МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук, фізики та математики
Кафедра інформатики, програмної інженерії та економічної кібернетики

ПРОЄКТУВАННЯ ТА РОЗРОБЛЕННЯ СЕРВІСНОЇ АРХІТЕКТУРИ УПРАВЛІННЯ БІЗНЕС-ПРОЦЕСАМИ УНІВЕРСИТЕТУ. СЕРВІС «ВІДДІЛ КАДРІВ»

Кваліфікаційна робота (проєкт)
на здобуття ступеня вищої освіти "магістр"

Виконав: студент 2 курсу
Спеціальності 121 Інженерія програмного забезпечення
Освітньо-професійної програми
«Інженерія програмного забезпечення»
другого (магістерського) рівня вищої освіти
Сенчишен Денис Олександрович
Керівник кандидат фізико-математичних наук,
доктор педагогічних наук, професор
Співаковський Олександр Володимирович
Рецензент кандидат педагогічних наук, доцент
Гончаренко Тетяна Леонідівна

Херсон – 2020

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

1. **API** — програмний інтерфейс додатку (application programming interface)
2. **HTML** — мова розмітки документів (Hypertext Markup Language).
3. **HTTPS** — захищений протокол передачі даних прикладного рівня (Hypertext Transfer Protocol Secure).
4. **HTTP** — протокол передачі даних прикладного рівня (Hypertext Transfer Protocol).
5. **JSON** — текстовий формат обміну даними на основі синтаксису ECMAScript (Javascript object notation).
6. **JWT** — коротке повідомлення з криптографічними перетвореннями, що використовується зокрема для аутентифікації користувачів (JSON Web Token).
7. **ORM** — відображення моделей даних на структури бази даних (Object-Relational Mapping)
8. **REST** — передача репрезентативного стану системи, підхід до організації API (Representational state transfe).
9. **SOAP** — протокол обміну структурованими повідомленнями, базується на XML (Simple Object Access Protocol).
10. **SQL** — мова запитів до реляційних баз даних (Structured Query Language).
11. **XML** — розширювана мова розмітки (Extensible Markup Language).
12. **Авторизація** — підтвердження користувачем права на виконання тих чи інших дій.
13. **Аутентифакація** — підтвердження користувачем своєї особи.
14. **Ідентифікація** — представлення користувачем себе.
15. **СКБД** — система керування базами даних.

ВСТУП

Всесторонній розвиток інформаційних технологій та їх поширення у всіх сферах життя суттєво спрощують не тільки поширення інформації від джерел до кінцевих користувачів, а й роблять ефективнішими всі етапи її створення, збереження, передачі та обробки.

Однією з сфер, що інтенсивно розпочали впровадження у своїй роботі інформаційних технологій, можна відзначити сферу надання адміністративних послуг державними та комунальними органами всіх гілок влади. Так, Кабінет Міністрів України декларує головною метою переведення надання адміністративних послуг в електронний формат збільшення ефективності управління. При цьому, ефективність забезпечується як власне модернізацією бізнес-процесів, так і сопутнім переглядом взаємодії окремих елементів та скороченням паперового документообігу, що у свою чергу сприяє дебюрократізації.

Окремо слід відзначити розвиток та поширення використання засобів цифрової ідентифікації, а саме систем BankID, MobileID, сервісів, що надають засоби для накладання електронного цифрового підпису.

Їх існування та використання як безпосередньо спрощує взаємодію громадян та організацій між собою, так і органічно входить до складу систем надання адміністративних та інших послуг, забезпечуючи необхідний рівень захисту персональних даних, описаний у Законі про захист персональних даних [27], так і міжнародними нормативно-правовими актами, зокрема GDPR [9].

Організації у своїй роботі можуть використовувати ІТ у двох напрямках — по перше, для забезпечення внутрішніх організаційних потреб, по друге — надання тієї чи іншої інформації зовнішнім користувачам.

Закладам освіти потрібно вирішувати обидві наведені задачі — як підтримувати всіх учасників освітнього процесу, так і публікувати інформацію про свою діяльність.

Якість підготовки спеціалістів у закладах освіти і особливо ефективність використання науково-педагогічного потенціалу залежать певною мірою від рівня організації навчального процесу.

Одні з головних складових цього процесу – навчальних план та розклад занять – регламентує трудовий ритм, впливає на творчу віддачу викладачів, тому його можна вважати фактором оптимізації використання обмежених ресурсів – викладацького складу і аудиторного фонду.

Проблему складання розкладу слід розглядати не тільки як трудомісткий процес, об’єкт автоматизації з використанням комп’ютера, але і як проблему оптимального керування.

В той же час, вирішення окремих питань у відриві від оновлення всієї системи має певні труднощі, а часто і взагалі не є можливим з використанням адекватної кількості ресурсів.

Зокрема, багаторазово проводились спроби реалізації та впровадження систем керування розкладом навчальніх занять, які не виявились успішними та наразі не використовуються.

Актуальність дослідження полягає в необхідності уніфікації підходів до управління електронними ресурсами, прискоренні об’єднання різнорівневих ресурсів навчального закладу в єдиний портал та забезпеченні учасників освітнього процесу доступом до персоніфікованої інформації.

Об’єкт дослідження — системи управління університетами та взаємодія між їх підрозділами.

Предмет дослідження — сервісна архітектура управління бізнес-процесами університету. Сервіс обліку студентів та персоналу.

Метою роботи є проектування та розробка архітектури розширюваної системи управління бізнес-процесами університету та реалізація відкритого API для взаємодії з системою, реалізація сервісу обліку студентів та персоналу.

Для реалізації мети поставлено наступні **завдання дослідження**:

а) Розглянути характеристики існуючих систем, зокрема обсяг їх можливостей.
б) Розглянути інформаційну інфраструктуру ХДУ.
в) Проаналізувати окремі бізнес-процеси ХДУ.
г) На основі проведеного аналізу розробити вимоги щодо можливостей серверної частини системи.
д) Обґрунтувати використані технології при проектуванні серверної частини.
е) Відповідно до створених вимог розробити серверну частину системи.
ж) Реалізувати публічний API.
з) Розробити документацію до публічного API.

Очікується, що спроектований продукт буде придатний до використання всіма учасниками освітнього процесу в закладаі вищої освіти.

Робота складається з 3 розділів, містить 3 рисунки та 1 таблицю.

РОЗДІЛ 1
ОГЛЯД СИСТЕМ УПРАВЛІННЯ БІЗНЕС-ПРОЦЕСАМИ

В рамках дослідження поставлено задачі з огляду підходів до організації інформаційних та бізнес-процесів організацій, як у цілому, так і на прикладі закладів вищої освіти, зокрема ХДУ.

1.1 Корпоративна інформаційна система

Корпоративна інформаційна система — це інформаційна система, котра надає підтримку у автоматизації тих чи інших функцій з управління бізнес-процесами на підприємстві і надає користувачам інформацію для прийняття управлінських рішень. Крім того, в залежності від реалізації, система може автоматизовувати і прийняття шаблонних рішень на тому чи іншому рівні та з певним рівнем ручного контролю. У ній реалізована управлінська ідеологія, яка об’єднує бізнес-стратегію підприємства, реалізацію підтримки та автоматизацій виконання його бізнес-процесів, контролю за їх виконанням. Для реалізації цих задач використовуються прогресивні інформаційні технології [13].

У загальному визначенні «автоматизована система» — сукупність керованого об’єкта й автоматичних керувальних пристроїв, у якій частину функцій керування виконує людина. Вона представляє собою організаційно-технічну систему, що забезпечує вироблення рішень на основі автоматизації інформаційних процесів у різних сферах діяльності.

Сучасні автоматизовані системи управління навчальним процесом у закладах вищої освіти здатні вирішувати велику кількість функцій [39], а саме:

- планування навчальної діяльності;
- контроль за навчальним процесом;
- аналіз результатів навчальної діяльності;
- доступ до інформації про хід навчального процесу;

- систему звітів на основі даних та статистичної інформації;
- системи забезпечення безпеки даних та контролю доступу до них з урахуванням вимог законодавства;
- облік контингенту студентів та співробітників;
- проведення вступної кампанії;
- формування пакетів даних з метою виготовлення тих чи інших документів.

Функціонування будь-якої автоматизованої системи можна швидко адаптувати до особливостей навчального процесу конкретного навчального закладу, до локальних мереж різного рівня, що допомагає розширити коло користувачів (адміністрації, викладачів і студентів) для оперативного забезпечення їх необхідною інформацією.

Отже, використання таких систем дає змогу не тільки удосконалити якість планування навчального процесу, а й оперативність управління ним.

Не зважаючи на всі переваги, які надає використання автоматизованих систем, досі далеко не в кожному закладі вони впроваджені чи використовуються в повній мірі з тих чи інших причин — інерційності поглядів адміністрації, супротив працівників або «саботаж» на місцях, відсутність фінансової або організаційної можливості.

1.1.1 Інформаційно-аналітична система

В ХДУ використовується корпоративна інтегрована система «Інформаційно-аналітична система (IAS)».

Вона дозволяє вести облік працівників і студентів, бухгалтерський облік, контроль за матеріальними цінностями тощо (рис. 1.1).

Система дозволяє вносити і ефективно стежити за будь-якими змінами. В основі системи лежить ядро, на основі ядра виконується розширення системи до будь-якої кількості компонентів. При цьому

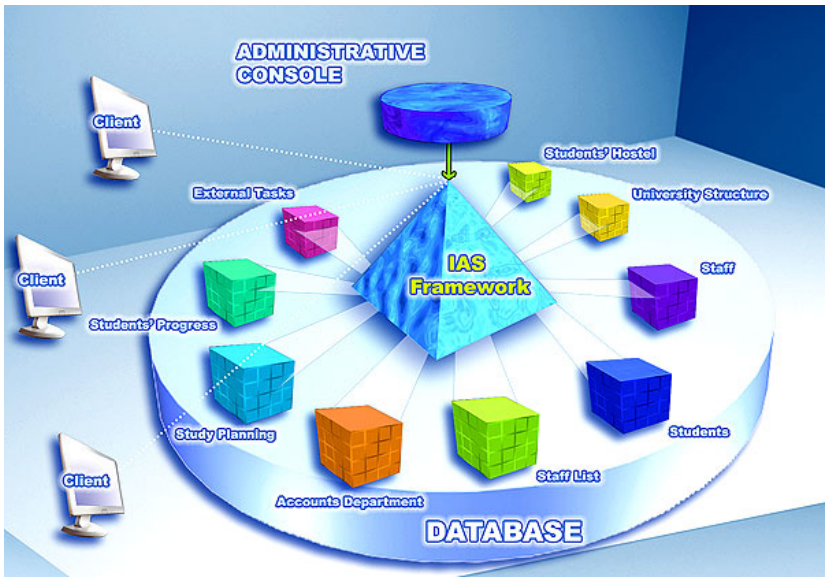

Рисунок 1.1 – Структура IAC

основна функціональність може бути розширена за рахунок додаткових компонентів [31].

Програма IAS орієнтована на платформу на базі операційної системи Windows з використанням системи керування базами даних MS SQL Server. Вона має багаторівневу архітектуру, що складається з бази даних, засобами маніпулювання даними, рівню бізнес-логіки, системи звітів та клієнтського інтерфейсу.

Журнал реєстрації подій зебезпечує збереження історії маніпулювання даними і дозволяє та слідкувати за записами, що стосуються усіх подій.

Відсутність компонентів, пов’язаних з формуванням розкладу занять, та відсутність у використанні сторонніх рішень ставить задачу з проектування власного додатку для забезпечення всіх учасників освітнього процесу доступом до актуальної версії розкладу занять у будь-який час, а також можливості спрощення процесу формування розкладу та подальшої інформатизації освітнього процесу.

1.2 Єдине інформаційно-освітнє серидовище ХДУ

Разом з осучасненням освітнього процесу у цілому актуальною є задача з об'єднання існуючих інформаційних систем університету.

В результаті проведеної інтеграції планується отримати так званий «Особистий кабінет студента», в котрому студент, як основний учасник освітнього процесу, матиме доступ до всієї необхідної інформації.

Наразі інформаційне середовище включає в себе низку в цілому незалежних один від одного проектів, а саме, деякі з них:

- web-портал університету [40];
- система підтримки дистанційного навчання «KSU Online» [37];
- система підтримки дистанційної освіти «Херсонський Віртуальний Університет» [29].
- програмний комплекс «ST-Абітурієнт», що використовується у університеті для підтримки процесу обробки заяв про зарахування на навчання, прийому документів абітурієнтів та результатів вступних іспитів тощо;
- програмний комплекс «Інформаційно-аналітична система (IAC)», що дозволяє вести облік співробітників, зокрема викладачів, і студентів, бухгалтерський облік, контроль за матеріальними цінностями (розділ 1.1.1), проте лише частково охоплює навчальний процес;
- сервіс «KSU Feedback», що забезпечує проведення анонімного або звичайного голосування за визначеними критеріями серед обраного переліку респондентів [35];
- сервіс «Пошук книг в електронному каталозі бібліотеки» надає доступ до каталогу в будь який момент [36];
- web-портал «Збірник наукових праць «Інформаційні технології в освіті» (ІТО)», що призначений для поширення та передачі знань про розробку та впровадження ІТ і використання їх у сфері освіти [22];

– web-портал «Чорноморський ботанічний журнал», що надає відкритий доступ до електронних версій статей у форматі pdf.

При цьому наразі відсутня будь-яка інтеграція між сервісами та сайтами, крім посилань на певну частину з них на головній сторінці web-порталу університету.

1.3 Огляд систем управління ЗВО

Проекти систем управлінської діяльності закладами освіти традиційно охоплюють широкий спектр завдань: від додаткової формалізації процедур збору та зберігання інформації до здійснення змін в організаційній структурі управління і перерозподілу обов'язків.

Ефективність кожної з окремо взятих розробок зазвичай є недостатньою, оскільки зазвичай відсутній єдиний системний підхід до управління навчальним закладом.

Основною негативною рисою є те, що програми від різних виробників не мають можливості ефективного обміну даними, а часто — взагалі не передбачають можливості інтеграції з іншими програмно-апаратними рішеннями.

Окремо стоять питання ліцензування і використання сторонніх розробок. Слід зазначити, що абсолютна більшість існуючих систем мають закритий вихідний код.

Безвідносно обраної нами ліцензії, автором наголошується у необхідності публікації вихідного коду системи.

Щодо ризиків виявлення сторонніми користувачами вразливостей у системі, всюди де можливо не має використовуватися підхід «Security through obscurity» — коли безпека системи забезпечується не криптографічно сильними алгоритмами, а лише фактом того, що сторонній користувач не бачить як обійти недосконалий захист.

Під керівництвом Владислава Гетьмана проведено огляд серії вітчизняних (АСУ «СТЕП 5 ПРОФ», АСУ навчальним процесом «Дире-

ктива», АСУ «Університет», Пакет комп'ютерних систем ПП «Політексофт», Програмний комплекс «АЛЬМА-МАТЕР» АСУ «Вищий навчальний заклад» НДІ ПІТ, ІАС «Університет» Херсонський державний університет, Електронна система управління ВНЗ «Сократ» Вінницький національний аграрний університет) та зарубіжних (Classter, Ellucian, PowerVista RollCall, iGradePlus, CampusAnyware, Administrative Solutions 3, mySkoolApp, Veracross) систем управління ЗВО [23].

Оглянуто основні типи функціоналів, які має та чи інша система, серед них слід відзначити нижченаведені.

Жирним виділено ті з пунктів, реалізація котрих ввійде в першу версію системи.

1. Управління навчальними матеріалами. Можливість зберігати, редагувати та поширювати лекції, лабораторні та семінари всередині системи управління закладу.
2. Управління харчуванням. Можливість замовити та оплатити продукти харчування на території університету. Відповідно наявність функціоналу для закладів харчування, які співпрацюють з ВНЗ .
3. Звітність / аналітика. Можливість вести звіт діяльності університету. Автоматичний збір та аналіз даних для внесення до звітності.
4. **Управління оцінками. Функціонал для викладачів та студентів, що дозволяє вести та контролювати журнал оцінок.**
5. Управління бібліотекою. Онлайн база книжок. Бронювання книг. Контроль за поверненням книг.
6. Управління кампусом.
7. **Управління навчальними програмами. Система створення та контролю за навчальними програмами дозволяє розподіляти навчальні години, планувати заняття, моніторити навчальний процес онлайн.**
8. Управління фінансовою допомогою. Контроль за грантами, соціальними виплатами та стипендіями.

9. Спеціальна та факультативна освіта.
10. Онлайн-платежі.
11. **Управління доступом.**
12. Фінансовий менеджмент ЗВО
13. Портал для абітурієнтів.
14. Управління випускниками. Моніторинг та аналіз успішності випускників.
15. **Факультет, управління персоналом.**
16. Управління житлом, гуртожитками.
17. Управління збору коштів. Благодійність в межах ЗВО.
18. Планування. Планування внутрішньої та зовнішньої діяльності ЗВО.
19. **Інформація для студентів / записи. Таблиці розкладів, культурних заходів та ін.**
20. Студентський портал. Функціонал для студентського самоврядування..
21. Інструменти зв’язку. Чати груп, факультетів. Контакти відділів ЗВО, викладацького складу.
22. Управління громадою. Функціонал для викладацького самоврядування, профспілок.
23. **Календар подій.**
24. Інтерактивне навчання. Онлайн лабораторії, додатковий медіаматеріал.
25. Навігація по території ЗВО.
26. **Опитування, голосування.**
27. **Управління безпекою.**
28. Управління студентською групою.
29. Електронний документообіг.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

2.1 Використані програмні рішення та бібліотеки

В результаті проведеного аналізу, окремі частини якого представлені нижче, було прийнято ряд рішень щодо вибору технологій, в тому числі і таких, що суттєво впливають на архітектуру додатку.

Як результат, до основних технологій ввійшли нижче наведені.

- ***Python*** та його стандартна бібліотека [11] як основа стеку технологій;
- ***Django*** як фреймворк для побудови основи додатку [20] та менеджменту бази даних;
- ***Django REST Framework*** як фреймворк для реалізації взаємодії між сервером та клієнтами [12] і сторонніми додатками з використання підходу REST (REpresentational State Transfer) [20];
- ***PostgreSQL*** як основна СКБД системи;
- ***Pip*** як менеджер пакетів та залежностей для python;
- ***Docker*** як програмний засіб автоматизації розвертання та управління додатками як контейнерами з віртуалізацією на рівні операційної системи [18];
- ***Git*** як найпоширеніша система контролю версій [21];
- ***GitHub*** [7] та ***GitLab*** [8] як віддалені сховища репозиторіїв;
- ***Docutils*** як генератор web-документації сервісу на основі програмного коду опису моделей даних та коментарів класів;
- ***PlantUML*** як мова оформлення супроводжуючих діаграм [33] до структури сервісу;
- ***Markdown*** як мінімалістичний засіб для оформленнія супровідних текстів, інструкцій (***README.md*** тощо);
- ***Swagger*** як специфікація [5] та технологія документування API;

– ***GrayLog*** як система менеджменту логів, що має практику використання при розробці інформаційно-комунікаційних систем [24];
– ***Bash*** як мова скриптів в *nix подібних операційних системах, що дозволяє описати примітивні послідовності консольних команд, наприклад розвертання сервісу або виклик послідовності менеджмент-команд Django;

2.2 Клієнт-серверна архітектура

Архітектура «клієнт-сервер» є одним шаблонів щодо реалізації архітектури програмного забезпечення. Крім того вона є домінуючою концепцією у реалізації розподілених у мережі застосунків. Передбачає собою взаємодію та обмін даними між частинами системи, при цьому окремі частини не є рівнозначними. Вона передбачає такі основні компоненти:

– набір серверів, які надають інформацію (у широкому значенні) додаткам, що звертаються до них;
– набір клієнтів, які використовують інфрмацію (сервіси), що надаються серверами;
– мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери зазвичай є незалежними один від одного. В окремих випадках сервери можуть самі бути клієнтами у відношенні до інших серверів. Клієнти функціонують паралельно і незалежно один від одного. Немає жорсткої прив’язки клієнтів до серверів. Типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; а клієнт звертається до різних серверів.

Клієнти мають знати про доступні сервери, але можуть не мати уявлення про існування інших клієнтів [17].

Загальноприйнятим є положення, що клієнти та сервери — це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп’ютерах, але бувають ситуації, коли обидві програми — і

клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним. Можна окремо розглянути ситуацію, коли клієнти та сервери розташовані на різних комп‘ютерах в середені локальної мережі, а сервери не доступні з глобальної мережі. Це може переслідувати різні цілі, зокрема, питання забезпечення безпеки. Прикладом остайнього є ІАС в Херсонському державному університеті.

2.2.1 REST API

REST — підхід до реалізації архітектури мережевих протоколів, що надають доступ до інформаційних ресурсів. Був описаний і популяризований одним із авторів стандарту протоколу HTTP.

В основі REST закладено принципи функціонування мережі Інтернет і, зокрема, можливості HTTP. Філдінг розробив REST паралельно з HTTP 1.1, при цьому базувався на попередньому протоколі HTTP 1.0.

Дані можуть передаватися у вигляді певних стандартних форматів (наприклад, HTML, XML, JSON). Будь-який REST протокол (HTTP в тому числі) має підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати стан системи між парами «запит-відповідь»[30].

Такий підхід забезпечує масштабовність системи і дозволяє їй розвиватись з отриманням нових вимог. Ці особливості сприяють використанню REST API при проектуванні мікросервісних додатків [30].

REST, як і кожен архітектурний стиль відповідає ряду обмежень архітектури додатку. Це гібридний стиль який успадковує обмеження з інших архітектурних стилів.

2.2.1.1. Клієнт-серверна архітектура систем

Основна архітектура від якої він успадковує обмеження — це клієнт – сервер. Вона вимагає розділення відповідальності між частинами системи, які займаються зберіганням та обробкою даних (сервером), і

тими компонентами, які займаються відображенням даних на стороні користувача та дії з цим інтерфейсом (клієнтом).

Таке розділення дозволяє компонентам розвиватись незалежно один від одного та спрощує внесення змін у систему.

2.2.1.2. Відсутність стану при клієнт-серверній взаємодії

Ще одним архітектурним обмеженням є те, що акти взаємодії між сервером та клієнтом не мають стану, а отже кожен окремий запит містить всю необхідну інформацію для його обробки, і не використовує при цьому те, що сервер знає ту чи іншу інформацію з попереднього запиту.

Відсутність стану не означає що стану немає. Відсутність стану означає, що сервер не знає про стан клієнта. Коли клієнт, наприклад, запитує головну сторінку сайту, сервер відповідає на запитання і забуває про клієнта.

Клієнт може залишити сторінку відкритою на певний час, перш ніж перейти за посиланням, і тоді сервер відповість на наступний запит. В той час сервер може відповідати на запити інших клієнтів, нічого не робити, або навіть бути фізично вимкненим частину часу — для клієнта це не має значення.

Таким чином, наприклад дані про стан сесії (користувача, який аутентифікувався системи, детальніше питання та варіанти аутентифікації буде розглянуто у пункті 2.3) зберігаються на клієнті, і передаються окремо разом з кожним запитом, наприклад, у заголовку запиту. Це покращує масштабовність, бо сервер після закінчення обробки запиту може звільнити всі ресурси, задіяні для цієї операції, для використання їх в обробці інших запитів, без жодного ризику втратити інформацію.

Також спрощується контроль і пошук помилок, бо для аналізу того, що відбувається в певному запиті, досить переглянути лише на той один

запит. Збільшується надійність, адже помилка в одному запиті не зачіпає інші.

2.2.2 Публічне API системи

В процесі проектування створено структуру роутів, котра може використовуватися як частинами системи, так і сторонніми сервісами за наявності ключів доступу.

Однією із поставлених задач було розроблення документації до публічного API. Саме це є необхідною умовою для забезпечення принципів перевикористання коду і можливості використання проекту або окремих його сервісів сторонніми учасниками.

Документацію опубліковано в репозиторії проекту [8]. При підготовці документації використано специфікацію OpenAPI [5] та технологію Swagger, що забезпечує можливість тестування API.

На рисунку 2.1 наведено фрагмент Swagger-специфікації API проекту, а саме — метод **api-auth** для створення токену доступу до системи за логіном і паролем користувача, а також методи **api/profile** та **api/schedule** для отримання даних про користувача та розклад.

Крім того, Swagger реалізує «self-documented» підхід, коли декларація коду у є його документацією.

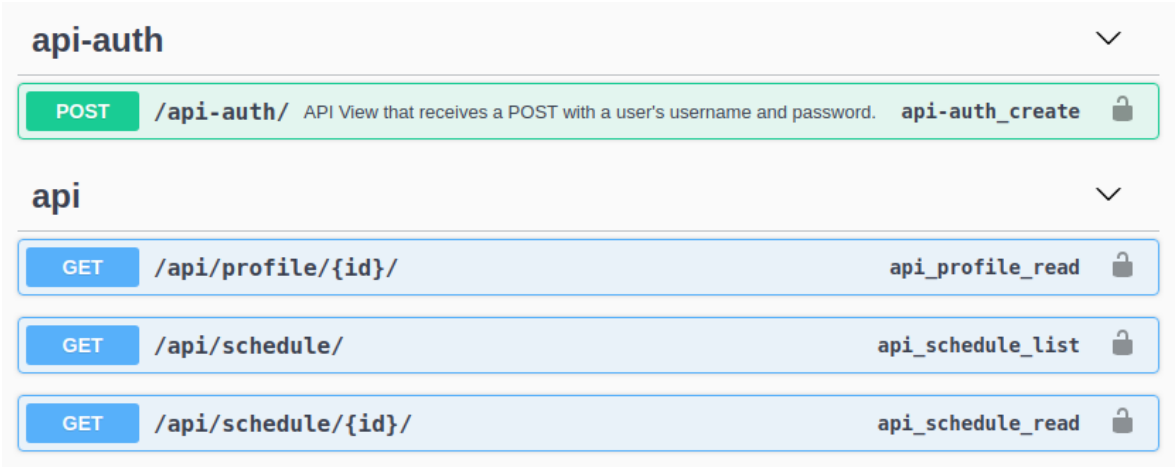

Рисунок 2.1 – Зразок Swagger-специфікації API проекту

Для автоматизації генерації OpenAPI документації API використано бібліотеку **drf-yasg**, котра використовує механізми Django REST Framework [12].

Аналогічний підхід використано у описі моделей даних, коли docstring класу [11] моделі та help-text поля класу створюють собою документацію структури даних. Останнє автоматизується бібліотекою **Docutils**.

2.2.3 CRUD-операції

В процесі проектування закладено серію моделей, що відповідають об’єктам предметної області. Для доступу до даних використовуються основні HTTP методи, що відповідають операціям CRUD (від Create, Read, Update, Delete), їх перелічено нижче.

2.2.3.1. GET

Запитує вказаний ресурс у сервера, який може приймати параметри, що передаються в URL. Згідно зі стандартом, ці запити є ідемпотентними — багатократне надсилання до сервера одного і того ж запиту GET має приводити до отримання однакових відповідей [2] (звісно, за умови, що сам ресурс не було змінено за час між виконаннями запитів). Слід зазначити, що не є рідкісними випадки порушенні цієї вимоги специфікації. Крім того, самі по собі запити можуть модифікувати не сам ресурс, а побічну інформацію, зокрема лічильники відвідувань сторінки.

В запропонованій реалізації запит GET має дві версії — з параметром (ID) та без нього. Останній виконує дію (надає користувачу) не до конкретного об’єкту, а до всієї множини, що є необхідним в певних ситуаціях (наприклад, відображення списку всіх викладачів за певним критерієм).

2.2.3.2. HEAD

Аналогічний GET, за винятком того, що у відповіді сервера на такий варіант запиту має бути відсутнє тіло. Це може бути необхідно для отримання мета-інформації [2].

2.2.3.3. POST

Передає дані (наприклад, з форми на веб-сторінці) заданому ресурсу. При цьому передані дані включаються в тіло запиту. На відміну від методу GET, метод POST не є ідемпотентним. В такому вигляді повторення одних і тих же запитів POST до того ж самого ресурсу буде повертати різні результати (якщо інше не передбачено логікою ресурсу).

На першому етапі відбувається перевірка доступу користувача до створення об’єкту цього типу (авторизація), відповідно до прав доступу (детальніше в пункті 2.3.1).

2.2.3.4. PUT

Завантажує вказаний ресурс на сервер. В розроблюваній системі використовується для редагування існуючих даних.

В процесі виконання, спочатку з бази даних силами ORM вибирається конкретний об’єкт, в нього вносяться зміни, після чого він записується до сховища на заміну попередньої версії.

Слід зазначити, що для багатьох ресурсів це не зовсум відповідає дійсності — ресурс не замінюється зміненою версією, а обидві з них зберігаються у сховищі непрозоро для клієнта. При доступі до ресурсу повертається остайня з версій у сховищі, проте, за потреби та при перегляді історії, будуть використані всі версії.

2.2.3.5. PATCH

Завантажує частину ресурсу на сервер. При розробці необхідності у використанні не знайдено.

Не зважаючи на це, специфікафія API підтримуватиме використання таких запитів у подальшому, а отже фронтенд команда може вільно використовувати їх, якщо такий підхід виявиться оптимальних для оновлення тих чи інших данних.

2.2.3.6. DELETE

Видаляє вказаний ресурс. Слід звернути увагу, що в процесі виконання запиту на видалення об’єкту в системі, видалення як такого не відбувається. Замість цього в окреме поле таблиці вноситься інформація про час виконання цієї процедури.

Такий спосіб реалізації дозволяє з однієї сторони приховати дані, відмічені як видалені від подальшого використання, а з іншої — зберегти їх там, де вони вже використовуються.

В іншому випадку, у зв’язку з реляційністю бази, потрібно було б вирішувати дилему — або проводити циклічне видалення для збереження цілісності даних, втрачаючи всі об’єкти, що посилаються на той, що видаляється; або ускладнювати структури даних, що потенційно призведе до дублювання даних.

2.3 Питання безпеки

В процесі проектування основних частин системи та реалізації підсистеми аутентифікації розглянуто основні способи аутентивікації користувачів та окремі теоретичні питання безпеки.

Проблеми безпеки детально проаналізовано Владиславом Гетьманом в [23].

2.3.1 Ідентифікація, авторизація, аутентифікація

В процесі доступу користувача до системи можна виділити наступні поняття, які нерідко помилково об’єднуються під одним з них або ж

їх відмінності ігноруються: ідентифікація, авторизація та аутентифікація.

а) **ідентифікація** — представлення користувачем себе;
б) **аутентифакація** — підтвердження користувачем своєї особи;
в) **авторизація** — підтвердження користувачем права на виконання тих чи інших дій.

Переважно авторизація та аутентифікація змішуються в один процес, проте можлива і авторизація без аутентифікації. Прикладом можна привести надання користувачеві секретного ключа, не привязаного до його особи.

2.3.2 Способи аутентифікації користувача

Виділяють три ключові фактори авторизації, що почали використовуватись задовго до початку ери цифрових інформаційних технологій.

1. **Щось, що нам відомо** — найпростіший у використанні і реалізації фактор — секретне слово, пароль, PIN-код.
2. **Щось, чим ми володіємо** — ключ, печатка, пластикова картка доступу, файл з сертифікатом, фізичний криптографічний токен або пристрій.
3. **Щось, що є частиною нас** — біометричні характеристики — голос, відбитки пальця, особливості радужної оболонки ока.

Зазвичай ці фактори так чи інакше комбінуються для посилення захисту (що можна назвати двухфакторною аутенифікацією в широкому сенсі, але не в наступних прикладах), наприклад:

а) **1+2** — банківська картка та PIN-код до неї для використання банкомату.
б) **1+2** — ID-картка громадянина України та PIN-код до неї для накладання цифрового підпису на документ.

в) **2+3** — смартфон та відбиток пальця для підтвердження фінансових операцій у мобільному додатку банку.

г) **2+3** — смартфон та FaceID для доступу до документів в хмарному сховищі.

д) **1+3** — секретне слово та обличчя і голос для підтвердження особи працівникові організацій.

е) **1+2+3** — графічний ключ, відбиток пальця та власне смартфон для розблокування доступу до акаунту після перезавантаження смартфону.

2.3.2.1. Криптографічні методи, хешування та JWT

При збереженні фактору авторизації на сервері є неприпустимим збереження його «як є». В такому випадку, якщо зловмисник отримує доступ до бази даних, то він одночасно отримує доступ до всіх факторів авторизації (наприклад паролів) всіх користувачів системи. Ключовою проблемою є те, що попри всі вимоги бізпеки користувачі схильні до повторного використання паролю в інших системах, що ставить під загрозу їх особисту інформацію у цілому.

Незалежно від того, які фактори використовуються та як вони зберігаються — над ними виконуються криптографічні перетворення, зокрема хешування.

Хешем є результат виконання хеш-функції над порцією даних. Хеш функція у загальному — функція, для якої не існує оберненої функції. Можна заявити, що розрахунок значення функції є відносно простим алгоритмом (переважно всі популярні алгоритми опубліковані як стандарти, а їх реалізації входять до стандартних бібліотек мов програмування).

В той же час, знаходження аргументу хеш-функції за її значенням не є можливим суттєво більш швидким способом, ніж шляхом повного перебору.

Як приклади алгоритмів хешування можна навести MD5 та сімейство алгоритмів SHA.

Фреймворк Django пропонуэ реалізацію парольної авторизації з використанням алгоритму PBKDF2 з хеш-функцією SHA256.

Одночасно з тим, для доступу до API використовуэться JWT (детальніше в пункті 2.3.3), який має ще складніші криптографічні перетворення у своїй основі.

Довжина хешів, котрі використовуються є достатньою, щоб виключити можливість перебору за практичний час.

Задачу пошуку паролю за хешем довшини такого порядку можна оцінити як близьку до трансобчислювальної — задачі, яку комп'ютер розміром з Землю, який працює на максимальній теоретично можливій швидкості (котру називають лімітом Бремерманна [6]), обчислюватиме довше часу існування Землі. Вважається, що задача є трансобчислювальною, якшо для її вирішення необхідно обробити більше 10^{93} біт інформації [6].

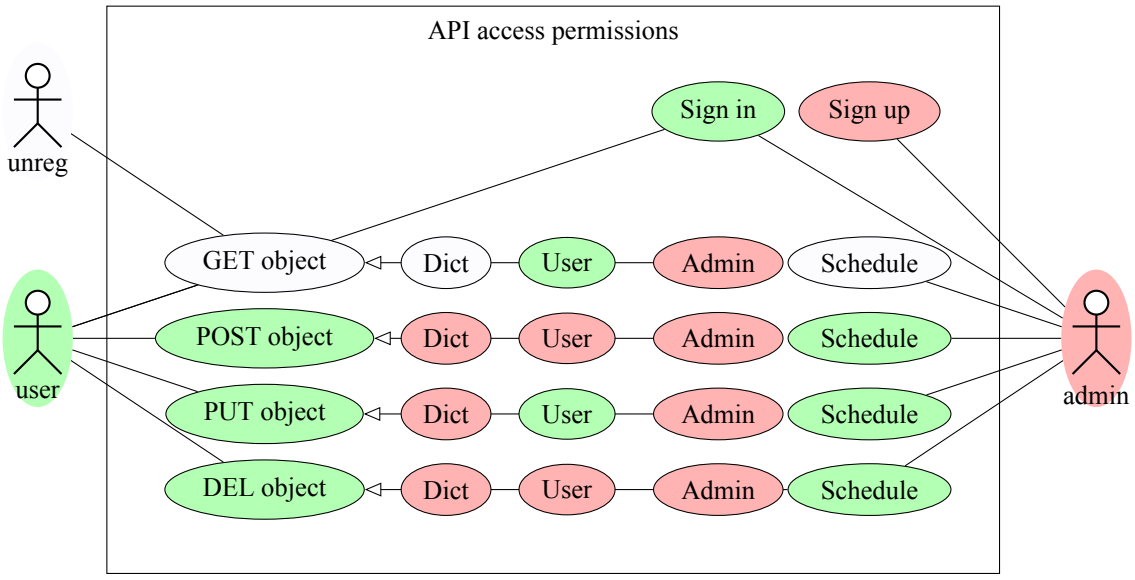

Рисунок 2.2 – Доступ на виконання запитів до системи

2.3.3 JSON Web Token

Для забезпечення конфіденційності при обміні даними використовується JSON Web Token. Можливу структуру роути, що обробляють реєстраційні, авторизаційні запити, та запити на доступ до інформації з різним рівнем доступу, представлено на рис. 2.2.

JSON Web Token це стандарт токена доступу на основі JSON, стандартизованого в RFC 7519 [14]. Використовується для верифікації тверджень. JSON Web Token складається з трьох частин: заголовка, вмісту і підпису.

В корисному навантаженні зберігається будь-яка інформація, яку потрібно перевірити. Кожен ключ в корисному навантаженні відомий як «заява». Як і заголовок, корисне навантаження кодується в base64. Після отримання заголовку і корисного навантаження, обчислюється підпис.

У несеріалізованном вигляді JWT складається з заголовка і корисного навантаження, які є звичайними JSON-об'єктами.

Тема (заголовок JOSE) в основному використовується для опису криптографічних функцій, які застосовуються для підпису або шифрування токена. Тут також можна вказати додаткові властивості, наприклад, тип вмісту, хоча це рідко потрібно.

Якщо JWT підписаний або зашифрований, в заголовку вказується ім'я алгоритму шифрування. Для цього призначена заявка alg.

Слово «заявка» в специфікації позначає просто частина інформації і аналогічна ключу JSON-об'єкта. Вона представлена у вигляді пари $name : value$, де $name$ завжди є рядком. Значним заявки може бути будь-який серіалізуємий тип даних. Наприклад, об'єкт JSON на рис. 2.3 складається з трьох заявок: iss, exp і $http : example.comis\_admin$.

Заявки бувають службовими і призначеними для користувача. Перші зазвичай є частиною будь-якого стандарту, наприклад, реєстру JSON Web Token Claims, і мають певні значення.

```
{
  ”iss”: ”joe”,
  ”exp”: 1300819380,
  ”http://example.com/is_root”: true
}
```

Рисунок 2.3 – Приклад об’єкту JSON

Токен можна підписати, щоб перевірити, чи не були змінені дані, що містяться в ньому. Підписаний веб-токен відомий як JWS (JSON Web Signature). У компактній серіалізовані формі у нього з’являється третій сегмент - підпис.

На відміну від підпису, який є засобом встановлення автентичності токена, шифрування забезпечує його нечитабельність.

Зашифрований JWT відомий як JWE (JSON Web Encryption). На відміну від JWS, його компактна форма має 5 сегментів, між якими ставиться крапка. Додатково до зашифрованого заголовку і корисного навантаження, він включає в себе зашифрований ключ, вектор ініціалізації і тег аутентифікації.

2.4 Система керування базою даних

База даних — сукупність даних, організованих відповідно до певної прийнятої концепції, яка описує характеристику цих даних і взаємозв’язки між їхніми елементами. Дані у базі організовують відповідно до моделі організації даних.

В загальному випадку базою даних можна вважати будь-який впорядкований набір даних, наприклад, паперову картотеку бібліотеки. Але все частіше термін «база даних» використовуєтьсяу контексті використання баз даних в інформаційних системах, як і самі бази даних переносяться в електронні системи в процесі інформатизації. На даний

час додатки для роботи з базами даних є одними з найпоширеніших прикладних програм [38].

Через тісний зв'язок баз даних з системами керування базами даних (СКБД) під терміном «база даних» нерідко неточно мається на увазі система керування базами даних. Але варто розрізняти базу даних — сховище даних, та СКБД — засоби для роботи з базою даних. Надалі, в роботі під терміном «база даних», в залежності від контексту, може матися на увазі як сукупність даних чи певні її параметри, так і СКБД, крім випадків де це не очевидно.

Розроблення перших баз даних розпочинається в 1960-ті роки. Переважно, дослідницькі роботи ведуться в проектах IBM та найбільших університетів. Пізніше, на початку 1970-х років Едгар Ф. Кодд обґрунтовує основи реляційної моделі [4]. Уперше цю модель було використано у бази даних Ingres та System R, що були лише дослідними прототипами. Проте вже в 1980-ті рр. з'являються перші комерційних версій реляційних БД Oracle та DB2. Реляційні бази даних починають успішно витісняти мережні та ієрархічні. Починаються дослідження розподілених (децентралізованих) баз даних.

2.4.1 Реляційна модель даних

Реляційна модель даних — логічна модель даних, вперше описана Едгаром Ф. Коддом [4]. В даний час ця модель є фактичним стандартом, на який орієнтуються більшість сучасних СКБД.

У реляційній моделі досягається більш високий рівень абстракції даних, ніж в ієрархічній або мережевій. Стверджується, що «реляційна модель надає засоби опису даних на основі тільки їх природної структури, тобто без потреби введення якоїсь додаткової структури для цілей машинного представлення» [4]. А це означає, що подання даних не залежить від способу їх фізичної організації, що забезпечується за рахунок використання математичного поняття відношення.

До складу реляційної моделі даних зазвичай включається теорія нормалізації. Дейт визначив наступні частини реляційної моделі даних [25]:

а) структурна;
б) маніпуляційна;
в) цілісна.

Структурна частина моделі визначає, що єдиною структурою даних є нормалізоване n-арне відношення.

2.4.2 Нормалізація бази даних

Нормалізація схеми бази даних — процес розбиття одного відношення (таблиці в поняттях СУБД) відповідно до алгоритму нормалізації на кілька відношень на основі функціональних залежностей.

Нормальна форма визначається як сукупність вимог, яким має задовольняти відношення, з точки зору надмірності, яка потенційно може призвести до логічно помилкових результатів вибірки.

Таким чином, схема реляційної бази даних покроково, у процесі виконання відповідного алгоритму, переходить у першу, другу, третю і так далі нормальні форми. Якщо відношення відповідає критеріям n-ої нормальної форми та всіх попередніх нормальних форм, тоді вважається, що це відношення знаходиться у нормальній формі n-ого рівня.

Проведено порівняння відомих і популярних СКБД, суттєвий фрагмент висновків до аналізу наведено в таблиці 2.1.

2.4.3 СКБД PostgreSQL

PostgreSQL — широко розповсюджена система керування базами даних з відкритим вихідним кодом. Прототип був розроблений в Каліфорнійському університеті Берклі в 1987 році, пізніше проект Берклі було зупинено, а реалізацію було викладено в Інтернет під назвою Postgres95 після вдосконалення вихідного коду. Наразі підтримкою й

розробкою займається група спеціалістів, які добровільно приєднались до проекту.

Сервер PostgreSQL написаний на мові C. Розповсюджується у вигляді вихідного коду, який необхідно відкомпілювати. Разом з кодом розповсюджується детальна документація.

2.4.4 Шаблон проектування ORM

ORM — шаблон програмування, який зв'язує бази даних з концепціями об'єктно – орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних». В об'єктно-орієнтованому програмуванні об'єкти в програмі представляють об'єкти з реального світу.

Суть проблеми полягає в перетворенні таких об'єктів у форму, в якій вони можуть бути збережені у файлах або базах даних, і які легко можуть бути витягнуті в подальшому, зі збереженням властивостей об'єктів і відношень між ними. Ці об'єкти називають «постійними». Існує кілька підходів до розв'язання цієї задачі. Деякі пакети вирішують цю проблему, надаючи бібліотеки класів, здатних виконувати такі перетворення автоматично. Маючи список таблиць в базі даних і об'єктів в програмі, вони автоматично перетворять запити з одного вигляду в інший.

Розробниками фреймворку Django запропоновано використання Django-ORM, котрий надає високорівневий механізм Queryset-ів. Вони дозволяють виконувати оптимізовані запити до бази даних та є оболонкою для генерації безпосередньо низькорівневих запитів до бази даних. В той же час, Django-ORM підтримує з незначними обмеженнями будь-які SQL СКБД, та дозволяє обмежено використовувати навіть NoSQL рішення не змінюючи структуру моделей даних [20].

2.4.5 NoSQL

Напротивагу SQL (та частково реляційних баз даних у цілому) виступають NoSQL рішення, які мають принципові відмінності у підході до збереження даних.

2.4.5.1. MongoDB

Не приймає безпосередньої участі у збереженні даних сервісів, проте є місцем збереження занних системи менеджменту логів GrayLog.

2.4.5.2. Neo4j

Графова система керування базами даних. Попри всі переваги необхідно описати методи доступу до даних в графі, що нівелює можливості Django-ORM [10]. NeoModel-ORM для Python знаходться в розробці.

Таблиця 2.1

Порівняння основних груп СКБД

	Реляційні	Графові	Ключ-значеннння
Структура і тип даних	Однозначно визначена структура	Вузли та ребра з довільними властивостями	Деревовидна структура вкладених значень
Мова запитів	SQL-стандарт	Cypher	Різні
Ефективність обробки складних зв’язків	Швидко падає з ростом кількості зв’язків	Не залежить від розміру даних	Відсутня на рівні СКБД

Таблиця 2.1

	Реляційні	Графові	Ключ-значенння
Попереднє моделювання структури	Потрібно побудувати модель даних	Може змінюватись якщо підтримує додаток	Може змінюватись якщо підтримує додаток
Розмір сховища даних	Залежить від об'єму даних і індексів	Залежить від кількості даних	Залежить від об'єму даних і індексів
Гнучкість	Має статичну модель	Модель не фіксована	Модель не фіксована
Забезпечення цілісності даних	Наявне забезпечення цілісності даних	Наявне забезпечення цілісності даних	Відсутні, має контролюватись додатком

2.5 Практики розробки програмного забезпечення

2.5.1 Системи контролю версій

Активну популярність мають розподілені системи контролю версій (SVC).

Найбільш поширеними з таких є Subversion (SVN), Microsoft Visual Source Safe (VSS), Revision Control System (RCS), Concurrent Versions System (CVS), Git та Mercurial.

Знання подібних систем підвищує затребуваність IT фахівців на ринку праці, покращує продуктивність розробників та полегшує рішення щоденних завдань. Саме передача знань є вирішальною у процесі експорту-імпорту технологій [28].

2.5.1.1. Система контролю версій Git

Під час дослідження було використано систему контролю версій Git з віддаленим репозиторієм на сервісі GitHub [7].

Система контролю дозволяє зберігати попередні версії файлів та завантажувати їх за потребою. Вона зберігає повну інформацію про версію кожного з файлів, а також повну структуру проекту на всіх стадіях розробки.

Місце зберігання даних файлів називають репозиторієм. В середині кожного з репозиторіїв можуть бути створені паралельні лінії розробки — гілки [16].

Git підтримує швидке розділення та злиття версій, містить можливості для візуалізації і навігації за нелінійною історією розробки.

2.5.1.2. Колективна робота з використанням Git

Для злиття гілок використовуються методи зміни історії rebase та merge. Суттєвою є вимога щодо використання fast-forward merge, викладена в описі до процедури злиття вкладу учасника розробки проекту в основну гілку (**CONTRIBUTING.rst**).

Це дозволяє уникати створення merge-комітів, що не несуть сенсу в плані історії розробки, а також дозволяє видаляти гілки після прийняття змін учасника, що є виправданим в рамках проекту відповідно до загальноприйнятих методих використання git [16].

2.5.2 Семантичне версіювання

В процесі розробки програмного забезпечення можливе виникнення проблеми під назвою «пекло залежностей».

Ця проблема полягає в тому, що при збільшенні розмірів програмної системи, збільшується кількість бібліотек та пакетів, що використовуються в ній. При цьому, кожен з них, зазвичай, вимагає для своєї роботи деякі інші бібліотеки певних версій. У разі, якщо документація

програмного забезпечення надто вільна, то рано чи піздно виникає проблема невідповідності між фактично необхідною версією, вказаною в документації та встановленою, що негативно позначається на всьому процесі розробки програмного забезпечення.

Для вирішення цієї проблеми пропонується простий набір правил і вимог, що визначають як встановлюються і збільнуються номери версій. Для роботи системи необхідно створити і описати публічне API програмного продукту. Після цього будь-які зміни в версії визначаються певною зміної її номера.

Розглянемо формат версій X.Y.Z (мажорна, мінорна, патч).

Зміни, що не впливають на API, змінюють номер патч-версії. Зворотньо-сумістні зміни та розширення API збільшують мінорну версію. І, нарешті, несумістні зміни API збільшують мажорну версію.

Ця система називатиметься «Семантичне версіювання».

Мажорна версія «0» (0.Y.Z) призначена для початкової розробки, публічний API не має розглядатися як стабільний. Версія 1.0.0 визначає публічний API, після цього релізу вона змінюватиметься відповідно до змін в API. Після чергової зміни мінорної версії патч-версія змінюється на «0», аналогічні зміни відбуваються зі зміною мажорної версії.

Крім зазначених правил, специфікація семантичного версіонування [19] визначає додатково певні деталі та поради щодо його практичного використання, зокрема для продуктів, що мають складну систему релізів та передрелізних версій.

2.5.3 LaTeX

LaTeX — це високоякісна набірна система; він включає функції, призначені для виготовлення технічної та наукової документації. LaTeX є фактичним стандартом для комунікації та публікації наукових документів [15]. LaTeX доступний як вільне програмне забезпечення.

TEX — це створена американським математиком і програмістом Дональдом Кнутом система для верстки текстів з формулами. Сам по собі TEX є спеціалізованою мовою програмування (Кнут не тільки придумав мову, а й написав для нього транслятор, причому таким чином, що він працює абсолютно однаково на різних комп'ютерах), на якому пишуться видавничі системи, що використовуються на практиці. Точніше кажучи, кожна видавнича система на базі TEXа є пакетом макросів (макропакет) цієї мови. LATEX — це створена Леслі Лампортом видавнича система на базі TEXа [32].

Всі видавничі системи на базі TEXа володіють перевагами, закладеними в самому TEX. Для новачка їх можна описати однією фразою: «надрукований текст виглядає «зовсім як у книзі»» [32].

LATEX, як видавнича система, надає зручні і гнучкі засоби досягти цього книжкового якості. Зокрема, вказавши за допомогою простих засобів структуру тексту, автор може не вникати в деталі оформлення, причому ці деталі при необхідності неважко змінити (щоб, скажімо, змінити шрифт, яким друкуються заголовки, не треба нишпорити по всьому тексту, змінюючи всі заголовки, як це трапляється при ігноруванні стилів у сучасних офісних пакетах, а досить замінити одну декларацію в преамбулі). Нумерація розділів, посилання, зміст і т.п. закладені в пакет макросів. Величезним плюсом систем на базі TEXа є висока якість та гнучкість форматування абзаців і математичних формул (в останньому зауваженні TEX є одним з напотужніших редакторів, його синтаксис закладено в редактори формул офісних пакетів).

TEX (і всі видавничі системи на його базі) невибагливий до використовуваної техніки. TEXовські файли мають високий ступінь переносимості: можливо підготувати вихідний текст і бути впевненими, що текст буде правильно оброблений і відображений на будь-якому іншому комп'ютері з встановленим редактором TEX (навідміну від документів, підготовлених в офисних пакетах).

Особливість TEXa, яка може не сподобатися тим, хто звик до традиційних редакторів офісних пакетів — це те, що він не є системою типу WYSIWYG («те що ти бачиш, те ти і отримаєш»): робота з вихідним текстом і перегляд того, як текст буде виглядати після друку — різні операції. Втім, завдяки цій особливості час на підготовку тексту істотно скорочується.

При роботі над звітом також використано сервіс Overleaf — сучасний інструмент, розроблений у 2012 році. Він був створений щоб допомогти редагувати свої наукові статті, технічні звіти, тези, презентації, блок-схеми та інші документи, написані на мові розмітки LaTeX [1].

РОЗДІЛ 3
СТРУКТУРА СЕРВІСІВ СИСТЕМИ

3.1 Структура закладу освіти

Університет поділяється на два рівні структурних підрозділів:

1. Факультет — очолюється деканом з числа співробітників.
2. Кафедра — очолюється завідувачем з числа співробітників, зазвичай входить до складу одного з факультетів.

Аудиторія визначеної вмістимості відноситься до певної кафедри та/або факультету (або жодного з них).

При розгляді альтернативних СКБД, відзначено, що певні залежності реального світу простіше описувати не з точка зору складноструктурованих об’єктів постійної структури, а з точки зору простих об’єктів, повязаних відношеннями.

Зокрема, працівник та структурний підрозділ організії можуть мати відношення «працює», «очолює». Причину, з якої довелось відмовитись від використання графової бази даних, наведено в пункті 2.4.5.2

3.2 Загальні дані про освітній процес

Основнною частиною освітнього процесу є освітня програма. Вона відноситься до конкретної спеціальності, за якою здійснюється підготовка здобувачів освіти.

Пов’язана з кафедрою, що здійснює підготовку. Окрема для кожного ступеня освіти.

Перелік спеціалностей визначено постановою КМУ. Кожна спеціальність відноситься до галузі знань, що об’єднує споріднені спеціальності [34]. На рис. 3.1 опис моделі даних на прикладі спеціальності.

```
1 class Speciality(DataObject):
2     speciality_group = models.ForeignKey(SpecialityGroup)
3     code = models.CharField(max_length=15)
4     name = models.CharField(max_length=255)
5
6     class Meta:
7         verbose_name = 'Speciality'
8         verbose_name_plural = 'Specialities'
```

Рисунок 3.1 – Приклад опису фрагменту моделі даних спеціальності

3.3 Освітній процес

Основна структура освітнього процесу - дисципліна, що є частиною освітньої програми.

Дисципліна ділиться на декілька блоків (що займають один семестр або модуль), протягом кожного з який викладач викладає а студент вивчає предмет на кожному з навчальних занять (лекціях, практичних та лабораторних заняттях, семінарах тощо).

Заняття проводиться в певній аудиторії у визначений час.

Студент може отримати оцінку на кожному з занять, оцінки відображаються в журналі для викладача і в картці для студента.

Вивчення дисципліни завершується контрольним заняттям (заліком, диференційованим заліком, екзаменом тощо).

На контрольному занятті студент отримує оцінку. Також можливе повторне складання контролю, тобто перездача. Обрахований остаточний результат відображається у відомості обліку успішності для викладача та у заліковій книжці для студента.

3.4 Людські ресурси

Основною одиницею обліку є власне особа, про яку зберігаються основні персональні дані.

В залежності від ролі особи в закладі освіти вона може здобувати одну або більше освіт за певиними освітніми програмами або працювати на одній або кількох посадах.

Відповідно до кожної окремої ролі зберігаються відповідні дані про особу (наприклад дата початку навчання та освітня програма для студента посада для співробітника тощо). На рисунку 3.2 наведено структура моделей даних сервісу відділу кадрів.

Дані про студентів та співробітників імпортуються до системи з зовнішніх ресурсів, зокрема Інформаційно-аналітичної системи університету та ЄДЕБО [26].

Після імпорту даних автоматично створюються профілі користувачів (для кожної особи окремий для кожної її ролі). Наприклад, здобувач аспірантури, котрий одночасно з тим навчається за іншою спеціальністю за заочною формою та працює в університеті матиме три профілі.

Для кожного учасника освітнього процесц створюється щонайменше три записи в базі даних.

Перший — в таблиці користувачів системи аутентифікації Django. Включає в себе відомості, необхідні для надання доступу до системи.

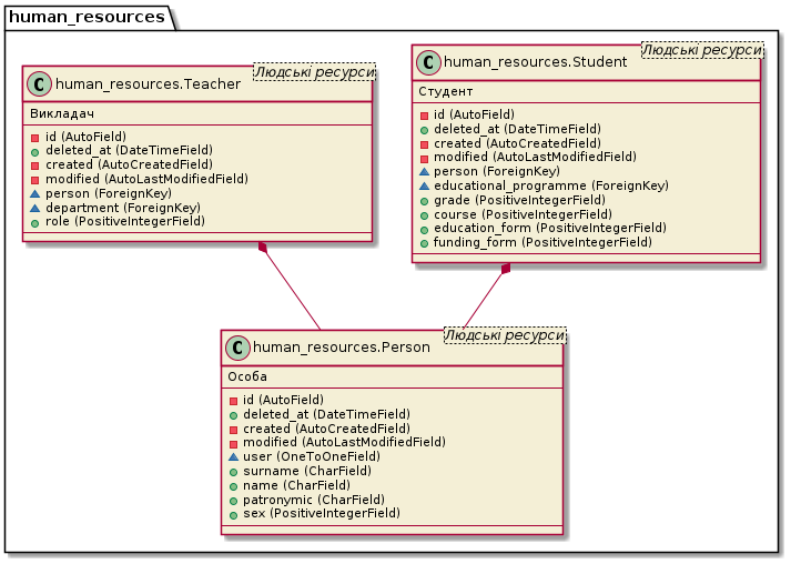

Рисунок 3.2 – Структура моделей даних сервісу відділу кадрів

З цим записом повязуються логи дій в системі. Крім того, всі технічні засоби груп да рівнів доступу (groups та permissions) також пов’язані з цим записом.

Ттаблиця користувачів системи аутентифікації Djang включає в себе:

1. Логін користувача.
2. Електронну пошту.
3. Прізвище.
4. Ім’я.
5. Пароль.
6. Належність до адміністраці системи.
7. Належність до суперкористувачів.
8. Активність запису.
9. Технічні відомості про створення запису.

Активність запису показує, чи може цей корістувач ввійти до системи. Для відключення доступу необхідно і достатньо зняти цей флаг.

Належність до адміністрації системи дозволяє користувачеві входити до сторінки адміністрування Django. Ця можливість обмежено необхідня для контролю за системою технічними адміністраторами та дає прямий доступ до даних в базі даних.

Належність до суперкористувачів надає доступ до всіх даних в адміністративній панелі Django, навіть якщо доступ до них обмежено лише певними групами або заборонено за замовчуванням. Не зважаючи на це, навіть суперкористувач не може внести зміни, що порушать консистентність даних або суперечать обмеженням моделей даних.

Слід відзначити, що пароль не зберігається в тому вигляді, в якому його вводить користувач. Загальноприйнятним є підхід, за якого в базі зберігається не сам пароль, а криптографічне перетворення над ним. Детальніше це було розглянуто в пункті 2.3.

Використовується схема Django за замовчуванням — пароль поєднується з так званною «сіллю», що представляє собою випадкову порцію даних.

Над цією комбінацією виконується криптографічне перетворення з використанням алгоритму PBKDF2 і хеш-функції SHA256.

Результатом є хеш — значення, яке однозначно отримується з вихідних даних, проте обернене перетворення неможливе суттєво швидше, ніж повним перебором всіх потенційних вхідних даних [3].

До бази даних вноситься строка виду 3.1, з якого, при використанні алгоритму за замовчуванням, з частково прихованими даними отримаємо представлення 3.2.

$$< algorithm > \$ < iterations > \$ < salt > \$ < hash > \tag{3.1}$$

$$pbkdf2\_sha256\$216000\$Q3Zjlr***\$Qzt16m** \tag{3.2}$$

Другий запис — в таблиці людей, включає в себе дані про фізичну особу:

1. Прізвище.
2. Ім’я.
3. По батькові.
4. Стать.
5. Дату народження.
6. Посилання на Користувача.
7. Технічні відомості про створення запису.

З цим записом повязуватимуться всі інші рольові записи, зокрема, записи викладачів і студентів.

На початковому етапі аутентифікація та авторизація здійснюються засобами Django. В процесі розгортання вони замінені на аутентифікацію з використанням JWT [23], що використовує засоби Django, проде

надає інші інтерфейси для взаємодії та змінює схему обміну даними для авторизації запитів.

Після аутентифікації користувача знаходяться його профілі. У разі якщо користувач має більше одного профілю йому пропонується обрати активний для роботи у поточній сесії.

Обраний профіль визначає доступну для перегляду інформацію, роль профілю частково визначає інтерфейс користувача (наприклад, інтерфейс буде дещо різним у студента та у викладача, зокрема у частині перегляду та редагування оцінок та іншої інформації про навчальний процес, зокрема розкладу).

3.5 Розклад та заняття

Складовими частинами дисципліни є навчальні та контрольні заняття.

Кожне з них проходить в певній аудиторії, його проводить викладач. Заняття розпочинається у певний час та має визначену тривалість. В занятті приймають участь студенти.

За результатами кожного навчального заняття у викладача є можливість виставити відмітку про присутність студента на ньому та відмітку до виконаної роботи за бажанням.

За результатами проведення контрольного заняття кожному учаснику має бути виставлена відмітка до результату контролю. Можу бути передбачено більше одного контрольного заняття за семестр (модуль для одномодульних дисциплін). В такому разі для студента має визначатись остаточна оцінка у разі якщо він приймав участь у повторній здачі контролю.

3.6 Контроль якості освітнього процесу

З метою контролю за якістю надання освітніх послуг проводяться опитування здобувачів освіти по прослуханих предметах.

Генерація опитувань відбувається напівавтоматично адміністратором системи з використанням даних про здобвувачів освіти; дисципліни, що викладалися протягом поточного семестру; залучення здобувачів до прослуховування дисципліни та здачі ними контролю.

Одне опитування стосується одного виду занять предмету, що викладався одним викладачем. У разі викладання предмету кількома викладачами (наприклад за наявності декількох підгруп однієї академічної групи) буде згенеровано відповідну кількість окремих опитувань.

Після генерації опитування в список запрошених додаються всі здобувачі, котрі приймали участь у прослуховуванні цієї дисципліни у цей семестр.

Одночасно з цим створюється повязанний з опитуванням об’єкт для збереження і швидкого пошуку у подальшому результатом за певним контекстом (факультет, кафедра, викладач тощо) (рисунок 3.3).

Опитування наповнюється запитаннями (переважно з зазделегіть створеного переліку генеруються запитання для кожного окремого опитування).

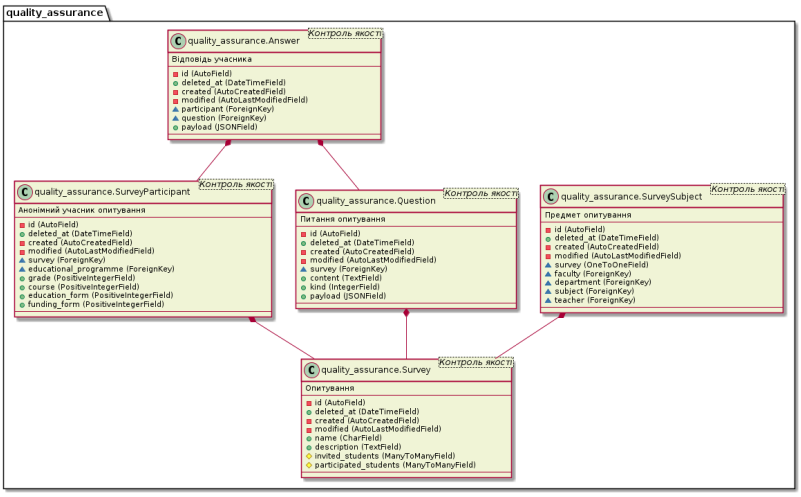

Рисунок 3.3 – Структура моделей даних сервісу контролю якості освітнього процесу

Запитання можуть мати різний вид (за видом відповіді - необхідно обрати серед варіантів, обрати значення на шкалі, коротко відповісти у відкритому вигляді тощо).

Користувачі, котрим адресовано опитування, отримують запрошення до участі в особистому кабінеті. При погодженні прийняти участь у опитуванні для користувача генерується форма (з типу питання та додаткового навантаження, котре рендериться клієнтом відповідно до типу). Наприклад, для питання типу ”Обрання декількох варіантів” у навантаженні буде задано перелік варіантів, мінімальна та максивальна кількість обраних, відповідно до чого буде відображено форму з чекбоксами. У разі запитання типу ”Відкрита відповідь” у навантаженні буде передано максимальну довжину тексту, відповідно до чого буде відображено текстове поле певного розміру.

Після завершення проходження опитування користувач додається в список тих, хто пройшов це опитування. Створюється копія профілю користувача, котра не містить персональних даних (а лише загальну інформацію - спеціальність, курс, групу тощо) і саме з цією анонімною карткою учасника опитування повязуються відповіді користувача. Повторне проходження опитування не доступно, як і внесення змін у відповіді після завершення проходження опитування. Перед завершенням проходження опитування користувачу пропонується перевірити і, у разі необхідності, внести виправлення у надані дані.

Після завершення проходження опитування користувачами адміністраторам ресурсу доступні деперсоніфіковані відповіді учасників, згруповані у звіти необходного формату.

ВИСНОВКИ

Для виконання поставлених завдань було проведено аналіз характеристик існуючих систем планування, зокрема обсяг їх можливостей.

До огляду було включено як зарубіжні, так і вітчизняні рішення. Крім того, було розглянуто Інформаційно-аналітичку систему ХДУ та її місце в бізнес-процесах университету.

Відзначено проблеми розширення існуючої інфраструктури та обгрунтовано необхідність розробки нової системи та її співвідношення з існуючою інфраструктурою.

Виділено основні набори можливостей, що надають розглянуті системи користувачам. Виокремлено ті з них, що необхідні та бажані в системі.

При підготовці до проектування було приділено увагу окремим бізнес-процесам Херсонського державного университету, зокрема на факультеті комп'ютерних наук, фізики та математики.

На основі проведеного аналізу розроблено базові вимоги щодо можливостей серверної частини. Суттєву частину роботи приділено аналізу існуючих технологій всіх рівнів для створення веб-додатків та веб-сервісів. Детально досліджено роботу клієнт-серверних додатків та супутніх технологій. Розглянуто та обгрунтовано використання підходу з використанням RESTfull API при проектуванні системи.

Приділено увагу забезпеченні безпеки сервісів, використано криптографічно стійкі рішення забезпечення доступу до інформації всередені сервісів. Розглянуто теоретичні питання забезпечення безпеки інформаційних систем.

Розглянуто популярні архітектурні рішення до реалізації інформаційних систем та їх необхідність до використання в системі, зокрема відкинуто ідею реалізації мікросервісної архітектури.

Розглянуто питання постійного збереження даних та забезпечення їх консистентності, преведено дослідження доцільності використання нереляційних баз даних в системі.

Як результат відкинуто радикально відмінну графову СУБД на користь традиційній реляційній у звязку з частковою несумісністю з іншими архітектурними рішеннями. В той же час використано NoSQL рішення в системі менеджменту логів.

Відповідно до створених вимог розроблено архітектуру серверної частину спроектованої системи. Після проведення аналізу популярних технологій розробки веб-сервісів для реалізації основної частини системи обрано Django з бібліотекою DRF. Реалізовано структуру бази даних засобами PostgreSQL, моделі з використанням Drango-ORM.

Розроблено публічний прикладний програмний інтерфейс (API), та сформовано документацію до нього. При написанні ключових частин використано спеціальну форму коментарів, що забезпечують інтеграцію опису функцій та їх параметрів в документацію популярних IDE.

Останнє є корисним при подальшій розробці, особливо при використанні існуючої кодової бази сторонніми розробниками, що є можливим, зважаючи на модульність проекту.

Реалізовано менеджмент-команди для маніпуляції даними сервісів адміністраторами системи, зокрема генерації опитувань для користувачів за обраними критеріями та імпорт даних студентів з інших систем.

При розробці проекту використовується система контролю версій git з публічними репозиторіями на сервісах GitHub [7] та GitLab університету [8], що дозволяє використовувати сучасні методи колективної роботи.

Підготовлено рекомендації для подальшої розробки системи. Налаштовано засоби автоматизації прийняття змін від розробників до основної гілки з перевіркої змін на відповдність прийнятим методам колективної розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Basu A.* How to write using rich text format and markdown in latex and overleaf // Universidad de Canterbury. — 2016.
2. *Berners-Lee T.*, *Fielding R.*, *Frystyk H.* Hypertext transfer protocol–HTTP/1.0. — 1996. — URL: `https://www.rfc-editor.org/rfc/pdfrfc/rfc1945.txt.pdf`.
3. *Carter J. L.*, *Wegman M. N.* Universal classes of hash functions // Journal of computer and system sciences. — 1979. — Т. 18, № 2. — С. 143—154.
4. *Codd E. F.* A relational model of data for large shared data banks // Communications of the ACM. — 1970. — Т. 13, № 6. — С. 377—387.
5. *Ed-Douibi H.*, *Izquierdo J. L. C.*, *Cabot J.* OpenAPItoUML: a tool to generate UML models from OpenAPI definitions // International Conference on Web Engineering. — Springer. 2018. — С. 487—491.
6. *Gatherer D.* Less is more: the battle of Moore's law against Bremermann's limit on the field of systems biology // BMC systems biology. — 2007. — Т. 1, S1. — P53.
7. GitHub-репозиторій серверної частини системи. — 2020. — URL: `https://github.com/sen-den/ksu.back`.
8. GitLab-репозиторій серверної частини системи. — 2020. — URL: `gitlab.ksu.tools/dsenchishen/ksu.backend`.
9. *Goddard M.* The EU General Data Protection Regulation (GDPR): European regulation that has a global impact // International Journal of Market Research. — 2017. — Т. 59, № 6. — С. 703—705.

10. *Gupta S.* Building Web Applications with Python and Neo4j. — Packt Publishing Ltd, 2015.

11. *Hellmann D.* The Python standard library by example. — Addison-Wesley Professional, 2011. — C. 1302.

12. *Hillar G. C.* Django RESTful Web Services: The Easiest Way to Build Python RESTful APIs and Web Services with Django. — Packt Publishing Ltd, 2018.

13. *Hoeven H. van der*. ERP and Business Processes. — Llumina Press, 2011. — C. 462.

14. *Jones M.*, *Campbell B.*, *Mortimore C.* JSON Web Token (JWT) profile for OAuth 2.0 client authentication and authorization Grants. — 2015. — URL: `https://tools.ietf.org/html/rfc7523` (дата зверн. 03.11.2020).

15. *Lamport L.* LATEX: a document preparation system: user's guide and reference manual. — Addison-wesley, 1994. — C. 184.

16. *Loeliger J.*, *McCullough M.* Version Control with Git: Powerful tools and techniques for collaborative software development. — O'Reilly Media, Inc., 2012. — C. 144.

17. *Lowe D.* Client/Server Computing for Dummies. — John Wiley & Sons Inc, 1997. — C. 140.

18. *Merkel D.* Docker: lightweight linux containers for consistent development and deployment // Linux journal. — 2014. — T. 2014, № 239. — C. 2.

19. *Preston-Werner T.* Semantic Versioning 2.0.0. — 2013. — URL: `https://semver.org/`.

20. *Rubio D.* REST services with Django // Beginning Django. — Springer, 2017. — C. 549—566.

21. *Torvalds L.*, *Hamano J.* Git: Fast version control system (2010). — 2010. — URL: http://git-scm.com.

22. Web-портал «Збірник наукових праць «Інформаційні технології в освіті» (ІТО)». — 2008. — URL: http://ite.kspu.edu/.

23. *Гетьман В.* Проєктування та розробка сервісної архітектури управління бізнес-процесами університету. Структура безпеки для системи управління університетом. — 2020.

24. *Горбась І.* Програмний комплекс для моніторингу процесів інформаційно-комунікаційної системи. — 2020.

25. *Дейт К. Д.* Введение в системы баз данных. — Вильямс, 2008. — С. 1328.

26. Єдина державна електронна база з питань освіти. — 2020. — URL: https://mon.gov.ua/ua/ministerstvo/yedebo.

27. Закон України «Про захист персональних даних» // Відомості Верховної Ради України (ВВР). — 2010. — № 34. — С. 2297.

28. *Киричек Г. Г.*, *Киричек О. О.* Модель оцінки плагіату програмного коду на основі системи контролю версій // Восточно-Европейский журнал передовых технологий. — 2012. — 2 (2). — С. 25—28.

29. *Кравцов Д.* Web-портал «Херсонський Віртуальний Університет». — 2010. — URL: http://dls.kherson.ua/.

30. *Кучер В.* Мікросервісна архітектура та її особливості. — 2018. — URL: http://eprints.zu.edu.ua/28192/1/34.pdf.

31. *Львов М.*, *Співаковський О.*, *Щедролосьєв Д.* Інформаційна система управління вищим навчальним закладом як платформа реалізації управління академічним процесом // Комп'ютер у школі та сім'ї. — 2007. — № 2. — С. 3—6.

32. *Львовский С.* LATEX: подробное описание. — 2003. — С. 242.

33. *Параничев А.* Опыт проектирования учебного программного продукта с помощью инструментария PlantUML // Системы проектирования, технологической подготовки производства и управления этапами жизненного цикла промышленного продукта (CAD/CAM/PDM–2017). — 2017. — С. 224—227.

34. Постанова Кабінету Міністрів України «Про затвердження переліку галузей знань і спеціальностей, за якими здійснюється підготовка здобувачів вищої освіти». — 2017. — URL: `https://zakon.rada.gov.ua/laws/show/266-2015`.

35. Сервіс «KSU Feedback». — URL: `http://feedback.kspu.edu/`.

36. Сервіс «Пошук книг в електронному каталозі бібліотеки». — 2010. — URL: `http://elibrary.kspu.edu/`.

37. Система дистанційного навчання «KSU Online». — URL: `http://ksuonline.kspu.edu/`.

38. *Ситник Н. В.*, *Краснюк М. Т.* Проектування баз і сховищ даних // навчально-методичний посібник для самост. вивч. дисц. — 2005. — С. 384.

39. *Співаковський О.*, *Вінник М.*, *Тарасіч Ю.* Побудова ІКТ інфраструктури ВНЗ: проблеми та шляхи вирішення // Інформаційні технології і засоби навчання. — 2014. — 39, вип. 1. — С. 99—116.

40. *Співаковський О. В.* Web-портал Херсонського державного університету. — 2013. — URL: `http://kspu.edu/`.

ДОДАТКИ

Додаток А

КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ ХЕРСОНСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ

Я, ***СЕНЧИШЕН ДЕНИС ОЛЕКСАНДРОВИЧ***,
учасник(ця) освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

– дотримуватися:
- вимог законодавства України та внутрішніх нормативних документів університету, зокрема Статуту Університету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:
- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційному репозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;

– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;

– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;

– своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивного іміджу;

– не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;

– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;

– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні й політичні переконання;

– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;

– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науково-дослідницькі завдання;

– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;

– не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;

– не підроблювати документи;

– не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів і співробітників;

– не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійснення впливу на зміну отриманої академічної оцінки ;

– не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;

– не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власності інших студентів та/або працівників;

– не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних з діяльністю університету;

– не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;

– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальності й до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

2020.05.05  Денис Сенчишен