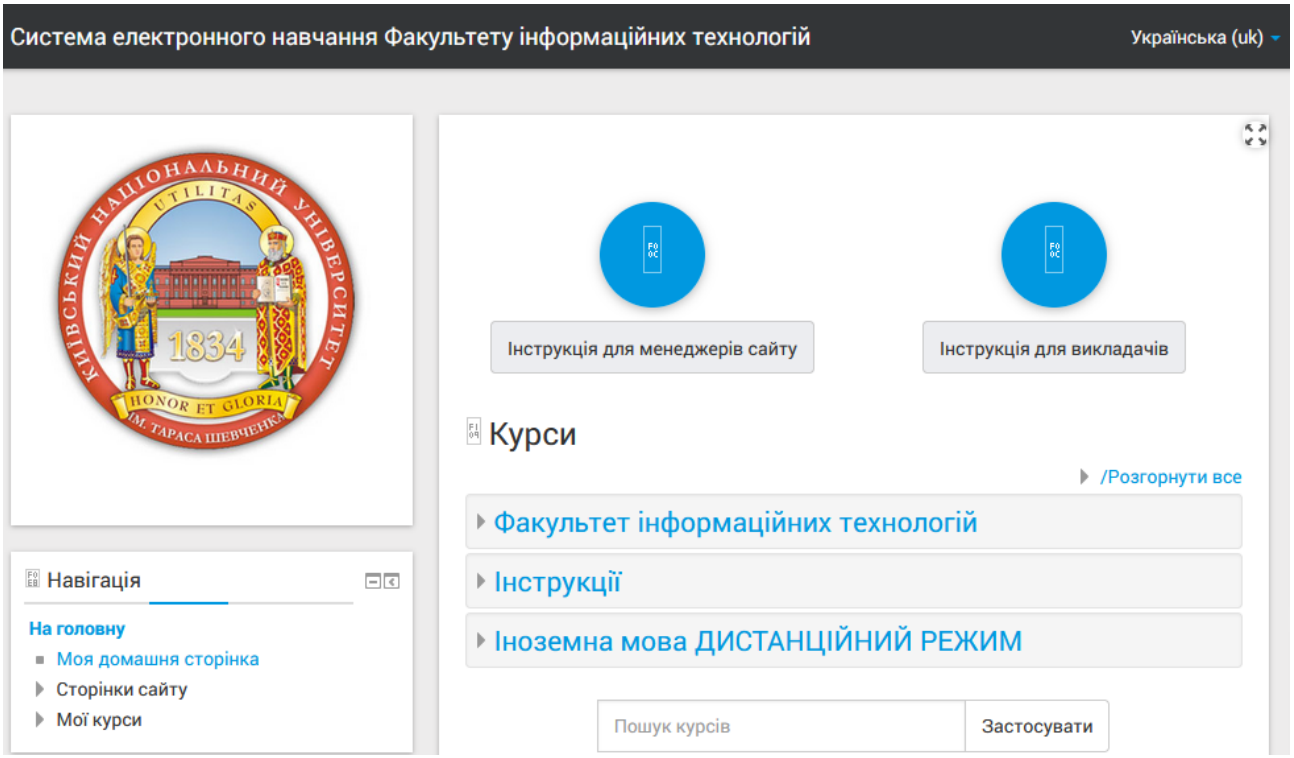


Рис. 1.2. Приклад сторінки у системі Moodle

1.3 Напрямки розвитку систем дистанційного навчання

Зі збільшенням орієнтації методів навчання на студентів та розвитком інформаційних технологій, системи управління навчанням також ставатимуть все більше необхідними та корисними; з розширенням пропускної спроможності, обсягу зберігання даних та обчислювальних можливостей мобільних пристроїв можливості LMS адаптуватимуться для задоволення нових потреб клієнтів.

Загальні тенденції систем керування навчанням полягають у збільшенні можливостей студентів щодо взаємодії з навчальним матеріалом, наприклад, використання смартфонів та персональних розумних пристроїв. Також одним із важливих напрямків розвитку LMS є вдосконалення вже існуючих та створення нових засобів синхронної комунікації, наприклад, покращення якості відеоконференцій.

До основних напрямів розвитку систем дистанційного навчання в майбутньому можна віднести:

1. Адаптивне навчання

   Адаптивні технології навчання дозволяють розробникам курсів пристосовувати навчальні завдання та матеріали до індивідуальних потреб учнів. Як приклади таких технологій можна навести рекомендації навчального матеріалу на основі попередніх досягнень студента, створення навчальних матеріалів студентами, адаптування тестів на основі успішності студентів.

2. Засоби аналітики

   Функції звітування, такі як контроль присутності та інформація про виконання завдань, є стандартом для сучасних LMS. Завдання майбутніх систем полягає у використанні різноманітних даних, зібраних системою управління навчанням, та використанні цієї інформації для прогнозування можливих проблем у студентів.

3. Засоби комунікації

   Крім знань, вмінь і навичок важливою складовою традиційного навчання є соціалізація та взаємодія з іншими студентами – часто системи дистанційного навчання критикують за відсутність або невідповідність засобів комунікації. Включення більш синхронних засобів спілкування, таких як відеоконференції в реальному часі та додатки в соціальних мережах у реальному часі, швидше за все, посилить привабливість онлайн-навчання як соціальної діяльності. Постачальники LMS вже використовують соціальні мережі, як от Facebook, Twitter і WhatsApp, а також засоби відеоконференцій, як Skype або Zoom, щоб забезпечити соціальну основу для студентів.

4. Хмарні технології

   Вже існує тенденція до орієнтованих на хмарні технології пропрієтарних LMS, оскільки часто постачальники пропонують власні системи клієнтам, які не мають інфраструктури або персоналу для управління внутрішнім хостингом. Однак безкоштовні системи з відкритим вихідним кодом, такі як Moodle, також розвиваються з урахуванням можливості існування своїх систем виключно в середовищі веб-хостингу. Наприклад, сервіс Softaculous можна використовувати для встановлення та налаштування таких систем керування навчанням як Moodle безпосередньо в обліковому записі веб-хостингу клієнта.

5. Гейміфікація

   Навчальні ігри, якщо вони правильно структуровані, можуть забезпечити цікавий та стимулюючий спосіб залучити учнів, винагороджуючи їхній прогрес. Майбутні ігрові функції LMS можуть присвоювати сертифікати або значки учням на основі їхнього володіння змістом курсу і навіть можуть бути використані для присвоєння звання або статусу окремим учням, якими можна поділитися у спільноті користувачів.

1.4 Огляд типової функціональності системи дистанційного навчання

Для ефективної організації дистанційного навчання система управління навчанням має бути надійною та якісною, повинна забезпечувати своєчасну та безперешкодну комунікацію студентів, викладачів та зацікавлених осіб. Незалежно від того, чи є LMS пропрієтарною або з відкритим вихідним кодом, для забезпечення якісного досвіду користувача система повинна надавати такі основні функції як розміщення навчальних матеріалів, оцінка компетентності студентів, відслідковування їх досягнень та результатів роботи, надання засобів комунікації та засобів безпеки. На рисунку 1.3 зображено основні категорії функцій, що повинна надавати якісна система керування навчанням.

Рис. 1.3. Основні категорії функцій типової LMS

Функції управління дисциплінами охоплюють здатність LMS надавати своєчасний та відповідний матеріал курсу студентам, управління вмістом курсу, планування занять та можливості аудиту навчального матеріалу. Також деякі системи надають можливість студентам розширювати вже існуючий матеріал.

Однією з найважливіших функцій системи дистанційного навчання є оцінювання результатів роботи студентів. Типова LMS повинна надавати можливість створювати завдання для оцінювання у вигляді тестів, проектів та задач, а також надавати оцінку й зворотний зв’язок студентам. Крім цього, бажаними можливостями LMS є надання студенту в режимі реального часу інформації про його досягнення по дисципліні, забезпечення зворотного зв’язку викладачів та студентів, формування звітності.

Зазвичай студенти на дистанційному навчанні зазнають складнощів у сприйнятті навчального матеріалу через відсутність безпосередньої взаємодії з викладачем, а також через наявність значної кількості відволікаючих факторів, які можуть бути відсутніми в традиційному освітньому закладі. Тому важливою функцією систем дистанційного навчання є відслідковування участі й прогресу студентів з дисципліни. За допомогою якісних функцій звітування та аналітики, що можуть включати відстеження присутності студента та часу обробки

навчального матеріалу, перевірку взаємодії студентів з викладачем тощо, викладачі дисципліни можуть виявляти як прогалини в навчальному матеріалі, так і проблеми з окремими студентами.

Безпека та конфіденційність надзвичайно важливі для успішної організації дистанційного навчання. Важливими функціями безпеки в системах управління вмістом є автентифікація користувачів, перевірка доступу, контроль цілісності паролів та виявлення зловмисників. Контроль конфіденційності також має важливе значення для того, щоб конфіденційна інформація була доступною лише для одержувача.

Система дистанційного навчання повинна надавати певні засоби комунікації між викладачами та студентами. Такі засоби можуть бути синхронними та асинхронними: до асинхронних відносяться засоби односторонньої комунікації, тобто, електронна пошта, сповіщення, дошки обговорень тощо; в той час синхронними вважаються засоби двосторонньої комунікації, що забезпечують обмін інформацією в режимі реального часу – відеоконференції, інтерактивні дошки, real-time чати.

Важливою характеристикою системи дистанційного навчання є її доступність: сьогодні все більше зростає роль портативної техніки (зокрема, ноутбуків, смартфонів та планшетів) у дистанційному навчанні, наприклад, для зручного перегляду розкладу навчання та завдань у мобільних додатках. Більшість систем керування навчанням розробляються за клієнт-серверною технологією, що дозволяє незалежно створювати серверні застосунки для зберігання й обробки даних та клієнтські застосунки для коректного відображення вмісту на цільових платформах. До того ж, для розробки клієнтських веб-застосувань використовуються фреймворки з адаптивною верстkою, що дозволяє відображати інтерфейс в зручному вигляді на різних платформах (наприклад, у мобільних браузерах без необхідності створювати мобільні застосунки).

Нижче представлена таблиця порівняння функціональності популярних систем керування навчанням для вищих навчальних закладів з відкритим вихідним кодом [8].

Таблиця 1

Система управління навчанням / **Функція**	**Moodle**	**ATutor**	**ILIAS**
Персоналізована дошка	наявна	наявна	наявна
Багатомовність	наявна	наявна	наявна
Резервне копіювання	наявна	наявна	відсутня
Користувацькі ролі	наявна	наявна	наявна
Оновлення безпеки	наявна	відсутня	наявна
Календар	наявна	наявна	наявна
Оголошення	наявна	наявна	наявна
Перекладач	відсутня	наявна	відсутня
Словник / глосарій	відсутня	наявна	наявна
Опитування	наявна	наявна	наявна
Чат	наявна	наявна	наявна
Електронна пошта	наявна	наявна	наявна
SMS-сповіщення	наявна	відсутня	відсутня

1.5 Висновки до розділу

В першому розділі було розглянуто предметну область, а саме поняття та особливості дистанційного навчання, поняття та характеристика систем дистанційного навчання, типові функції систем керування навчанням.

РОЗДІЛ 2
РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ОРГАНІЗАЦІЇ РОБОТИ СТУДЕНТІВ ЗА УМОВ ДИСТАНЦІЙНОГО НАВЧАННЯ

2.1 Цілі системи

Система для організації роботи студентів за умов дистанційного навчання повинна:

1. Вміти створювати та зберігати користувачів системи, розподіляти їх на відповідні ролі.
2. Дати змогу користувачам автентифікуватись та авторизуватись.
3. Вміти створювати та зберігати дані про дисципліни, навчальні графіки, завдання для студентів.
4. Дати змогу управляти даними про дисципліни, навчальними графіками, завданнями для студентів.
5. Дати змогу викладачам переглядати навчальні графіки за обраними дисциплінами, формувати завдання для студентів та контролювати їх виконання.
6. Дати змогу студентам переглядати навчальні графіки за обраними дисциплінами та виконувати призначені викладачами завдання.

2.2 Архітектура системи

Серверна частина розроблюваної системи складатиметься з 3 компонентів:

1. База даних
2. Веб-сервер
3. Серверний веб-API застосунок

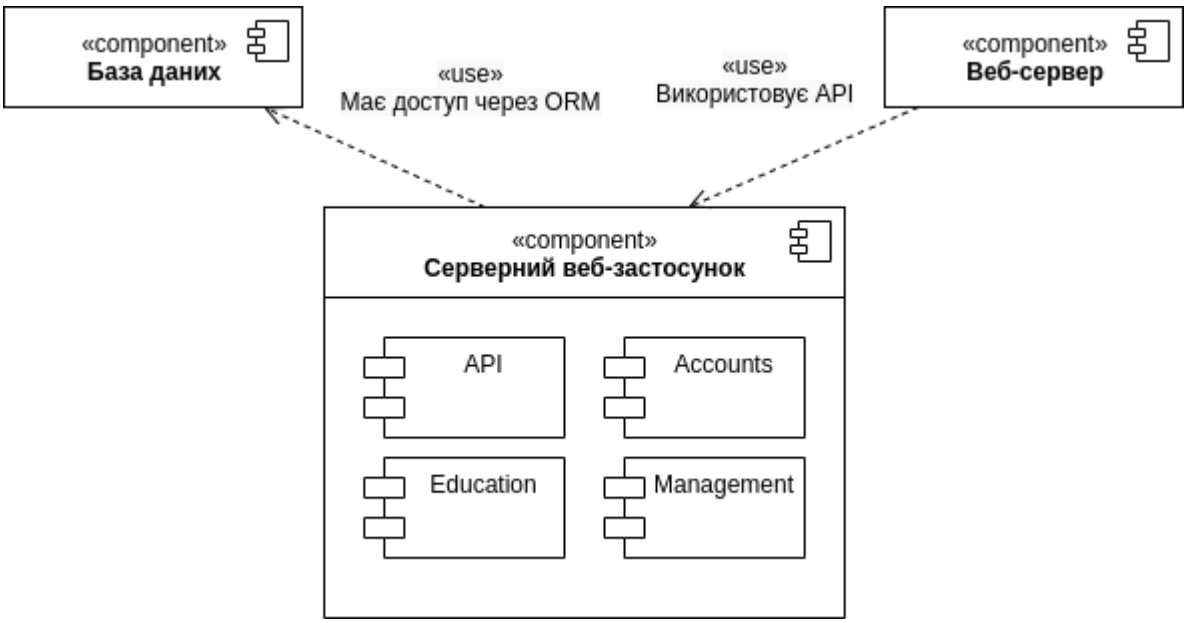


Рис 2.1. Діаграма компонентів системи

2.3 Ролі в системі

В системі є 4 ролі:

1. Адміністратор користувачів

   Роль, що надає можливість управляти обліковими записами інших користувачів системи (тобто, додавати нові облікові записи або змінювати дані вже наявних в системі) та керувати права доступу користувачів.

2. Адміністратор дисциплін

   Основна роль в системі, що надає можливість управляти інформацією про дисципліни, формувати навчальні графіки студентів та викладачів.

3. Викладач

   Роль, що надає можливість переглядати навчальні графіки та інформацію з дисциплін, створювати завдання для студентів та перевіряти і оцінювати його виконання.

4. Студент

   Роль, що надає можливість переглядати навчальні графіки та інформацію з дисциплін, виконувати призначені завдання.

2.4 Моделі даних

У системі зберігатиметься 9 основних сутностей:

1. Користувач (User).

   Поля: id, uuid, роль, хеш паролю, ім’я, email, дата створення, статус активності.

   Є базовою сутністю для облікових записів системи.; пов’язана з базовою сутністю “Профіль користувача” як “один-до-одного”.

2. Профіль користувача (UserProfile).

   Поля: id, uuid, дата створення.

   Є базовою сутністю для зберігання даних користувачів; має зв’язок “один-до-одного” з сутністю “Користувач”.

3. Академічна група (StudentGroup).

   Поля: id, uuid, код групи.

   Представляє академічну групу, тобто групу студентів; має зв’язок “багато-до-одного” з сутністю “Профіль користувача”.

4. Академічна дисципліна (Course).

   Поля: id, uuid, код дисципліни, назва дисципліни, опис дисципліни.

   Містить інформацію про академічну дисципліну; має зв’язки “багато-до-багатьох” з сутностями “Профіль користувача” та “Академічна група”.

5. Розклад (Timetable).

   Поля: id, uuid, код розкладу, назва розкладу, початкова дата, кінцева дата.

   Використовується для формування графіку подій академічної дисципліни (лекції, практичні заняття, консультації тощо); містить зв’язок “багато-до-одного” з сутністю “Академічна дисципліна”.

6. Академічна подія (Event).

   Поля: id, uuid, назва події, опис події, тип події, час початку, час закінчення.

   Є базовою сутністю для зберігання інформації про подію з дисципліни (лекції, практичні заняття, консультації тощо); містить зв’язки “багато-до-одного“ та “багато-до-багатьох” з сутністю “Профіль користувача” для посилання на викладачів і студентів відповідно, а також “багато-до-багатьох” з сутністю “Розклад”.

7. Завдання (Assignment).

   Поля: id, uuid, назва завдання, опис завдання, час початку, час закінчення, призначена дата.

   Є базовою сутністю для зберігання інформації про завдання з дисципліни; містить зв’язки “багато-до-одного“ та “багато-до-багатьох” з сутністю “Профіль користувача” для посилання на викладачів і студентів відповідно, а також “багато-до-багатьох” з сутністю “Розклад”.

8. Розв’язок (Solution).

   Поля: id, uuid, час та дата створення, коментар.

   Є базовою сутністю для зберігання інформації про відповідь або розв’язок студентом завдання з дисципліни; містить зв’язки “багато-до-одного“ з пов’язаними сутностями “Завдання” та “Профіль користувача”.

9. Оцінка (Grade).

   Поля: id, uuid, бал, час та дата створення, коментар.

   Є базовою сутністю для зберігання інформації про оцінювання відповіді студента; містить зв’язки “один-до-одного” з пов’язаною сутністю “Розв'язок” та “багато-до-одного“ з сутністю “Профіль користувача”.

2.5 Огляд технологій та мов програмування, що були використані для побудови практичної частини

Для створення серверної частини веб-застосунку для організації роботи студентів було вирішено використати такі засоби: платформу Django, а саме фреймворки для розробки веб-рішень Django та Django Rest Framework, систему управління базами даних PostgreSQL; як мову розробки серверної частини обрано Python (основна мова програмування для обраних фреймворків).

Згідно офіційної документації, Django є безкоштовним, високорівневим, Python-based веб-фреймворком з відкритим вихідним кодом, що призначений для швидкої розробки веб-застосунків з потенціалом для масштабування. Розробники даного фреймворку притримувались принципів повторного використання коду, модульності, зменшення залежностей та зв'язності між модулями / компонентами, а також загального для всіх Python-застосунків "Don't Repeat Yourself". Даний фреймворк має значну спільноту прихильників та використовувався для створення таких веб-застосунків як Instagram, Bitbucket, Mozilla тощо [9].

У Django Framework використовується стандартний для більшості монолітних веб-застосунків архітектурний шаблон "Модель – Представлення – Контролер", що полягає в розподілі усього коду застосунку на 3 логічні шари (рис. 2.2):

1) шар даних (Model) – відповідає за бізнес-логіку обробки даних, містить структури даних для представлення наявних в системі сутностей та засоби для взаємодії з базою даних напряму або через бібліотеку ORM;
2) шар представлення (View) – використовується для представлення даних через визначений користувацький інтерфейс, найчастіше – графічний;
3) шар контролерів (Controller) – виступає інтерфейсом між шарами даних та представлень, інтерпретує отримані з користувацького інтерфейсу команди для коректного доступу до даних та повертає одержані дані.

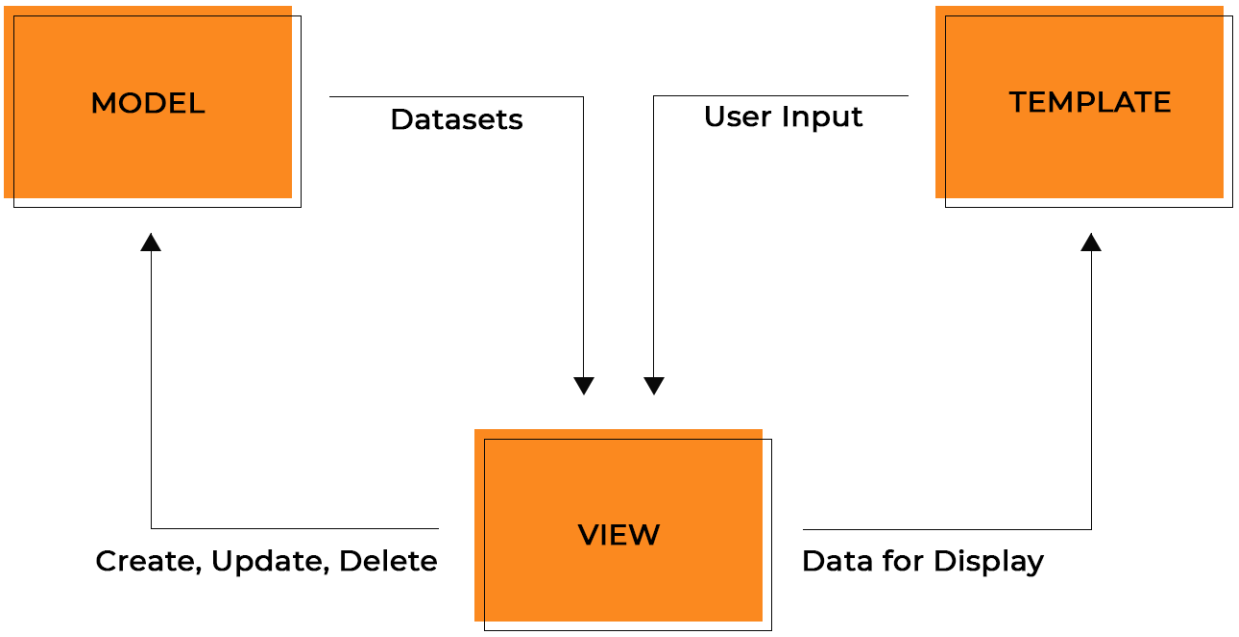


Рис. 2.2. Структура шарів фреймворку Django

До переваг даного фреймворку можна віднести:

- простота розробки застосунків та вивчення фреймворку за рахунок Python;
- принцип модульності та підтримка значної кількості сторонніх бібліотек
- принцип “Batteries included”, тобто постачання з фреймворком загальних для більшості веб-застосунків компонентів, таких як бібліотека ORM, засоби автентифікації та авторизації, панель адміністратора тощо;
- стандартизована структура веб-застосунків, що дозволяє не витрачати багато часу на проектування та зменшує кількість архітектурних помилок;
- низька зв’язність між окремими модулями / компонентами та загальна масштабованість;
- підтримка вбудованих засобів безпеки та механізмів запобігання поширених атак, таких як SQL ін’єкції та міжсайтова підробка запиту.

Недоліками Django є:

- монолітність фреймворку та наявність значної кількості вбудованих модулів не підходять усім проектам;

- використання досить повільної, порівняно з іншими, ORM та відсутність засобів для безпосередньої взаємодії з базою даних через SQL;
- низька загальна продуктивність фреймворку та розроблюваних веб-застосунків порівняно з аналогічними засобами на C#, Java або GoLang.

Так як Django Framework не має вбудованої підтримки веб-API, для створення серверного веб-застосунку було обрано найбільш поширену серед Django-розробників бібліотеку Django Rest.

Django REST – це Python-бібліотека з відкритим вихідним кодом, орієнтована на створення веб-API застосунків за допомогою фреймворку Django. Як і фреймворк Django, на якому дана бібліотека базується, Django REST притримується принципів модульності, повторного використання коду та надання користувачу значної кількості вбудованих засобів. Основною ідеєю Django REST є розділення коду на декілька логічних частин, схоже до підходу MVC в Django:

1) шар моделей – містить структури даних для представлення сутностей веб-застосунку та засоби для взаємодії з ними;
2) шар серіалізаторів – використовується для коректного перетворення даних з формату об’єктів Python у поширені формати передачі й збереження даних, такі як JSON, XML тощо;
3) шар контролерів – обробляє користувацькі запити, передає та приймає дані, перетворює отримані дані у формат об’єктів Python і навпаки за допомогою серіалізатора.

Основними особливостями Django REST є надання розробнику можливості власноруч спроектувати URL-структуру та визначити кінцеві точки, наявність вбудованих засобів перегляду та перевірки API, підтримка поширених протоколів автентифікації та авторизації.

Основними перевагами фреймворку Django REST є:

- простота та гнучкість вихідного коду;
- потужний механізм серіалізації, сумісний як з ORM-орієнтованими джерелами даних, так і без них;
- наявність простих для налаштування валідаторів, автентифікаторів та парсерів;
- наявність вбудованого пагінатора для зручної передачі даних пакетами.

Через те, що Django REST базується на засобах Django, крім переваг даний фреймворк успадковує також і такі недоліки як низька продуктивність, монолітність структури, повільна бібліотека ORM.

PostgreSQL, або Postgres (рис. 2.3) – це безкоштовна об’єктно-реляційна система управління базами даних з відкритим вихідним кодом, заснована на розробленій у 70-х роках в Каліфорнійському університеті Берклі системі Ingres. Хоча прототип системи під назвою Postgres був розроблений ще в 1987 році, перший повноцінний реліз системи відбувся у 1996 році під назвою PostgreSQL та початковою версією 6.0, що підкреслює підтримку SQL [10]. СКБД PostgreSQL має такі особливості:

- сумісність з різними платформами з використанням більшості популярних мов програмування;
- повна підтримка клієнт-серверної архітектури;
- відповідність стандарту ANSI SQL;
- реплікація SSL, орієнтована на журналювання та тригери;
- підтримка управління багатоверсійного паралелізму;
- підтримка формату JSON для взаємодії з іншими сховищами даних (наприклад, NoSQL);
- висока доступність.

Основними перевагами СКБД PostgreSQL є:

- PostgreSQL є безкоштовним програмним забезпеченням з відкритим вихідним кодом;

– підтримка низькорівневого обслуговування та адміністрування як для вбудованого, так і для корпоративного використання PostgreSQL;
– можливість запускати динамічні веб-сайти та веб-застосунки через LAMP (Linux, Apache, MySQL, PHP/Perl/Python) стек;
– стійкість до несправностей завдяки функції журналювання із записом наперед.

До основних недоліків PostgreSQL відносять нижчу продуктивність порівняно з MySQL, відсутність вбудованого графічного інтерфейсу, необхідність знання SQL для створення ендпоінтів.

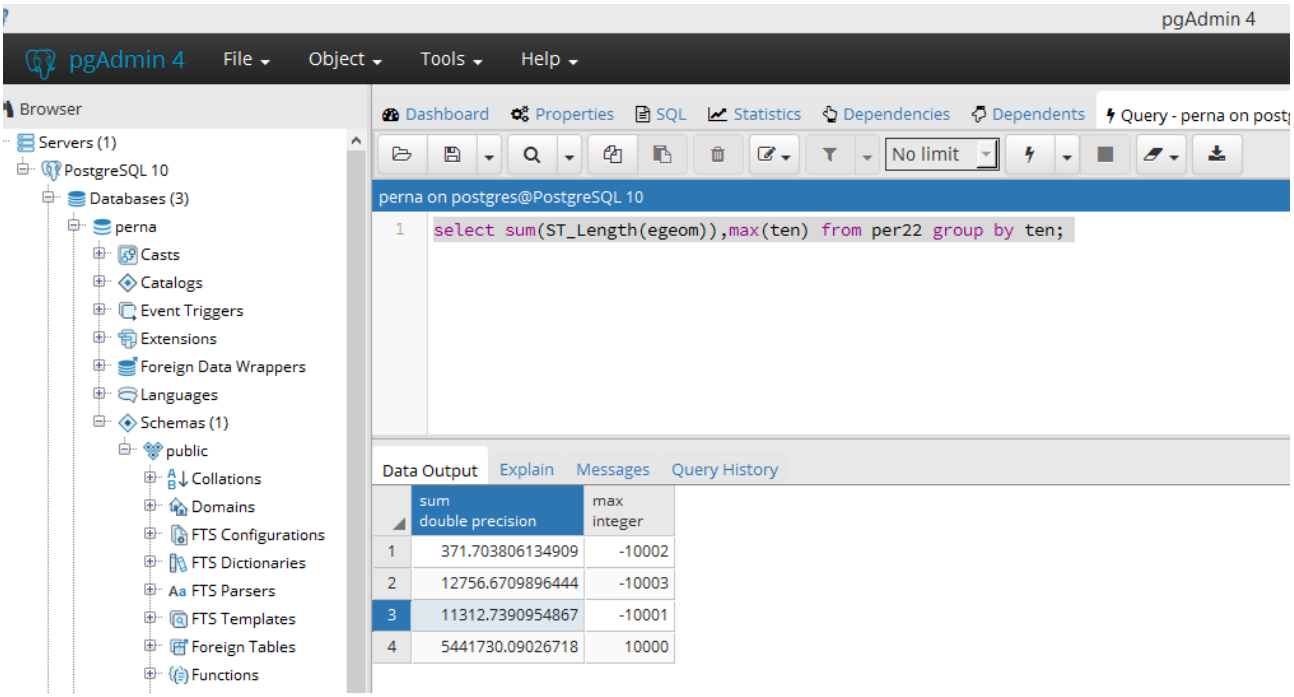


Рис. 2.3. Приклад роботи з PostgreSQL в середовищі pgAdmin

2.6 Огляд архітектури серверного застосунку

Основною одиницею веб-застосунку Django є проект Django, що включає один або декілька Django-застосунків. Додаток Django – це Python-модуль, що містить набір моделей, шаблонів, контролерів, статичних файлів, файлів з URL-адресами та який може використовуватись незалежно від проекту. Загалом кожен проект Django складається з головного модуля, в якому містяться файли налаштування проекта та HTTP-сервера, кореневі URL-адреси тощо, а також з

додаткових модулів, в кожному з яких описується той чи інший компонент системи. На рисунку 2.4 представлено структуру розробленого серверного застосунку.

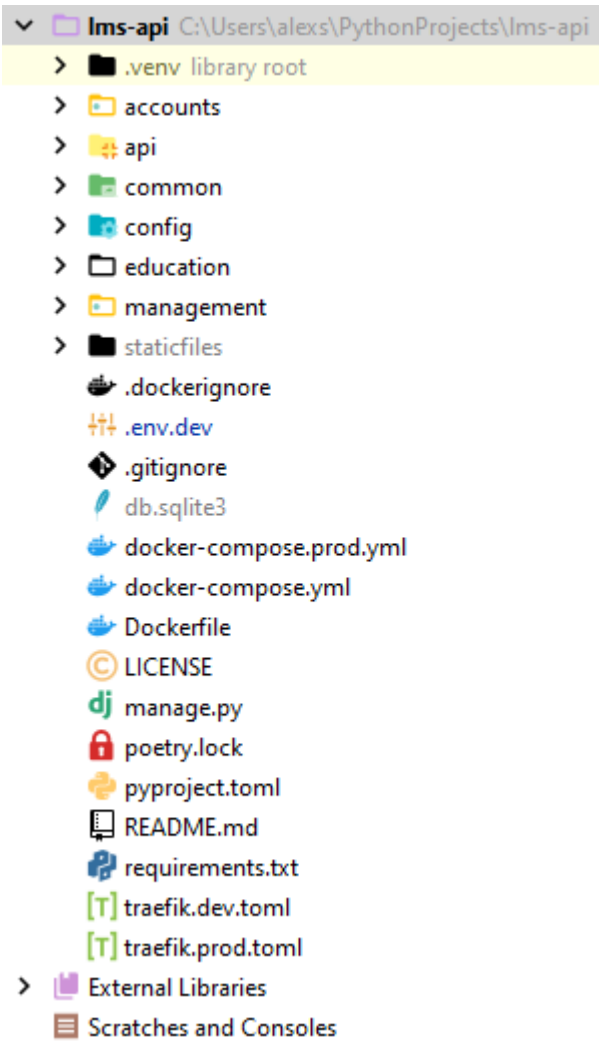


Рис. 2.4. Структура проекта розробленого веб-застосунка

Як видно з рисунку, розроблений застосунок також є модульним та складається з таких Django-додатків:

- модуль config містить налаштування проекту;
- модуль common містить спільні для інших модулів компоненти, наприклад, моделі та серіалізатори;
- модуль accounts містить моделі для керування користувацькими обліковими записами;
- модуль education містить моделі для керування навчальними дисциплінами, графіками подій, розкладом тощо.

– модуль api реалізує бізнес-логіку застосунку та містить серіалізатори, контролери і класи фільтрування запитів.

Також у проекті зберігаються додаткові файли налаштування Nginx-сервера, файли для керування контейнеризацією застосунку через Docker та Docker-compose, файли з налаштуваннями змінних середовища.

Нижче на рисунку 2.5 зображено схему сутностей та залежності між ними (інші діаграми містяться в додатку А). Наприклад, з рисунку видно використання моделей профілів користувача для зберігання записаних на курс студентів та інструкторів курсу, формування списку студентських груп, визначення списку приналежних до створеної події користувачів, створення та перевірки завдань тощо.

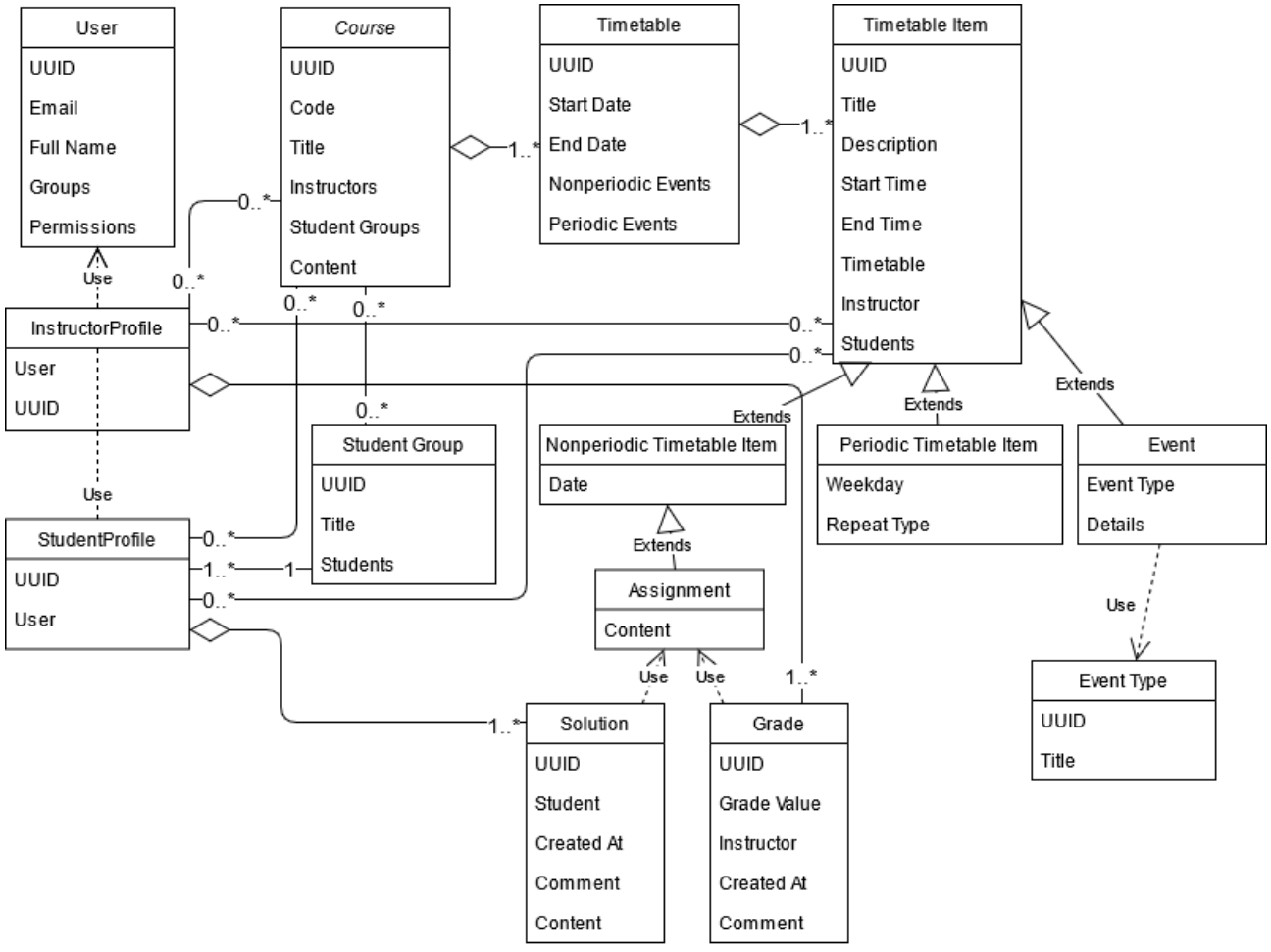


Рис. 2.5. Схема основних сутностей в системі

В модулі common містяться спільні для всіх модулів компонентів елементів, такі як розширення моделі з UUID-полем, модель різних типів контенту (зокрема файлів); на рисунках нижче представлено відповідний код. Клас UUIDFieldMixin успадковується від вбудованого в Django класу для визначення моделей даних Model. Клас Model містить набір методів для керування полями моделі та використовується для реєстрації моделі даних у базі даних через ORM. Як видно з рисунку, доступ до даного класу можна отримати з модуля models, що знаходиться в просторі django.db та містить також класи полів для моделей. В даному випадку було обрано поле UUID, тобто універсальний унікальний ідентифікатор, та задано параметри для моделювання поведінки публічного ключа. За рахунок властивості abstract у вкладеному класі Meta клас UUIDFieldMixin не буде зареєстровано в базі даних, а лише використовуватиметься як міксін, тобто додаватиме певну логіку в класи-спадкоємці.

```
class UUIDFieldMixin(models.Model):
    uuid = models.UUIDField(
        unique=True,
        editable=False,
        default=uuid.uuid4,
        verbose_name='Public identifier'
    )

    class Meta:
        abstract = True
```

Рис. 2.6. Клас спеціального поля з UUID

На рисунку 2.7 зображено класи для представлення контенту в системі – класс Content виступає сховищем, а ContentItem – конкретною одиницею контенту, наприклад файлом; як і в попередньому прикладі, ці класи є абстрактними. В класі Content реалізовано принцип Generic Relation (узагальнене відношення), завдяки якому можна посилатись більш ніж на одну модель через зовнішній ключ. Реалізується цей механізм за допомогою узагальненого зовнішнього ключа (GenericForeignKey) та об’єкта типу контенту.

```python
class Content(UUIDFieldMixin, models.Model):
    """Basic model for course contents."""
    content_type = models.ForeignKey(
        ContentType,
        limit_choices_to={
            'model__in': (
                'text',
                'video',
                'image',
                'file',
            )
        },
        on_delete=models.CASCADE,
    )
    object_id = models.PositiveIntegerField()
    item = GenericForeignKey('content_type', 'object_id')

    class Meta:
        abstract = True


class ContentItem(UUIDFieldMixin, models.Model):
    """Basic model for content items."""
    title = models.CharField(max_length=250)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        abstract = True

    def __str__(self):
        return self.title
```

Рис. 2.7. Моделі для представлення контенту в системі

В модулі accounts реалізовано модель користувача системи на базі вбудованої у Django моделі AbstractUser та розширень з дозволами; створено моделі профілів користувачів та академічної групи; також реалізовано клас-менеджер для керування процесом створення користувачів. В модулі

education реалізовано моделі для представлення дисциплін, графіків подій, власне подій, завдань з дисциплін. Приклад моделі даних та відповідний клас-менеджер представлено нижче.

Клас CustomUser успадковує вбудовану в Django модель AbstractBaseUser, що вже реалізує деякі поля, такі як поле паролю і час останнього входу, та методи, наприклад, для створення паролю, перевірки автентифікований користувач або анонімний тощо. В даному випадку в моделі CustomUser реалізовано поля імені та електронної пошти користувача, дати створення і останньої модифікації, булевих полів активності облікового запису та статусу адміністратора. Крім того визначені обов’язкові поля та поле, що використовується як дані для входу, а також вказано клас-менеджер для створення об’єктів класу CustomUser.

На рисунку 2.9 представлено визначений клас-менеджер класу CustomUser. В Django використовується поділ користувачів на простих та супер-користувачів, що мають усі права за замовчуванням – для створення кожного з цих типів необхідно визначити окремий метод. В класі CustomUserManager реалізовано відповідні методи create_user та create_superuser: в методі create_superuser булеві поля статусу адміністратора та суперкористувача встановлюються як True, а також перевіряється коректність даних полів, після чого викликається метод create_user; в методі create_user перевіряється коректність переданих даних (імені, електронної пошти та паролю користувача, після чого відбувається задання усіх інших полів класу CustomUser та шифрування паролю за допомогою відповідного метода.

```python
class CustomUser(AbstractBaseUser, PermissionsMixin, UUIDFieldMixin):
    """Custom user model with roles and email address is the unique identifier."""
    full_name = models.CharField(max_length=255)
    email = models.EmailField(unique=True, validators=[validate_email])
    created_date = models.DateTimeField(default=timezone.now, editable=False)
    modified_date = models.DateTimeField(default=timezone.now, editable=False)
    is_staff = models.BooleanField(
        _('staff status'),
        default=False,
        help_text=_(
            'Designates whether the user can log into this admin site.'),
    )
    is_active = models.BooleanField(
        _('active'),
        default=True,
        help_text=_(
            'Designates whether this user should be treated as active. '
            'Unselect this instead of deleting accounts.'
        ),
    )
    date_joined = models.DateTimeField(
        _('date joined'), default=timezone.now, editable=False)
    last_login = models.DateTimeField(
        _('last login'), blank=True, null=True, editable=False)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['full_name']

    objects = CustomUserManager()

    def __str__(self):
        return f'{self.full_name} ({self.email})'

    class Meta:
        verbose_name = _('user')
        verbose_name_plural = _('users')
```

Рис. 2.8. Модель для представлення облікового запису користувача

```python
class CustomUserManager(BaseUserManager):
    """Custom manager for creating users of type CustomUser."""

    def create_user(self, full_name, email, password, **extra_fields):
        """..."""
        if not full_name:
            raise ValueError(_("The full name must be set"))
        if not email:
            raise ValueError(_("The email must be set"))
        if not password:
            raise ValueError(_("The password must be set"))

        extra_fields.setdefault('is_superuser', False)
        extra_fields.setdefault('is_active', True)
        email = self.normalize_email(email)
        user = self.model(email=email, full_name=full_name, **extra_fields)
        user.set_password(password)
        user.save()
        return user

    def create_superuser(self, full_name, email, password, **extra_fields):
        """..."""
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)

        if extra_fields.get('is_staff') is not True:
            raise ValueError('Superuser must have is_staff=True.')
        if extra_fields.get('is_superuser') is not True:
            raise ValueError('Superuser must have is_superuser=True.')
        return self.create_user(full_name, email, password, **extra_fields)
```

Рис. 2.9. Клас-менеджер для моделі користувача

Центральною частиною API виступає модуль api, що містить серіалізатори для переведення даних з формату класів Python у формат JSON або XML та навпаки, класи для реалізації логіки фільтрації запитів, контролери для обробки запитів і повернення необхідних даних за допомогою серіалізаторів. Також в даному модулі розміщуються кінцеві точки, пов’язані з API системи. Нижче представлено приклад пов’язаних між собою серіалізатора,

класу фільтрації та контролера. Клас серіалізатора успадковується від вбудованого у фреймворк Django REST серіалізатора HyperlinkedModelSerializer, що надає url-поле з посиланням на поточним об’єкт. У вкладеному класі Meta задається пов’язана модель, тобто клас Course, та поля класу Course, які необхідно перетворювати у формат JSON. Для представлення залежностей з іншими моделями використовується реалізоване в модулі common поле UUIDHyperlinkedRelatedField, що використовує UUID для посилання на необхідні об’єкти.

```
class CourseSerializer(serializers.HyperlinkedModelSerializer):
    instructors = UUIDHyperlinkedRelatedField(
        view_name='instructor-detail',
        queryset=InstructorProfile.objects.all(),
        many=True,
    )

    student_groups = UUIDHyperlinkedRelatedField(
        view_name='student-group-detail',
        queryset=StudentGroup.objects.all(),
        many=True,
    )

    timetables = UUIDHyperlinkedRelatedField(
        view_name='timetable-detail',
        read_only=True,
        many=True,
    )

    contents = CourseContentSerializer(many=True, required=False, )

    class Meta:
        model = Course
        fields = (
            ...
        )
        extra_kwargs = {...}
```

Рис. 2.10. Серіалізатор моделі для представлення дисциплін

Клас для фільтрації запитів успадковується від класу FilterSet, що надає бібліотека django-filter. В класі визначено пов’язану модель та поля, за якими користувач матиме змогу фільтрувати запит. В перевизначеному методі qs реалізовано логіку фільтрації списку об’єктів залежно від типу авторизованого користувача: наприклад, для студента відображатимуться лише ті дисципліни, в інформації яких вказано групу студента. Після перевірки користувача запит фільтрується за допомогою метода filter та повертається як результат метода.

```
class CourseFilter(filters.FilterSet):
    instructors = filters.ModelMultipleChoiceFilter(
        label='Instructors', field_name='uuid', to_field_name='uuid',
        queryset=InstructorProfile.objects.all(),
    )

    student_groups = filters.ModelMultipleChoiceFilter(
        label='Student Groups', field_name='uuid', to_field_name='uuid',
        queryset=StudentGroup.objects.all(),
    )

    class Meta:
        model = Course
        fields = ('code', 'title', 'instructors', 'student_groups',)

    @property
    def qs(self):
        parent = super().qs
        user = getattr(self.request, 'user', None)
        students = StudentProfile.objects.filter(user=user)
        instructors = InstructorProfile.objects.filter(user=user)
        if students:
            student = students.first()
            student_group = student.groups.first()
            return parent.filter(student_groups=student_group)
        if instructors:
            instructor = instructors.first()
            return parent.filter(instructors=instructor)
        return parent
```

Рис. 2.11. Клас для фільтрації дисциплін

```
class CourseViewSet(viewsets.ModelViewSet, MultiSerializerMixin,
                    UUIDLookupFieldMixin, AutoPrefetchViewSetMixin):
    queryset = Course.objects.all()
    serializers = {
        'default': CourseSerializer,
    }
    filterset_class = CourseFilter
    search_fields = ('code', 'title',)
```

Рис. 2.12. CRUD-контролер для роботи з моделлю дисциплін

2.7 Інструкція користувача

Основні характеристики застосунку:

Мова програмування: Python

Середовище розробки: Visual Studio Code та PyCharm Professional

Мінімальні системні вимоги:

- 64-бітна операційна система Windows, macOS або дистрибутив Linux;
- Python 3.6 або вище (в разі запуску через інтерпретатор Python);
- СУБД PostgreSQL 13 (в разі запуску через інтерпретатор Python);
- Docker та Docker-compose (в разі запуску за допомогою Docker);
- 300 КБ доступного простору на жорсткому диску.

Вхідні дані:

- HTTP-запити з адресами кінцевих точок;
- параметри: дані у JSON-форматі.

Вихідні дані:

- HTTP-відповіді з даними у JSON-форматі.

Запустити застосунок можна двома способами: безпосередньо через інтерпретатор Python або за допомогою засобів Docker; в обох випадках необхідно створити файл змінних середовища .env, в якому потрібно вказати налаштування застосунку (наприклад, секретний ключ та дозволені домени) та дані для доступу до бази даних (назву бази даних, ім’я та пароль користувача,

посилання на базу даних у форматі psql://ім’я_користувача:пароль@ім’я_хосту:порт/назва_бази_даних).

Для запуску через Python необхідно мати встановлений інтерпретатор Python версії 3.6 або вище та СУБД PostgreSQL 13. Для запуску застосунку обов’язково необхідно створити відповідну базу даних за допомогою PostgreSQL та вказати її дані у файлі змінних середовища. Застосунок запускається командою “python manage.py runserver”, додатково можна виконати команди “python manage.py collectstatic --no-input” для генерації статичних файлів та “python manage.py migrate” для оновлення схеми бази даних.

За допомогою Docker запустити застосунок можна в двох режимах: з незахищеним з’єднанням та з підтримкою HTTPS/SSL. В першому випадку необхідно створити контейнери за допомогою команди “docker-compose build” (при першому запуску) та запустити їх командою “docker-compose up”; для призупинення роботи контейнерів необхідно ввести команду “docker-compose stop”. В другому випадку у файлі змінних середовища .env необхідно додатково вказати ім’я домену як параметр DOMAIN_NAME та виконувати ті ж команди з додаванням параметру “-f docker-compose.prod.yml”.

Взаємодія з серверним веб-застосунком представляє собою процес надсилання HTTP-запитів та обробки HTTP-відповідей, що повертає сервер. Для роботи з застосунком необхідно мати клієнтський застосунок, здатний працювати за протоколом HTTP та виконувати вищевказані операції. Доступ до API здійснюється за кінцевою точкою /api/; більшість з ресурсів, що надає серверний веб-застосунок, вимагають автентифікації користувача та відповідних прав дозволу. Для автентифікації необхідно надіслати HTTP-запит із даними автентифікації у форматі JSON за кінцевою точкою /api/auth/login/. Отримати схему API або документацію у форматах Swagger та Redoc можна за кінцевими точками /api/docs/schema/, /api/docs/swagger/ та /api/docs/redoc/ відповідно.

2.8 Тестування програмного забезпечення

Переглянути кореневі кінцеві точки, що доступні в API, можна перейшовши за кінцевою точкою /api/. В разі коректного запиту сервер поверне відповідь з кодом 200, що позначає успішне виконання запиту, та списком кінцевих точок URI у JSON-форматі. Результат представлено нижче на рисунку 2.13.

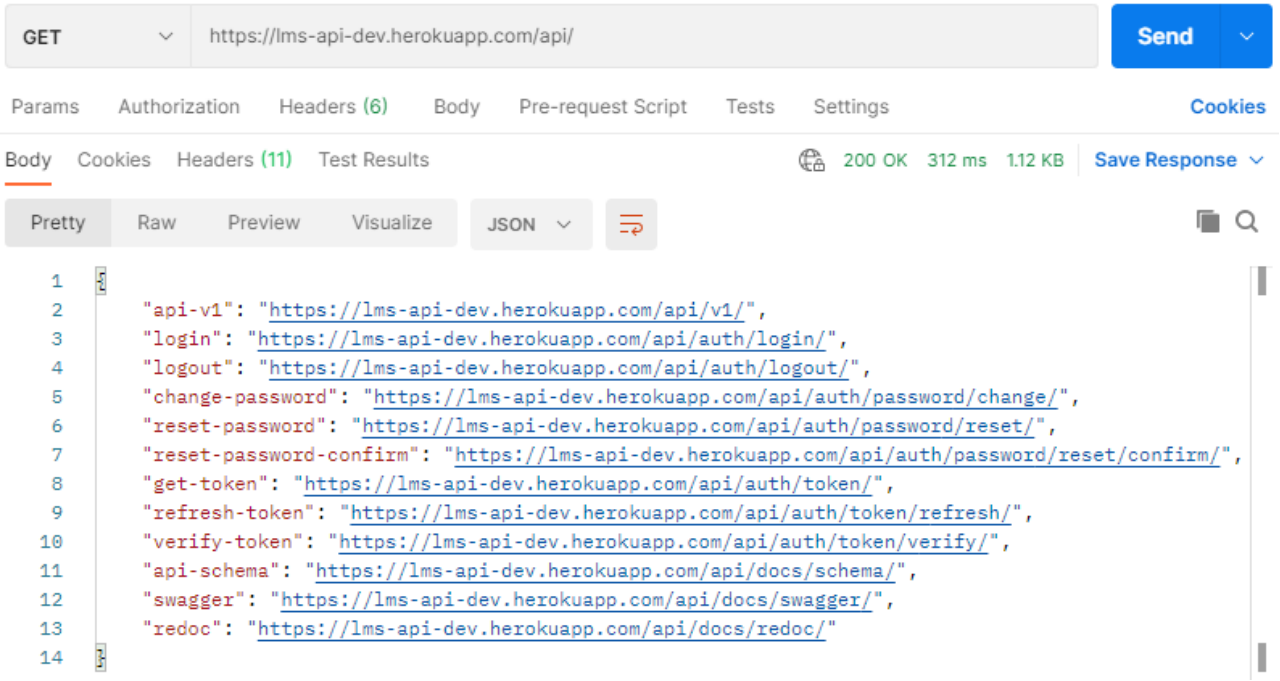


Рис. 2.13. Запит на отримання кореневих кінцевих точок API

Для використання функціональності застосунку необхідно автентифікуватись, тобто увійти в користувацький обліковий запис, та авторизуватись для перевірки прав доступу. Авторизація в Django відбувається за допомогою механізма JSON Web Token: при запиті на автентифікацію сервер надсилає клієнту необхідні токени, які клієнт використовує для підтвердження облікового запису при кожному наступному запиті; вхід в обліковий запис відбувається через кінцеву точку /api/auth/login, вихід з облікового запису – /api/auth/logout. Якщо було надано коректні дані для входу, застосунок поверне ключ авторизації у відповідь. Приклад автентифікації представлено на рисунку 2.14.

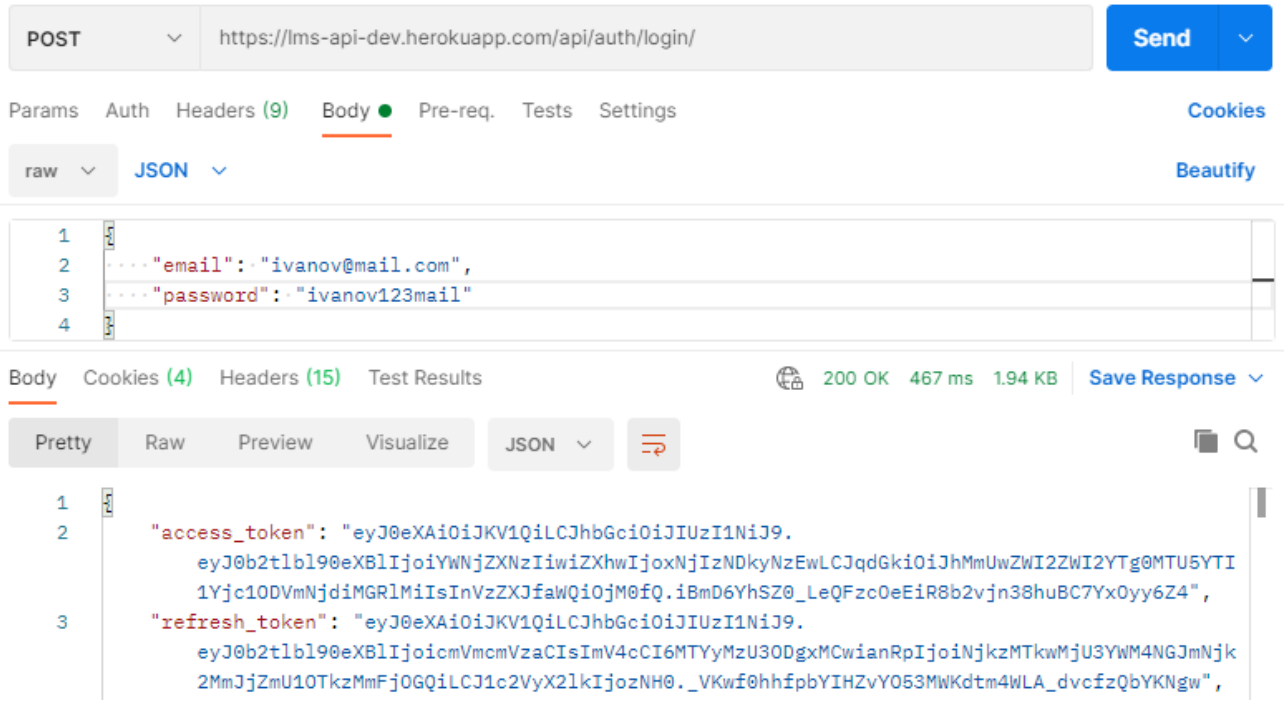


Рис. 2.14. Приклад автентифікації користувача

В разі наявності відповідних прав, за кінцевою точкою /api/{версія_api}/{ресурс} можна переглядати список створених об’єктів обраного ресурсу та інформацію про кожен з них за UUID, додавати нові об’єкти, редагувати та видаляти вже існуючі. Нижче на рисунках 2.15 та 2.16 представлено створення нової дисципліни та перегляду списку дисциплін.

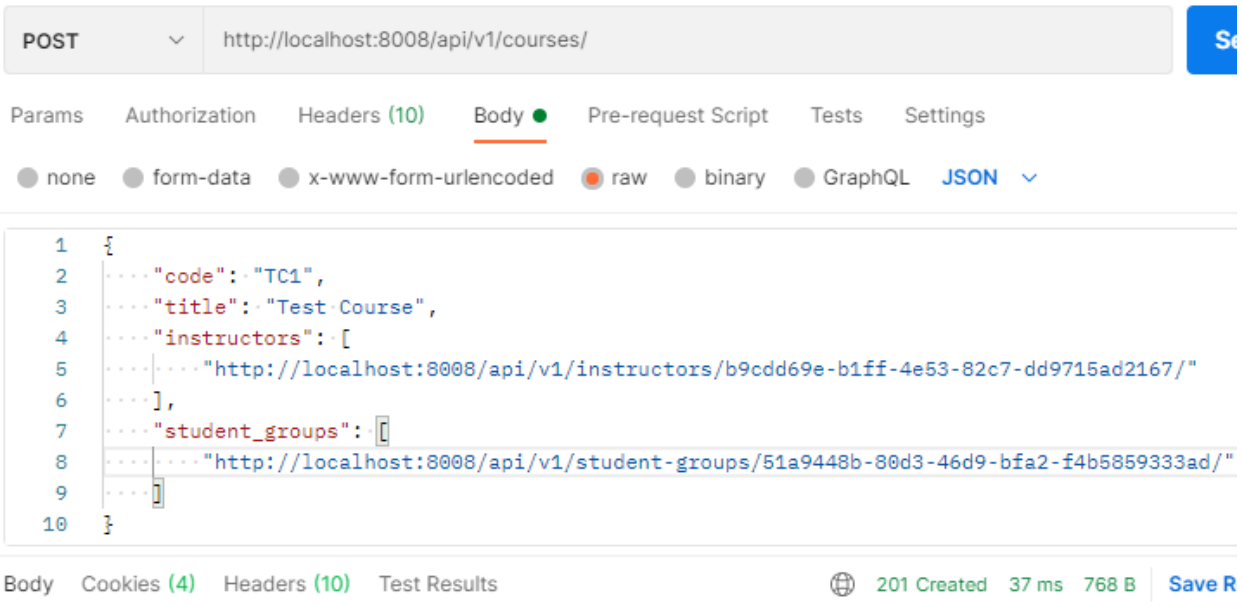


Рисунок 2.15. Створення нової дисципліни

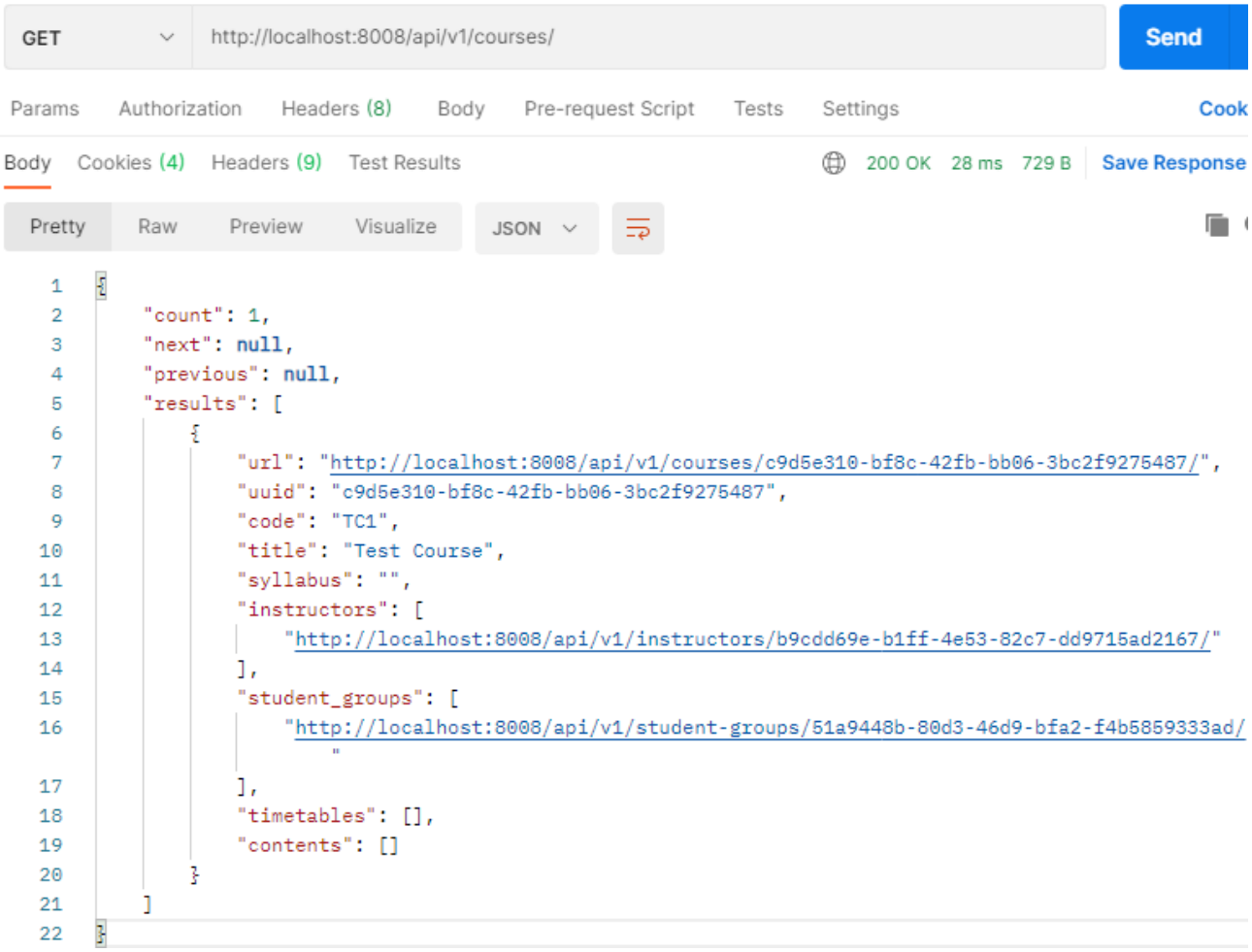


Рисунок 2.16. Перегляд списку дисциплін

Для зручного представлення усіх кінцевих точок API в застосунок було інтегровано бібліотеки для генерації API-схеми Swagger, Redoc та BrowsableAPI. Приклади відображення кінцевих точок в згенерованих схемах представлено на рисунках 2.17, 2.18 та 2.19.

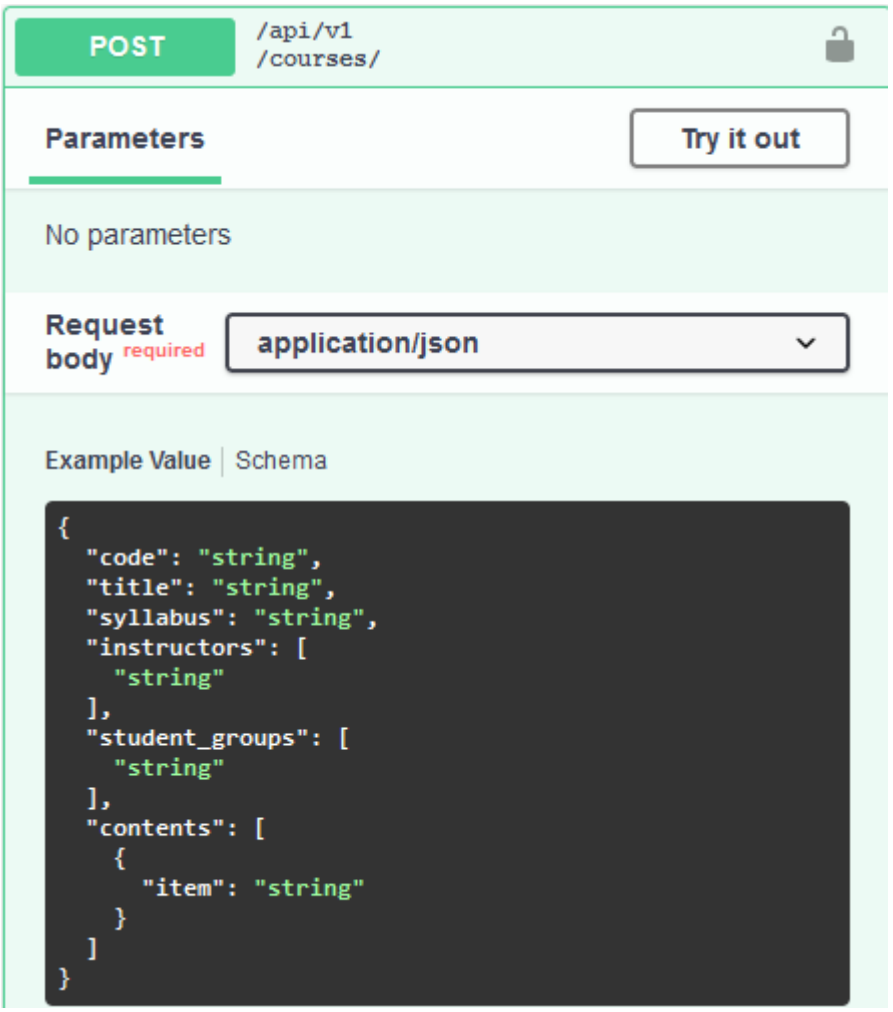


Рисунок 2.17. Приклад кінцевої точки в Swagger

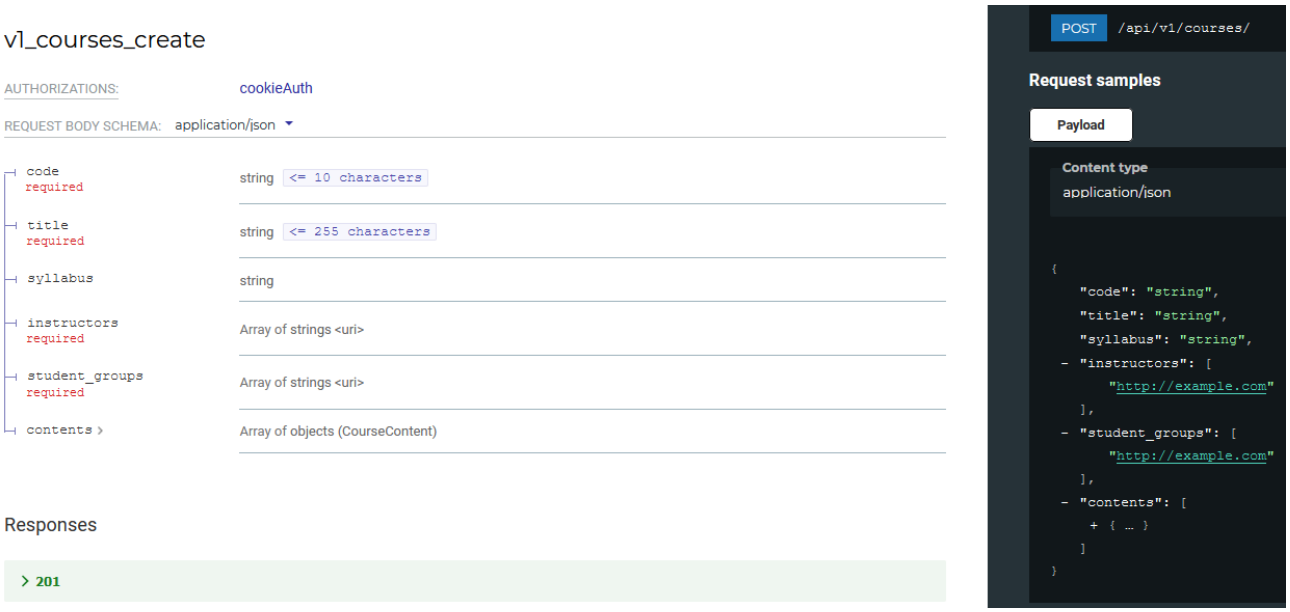


Рисунок 2.18. Приклад кінцевої точки в Redoc

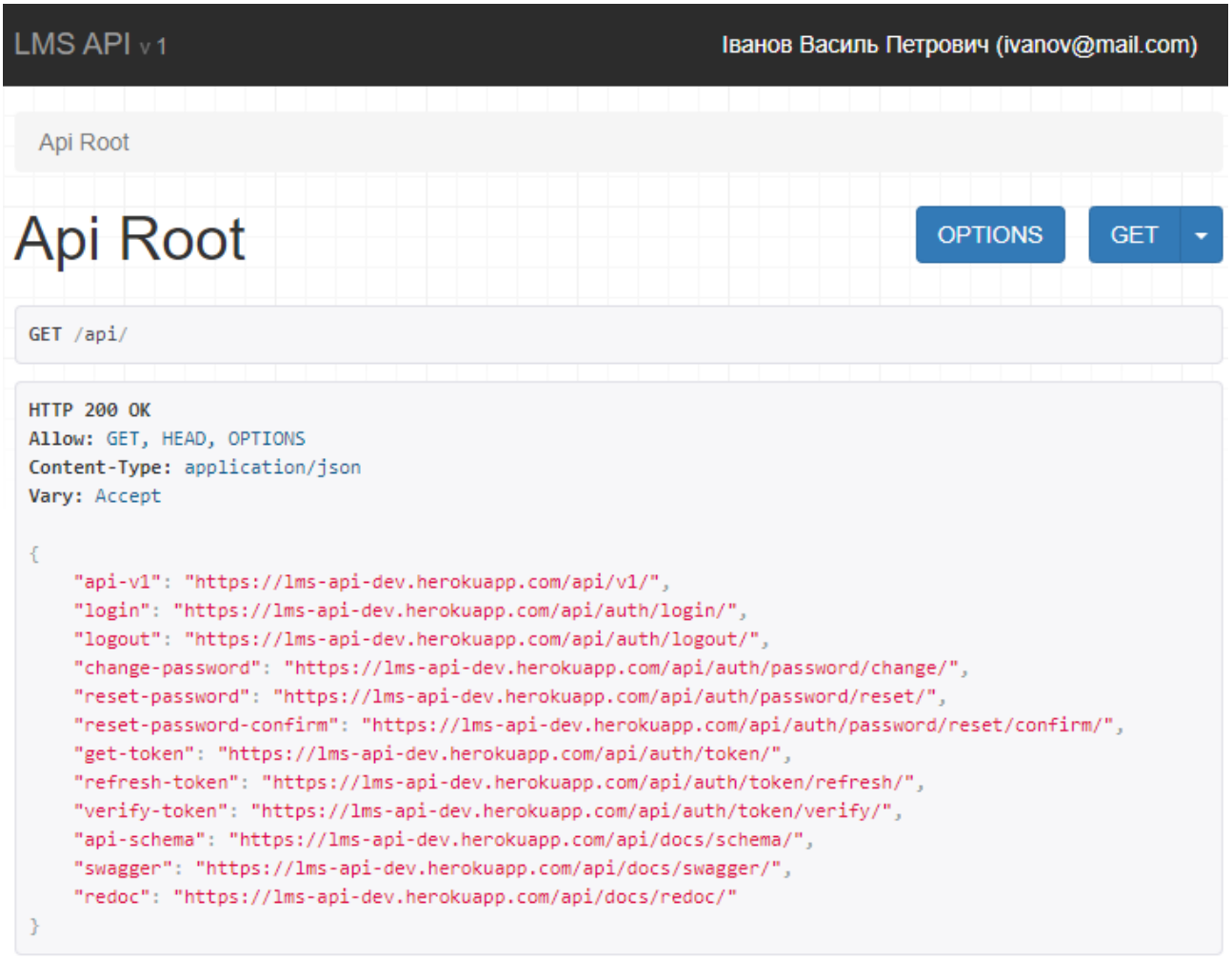


Рисунок 2.19. Приклад кореневої кінцевої точки в BrowsableAPI

2.9 Впровадження розробленої системи

Розроблений серверний веб-застосунок був успішно застосований під час практики студентів 3-го курсу та використаний для досліджень щодо проведення практики студентів кафедри ПСТ [11]. Студентами було запропоновано таке рішення по режимам функціонування та діагностуванню роботи системи:

– введення двох режим роботи: основного режиму, в якому всі підсистеми виконують свої основні функції, та профілактичного, в якому одна або всі підсистеми не виконують своїх функцій;
– в основному режимі система забезпечує роботу користувачів в режимі 24 години на добу, 7 днів на тиждень та виконання своїх функцій по збору,

обробці, завантаженню та зберіганню даних, а також надання звітності за показниками;

– у профілактичному режимі система забезпечує можливість проведення технічного обслуговування, модернізації апаратно-програмного комплексу та усунення аварійних ситуацій.

Також авторами роботи було надано рекомендації щодо вдосконалення системи та розширення функціональності серверного веб-застосунку, деякі з яких було реалізовано після проведення дослідження. У дослідження було виділено такі можливі доповнення:

– аналітика відвідуваності студентів;
– аналітика успішності студентів (частково реалізовано через систему завдань);
– проведення контролю знань (реалізовано у вигляді системи завдань);
– прогнозування середнього балу;
– розклад екзаменаційної сесії (частково реалізовано через систему подій);
– зворотній зв’язок з адміністрацією навчального закладу;
– отримання додаткових балів.

2.10 Висновки до розділу

У другому розділі було розглянуто технології, що використовувалися для створення системи. Було обрано мову програмування Python, фреймворки Django та Django REST; як СУБД було обрано PostgreSQL. Також, було наведено архітектурну схему розроблюваної системи, описано основні ролі та моделі в системі. Далі було більш детально розглянуто основні складові серверної частини та наведено приклади окремих компонентів, а також проведено тестування функціональності застосунку.

ВИСНОВКИ

В ході виконання дипломної роботи було розібрано достатню кількість інформації на такі теми: концепція дистанційного навчання та систем керування навчанням, типова функціональність систем керування навчанням, розробка веб-додатків за допомогою фреймворків Django та Django REST, шаблони проектування для створення веб-API та роботи з базою даних. Для реалізації серверного веб-API застосунку для організації роботи студентів було використано мову програмування Python, фреймворки Django та Django REST, СУБД PostgreSQL, засоби контейнеризації Docker та Docker-compose.

Як результат, було розроблено макет програмного забезпечення, що дозволяє вирішити такі проблеми:

1. Автоматизоване формування розкладу лекцій, семінарів та інших подій з дисциплін.
2. Відображення релевантного для студентів та викладачів розкладу без необхідності шукати потрібну інформацію у загальному Excel-файлі.
3. Об'єднання навчального матеріалу та завдань з дисциплін в одному місці для зручного доступу до них. Створення завдань з дисциплін з можливістю прикріплення файлів та їх перевірка.

Розроблений веб-застосунок надає такі можливості з управління обліковими записами користувачів системи, ролями та правами доступу; керування розкладом та інформацією про дисципліни; створення та перевірки завдань з дисциплін. Усім користувачам системи призначаються ролі з відповідними правами доступу та можливостями.

Перспективою даної роботи є імплементація додаткових засобів комунікації студентів та викладачів, як от real-time чати та відеоконференції, ведення документації навчального закладу, створення автоматизованих засобів перевірки коду та завдань, аналітика відвідуваності та успішності студентів, зворотній зв’язок з адміністрацією навчального закладу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Traditional education – Wikipedia, the free encyclopedia. – [Електронний ресурс]. – Режим доступу: en.wikipedia.org/wiki/Traditional_education

2. Distance education – Wikipedia, the free encyclopedia. – [Електронний ресурс]. – Режим доступу: en.wikipedia.org/wiki/Distance_education

3. Shubhangi K. Jadhav Virtual Classroom. The Center for Development of Advanced Computing, Sterling Institute of Technology and Management. Mumbai, 2010.

4. Watson WR., Watson SL., An Argument for Clarity: What are Learning Management Systems, What are They Not, and What Should They Become? // TechTrends, 2007. – 34 p.

5. Learning management system – Wikipedia, the free encyclopedia. – [Електронний ресурс]. – Режим доступу: en.wikipedia.org/wiki/Learning_management_system

6. WebCT – Wikipedia, the free encyclopedia. – [Електронний ресурс]. – Режим доступу: en.wikipedia.org/wiki/WebCT

7. Moodle – Wikipedia, the free encyclopedia. – [Електронний ресурс]. – Режим доступу: en.wikipedia.org/wiki/Moodle

8. Assessing the value of e-Learning Management System for Higher Education [Електронний ресурс]. – Режим доступу: www.slideshare.net/kaustav2012/assessing-the-value-of-e-learning-management-systems-for-higher-education

9. Django (web framework) – Wikipedia, the free encyclopedia. – [Електронний ресурс]. – Режим доступу: en.wikipedia.org/wiki/Django_(web_framework)

10. PostgreSQL – Wikipedia, the free encyclopedia. – [Електронний ресурс]. – Режим доступу: en.wikipedia.org/wiki/PostgreSQL

11. Білодід М.М., Білоусова Н.О., Іванчук В.С., Коржова Є.В., Прищепа В.В., Танасійчук Д.О., Хлопенко О.О. Проведення практики студентів кафедри ПСТ, погляд сучасних студентів: аналітичне дослідження, 2021.

ДОДАТОК А

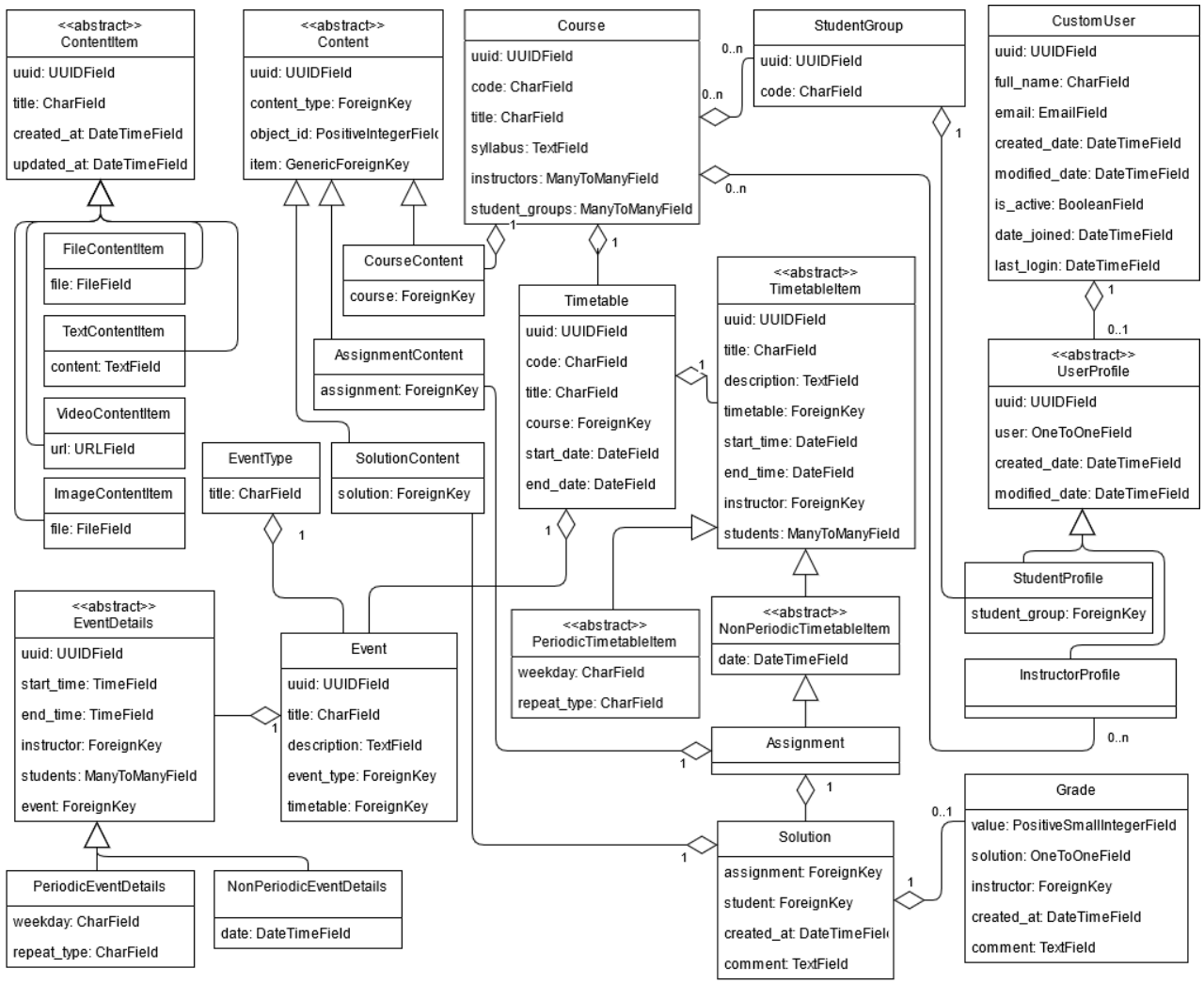


Рис. А.1. Діаграма класів, що представляють моделі даних

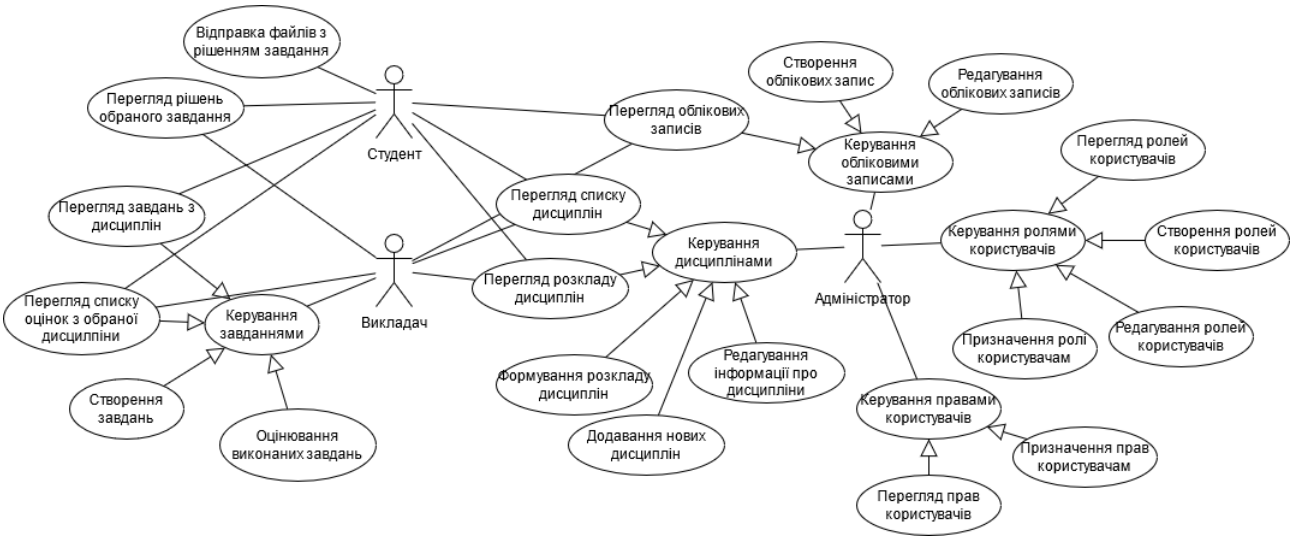


Рис. А.2. Діаграма можливостей користувачів системи

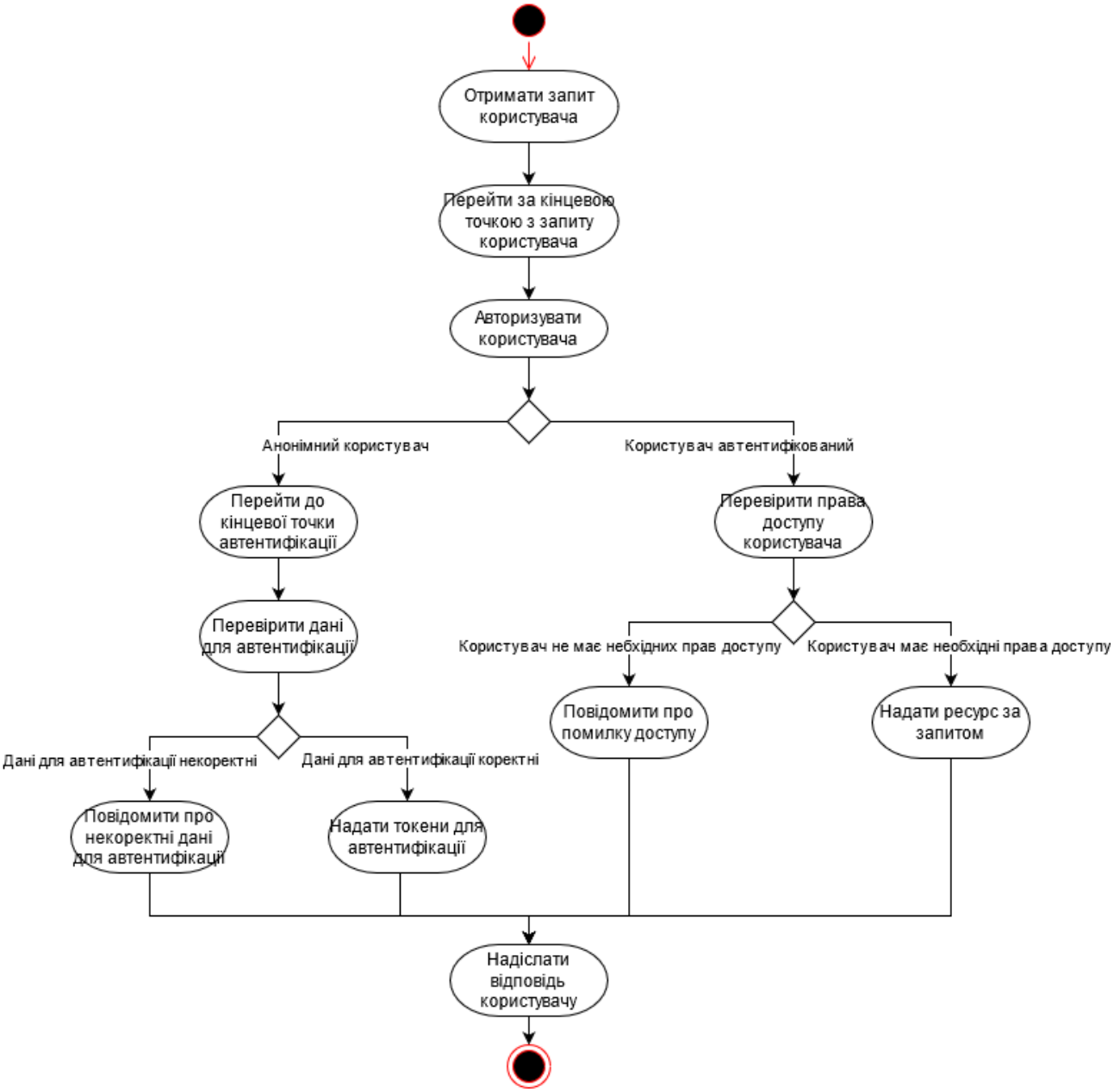


Рис. А.3. Діаграма взаємодії користувача з сервером через клієнт

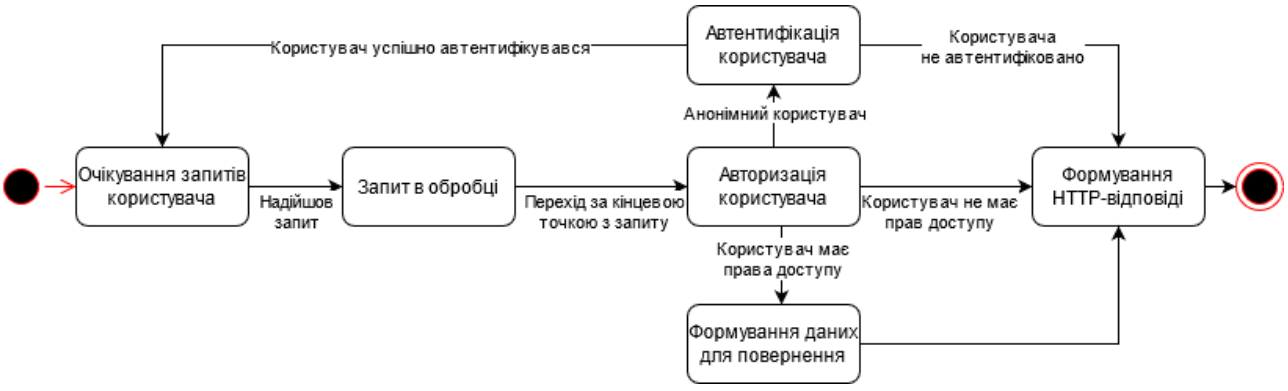


Рис. А.4. Діаграма станів під час взаємодії користувача з сервером

ДОДАТОК Б

Приклад класів для представлення моделей даних

```
from django.db import models
from django.utils.translation import gettext_lazy as _

from accounts.models import InstructorProfile, StudentProfile, StudentGroup
from common.models import Content, UUIDFieldMixin


class Course(UUIDFieldMixin, models.Model):
	"""Represent an academic course."""
	code = models.CharField(max_length=10, unique=True)
	title = models.CharField(max_length=255)
	syllabus = models.TextField(blank=True)
	instructors = models.ManyToManyField(
	InstructorProfile,
	related_name='instructed_courses',
	)
	student_groups = models.ManyToManyField(
	StudentGroup,
	related_name='joined_courses',
	)

	def __str__(self):
	return f'{self.code} - {self.title}'


class Timetable(UUIDFieldMixin, models.Model):
	"""Represent an event timetable related to specific course."""
	code = models.CharField(max_length=10, unique=True)
	title = models.CharField(max_length=255, blank=True)
	course = models.ForeignKey(
	Course,
	on_delete=models.CASCADE,
	related_name='timetables',
	)
	start_date = models.DateField(verbose_name='Course start date')
	end_date = models.DateField(verbose_name='Course end date')

	def __str__(self):
	return f'{self.title} ({self.course.code} course)'


class TimetableItem(UUIDFieldMixin, models.Model):
	"""Basic model for course events."""
	title = models.CharField(max_length=255)
	description = models.TextField(blank=True)
	timetable = models.ForeignKey(
	Timetable,
	on_delete=models.CASCADE,
	related_name='%(class)ss',
```

```
        )
        start_time = models.TimeField(verbose_name='Course event start time')
        end_time = models.TimeField(verbose_name='Course event end time')
        instructor = models.ForeignKey(
        InstructorProfile,
        on_delete=models.SET_NULL,
        null=True,
        related_name='%(class)ss',
        )
        students = models.ManyToManyField(StudentProfile, related_name='%(class)ss')

        class Meta:
        abstract = True


class PeriodicTimetableItem(TimetableItem):
        """Basic model for periodic timetable items."""

        class WeekDay(models.TextChoices):
        Monday = 'MO', _('Monday')
        Tuesday = 'TU', _('Tuesday')
        Wednesday = 'WE', _('Wednesday')
        Thursday = 'TH', _('Thursday')
        Friday = 'FR', _('Friday')
        Saturday = 'SA', _('Saturday')
        Sunday = 'SU', _('Sunday')

        class RepeatType(models.TextChoices):
        Weekly = 'W', _('Weekly')
        Even = 'E', _('Even')
        Odd = 'O', _('Odd')

        weekday = models.CharField(
        choices=WeekDay.choices,
        default=WeekDay.Monday,
        max_length=2,
        )
        repeat_type = models.CharField(
        choices=RepeatType.choices,
        default=RepeatType.Weekly,
        max_length=1,
        )

        class Meta:
        abstract = True


class NonPeriodicTimetableItem(TimetableItem):
        """Basic model for non-periodic timetable items."""
        date = models.DateTimeField()

        class Meta:
        abstract = True
```

```
class Assignment(NonPeriodicTimetableItem):
	"""Represent a course assignment related to event."""

	def __str__(self) -> str:
	return self.title


class Solution(UUIDFieldMixin, models.Model):
	"""Represent student's solution to course assignment."""
	assignment = models.ForeignKey(
	Assignment,
	related_name='solutions',
	on_delete=models.CASCADE,
	)
	student = models.ForeignKey(
	StudentProfile,
	related_name='solutions',
	on_delete=models.SET_NULL,
	null=True,
	)
	created_at = models.DateTimeField(auto_now_add=True)
	comment = models.TextField(blank=True)


class Grade(UUIDFieldMixin, models.Model):
	"""Represent a response to the student's solution."""
	value = models.PositiveSmallIntegerField(default=0)
	solution = models.OneToOneField(
	Solution,
	related_name='grade',
	on_delete=models.CASCADE,
	)
	instructor = models.ForeignKey(
	InstructorProfile,
	related_name='grades',
	on_delete=models.SET_NULL,
	null=True,
	)
	created_at = models.DateTimeField(auto_now_add=True)
	comment = models.TextField(blank=True)


class CourseContent(Content):
	course = models.ForeignKey(
	Course,
	on_delete=models.CASCADE,
	related_name='contents',
	)


class AssignmentContent(Content):
	assignment = models.ForeignKey(
	Assignment,
```

```
        on_delete=models.CASCADE,
        related_name='contents',
        )


class SolutionContent(Content):
        solution = models.ForeignKey(
        Solution,
        on_delete=models.CASCADE,
        related_name='contents',
        )
```

ДОДАТОК В

Приклад класів для фільтрації запитів

```
from django.db.models import Q
from django_filters import rest_framework as filters

from accounts.models import InstructorProfile, StudentProfile
from education.models import (
    Assignment,
    Course,
    Event,
    EventType,
    Grade,
    NonPeriodicEventDetails,
    PeriodicEventDetails,
    Solution,
    StudentGroup,
    Timetable,
)


class CourseFilter(filters.FilterSet):
    instructors = filters.ModelMultipleChoiceFilter(
        label='Instructors',
        field_name='uuid',
        to_field_name='uuid',
        queryset=InstructorProfile.objects.all(),
    )

    student_groups = filters.ModelMultipleChoiceFilter(
        label='Student Groups',
        field_name='uuid',
        to_field_name='uuid',
        queryset=StudentGroup.objects.all(),
    )

    class Meta:
        model = Course
        fields = ('code', 'title', 'instructors', 'student_groups',)

    @property
    def qs(self):
        parent = super().qs
        user = getattr(self.request, 'user', None)

        students = StudentProfile.objects.filter(user=user)
        instructors = InstructorProfile.objects.filter(user=user)

        if students:
            student = students.first()
            student_group = student.groups.first()
            return parent.filter(student_groups=student_group)
        if instructors:
            instructor = instructors.first()
            return parent.filter(instructors=instructor)
        return parent


class TimetableFilter(filters.FilterSet):
    course = filters.ModelChoiceFilter(
```

```
        label='Course',
        field_name='uuid',
        to_field_name='uuid',
        queryset=Course.objects.all(),
    )

    class Meta:
        model = Timetable
        fields = (
            'code', 'title', 'course', 'course__code', 'course__title',
            'start_date', 'end_date',
        )


class TimetableItemFilter(filters.FilterSet):
    timetable = filters.ModelChoiceFilter(
        label='Timetable',
        field_name='uuid',
        to_field_name='uuid',
        queryset=Timetable.objects.all(),
    )

    instructor = filters.ModelChoiceFilter(
        label='Instructor',
        field_name='uuid',
        to_field_name='uuid',
        queryset=InstructorProfile.objects.all(),
    )

    students = filters.ModelMultipleChoiceFilter(
        label='Students',
        field_name='uuid',
        to_field_name='uuid',
        queryset=StudentProfile.objects.all(),
    )

    class Meta:
        abstract = True
        fields = (
            'title', 'timetable', 'instructor', 'students', 'start_time',
            'end_time',
        )

    @property
    def qs(self):
        parent = super().qs
        user = getattr(self.request, 'user', None)

        students = StudentProfile.objects.filter(user=user)
        instructors = InstructorProfile.objects.filter(user=user)

        if students:
            student = students.first()
            return parent.filter(students=student)
        if instructors:
            instructor = instructors.first()
            return parent.filter(instructors=instructor)
        return parent


class PeriodicTimetableItemFilter(TimetableItemFilter):
    class Meta:
```

```
        abstract = True
        fields = TimetableItemFilter.Meta.fields + ('weekday', 'repeat_type',)


class NonPeriodicTimetableItemFilter(TimetableItemFilter):
    class Meta:
        abstract = True
        fields = TimetableItemFilter.Meta.fields + ('date',)


class AssignmentFilter(NonPeriodicTimetableItemFilter):
    class Meta:
        model = Assignment
        fields = NonPeriodicTimetableItemFilter.Meta.fields + ()


class SolutionFilter(filters.FilterSet):
    assignment = filters.ModelChoiceFilter(
        label='Assignment',
        field_name='uuid',
        to_field_name='uuid',
        queryset=Assignment.objects.all(),
    )

    student = filters.ModelChoiceFilter(
        label='Student',
        field_name='uuid',
        to_field_name='uuid',
        queryset=StudentProfile.objects.all(),
    )

    class Meta:
        model = Solution
        fields = ('assignment', 'student', 'created_at',)

    @property
    def qs(self):
        parent = super().qs
        user = getattr(self.request, 'user', None)

        students = StudentProfile.objects.filter(user=user)
        instructors = InstructorProfile.objects.filter(user=user)

        if students:
            student = students.first()
            return parent.filter(student=student)
        if instructors:
            instructor = instructors.first()
            return parent.filter(assignment__instructor=instructor)
        return parent


class GradeFilter(filters.FilterSet):
    solution = filters.ModelChoiceFilter(
        label='Solution',
        field_name='uuid',
        to_field_name='uuid',
        queryset=Solution.objects.all(),
    )

    instructor = filters.ModelChoiceFilter(
        label='Instructor',
```

```
    field_name='uuid',
    to_field_name='uuid',
    queryset=InstructorProfile.objects.all(),
)

class Meta:
    model = Grade
    fields = ('value', 'solution', 'instructor', 'created_at',)

@property
def qs(self):
    parent = super().qs
    user = getattr(self.request, 'user', None)

    students = StudentProfile.objects.filter(user=user)
    instructors = InstructorProfile.objects.filter(user=user)

    if students:
        student = students.first()
        return parent.filter(solution__student=student)
    if instructors:
        instructor = instructors.first()
        return parent.filter(instructor=instructor)
    return parent
```

ДОДАТОК Д

Приклад класів для серіалізації даних

```
from django_auto_prefetching import AutoPrefetchViewSetMixin
from rest_framework import viewsets

from api.common.views import MultiSerializerMixin, UUIDLookupFieldMixin
from api.education.filters import AssignmentFilter, CourseFilter, EventFilter,
GradeFilter, SolutionFilter, \
       TimetableFilter
from api.education.serializers import (
       AssignmentSerializer,
       EventSerializer,
       CourseSerializer,
       EventTypeSerializer,
       GradeSerializer,
       SolutionSerializer,
       TimetableSerializer,
)
from education.models import (
       Assignment,
       Course,
       Event,
       Grade,
       Solution,
       Timetable,
       EventType,
)


class CourseViewSet(viewsets.ModelViewSet, MultiSerializerMixin,
UUIDLookupFieldMixin, AutoPrefetchViewSetMixin):
       queryset = Course.objects.all()
       serializers = {
       'default': CourseSerializer,
       }
       filterset_class = CourseFilter
       search_fields = ('code', 'title',)


class TimetableViewSet(viewsets.ModelViewSet, MultiSerializerMixin,
UUIDLookupFieldMixin, AutoPrefetchViewSetMixin):
       queryset = Timetable.objects.all()
       serializers = {
       'default': TimetableSerializer,
       }
       filterset_class = TimetableFilter
       search_fields = ('code', 'title',)


class AssignmentViewSet(viewsets.ModelViewSet, MultiSerializerMixin,
UUIDLookupFieldMixin):
```

```
        queryset = Assignment.objects.all()
        serializers = {
        'default': AssignmentSerializer,
        }
        filterset_class = AssignmentFilter
        search_fields = ('title',)


class SolutionViewSet(viewsets.ModelViewSet, MultiSerializerMixin,
UUIDLookupFieldMixin):
        queryset = Solution.objects.all()
        serializers = {
        'default': SolutionSerializer,
        }
        filterset_class = SolutionFilter


class GradeViewSet(viewsets.ModelViewSet, MultiSerializerMixin,
UUIDLookupFieldMixin):
        queryset = Grade.objects.all()
        serializers = {
        'default': GradeSerializer,
        }
        filterset_class = GradeFilter


class EventViewSet(viewsets.ModelViewSet, MultiSerializerMixin, UUIDLookupFieldMixin,
AutoPrefetchViewSetMixin):
        queryset = Event.objects.all()
        serializers = {
        'default': EventSerializer,
        }
        filterset_class = EventFilter
        search_fields = ('title',)


class EventTypeViewSet(viewsets.ModelViewSet, MultiSerializerMixin,
UUIDLookupFieldMixin, AutoPrefetchViewSetMixin):
        queryset = EventType.objects.all()
        serializers = {
        'default': EventTypeSerializer,
        }
        filterset_fields = ('title',)
        search_fields = ('title',)
```

ДОДАТОК Е

Software Architecture Document

Server-side software (SsS) for organizing the work of students at the academic department (AD) under conditions of distance learning (DL)

Oleksandr Sviatetskyi

Version 1.2

May 2021

Revision History

NOTE: *The revision history cycle begins once changes or enhancements are requested after the initial version of the Software Architecture Document has been completed.*

Date	Version	Description	Author
25/05/2021	1.0	Initial version of SAD	Oleksandr Sviatetskyi
27/05/2021	1.1	Some changes after first review	Oleksandr Sviatetskyi
28/05/2021	1.2	Ready for the next review	Oleksandr Sviatetskyi

Table of Contents

Software Architecture Document

1. Introduction

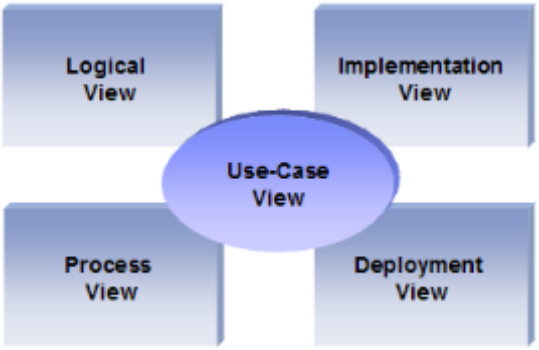
This document provides a high level overview and explains the whole architecture of Server-side software for organizing the work of students at the academic department under conditions of distance learning. It explains how an online user will be able to create and maintain software development process definitions and includes the underlying architecture of the tool.

The document provides a high-level description of the goals of the architecture, the use cases supported by the system and architectural styles and components that have been selected to best achieve the use cases. This framework then allows for the development of the design criteria and documents that define the technical and domain standards in detail.

1.1. Purpose

The Software Architecture Document (SAD) provides a comprehensive architectural overview of Server-side software for organizing the work of students at the academic department under conditions of distance learning. It presents a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

In order to depict the software as accurately as possible, the structure of this document is based on the “4+1” model view of architecture [KRU41].



The “4+1” View Model allows various stakeholders to find what they need in the software architecture.

1.2. Scope

The scope of this SAD is to depict the architecture of the Server-side software for organizing the work of students at the academic department under conditions of distance learning created by the student of Taras Shevchenko National University of Kyiv.

This document describes the aspects of Server-side software for organizing the work of students at the academic department under conditions of distance learning design that are considered to be architecturally significant; that is, those elements and behaviors that are most fundamental for guiding the construction Server-side software for organizing the work of students at the academic department under conditions of distance learning and for understanding this project as a whole. Stakeholders who require a technical understanding of Server-side software for organizing the work of students at the academic department under

conditions of distance learning are encouraged to start by reading this document, then reviewing the Server-side software for organizing the work of students at the academic department under conditions of distance learning UML model, and then by reviewing the source code.

1.3. Definitions, Acronyms, and Abbreviations

- **SsS** – Server-side software
- **AD** – Academic department
- **DL** – Distance learning
- **API** – Application Programming Interface
- **HTTP** – HyperText Transfer Protocol
- **REST** – Representational State Transfer
- **ORM** – Object Relational Mapping
- **RDBMS** – Relational Database Management System
- **SAD -** Software Architecture Document
- **RUP -** Rational Unified Process
- **UML** – Unified Modeling Language
- **Instructor** – User who manages assignments and grades
- **Student** – User who creates assignment solutions
- **Admin** – User who manages courses, timetables, accounts, user roles and permissions

1.4. References

[SRS]: Software Requirements Specification

[MedBiquitous]: Sample SAD,
http://medbiq.org/std_specs/techguidelines/softwarearchitecture.pdf

[KRU41]: The “4+1” view model of software architecture, Philippe Kruchten, November 1995,
http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Pbk4p1.pdf

1.5. Overview

In order to fully document all the aspects of the architecture, the Software Architecture Document contains the following subsections.

Section 2: describes the use of each view

Section 3: describes the architectural constraints of the system

Section 4: describes the functional requirements with a significant impact on the architecture

Section 5: describes the most important use-case realization.

Section 6: describes design’s concurrency aspects

Section 7: describes how the system will be deployed.

Section 8: describes any significant persistent element.

Section 9: describes any performance issues and constraints

Section 10: describes any aspects related to the quality of service (QoS) attributes

2. Architectural Representation

This document details the architecture using the views defined in the “4+1” model [KRU41], but using the RUP naming convention. The views used to document Server-side software for organizing the work of students at the academic department under conditions of distance learning application are:

Use Case view

Audience: all the stakeholders of the system, including the end-users.
Area: describes the set of scenarios and/or use cases that represent some significant, central functionality of the system. Describes the actors and use cases for the system, this view presents the needs of the user and is elaborated further at the design level to describe discrete flows and constraints in more detail. This domain vocabulary is independent of any processing model or representational syntax (i.e. XML).
Related Artifacts : Use-Case Model, Use-Case documents

Logical view

Audience: Designers.
Area: Functional Requirements: describes the design's object model. Also describes the most important use-case realizations and business requirements of the system.
Related Artifacts: Design model

Process view

Audience: Integrators.
Area: Non-functional requirements: describes the design's concurrency and synchronization aspects.
Related Artifacts: (no specific artifact).

Module Decomposition view

Audience: Programmers.
Area: Software components: describes the modules and subsystems of the application.
Related Artifacts: Implementation model, components

Data view

Audience: Data specialists, Database administrators
Area: Persistence: describes the architecturally significant persistent elements in the data model
Related Artifacts: Data model.

Deployment view

Audience: Deployment managers.
Area: Topology: describes the mapping of the software onto the hardware and shows the system's distributed aspects. Describes potential deployment structures, by including known and anticipated deployment scenarios in the architecture we allow the implementers to make certain assumptions on network performance, system interaction and so forth.
Related Artifacts: Deployment model.

3. Architectural Goals and Constraints

Server-side software for organizing the work of students at the academic department under conditions of distance learning can be hosted on any Operation System (Windows, Linux, others), and connecting to PostgreSQL Database servers. All communication with client has to comply with public HTTPS, TCP/IP communication protocol standards.

3.1. Security

For security uses JWT tokens and role system.

3.2. Persistence

Data persistence will be addressed using a relational database.

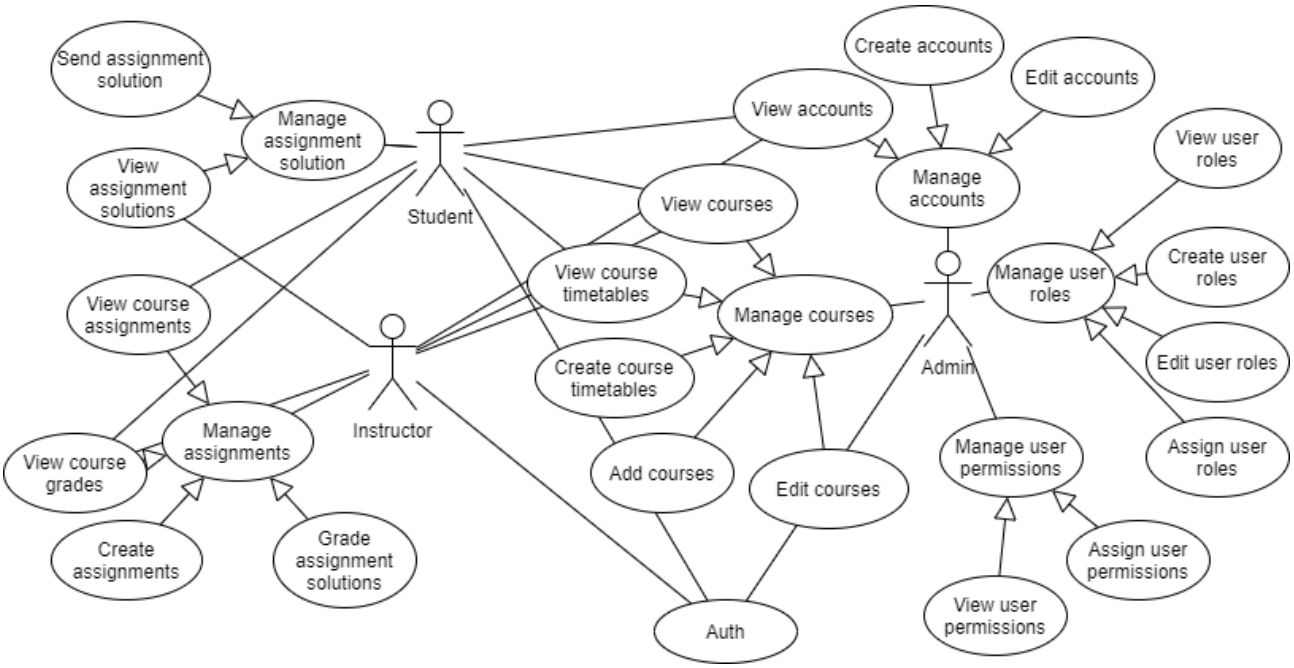
3.3. Performance

There is no particular constrains related to system performance.
It is anticipated that the system should respond to any request well under standard timeouts (20 seconds), also system performance can depend on available hardware. Therefore, actual performance can be determined only after system deployment and testing.

4. Use-Case View

This is a list of use-cases that represent major functionality of the final system [SRS]:

- Auth
- Manage accounts
- Manage user roles
- Manage user permissions
- Manage courses
- Manage assignments
- Manage assignment solutions



4.1. Actors

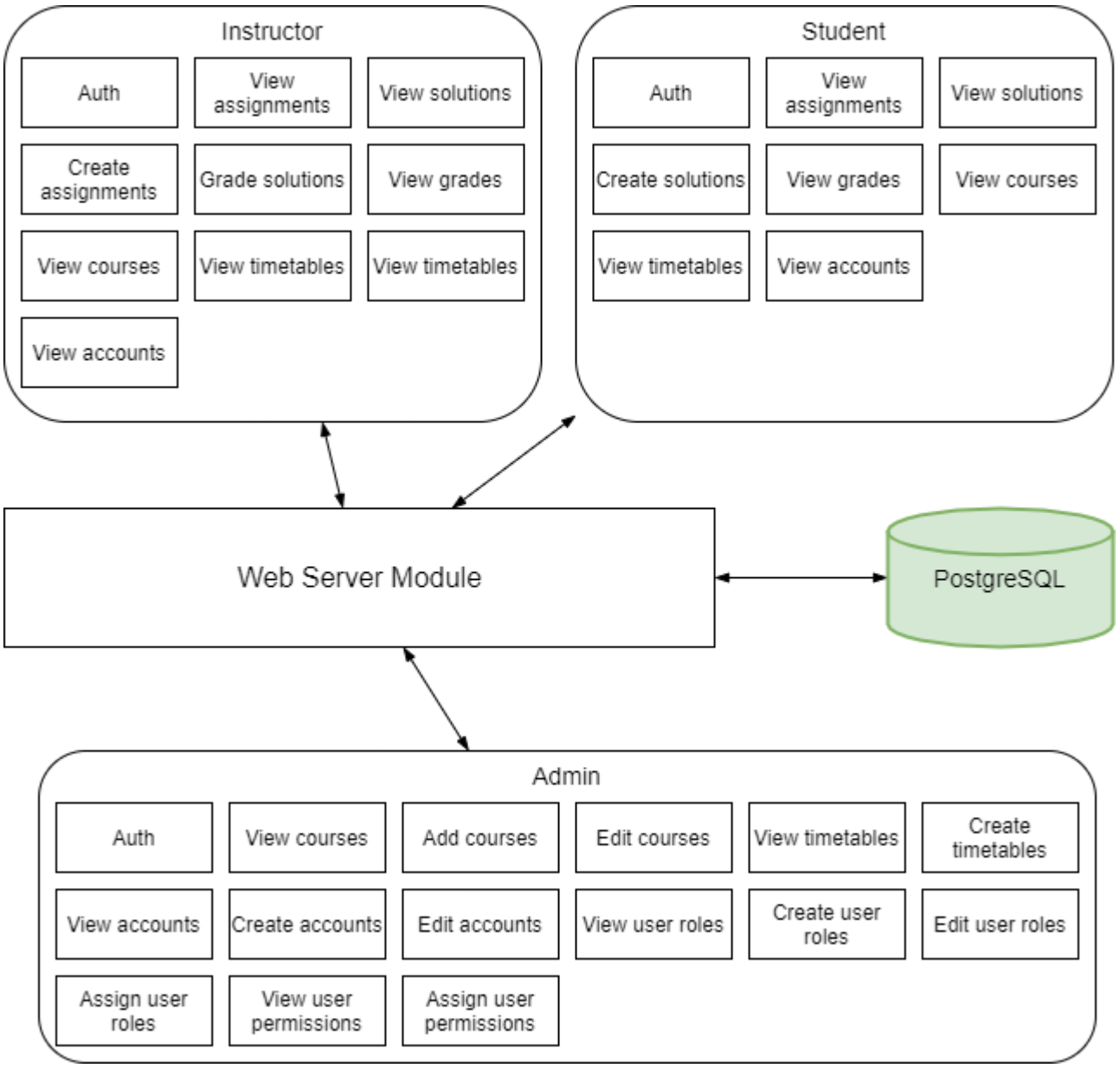
As described in the actors' correspondence diagram below, web user could be one of three types:

1. Instructor

2. Student
3. Admin

4.2. Use-Case Realizations

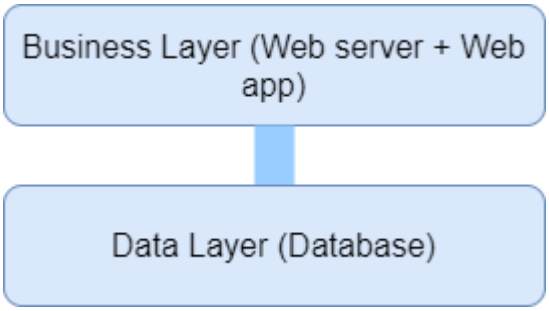
Use case functionality diagram below describes how design elements provide the functionalities identified in the significant use-cases. Use cases are displayed as functionalities for the system. Functionality may enclose more than one use-case.



5. Logical View

5.1. Overview

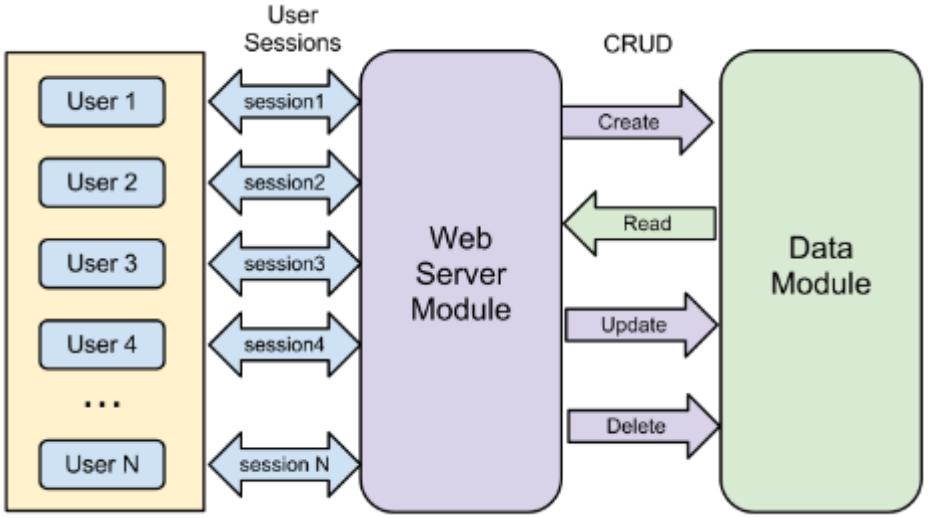
Automated system for checking tasks from the basics of programming is divided into layers based on the N-tier architecture [KRU41].



The layering model of the Server-side software for organizing the work of students at the academic department under conditions of distance learning application is based on a responsibility layering strategy that associates each layer with a particular responsibility. This strategy has been chosen because it isolates various system responsibilities from one another, so that it improves both system development and maintenance.

6. Process View

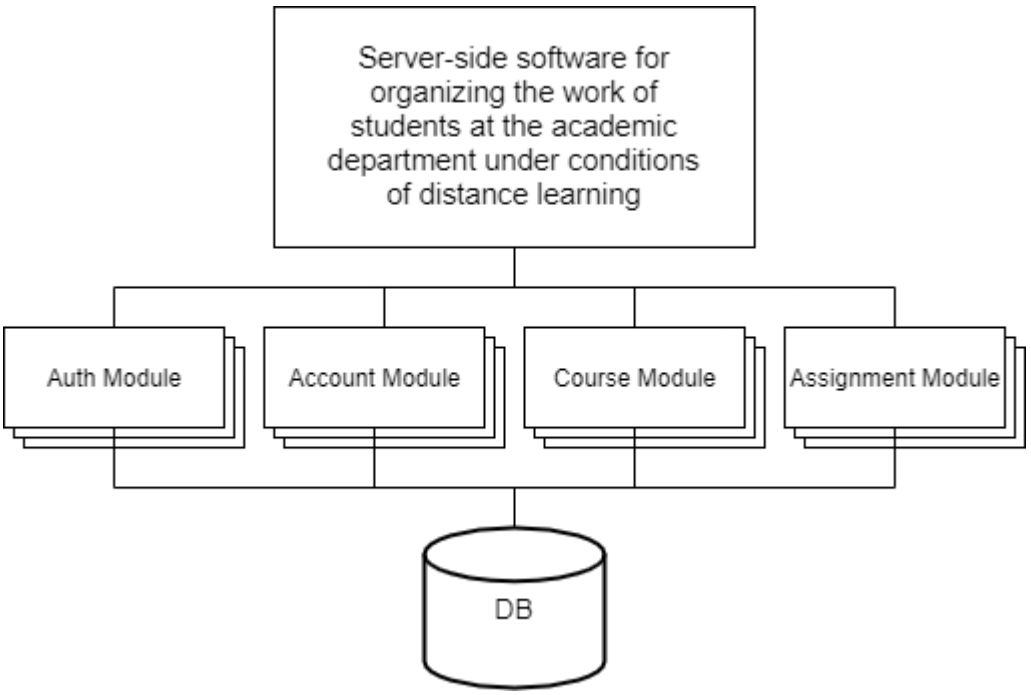
Due to disconnected nature of HTTP request / response and ability of relational database management system (RDMS), Server-side software for organizing the work of students at the academic department under conditions of distance learning will handle multiple users simultaneously. Therefore, concurrency issues such as synchronous versus asynchronous mechanisms will be not considered in this document.



- User – Instructor, Student or Admin
- Session – HTTP session assigned by web server automatically
- CRUD – Create-Read-Update-Delete

7. Module Decomposition View

Module decomposition view based on principles separation of concerns and abstraction and supports goals of modifiability and usability.



8. Data View

The data view represents a significant part of the Server-side software for organizing the work of students at the academic department under conditions of distance learning I. Modularization (normalization) is selected as the design approach of a physical data model. Data consistency and quality are enforced through the series of Primary and Foreign Key constraints.

9. Size and Performance

Volumes

- Simultaneous users 1000 max
- Data storage under 30MB per user

Performance

- With maximum load all transactions well under standard server script / database connection timeout – 20 seconds.

10. Issues and Concerns

- ➢ Data import and export
- ➢ University docs management