Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Розробка програмного проекту призначеного для інформаційної підтримки роботи мережі спортзалів

Текстова частина до курсової роботи
за спеціальністю „Комп'ютерні науки" 6.050103

Керівник курсової роботи
ст.викладач Вовк Н.Є
_____
(підпис)
"____" _____ 2021 р.

Виконав студент
Федюченко М.І
"____" _____ 2021 р.

Київ 2021

ЗМІСТ

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри мультімедійних систем,
Доцент., к. ф.-м. н. О.П.Жижерун
_____
(підпис)
«___»_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Федюченку Михайлу Ігоровичу факультету інформатики 4-го курсу
ТЕМА Розробка програмного проєкту призначеного для інформаційної

підтримки роботи мережі спортзалів

Зміст ТЧ до курсової роботи:

1. Індивідуальне завдання
2. Календарний план
3. Анотація
4. Вступ
5. Аналіз предметної області
6. Аналіз найкращих практик для проектування та побудови програмного проекту
7. Технічне рішення, щодо архітектури та структури API
8. Огляд створеної системи та використаних технологій
9. Висновки
10. Список використаної джерел

Дата видачі „___” _____ 2021 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Календарний план виконання роботи

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання теми курсової роботи	29.09.2020	
2.	Огляд технічної літератури за темою роботи	18.10.2020	
3.	Виконання аналізу предметної області	21.11.2020	
4.	Написання першого розділу	20.12.2020	
5.	Аналіз найкращих практик для проектування та побудови API	10.01.2021	
6.	Написання другого розділу	30.01.2021	
7.	Проектування базової системи	04.02.2021	
	Розробка API призначеного для інформаційної підтримки мережі спортзалів	01.03.2021	
8.	Написання останнього розділу	01.04.2021	
9.	Створення презентації	06.04.2021	
10.	Корегування роботи згідно із зауваженнями керівника	08.04.2021	

Студент Федюченко М.І.

Керівник Вовк Н.Є.

“_____”_____

Анотація

У даній роботі розглянуто побудову програмного проекту для інформаційної підтримки мережі спортзалів із перспективою подальшого розширення та створення нових сервісів, що дозволять автоматизувати процеси менеджменту спортивної мережі. Окрім того робота розглядає процес створення системи з використанням сучасних технічних рішень, методів та інструментів розробки.

Вступ

Вже декілька років здоровий образ життя захоплює планету і стає супутником сучасної людини. В зв’язку з цим простежується очевидна тенденція на популяризацію способів оздоровлення та підтримки здорового, міцного тіла, наприклад через збільшення кількості спортивних залів та розширення вже існуючих мереж. У свою чергу різке збільшення кількості спортивних залів та спортивного обладнання загалом викликає складність моніторингу та підтримки якості - і, якщо власник бізнесу не в змозі все тримати під контролем, виникає колапс.

Тому головною тенденцією в спортивній сфері на сьогодні є автоматизація процесів як для клієнта, так для персоналу та адміністрації. Аби бути більш клієнто-орієнтованими та привабливими для нової аудиторії – спортивні мережі створюють власні додатки, ведуть автоматизовані бази клієнтів. Все це не тільки спрощує комунікацію, але й допомагає в більш ефективному веденні бізнесу – наприклад, обліки та звіти не на папірцях, а в одному вікні додатку.

Окрім цього, пришвидшення переходу до діджиталізації наштовхнули обмеження, введені під час пандемії. Чим більше функціоналу надає продукт – тим менше живого контакту потрібно.

Тому все більша частина бізнесу переходить на створення додатків, які будуть зручні як клієнтам, так і персоналу. Що у свою чергу покращує ведення бізнесу, залучає нових клієнтів, в умовах карантину дозволяє регулювати потік відвідувачів, дозволяє збирати та зберігати дані для майбутнього бізнес-аналізу та маркетингових досліджень.

Кінцевою метою курсової роботи є готовий програмний проект, що дозволить здійснювати інформаційну підтримку мережі спортивних залів і його можливо буде інтегрувати у вже існуючі бізнес моделі спортивних мереж.

Текстова частина курсової роботи складається з трьох розділів.

У першому розділі виконано аналіз предметної області з визначенням та описом основних бізнес потреб.

У другому розділі зазначено шляхи підвищення ефективності при побудові архітектури REST API, застосовуючи сучасні архітектурні підходи та методи побудови.

У третьому розділі пояснюється процес побудови основних частин REST API на основі досліджених архітектурних моделей.

Постановка задачі.

1. Виконати аналіз предметної області, зокрема:
   - детально визначити призначення системи
   - описати дані предметної області опираючись на досліджені джерела
   - описати групи користувачів
   - визначити функції системи та функціональні вимоги
2. Виконати аналіз архітектурних підходів відповідно до потреб предметної області.
3. Дослідити сучасні архітектурні підходи й аргументувати їх використання для даного програмного проекту.
4. На основі досліджених архітектурних та структурних практик розробити програмний проект.
5. Зробити висновки по результатам експлуатації системи, визначити можливі модифікації та перспективи її використання й інтеграції.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Для побудови цілісної системи та введення її в експлуатацію перш за все потрібно проаналізувати предметну область, адже це допоможе зрозуміти, як краще зберігати та організувати дані. Також це дозволяє передбачити які бізнес процеси потрібні для належного функціонування проекту і відповідно до цього розробити потрібний функціонал та моделі даних.

Оскільки ідея побудови даної автоматизованої інформаційної системи вже могла бути реалізована, доцільним є рішення проаналізувати аналогічні готові системи для визначення можливих проблем, що виникнуть під час експлуатації та створенням нових вимог та функціоналу відповідно до результатів аналізу.

1.1 Призначення системи

Для того щоб керувати великої мережею спортклубів чи спортзалів потрібно вести облік обладнання, журнал реєстру клієнтів, персоналу. У випадках коли для кожного залу ведеться паперовий облік дуже важко проаналізувати динамічну статистику усієї мережі загалом, адже це займає багато часу навіть у випадках коли існують сервіси, окремі для певного клубу. Також це викликає складність для клієнтів, що хочуть відвідати одне з представництв мережі, але не бачать актуальної інформації по обладнанню, наявному персоналу та своєму прогресу тренувань.

Система призначена для ведення обліку відвідування мережі спортзалів, зручного планування клієнтами занять, отримання динамічних статистичних даних по клієнтам, тренерам, спортзалам та проведеним заняття. Також сервіс дозволяє вести облік спорядження у спортзалах.

Відстеження потоку людей у конкретні проміжки часу дозволить оптимізувати робочі години спортзалів. Наприклад, відкриття залу раніше,

якщо попит людей на заняття перед роботою підвищиться. Зручні умови для клієнтів — запис на заняття в обраному спортзалі з обраним тренером (чи без нього) залежно від дати та часу, можливість отримати інформацію про наявне обладнання та його стан, отримання статистики відвідування та порад щодо поліпшення ефективності тренувань, автоматизація системи обліку занять Автоматизований запис задовольнить потреби користувачів щодо безпечного відвідування спортзалів під час карантину. Залучення більшої кількості клієнтів — відправлення повідомлень наявним клієнтам про акції та знижки, облік клієнтів мережі спортзалів. А також дозволить регулювати потік клієнтів. За потреби можливо буде закривати запис, якщо кількість клієнтів на цей проміжок часу перевищуватиме дозволену кількість людей, яку визначає МОЗ[1] у такий карантинний період. Дані щодо відвідування спортзалів допоможуть скласти бізнес-план щодо розширення мережі, для майбутнього бізнес-аналізу та маркетингових досліджень. Якщо у цьому районі спостерігається великий попит, то існує сенс відкрити ще один спортзал чи розширити вже існуючий.

Основні задачі:

- Ведення обліку відвідування клієнтами мережі спортзалів;
- Створення запис на заняття, можливість коригування часу та місця, відміни запису;
- Перегляд зайнятості тренерів;
- Формування гнучкого розкладу тренерів в окремих спортзалах;
- Визначення попиту окремих спортзалів у мережі;
- Отримання статистичних даних про клієнтів, тренерів, спортзали та проведені заняття;
- Ведення обліку спорядження у спортзалах;

[1] Міністерство охорони здоров’я України

1.2 Опис даних предметної області

Аналіз предметної області виконувався через пряме спостереження функціонування існуючих великих мереж спортзалів, аналіз документації окремих спортклубів та спортзалів, що надали зразки журналів реєстрації клієнтів, зразки журналів відвідування, чи паспорти спортзалів, а також шляхом дослідження сайтів подібних організацій та існуючих бізнес процесів потенційних замовників програмного проекту.

1.2.1 Вивчення документації у паперовому вигляді

Було проаналізовано надану за проханням документацію тренажерного залу “Спорткомплекс Сирець” та клубу “FitCurves” до котрої входили такі документи:

- Зразок журналу реєстрації клієнтів
- Паспорт спортзалу
- Зразок журналу відвідування клієнтів та тренерів
- Зразок журналу обліку абонементів

Це допомогло визначити моделі даних, склад атрибутів таких сутностей моделей даних як *Клієнт*, *Тренер, Заняття*.

1.2.2 Дослідження сайту подібної організації

Для визначення додаткових сутностей «Обладнання» та структури побудови зв’язку «Належить» між сутностями «Клієнт» та «Абонемент» було проаналізовано веб-ресурси:

- http://www.gym4u.com.ua/
- http://sportandspa.com.ua
- https://ebsh.ua

Під час аналізу даних веб-ресурсів було визначено додаткові корпоративні обмеження цілісності.

1.2.3 Спостереження функціонування існуючих мереж

Після відвідування представництв двох відомих мереж “ЕБШ” та “Sportlife” було зазначено такі пункти, що визначили додаткові умови реалізації:

- Заняття бувають з тренером або без нього.
- Одному клієнту призначається один тренер.
- Умови доплати.
- Процедура створення графіків.
- Процедура створення занять.

1.3 Опис системи

1.3.1 Опис груп користувачів

Для кожної групи користувачів було складено необхідні функціональні можливості від котрих залежать відповідні бізнес процеси.

Можливості клієнта:

- переглядати список проведених занять;
- дізнатись у якому спортзалі і з яким тренером конкретне заняття;
- побачити список вільних тренерів у конкретному спортзалі у конкретний час і створювати заплановане заняття у конкретному спортзалі (з тренером або без) у конкретний час;
- дізнатись про умови чинного абонементу, побачити список можливих абонементів;
- побачити список та стан обладнання у конкретному спортзалі;
- переглянути інформацію про кожного тренера;

- дізнатись список спортзалів мережі і побачити детальну інформацію про кожен з них;

Можливості тренера:

- створити розклад роботи на будь-який період часу в обраному спортзалі;
- переглянути свій розклад;
- побачити заплановані з ним заняття (інформацію про клієнта з яким заплановане заняття та спортзал, де воно буде відбуватись);
- побачити список проведених занять, та деталі про кожне з них;

Можливості адміністратора спортзалу:

- може підтверджувати здійснення заняття;
- реєструвати нові спортзали або оновлювати інформацію про існуючі;
- реєструвати нових тренерів
- реєструвати нове обладнання або оновлювати інформацію про існуюче;
- бачити статистику відвідування кожного із спортзалів, котрими керує;
- переглядати заняття, що відбуваються у спортзалі, котрим керує;

Можливості анонімного користувача:

- може переглянути спортзали та обладнання в них;
- може отримати список тренерів та переглянути детальну інформацію
- може виконати авторизацію в системі

Варто розуміти, що можливі випадки коли одна й та сама особа буде суміщати різні ролі. Це може відноситися до клієнтів, що одночасно можуть

бути й тренерами, адже можливий випадок, коли тренерам потрібно буде мати доступ до системи у ролі клієнта, для того, щоб прибрати абонемент, якщо буде існувати дана бізнес потреба. Тому такі особи матимуть можливість використовувати увесь функціонал, що передбачається для їх фактичних ролей. Крім того дана потреба створює нову задачу для зменшення кількості облікових записів однієї особи з різними ролями. Адже концепція отримання максимального доступу до необхідного особі функціоналу зі спрощеною авторизацією відповідає клієнто-орієнтованості програмного проекту.

1.3.2 Опис функції системи

Під час аналізу предметної області було виділено основні функції та підфункції системи, що відповідають бізнес потребам:

- Функція ведення обліку занять.
    - Збереження розкладу тренера.
    - Створення заняття.
    - Формування власного заняття
    - Автоматичне формування рахунків за надані послуги.
    - Підтвердження виконання заняття
- Функція менеджменту спортивного залу
    - Ведення обліку обладнання з урахуванням його стану.
    - Формування статистики по окремому залу
- Функція формування особистого розкладу тренером
- Функція ведення реєстру користувачів.
    - Підфункція ведення реєстру тренерів.
    - Підфункція ведення реєстру клієнтів

1.3.2.1 Функціональні вимоги

До зазначених вище функцій системи було сформовано вимоги, котрі накладаються дослідженими та виокремленими бізнес процесами. Зазначено, що для функції «Ведення обліку занять» кожен тренер зобов’язаний заповнювати свій розклад мінімум на 2 тижні вперед чітко обираючи час роботи та місце, особа з роллю клієнт не зобов’язаний створювати заплановані заняття, тобто він може приходити в спортзал і не по запису, тоді заняття формується менеджером спортивного залу, чи клубу, також клієнт має мати можливість заздалегідь переглянути суму доплати запланованого заняття. Проведене, або заняття, що почалося має бути підтверджене вручну наявним у залі менеджером, чи автоматизованим шляхом (через особисту картку клієнта при вході до спортивного залу). Вимоги до функції «Менеджменту спортивного залу» передбачають виконання перевірки спортивного обладнання кожні два місяці для підтримки актуальності інформації про стан обладнання. У вимогах до функції «Реєстру користувачів» зазначено, що при першому відвідуванню спортзалу клієнта реєструють у системі, якщо в нього немає існуючого аккаунту. Також при прийомі на роботу тренера , виконується його реєстрація в системі із заповненням усіх потрібних даних про нього, при звільненні тренера – його обліковий запис видаляється.

2 ВИЗНАЧЕННЯ СПОСОБУ ВИКОНАННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

Після аналізу предметної області стало очевидним, що найбільш практичним буде створення веб-сервісу *Open REST API*, адже цей підхід дозволить інтегрувати систему для різним потенційних клієнтів з мінімальними змінами у технічній складовій програмного проекту. *API* (Application Programming Interface) підтримує взаємодію між двома системами, *API* доволі часто отримує запити чи дані з користувацького інтерфейсу обробляє отримані дані, чи генерує нові у відповідь на запит клієнта. REST – це такі принципи побудови розподілених гіпермедіа систем, котрі мають універсальні способи обробки й передачі станів ресурсів за допомогою HTTP[2]. За умовами, щоб розподілена система була сконструйована за REST архітектурою (RESTful), необхідно, щоб вона відповідала наступним критеріям:

- Client-Server – система має бути розділена на клієнтів та сервери. Тобто сервери ніяк не пов'язані з користувацьким інтерфейсом чи станом , так що вони можуть бути легко масштабовані. Сервери та клієнти можуть замінятися та розроблюватися окремо узгоджуючись через Uniform Interface.
- Cache – у кожній відповіді на запит має бути позначено чи кешується він чи ні, для того щоб клієнт не використовував застарівші, чи некоректні дані у відповідь на майбутні запити.
- Uniform Interface – створення єдиного інтерфейсу між клієнтами та серверами
- Stateless – сервер не повинен зберігати інформацію про клієнта. Запит має усю потрібну інформацію для його обробки, а якщо потрібно й для ідентифікації клієнта

[2] HyperText Transfer Protocol - протокол передачі даних, що використовується в комп'ютерних мережах.

- Layered System – розділення системи на ієрархію рівнів, за умови що кожен рівень може бачити та взаємодіяти з компонентами тільки наступного рівня

Також при такому підході власник бізнесу сам зможе обрати користувацький інтерфейс та за бажанням використовувати веб-застосунок для того, щоб користувачі могли мати доступ до системи з будь-яких пристроїв, що мають доступ до мережі, незалежно від операційної системи.

Вибір технологій розробки став одним з найважливіших факторів, адже зважаючи на поширеність, популярність та актуальність обраних технологій можна розраховувати на залучення якісних спеціалістів для підтримки сервісу, реалізації виникаючих бізнес потреб. Додатково перевагою використання актуальних технологій буде наявність великої кількості матеріалів щодо розробки та підтримки спільнотою певної технології при виникненні складних проблем.

Зважаючи на невеликий в порівнянні з досвідченними розробниками досвід, але існуюче бажання й стимул, було вирішено обрати для розробки API одну з найпопулярніших на 2021 рік мов програмування *Java* за рейтингом, котрий можна побачити на малюнку 1.1. Також сама мова загального призначення здатна підтримувати широкий спектр програм. Тим паче Oracle Corporation активно працює над вдосконаленням та реалізацією функцій, що просить спільнота.

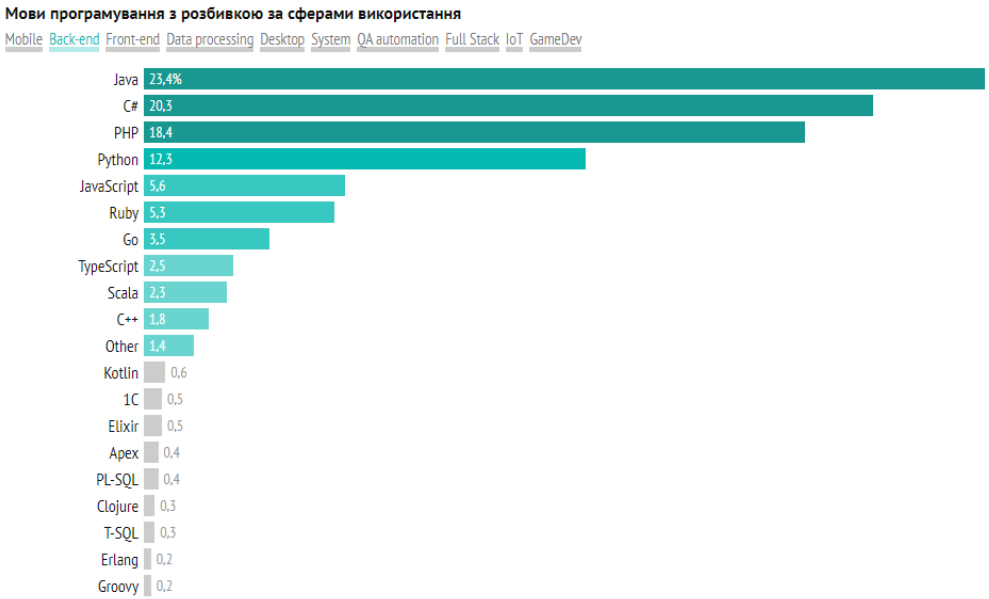


Рисунок 2.1 – рейтинг мов програмування за версією DOU.ua

Чи потрібне використання фреймворку для побудови систему? На дане риторичне питання легко було дати відповідь, маючи попередній досвід розробки програмних проектів без застосування фреймворків. Застосування фреймворку допомагає сфокусуватися радше на реалізації бізнес процесів, аніж на написанні типових шаблонів чи однотипного коду, зменшує час розробки відповідно й ціну розробки, вносить певні стандарти, дає проекту продуманий архітектурний дизайн та структуру. Зважаючи на значні переваги розробки з використанням фреймворків було вирішено використати їх. У ролі основного фреймворку для побудови *API* на мові *Java* було розглянуто *Spring*, *Play* та *Jersey*. Беручи до уваги чудову організацію модулів, якісну екосистему, популярність використання, велику кількість документації, активну підтримку розробниками було обрано *Spring*. Однак для спрощення створення API, зважаючи на слабкі сторони *Spring*, наприклад як особливо складні конфігураційні *XML*[3] файли, під час розробки було використано *Spring Boot*. Значною перевагою використання *Spring Boot* стала відсутність потреби в XML

[3] запропонований консорціумом W3C стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет

конфігураціях та генерації коду, наявність *DevTools* для автоматичного перезавантаження серверу відповідно до оновлень, вже інтегрований Tomcat[4], спрощене керування залежностями, автоматична конфігурація сторонніх бібліотек, наявність власних стартових залежностей для спрощення конфігурації збірки. Для створення збірки та зручного керування залежностями обрано *Maven*. Одна з головних особливостей інструменту атоматизації - декларативний опис проекту. Це означає, що розробнику не потрібно приділяти увагу кожному аспекту збірки - всі необхідні параметри налаштовані за замовчуванням. Зміни потрібно вносити лише в тому обсязі, в якому програміст хоче відхилитися від стандартних налаштувань. Ще одна перевага - гнучке управління залежностями. *Maven* вміє довантажувати в свій локальний репозиторій сторонні бібліотеки, вибирати необхідну версію пакету, обробляти транзитивні залежності.

Розглядаючи моделі даних системи очевидним вибором стало використання реляційних баз даних. Дані передбачається зберігати в СКБД *MySQL*, адже вона ідеально підходить для обробки великих баз даних. Також, це одна з найпоширеніших систем керування базами даних, що означає чудову підтримку з боку мови програмування *Java*.

2.1 Архітектура проекту

Одне з найважливіших питань, що постають перед розробником під час створення REST API це побудова правильної архітектури проекту. Адже це фундамент на котрому будується увесь подальший процес розробки. Проблеми з марним повторенням, що ускладнює підтримку, труднощі з впровадженням нових функцій, нечитабельний код, масштабованість можуть бути вирішені правильно обраної архітектурою. Головні критерії котрим має відповідати правильно побудована структура проекту – це читабельність та чистота коду,

[4] контейнер сервлетів, розроблений Apache Software Foundation.

можливість багаторазового використання, уникнення повторів, можливість інтеграції нових функцій без порушення існуючого коду.

Розроблену структуру проекту можна побачити на малюнку 2.1

Рисунок 2.2 – основна частина структури проекту

Для репрезентації сутностей бази даних та роботи з ними було вирішено використати об’єкти Java Beans, а точніше спеціальний тип об’єктів POJO з певними суворими правилами навколо імплементації об’єкту. POJO (Plain Old Java Object) – це об’єкт, що інкапсулює бізнес логіку. На малюнку 2.3 зображено робочий приклад POJO класу, де контролери взаємодіють із бізнес логікою, котра в свою чергу взаємодіє з POJO для доступу до бази даних. Це підвищує розуміння, читабельність та багаторазовість використання. Java Beans це такий тип POJO, у котрому рівень доступу до полів має бути приватним, доступ до полів виконується тільки через функції *get* та *set*, а сам об’єкт імплементує *Serializable* інтерфейс, що дозволяє зберігати стан (через використання Hibernate таке правило стало зайвим). Такі об’єкти зберігаються в папці *model* та згруповані відповідно до їх бізнес логіки, наприклад *model/workout*. Оскільки може виникати потреба пошуку певних специфікацій сутностей, наприклад залу, що знаходиться у місті *X* на вулиці *Y*, було вирішено у цій же папці *model/specification* розмістити специфікації для пошуку.

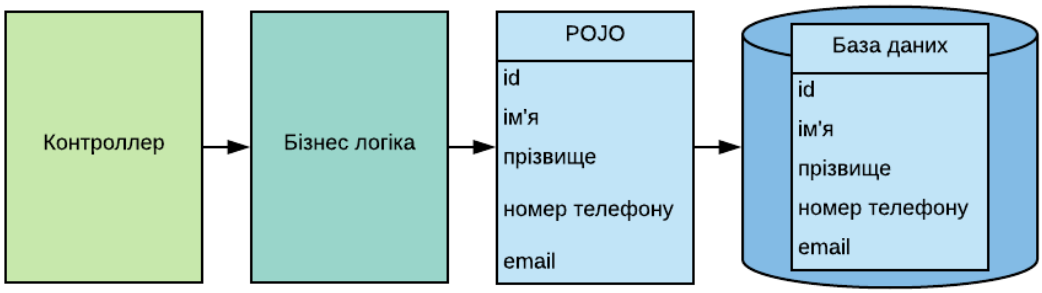


Рисунок 2.3 – діаграма взаємодії з POJO

Для розмежування модельного рівня з рівнем інтерфейсу в реалізації системи було вирішено використовувати об’єкти передачі даних DTO (Data transfer objects). DTO дозволяють передавати тільки ті дані, котрі потрібні для відповіді API, а не увесь об’єкт моделі, який можна було б агрегувати за допомогою декількох під-об’єктів і зберігати в базі даних. Такі об’єкти зберігаються в окремій папці *dto*. Зіставлення моделей даних POJO із DTO виконувалося за допомогою спеціальних допоміжних утиліт, приклади котрих можна знайти у папці *dto/mappers*. З одного боку недоліком використання DTO є певні дублікати коду, потреба в синхронізації DTO та POJO, потреба у використання додаткових інструментів для зіставлення, що збільшує час та ціну розробки. Однак без використання DTO рівень репрезентація/відповідей API буде міцно об’єднаний з доменом (моделями даних).

Об’єкти доступу до даних DAO було розміщено в папці *repository*. DAO (Data Access Object) являють собою об'єкти, або інтерфейси, що використовуються для доступу до сховища даних база даних. Це дозволяє абстрагувати пошук даних від механізму доступу до даних, адже при зміні джерела даних не потрібно нічого переписувати через використання DAO патерну. Такий патерн складається з наступних трьох частин:

- DAO інтерфейсу – визначає список операцій, що можливо виконати над об’єктами моделі

- Конкретний клас DAO – клас, що реалізовує вищевказаний інтерфейс, відповідає за отримання даних з джерела даних незалежно від механізму зберігання, адже це може бути як і база даних, так і xml файл наприклад.
- Value Object – відповідний POJO об’єкт, що містить методи для зберігання даних, отриманих з використанням класу DAO.

Рівень сервісів у котрому реалізовано основна обробка бізнес логіки розміщено у папці *service*. При такій архітектурі сервіси виступають у ролі бар’єру, що ізолює рівень відповідей API і дає додаткову свободу для змін внутрішньої структури моделей даних у майбутньому. Додатково рівень сервісів оперує з моделями даних з різних джерел, наприклад за допомогою DAO, у ньому можуть виконуватися взаємодії зі сторонніми сервісами потрібними для виконання бізнес потреб. За словами Мартіна Фаулера[5]: рівень послуг визначає обмеження програми, він інкапсулює домен. Іншими словами, він захищає домен. Важливо зазначити, що рівень сервісів ніколи не приймає модель даних на вхід й ніколи не повертає її. Рівень контролера взаємодіє зі службовим рівнем (сервісним), кожного разу, коли отримує запит від API, під час того як це відбувається, йому не має бути надано доступу до об’єктів моделі даних, а взаємодія має виконуватися через нейтральні, з точки зору, DTO. Отримавши об’єкт DTO функція сервісного рівня має виконати усі потрібні операції з даним об’єктом й створити відповідь у вигляді DTO, щоб надіслати відповідь контролеру, що його викликав. Рівень сервісів складається із інтерфейсів, що визначають потрібний функціонал. Класи, що імплементують зазначені інтерфейси розміщено у *service/impl*, додатковий сервіс для надсилання листів для підтвердження авторизації розміщений так само тут *service/email*.

[5] британський інженер-програміст, лектор з розробки програмного забезпечення, автор багатьох книг і статей з архітектури програмного забезпечення, об'єктно-орієнтованого аналізу та розробок, мови UML, екстремального програмування, рефакторингу, предметно-орієнтованих мов програмування.

Рівень, що пов’язує сервіс із користувачем починаючи від отриманого запиту, закінчуючи підготовленою та відправленою запитувачу відповіді – рівень контролера розміщено в папці *controller*. Даний рівень не виконує ніякої бізнес логіки, окрім маршрутизації та делегування дій відповідним службам чи сервісам. Вони максимально відокремлені, щоб бути максимально легкими для полегшення тестування.

Усі потрібні налаштування проекту та сторонніх фреймворків розміщено в папці config, але налаштування, що відповідають за безпеку зберігаються в папці *security* приклад організації можна побачити на малюнку 2.4.

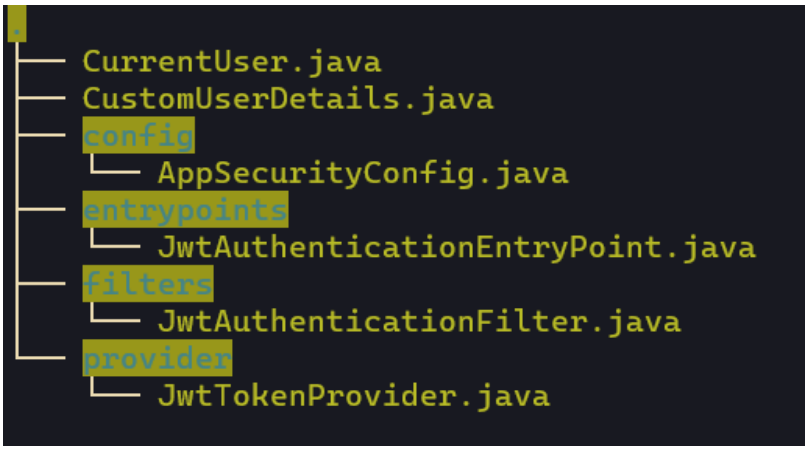


Рисунок 2.4 – структура папки security

Конфігураційний файл, що відповідає за безпеку API розміщено у відповідній папці *security/config* й оскільки механізм аутентифікації було реалізовано через JwtToken, то потрібний фільтр запитів зберігається у папці *security/filter*.

2.2 Проектування ER-моделі

Для конструювання вірної бази даних для програмного проекту, перевірки правильності функціонування та можливості отримання усієї потрібної інформації залежно від вимог бізнес процесу було побудовано модель

даних високого рівня ER-модель (Додаток А) Дана модель була використана для визначення елементів даних та взаємозв’язків потрібних для запланованої системи.

2.2.1 Опис сутностей

2.2.1.1 Спортивні зали

У цій сутності тільки 1 потенційний ключ, що є первинним, адже відповідає умові мінімальності та однозначно ідентифікує спортзал. Сутність «Спортивний зал» репрезентує відділення мережі спортзалів.

2.2.1.2 Обладнання

Дізнатися наявність та стан обладнання у певному спортзалі, побачити його фотографії. У цій сутності тільки 1 потенційний ключ, що є первинним, адже відповідає умові мінімальності та однозначно ідентифікує тренажер.

2.2.1.3 Заняття

Дозволяє отримати список занять з клієнтом, тренером або спортзалом, запланувати заняття, отримати детальну інформацію про дату заняття, місце, дізнатися суму можливої доплати за заплановане заняття. У цій сутності тільки 1 потенційний ключ, що є первинним, адже відповідає умові мінімальності та однозначно ідентифікує заняття. Похідний атрибут «доплата» визначається таким чином: якщо у клієнта є абонемент, що відповідає запланованому заняттю й заняття відбувається без тренера, тоді доплата не потрібна, у разі заняття не за абонементом з тренером, тоді виконується обчислення за формулою: *(кінець заняття – початок заняття)* * *(плата тренера за годину + штраф за годину)*, якщо без тренера але не у години роботи абонементу формула визначається так: *(кінець заняття – початок заняття)* * *штраф* за

годину, у випадку, коли немає дійсного абонементу створення заняття забороняється.

2.2.1.4 Користувачі

Відповідає за аутентифікацію та реєстрацію користувачів у системі. У цій сутності тільки 1 потенційний ключ, що є первинним, адже відповідає умові мінімальності та однозначно ідентифікує користувача.

2.2.1.5 Клієнти

Отримання інформації про клієнта, запланувати заняття для даного клієнта, дізнатися наявність абонементу, побачити статистику тренувань даного клієнту. У цій сутності тільки 1 потенційний ключ, що є зовнішнім ключем й відповідно первинним ключем сутності *Користувач*, адже відповідає умові мінімальності та однозначно ідентифікує клієнта.

2.2.1.6 Абонементи

Клієнт може отримати інформацію про власні абонементи, використовується під час підтвердження заняття та його формуванні. У цій сутності тільки 1 потенційний ключ, що є первинним, адже відповідає умові мінімальності та однозначно ідентифікує абонементи.

2.2.1.7 Тренери

Дозволяє отримати список тренерів за певною специфікацією, дізнатися плату за годину певного тренера, побачити тренера що проводив певне заняття, бере участь у створені заняття з тренером, а також у формуванні розкладу у

спортивному залі. У цій сутності тільки один потенційний ключ, що є зовнішнім ключем й відповідно первинним ключем сутності *Користувач*, адже відповідає умові мінімальності та однозначно ідентифікує тренера.

2.2.1.8 Менеджери спортзалу

Дозволяє керувати відділеннями мережі з котрими має зв’язок, відповідає за підтвердження здійснення заняття, має доступ до статистики спортивних залів. Дана сутність має тільки один потенційний ключ, що є зовнішнім ключем й відповідно первинним ключем сутності *Користувач*, адже відповідає умові мінімальності та однозначно ідентифікує тренера.

2.2.2 Опис зв’язків

2.2.2.1 Опис зв’язку «Спортивні зали» - «Тренери»

Мнемонічне подання зі стислим визначенням: |Спортивні зали|-o-n---працює---m-|-|тренери|

Для отримання годин роботи тренерів у спортзалах, не в кожному спортзалі працюють тренери але в спортзалі можуть працювати кілька тренерів, кожен тренер може працювати у кількох спортзалах, клієнт може подивитися у якому спортзалі працює певний тренер, чи отримати список залів у котрих працює певний тренер у заданий проміжок часу, залежно від дати та часу можна отримати список доступних вільних тренерів.

Пряме відношення має назву: "У ... працює ...". а зворотнє "... працює у ...".

Потужність : один до багатьох, адже у спортзалі може працювати кілька тренерів, а також тренер може працювати у кількох спортзалах за певним розкладом.

Прямий напрям: необов’язково сутність «Спортивні Зали» приймає участь у зв'язку. Адже існують зали у котрих не працюють тренери, наприклад ті що за містом.

Зворотній напрям: обов'язковий «Тренери», адже тренер не має міститися у нашій базі даних, якщо не працює у якомусь спортзалі.

2.2.2.2 Опис зв’язку «Спортивний зал» - «Заняття»

Мнемонічне подання зі стислим визначенням: |Спортивні зали|-o-1---відбувається---n-|-|Заняття|

Для планування заняття у певному спортзалі, для отримання усіх занять у певному спортзалі, для отримання інформації у якому спортзалі відбудеться (відбулося) заняття.

Пряме відношення має назву: "У ... відбувається ...". а зворотнє "... відбувається у ...".

Потужність : один до багатьох (багато до одного), адже у спортзалі може відбуватися кілька занять, а також одне заняття може відбуватися тільки в одному спортзалі.

Прямий напрям: необов’язково сутність «Спортивні Зали» приймає участь у зв'язку. Адже існують зали у котрих не відбувається занять.

Зворотній напрям: обов'язковий «Заняття», адже заняття не може відбуватися не у спортзалі.

2.2.2.3 Опис зв’язку «Спортивний зал» - «Обладнання»

Мнемонічне подання зі стислим визначенням: |Спортивні зали|-|-1---належить---n-|-|Обладнання|

Для перегляду обладнання у певному спортзалі, а також для перевірки стану обладнання, для отримання списку спортзалів залежно від типу обладнання.

Пряме відношення має назву: " ... має ...". а зворотнє "... належить ...".

Потужність : один до багатьох (багато до одного), адже у спортзал має кілька тренажерів, а також один тренажер може належати тільки одному спортзалу.

Прямий напрям: обов’язково «Спортивні Зали» приймає участь у зв'язку. Адже не існує спортивних залів без обладнання.

Зворотній напрям: обов'язковий «Обладнання», адже обладнання має належати якомусь спортзалу.

2.2.2.4 Опис зв’язку «Тренер» - «Заняття»

Мнемонічне подання зі стислим визначенням: |Тренер|-o-1---тренує---o-|-|Заняття|

Для отримання списку усіх занять тренера, для перевірки чи було заняття з тренером.

Пряме відношення має назву: " ... тренує на ...". а зворотнє "... тренує ...".

Потужність : один до багатьох (багато до одного), адже тренер може проводити декілька тренувань, а також одне тренування проводиться тільки одним тренером.

Прямий напрям: необов’язково «Тренери» приймає участь у зв'язку. Адже тренер може й не тренувати.

Зворотній напрям: необов'язковий «Заняття», адже заняття може відбуватися без тренера.

2.2.2.5 Опис зв’язку «Клієнт» - «Заняття»

Мнемонічне подання зі стислим визначенням: |Клієнт|-|-1---відвідує---n-|-|Заняття|

Для отримання статистики відвідування клієнтом занять, для планування заняття клієнтом у певний час з обраним тренером.

Пряме відношення має назву: " ... відвідує ...". а зворотнє "... відвідує ...".

Потужність : один до багатьох (багато до одного), адже клієнт відвідує декілька занять, але одне заняття доступно тільки одному клієнту.

Прямий напрям: обов’язково «Клієнт» приймає участь у зв'язку. Адже щоб бути клієнтом він має запланувати заняття.

Зворотній напрям: обов'язковий «Заняття», адже заняття має відбуватися тільки для певного клієнта.

2.2.2.6 Опис зв’язку «Абонементи» - «Клієнти»

Мнемонічне подання зі стислим визначенням: |Абонементи|-|-n---належить---1-o-|Клієнти|

Для надання клієнту абонемента, чи перевірки його наявності у разі розрахунку доплати.

Пряме відношення має назву: " ... належить ...". а зворотнє "... має ...".

Потужність : багато до одного (один до багатьох), адже ми зберігаємо усі абонементи клієнта, яких може бути декілька.

Прямий напрям: обов’язково «Абонемент» приймає участь у зв'язку. Адже не може існувати абонементів без зареєстрованих власників.

Зворотній напрям: необов'язковий, адже клієнт може користуватися системою без наявності абонементу.

2.3 Реляційна модель даних

На основі ER-моделі було побудовано реляційні моделі даних (Додаток Б, В, Г, Ґ, Д, Е, Є, Ж)

Під час проектування виділено певні обмеження цілісності:

- Години роботи тренера не можуть перетинатися у кількох залах.
- У тренера не може бути запланованих тренувань на години, коли він не працює.
- Клієнт не може запланувати заняття з тренером у певному спортзалі, якщо він там не працює.
- Атрибут «підтвердження заняття» у сутності “Заняття” дорівнює *false*, якщо заняття ще не відбулося.
- Атрибут «доплата» у сутності “Заняття” дорівнює *null*, якщо клієнт з правильним абонементом та займається без тренера.
- Атрибути «початок» у відповідних сутностях мають бути менші за атрибути «кінець»

Додатково до кожної реляційної моделі було визначено цілісність посилань, що є рисою реляційної бази даних , котра полягає у відсутності в будь-якому з її відношень зовнішніх ключів, які посилаються на відсутні кортежі. В базі даних цілісність посилань гарантується завдяки правильно обраним стратегіям для *FOREIGN KEY ON DELETE* у відношеннях.

Також в процесі проектування для підтримки цілісності посилань було визначено обмеження домену – одну з умов структурної цілісності бази даних. Дана умова передбачає визначення кожного атрибуту кожної сутності на своєму домені, тобто на наборі усіх можливих значень, що може мати даний атрибут. Такі обмеження значень можуть включати також визначення певних форматів для полів, відповідність значень бізнес правилам предметної області, чи певним статистичним умовам.

3 ІМПЛЕМЕНТАЦІЯ СИСТЕМИ

3.1 Створення та наповнення бази даних

Раніше під час розробки сервісів та підтримки синхронізації структур баз даних та їх наповнення розробники просто обмінювалися SQL-файлами котрі зберігали схему актуальної бази даних. Схеми баз даних визначають структуру та взаємозв'язки даних, керують реляціями бази даних. Однак такий підхід загрожує виникненням великої кількості колізій, що можуть призвести до проблем з функціюванням системи, чи навіть до втрати даних. Тому схеми що керують формою та межами даних слід обережно застосовувати, щоб відповідати очікуванням системи та уникати втрати даних, що зберігаються в даний час системою баз даних. Одним з підходів, що вирішує проблему синхронізації – це міграції бази даних.

3.1.1 Міграції бази даних

Міграції баз даних, також відомі як міграції схем, міграції схем баз даних або просто міграції, - це контрольовані набори змін, розроблені для модифікації структури об’єктів у реляційній базі даних. Міграції допомагають перевести схеми баз даних із їх поточного стану в новий бажаний стан, незалежно від того, чи це включає додавання таблиць і стовпців, видалення елементів, розділення полів або зміну типів та обмежень. Міграції програмно управляють поступовими, часто оборотними, змінами структур даних. Цілі програмного забезпечення для міграції баз даних - зробити зміни бази даних повторюваними, спільними та перевіреними без втрати даних. Як правило, програмне забезпечення для міграції виробляє артефакти, що описують точний набір операцій, необхідних для перетворення бази даних з відомого стану в новий. Їх можна перевірити та керувати ними за допомогою звичайного програмного забезпечення для контролю версій, щоб відстежувати зміни та

ділитися між членами команди. Зазвичай виконання міграцій – це контрольований процес, що виконує перевірку отриманих сценаріїв змін та внесення будь-яких змін, котрі необхідні для збереження важливої інформації.

Для створення та виконання зазначених міграцій в програмному проекті було використано інструмент міграції баз даних з відкритим кодом *Flyway* шляхом додання його в залежності проекту, приклад якого можна побачити на малюнку 3.1 з додатковими налаштуваннями у *yml*[6] файлі конфігурацій проекту *application-*.yml*

```
    <dependency>
        <groupId>org.mapstruct</groupId>
        <artifactId>mapstruct</artifactId>
        <version>${mapstruct.version}</version>
    </dependency>
# Flyway
flyway:
  enabled: true
  baseline-on-migrate: true
  check-location: true
  encoding: UTF-8
```

Рисунок 3.1 – налаштування Flyway

Код для генерації структури бази даних та її наповнення було розміщено в папці статичних ресурсів *resources/db.migration* у файлах з розширенням *.sql* для зручного написання використовуючи SQL синтаксис, що є спицифічним для бази даних. Приклад файлу з кодом генерації структури бази даних побачити на малюнку 3.2.

[6] зручний для читання людиною формат серіалізації даних, концептуально близький до мов розмітки, але орієнтований на зручність введення-виведення типових структур даних багатьох мов програмування.

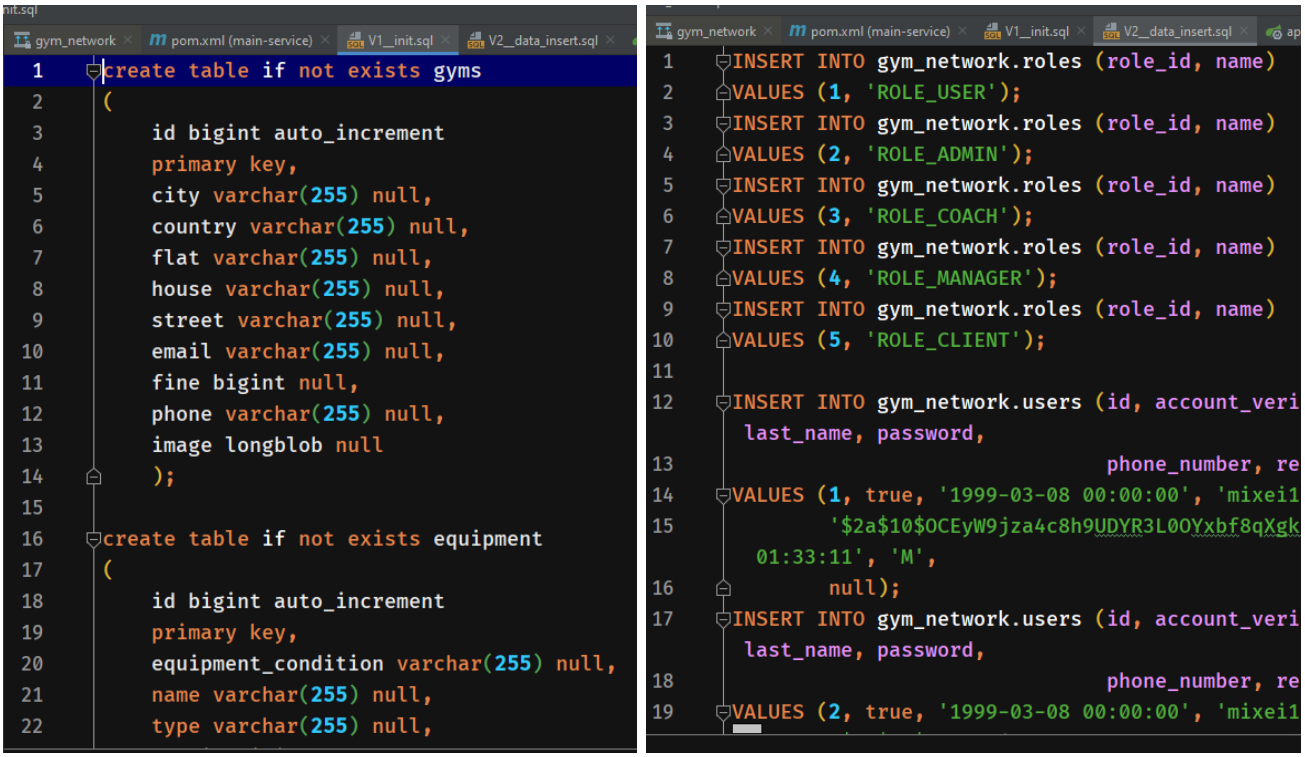


Рисунок 3.2 – приклад коду versioned міграцій

Оскільки *Flyway* підтримує декілька типів міграції, а саме *Versioned*, *Undo* та *Repeatable* правильно названий файл з розширенням *.sql* буде виконувати відповідний тип міграції. Для *Versioned* типу, що потрібен для базвої генерації системи, до назви файлів було додано префікс *V* з числом, що відповідало за версію даного файлу.

Для взаємодії з базою даних було використано *mysql-connctor-java* та відповідні налаштування у конфігураційному файлі проекту. Після запуску міграцій Flyway створив очікувані таблиці та додаткову таблицю *flyway_schema_history* (Додаток З) для відстеження того, які міграції вже були застосовані, коли й ким, а також для відстеження контрольних сум та чи були міграції успішні.

3.2 Створення POJO

Один з найбільших недоліків, що зазначають розробники щодо Java, це кількість повторюваного коду, що може міститися в одному класі. Особливо при створенні *Java Beans* та інших POJO потрібно писати величезну кількість однотипного коду, наприклад для функцій доступу до приватних полів класу, *hashcode, toString, equals*. Це збільшує час та ціну розробки, а також додає додаткову складність у вигляді “шуму” в класах, розробник зосереджується не на імплементації бізнес логіки, а на імплементації допоміжних локальних однотипних функцій.

Для вирішення такої проблеми в проекті було використано плагін компілятора Lombok, котрий дозволяє перетворювати анотації у вихідному коді в Java-оператори до того, як компілятор їх обробить, що зменшує зусилля на розробку й забезпечує додаткову функціональність. Залежності Lombok відсутні в рантаймі, тому використання плагіну не збільшує розміру вихідної збірки. Скориставшись такими анотаціями, як *@Getter*, *@Setter*, *@ToString, @Data*, *@NoArgsConstructor* вищезазначеного плагіну вдалося зробити код набагато зрозумілішим та зручнішим для майбутніх модифікацій (Додаток И).

Технологія програмування, котра зв’язує базу даних з концепціями об’єктно-орієнтованих мов програмування називається ORM (Object-Relational Mapping). Вона призначена для перетворення об'єктів в певний формат для збереження в файлах або базах даних, а також їх подальшого отримання в програмі зі збереженням властивостей об'єктів і відносин між ними. Такі об'єкти називають persistent. ORM звільняє програміста від роботи з SQL-скриптами і дозволяє зосередитися на ООП. .Для збереження POJO об’єктів у базі даних та читання з БД було використано програмний інтерфейс API, що входить до складу Java SE/EE - JPA (Java Persistence API), а точніше реалізацію його інтерфейсу Hibernate. Hibernate і оперує створеними POJO з анотацією *@Entity*, що й поєднує об’єкт з базою даних. Деякі сутності бази даних ьули

декомпозовані, наприклад через анотацію *@Embeddable* для POJO *Address*, адже цей об’єкт має атрибути спільні для декількох сутностей. Атрибути сутностей, що можуть приймати значення з чітко визначеного списку були позначені *@Enumerated(EnumType.String)* та для них були створені колекції статичних констант, наприклад *ESportRang* для поля *rang* класу POJO *CoahEntity*.

3.3 Імплементація рівня DAO

Для спрощення великої кількості типового коду (*boilerplate*), для узгодженості шаблонів доступу до даних та для узгодженості конфігурації було використано модуль Spring - Spring Data. Даний модуль дозволяє повністю видалити реалізації DAO і оперувати єдиним артефактом, що потрібно чітко визначити, а саме інтерфейсом DAO. Для того, щоб почати використовувати модель програмування Spring Data з JPA, інтерфейс DAO було розширено специфічний інтерфейс сховища JPA - *JpaRepository*. Це дозволило Spring Data знайти цей інтерфейс і автоматично створити реалізацію для нього. Розширивши цей інтерфейс, вдалося отримати доступ до найбільш релевантних CRUD[7] методів призначених для стандартного доступу до даних, що доступні в стандартному DAO.

Оскільки коли Spring Data створює нову реалізацію Repository, він аналізує всі методи, визначені інтерфейсами, і намагається автоматично генерувати запити з імен методів – було створено методи, що будуть потрібні для імплементації бізнес логіки. Деякі методи приймають у ролі параметрів специфікацію та пагінацію, для того щоб отримати POJO об’єкти відповідно до бізнес потреби, без створення зайвих додаткових методів чи JPQL[8] запитів, що можна побачити на малюнку 3.3.

[7] create read update delete - 4 основні функції управління даними «створення, читання, оновлення і вилучення».
[8] це платформо-незалежна об'єктно-орієнтована мова запитів, що є частиною специфікації Java Persistence API

CoachRepository.java

```java
package com.spp.gym_network.mainservice.repository;

import com.spp.gym_network.mainservice.model.coach.CoachEntity;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface CoachRepository extends JpaRepository<CoachEntity, Long> {
    Page<CoachEntity> findAll(Specification<CoachEntity> spec, Pageable page);
}
```

Рисунок 3.3 – приклад класу репозиторію

Для автоматичного налаштування *DataSource* у проекті було використано Sprong Boot Starter Data JPA залежність. Після цього явна конфігурація для стандартної програми Spring тепер включена як частина автоконфігурації Spring Boot.

3.4 Сервісний рівень

Реалізація сервісного рівня почалася зі створення інтерфейсів з методами, що будуть задовольняти поставлені бізнес потреби. Імплементація цих інтерфейсів здійснювалася за бізнес логікою виділеною в процесі аналізу предметної області. Кожен клас був згрупований за доменом, котрий був основним для цього класу. Усі класи використовують DAO для отримання та збереження інформації до джерела даних. Додатково сервіси можуть взаємодіяти з іншими сервісами через інкапсульовані об’єкти відповідних сервісів.

Наприклад сервіс, що відповідає за реєстрацію користувачів, верифікацію реєстрації користувачів та отримання інформації по обраному користувачу – це *UserService*. На Додатку І можна ознайомитися з реалізацією метода *register()*, що виконує реєстрацію користувача в системі та реалізацією допоміжного

метода *sendRegistrationConfirmationEmail()* для надсилання листа підтвердження. Спочатку виконується перевірка, чи існує користувач в системі шляхом звернення до відповідного UserRepository у випадку реєстрації користувача з аналогічної поштовою адресою кидається завчасно створена помилка *UserAlreadyExistException*, котра розширює клас *Exception* з пакету *java.lang*. В іншому випадку створюється POJO об'єкт користувача та зберігається до бази даних з паралельним створенням токену й надсилання листа з токеном для верифікації реєстрації за допомогою сервісу розсилки *EmailService*. Після того як користувач отримав листа з підтвердженням й перейшов за посиланням контролер передає токен сервісу користувача, що у разі успішної перевірки токену активує аккаунт користувача.

Додатково сервісні класи використовують об'єкти мапери для перетворювання DTO у POJO та навпаки. Потреба у таких об'єктах виникла через помічену велику кількість надмірної логіки по перетворенню у самому сервісі, адже воно погіршувало читабельність та розуміння коду загалом. Через те, що DTO та Java Bean (POJO) дуже пов'язані і зміни в одному об'єкті потребують змін в іншому, щоб не додавати ще одну додаткову залежність у вигляді класу меперу, а також для зменшення кількості рутинних завдань та можливих помилок було вирішено використати генератор коду MapStruct. MapStruct – це процесор анотацій, що підключений до компілятору Java і використовується з у збірці з командного рядка Maven. На відміну від інших фреймворків відображення він генерує зіставлення компонентів під час компіляції, що забезпечує високу продуктивність та ретельну перевірку помилок. Наприклад зайву логігу по шифруванню картинок для передачі клієнту так само було перенесено з сервісу до мепперу як в прикладі інтерфейсу *UserMapper*.

3.5 Рівень контролерів

Рівень контролерів, що відповідає за взаємодію клієнта та API був розроблений таким чином, щоб усі потрібні бізнес потреби можна було виконати з мінімальною кількістю запитів, але й без нагромадження складного коду, що важко підтримувати. На малюнку 3.4 можна побачити згруповані можливі запити до створеного REST API для отримання або внесення змін.

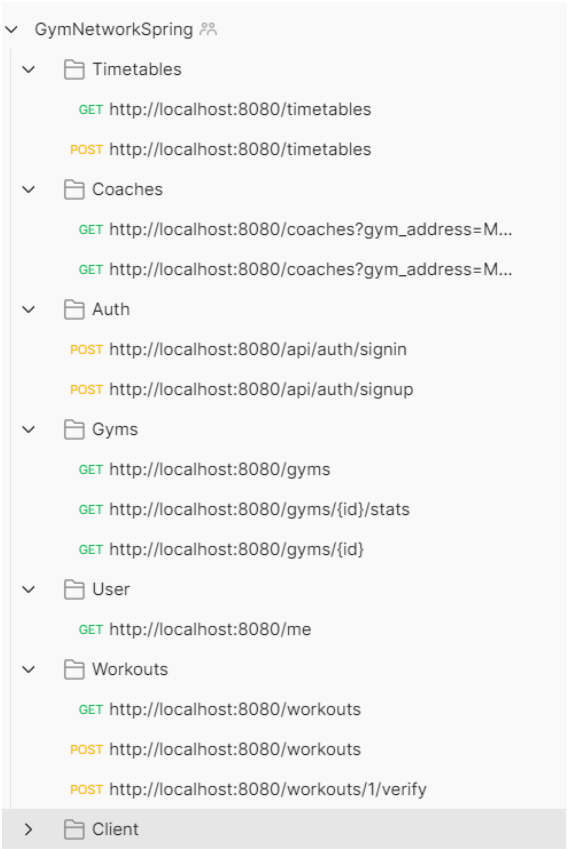


Рисунок 3.4 – workspace програми Postman із згрупованими запитами

Усі класи контролерів позначені анотацією @RestController, що є спиціалізацією анотації @Component, котра дозаоляє автоматично визначити класи реалізації через сканування шляхів до класу. У методах контролерів за допомогою анотації *@Valid* вимагається валідація DTO, що приходять на вхід. Для валідації запитів використовується залежність *spring-boot-starter-validation* й відповідні анотації у класах DTO. Наприклад під час реєстрації користувача

вимагається наявність імейлу та паролю в запиті клієнта, з імплементацією можна ознайомитися на малюнку 3.5

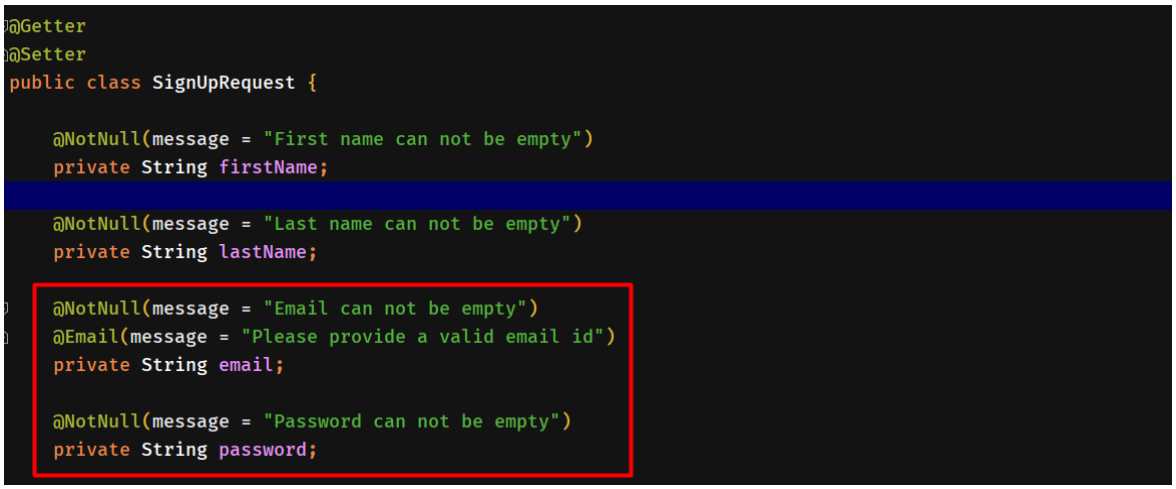


Рисунок 3.5 – приклад валідації DTO

Для деяких бізнес потреб, наприклад для отримання усіх тренерів, котрих звуть Андрій чи списку спортзалів, що знаходяться у місті Київ вимагалося створення ефективного способу пошуку за критеріями з підтримкою пагінації. З використанням Spring Data доволі просто створити репозиторій з власним методом пошуку, наприклад по імені, а далі відповідно створити контролер, що приймає HTTP query параметр ім’я тренера. На жаль, доводиться вручну керувати іншими параметрами запиту, щоб визначити відповідні методи сховища. Хоча це не велика проблема для одного параметра запиту, цей підхід стає неприйнятним, коли є більше змінних. Припустимо, що потрібно відфільтрувати тренерів за іменем та прізвищем, а також їх рангом. Усі фільтри стають необов’язковими. Це призводить до перенасичення та неприємної кількості логічних умовних конструкцій в контролері. Вирішенням даної проблеми може бути використання інтерфейсу *Specification* введеного Spring Data, що може використовуватися разом із репозиторієм. *Specification* - це простий інтерфейс, який надає метод *toPredicate*, що повинен повертати предикат критеріїв JPA2. Дане рішення набагато краще, але все ж залишається проблема у рутинній роботі по написанню власних специфікацій, а потім їх

обробці. А якщо відбудуться зміни у POJO до котрих дана специфікація відноситься, то доведеться вносити зміни й у саму специфікацію. Тому для вирішення даної проблеми у проекті було використано API, що надає власний *HandlerMethodArgumentResolver*, який перетворює параметри HTTP в об'єкт *Specification*, готовий до використання зі сховищами Spring Data. Наприклад для отримання тренерів, що мають розклад в певні проміжки часу було створено анотацію на малюнку 3.6. Де для порівняння часу використовуються спеціальні класи *LessThanOrEqual* та *GreaterThanOrEqual* від API Specification Argument Resolver, що автоматично в рантаймі згенерує імплементації інтерфейсів специфікацій базуючись на доданих анотаціях.

```
@Join(path = "timetables", alias = "t")
@And({
        @Spec(path = "payment", spec = Like.class),
        @Spec(path = "rang", spec = Like.class),
        @Spec(path = "email", spec = Like.class),
        @Spec(path = "t.gym.id", params = "gym_id", spec = Equal.class),
        @Spec(
                path = "t.startTime",
                params = "start_time",
                spec = LessThanOrEqual.class,
                config = "yyyy-MM-dd HH:mm:ss"
        ),
        @Spec(
                path = "t.endTime",
                params = "end_time",
                spec = GreaterThanOrEqual.class,
                config = "yyyy-MM-dd HH:mm:ss"
        ),
})
```

Рисунок 3.6 – приклад анотації для генерації специфікації

3.6 Аутентифікація користувачів

Механізм ідентифікації та подальшої аутентифікації користувачів відбувається за допомогою JWT токену. JWT або JSON Web Tokens - це

відкритий стандарт, який визначає компактний та автономний спосіб безпечної передачі інформації між сторонами через об’єкт *JSON*[9]. Цю інформацію можна перевірити та довіряти їй, оскільки вона має цифровий підпис. JWT можна підписати, використовуючи secret (з алгоритмом HMAC[10]) або пару відкритих / приватних ключів, використовуючи RSA[11] або ECDSA[12]. Він в основному використовується для захисту REST API. В нашій реалізації secret та час життя токену зберігається у файлі конфігурації yml. Оскільки JSON менш багатослівний, ніж XML, при кодуванні його розмір також менший, що робить JWT компактнішим, ніж *SAML*[13]. Це робить JWT хорошим вибором для передачі в середовищах HTML і HTTP. У процесі автентифікації JWT клієнт спочатку надсилає деякі облікові дані для автентифікації (ім’я користувача та пароль). Потім Spring додаток перевіряє ці облікові дані, і якщо вони дійсні, генерує JWT і повертає їх. Після цього кроку клієнт повинен надати цей токен у заголовку авторизації запиту у формі «Bearer TOKEN». Бек-енд перевіряє дійсність цього токену та авторизує або відхиляє запити (Додаток І). Токен може також зберігати ролі користувачів та авторизувати запити на основі даних повноважень, але в реалізації програмного проекту у контролері ми отримуємо облікові дані користувача і в сервісі визначаємо його роль. Код, що відповідає за автентифікацію користувачів у системі зберігається окремо в папці *security*.

3.7 Обробка помилок

Для правильної обробки помилок, що можуть виникати в рівні контролера, так і помилок, що виникають на рівні бізнес логіки було створено клас *RestExceptionHandler*, щорозміщується в папці *exception*. Даний клас

[9] текстовий формат обміну даними між комп'ютерами
[10] механізм перевірки цілісності інформації, що передається або зберігається в ненадійному середовищі.
[11] криптографічний алгоритм з відкритим ключем, що базується на обчислювальній складності задачі факторизації великих цілих чисел.
[12] алгоритм з відкритим ключем для створення цифрового підпису, аналогічний за своєю будовою DSA, але визначений, на відміну від нього, не над кільцем цілих чисел, а в групі точок еліптичної кривої.
[13] Security Assertion Markup Language - мова розмітки, заснована на мові XML.

використовує анотацію, котра була введена у Spring 3.2 @ControllerAdvice. Дана анотація робить цей клас глобальним обобником усіх помилок. Це дає можливість механізму, який відмовляється від старої моделі MVC і використовує ResponseEntity разом із безпекою типів та гнучкістю @ExceptionHandler. Насправді механізм надзвичайно простий, але також дуже гнучкий. Це дає повний контроль над тілом response, а також кодом стану. Він забезпечує відображення кількох винятків до одного і того ж методу, які слід обробляти разом. А також добре використовує новіший response RESTful ResposeEntity (Додаток Ї)

3.8 Документація API

Розробка RESTful API створює певну область проблем і потреб, яка виходить за рамки реалізації лише ендпоїнтів. У системи будуть клієнти, що її використовуватимуть. Клієнти повинні знати, як взаємодіяти зі створеним API. Таким чином, документація щодо API стає більш критичною. Документація API повинна бути структурована таким чином, щоб вона була інформативною, стислою та легкою для читання. Тому для генерації документації для Spring Boot програмного проекту було використано *Swagger 2*. *Swagger 2* - це проект з відкритим кодом, який використовується для опису та документування RESTful API. *Swagger 2* є мовно-агностичним і розширюється на нові технології та протоколи, не тільки HTTP. Поточна версія визначає набір ресурсів HTML, JavaScript та CSS для динамічного генерування документації із сумісного з *Swagger* API. Ці файли групуються проектом інтерфейсу *Swagger* для відображення API у браузері. Окрім надання документації, інтерфейс користувача *Swagger* дозволяє іншим розробникам API або користувачам взаємодіяти з ресурсами API, не маючи жодної логіки реалізації. Специфікація *Swagger 2*, яка відома як специфікація *OpenAPI*, має кілька реалізацій. В даний

час *Springfox*, який замінив Swagger-SpringMVC є найпопулярнішим для програм Spring Boot і використовується в даному проекті (Додаток Й).

3.9 Створення механізму верифікації заняття

Бізнес процес верифікації запланованого заняття покладається на роботу менеджера, або адміністратора спортивного залу в котрому відбувається заняття. Тобто коли клієнт приходить на заняття у запланований час менеджер спортивного залу має підтвердити, що заняття почалося скориставшись ендпоїнтом *workouts/{id}/verify*. Проте даний процес вводить багато зайвих обмежень й загрожує виникненням великої кількості невідповідностей у системи у випадку, якщо адміністратор не верифікував заняття. Тобто для зведення людського впливу й відповідно людської помилки в системі потрібно було розробити певний механізм для автоматизації процесу верифікації.

Під час аналізу даної проблеми було переглянуто бізнес процес верифікації запланованого заняття й додано додаткові умови. Розглядається, що клієнт, котрий відвідує відділення мережі спортзалів має при собі активний абонемент. Картка абонементу представляє собою RFID[14] з частотою 13.56 МГц та пам’яттю EEPROM[15] 1 кб на котрій нанесена додаткова інформація про умови дії даного абонементу. Коли клієнт заходить до відділення спортивного залу мережі він підносить абонемент до спеціального RFID-модулю, що зчитує інформацію про UID[16] картки виконується перевірка, чи він має заплановане заняття в даний час шляхом комунікації з побудованим REST API через ендпоїнт clients/verification. Оскільки номер UID картки зберігається в сутності абонементу це дозволяє однозначно підтвердити чи має право клієнт займатися в залі. Також при втраті картки, якщо клієнт її загубив –

[14] радіочастотна ідентифікація
[15] постійний запам'ятовувач, що програмується та очищується за допомогою електрики, один з видів енергонезалежної пам'яті.
[16] унікальний ідентифікатор

є можливість змінити номер UID зберігши відповідний абонемент. Також при потребі це дає можливість у мабутньому розробити додатковий сервіс, що буде перевіряти, чи має даний клієнт з відповідним абонементом доступ до спортивного залу, котрий хоче відвідати, при виникненні такої бізнес потреби

3.9.1 Моделювання бізнес процесу з розумним сканером

Для моделювання бізнес процесу було використано плату Arduino. Arduino - це електронна платформа з відкритим кодом, заснована на простому у використанні апаратному та програмному забезпеченні. Плати Arduino здатні зчитувати входи - світло на датчику, палець на кнопці - і перетворювати його на вихід - активуючи двигун, включаючи світлодіод, публікуючи щось в Інтернеті. Можна сказати платі, що робити, надіславши набір інструкцій мікроконтролеру на неї.

У ролі головного мікроконтролера було використано найпопулярнішу плату з сімейства Arduino – UNO. Вона містить усе потрібне для підтримки мікроконтролера і її легко підключити до комп’ютера за допомогою кабелю USB або до адаптеру змінного/постійного струму або акамулятора. Оскільки у використаній стандартній версії UNO немає модулю WiFi[17], для під’єднання до мережі та взаємодії з REST API додатково було використано WiFi модуль ESP8266 версії ESP-01S на базі мікросхеми ESP8266EX, а для спрощення його підключення додатково модуль адептера для нього зі стабілізатором напруги 3.3 В. Для сканування RFID карток використовувався заснований на мікросхемі MFRC522 RFID-модуль 13.56 МГц з SPI-інтерфейсом. Даний модуль також може бути використаний для різних радіоаматорських і комерційних застосувань, в тому числі для контролю доступу, автоматичної ідентифікації, робототехніки, відстеження речей, платіжних систем. Після створення схеми

[17] загальновживана назва для стандарту IEEE 802.11 передачі цифрових потоків даних по радіоканалах

з’єднань (Додаток К) було відтворено схему у живу, що можна побачити на малюнку 3.7.

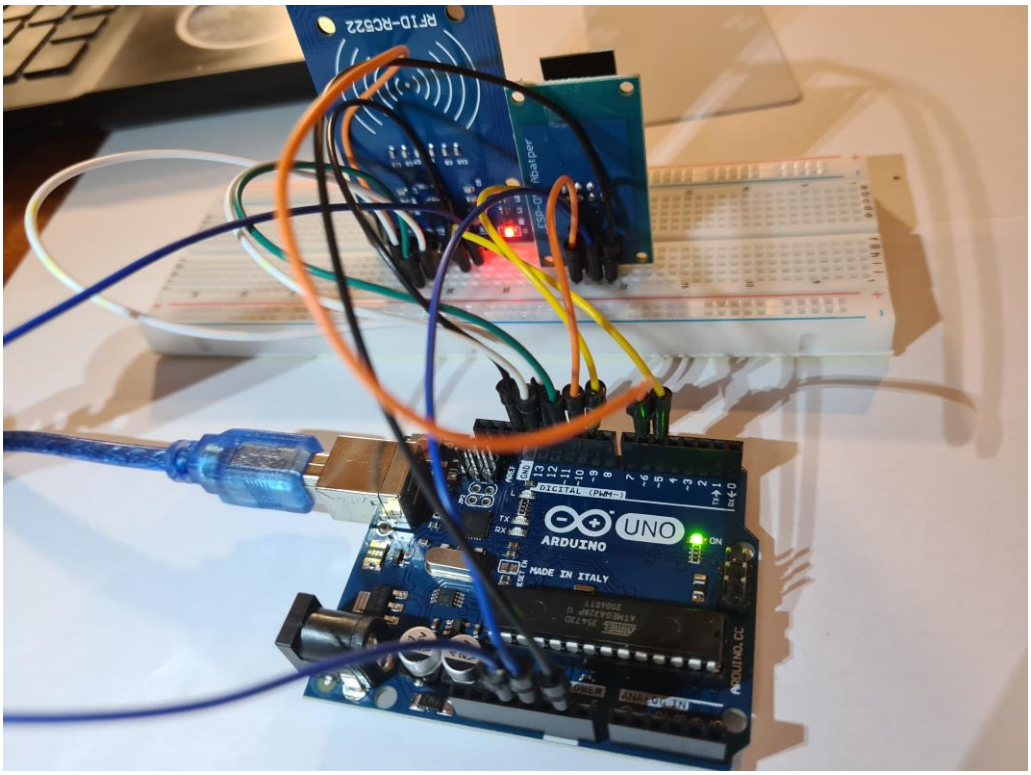


Рисунок 3.7 – фотографія реалізованої схеми на базі мікроконтролера Arduino

Код до мікроконтролеру було написано на мові *C++* у Arduino IDE[18]. Мікроконтролер виконує з’єднання з локальною WiFi мережею через модуль ESP8266, далі у разі отримання сигналу про зчитування картки від RFID модулю виконується post [19]запит для верифікації заняття користувача.

На стороні програмного проекту виконується перевірка, чи існує заняття для даного клієнта, чи абонемент з UID номером дійсний і при успішній перевірці підтверджує заняття. Код для прошивання мікроконтролера зберігається у кореневій папці проекту *microcontroller-rfid.ino*.

[18] це крос-платформний додаток, що використовується для написання та завантаження програм на Arduino-сумісні плати

[19] один з багатьох методів запиту, що підтримуються протоколом передачі даних HTTP, який використовується у всесвітній мережі Інтернет.

3.10 Докеризація програмного проекту

Для того, щоб легко збирати, доставляти програмний проект без додаткових налаштувань операційної системи та компонент з котрими буде працювати програмний проект варто скористатися Docker контейнерами. Контейнери дозволяють ізолювати код в один контейнер. Це полегшує модифікацію та оновлення програми. Контейнери працюють дуже схоже до віртуальні машини, але набагато більш конкретно і детально. Вони ізолюють одну програму та її залежності - усі зовнішні бібліотеки програмного забезпечення, які потрібна для запуску програми, - як від базової операційної системи, так і від інших контейнерів. Усі контейнерні програми мають єдину загальну операційну систему (Linux[20] або Windows[21]), але вони розділені між собою та в цілому від системи. Тому для докеризації програмного проекту було створено Dockerfile зі списком відповідних команд, що використовує профайл docker, щоб застосувати *application-docker.yml* конфігурації. А для того, щоб зменшити можливість виникнення помилок під час приєднання до бази даних та одночасного керування декількома контейнерами застосовано інструмент, що входить до складу Docker – Docker Compose. Відповідні конфігурації знаходяться у файлі docker-compose.yml

3.11 Результати, недоліки та перспективи використання API

Попри відсутність можливості запуску системи з реальними даними та на справжній мережі спортивних залів, вдалося успішно протестувати усю розроблену бізнес логіку на тестових даних. Механізми авторизації були зручними для клієнта, що полегшило розробку клієнтської частини. Проаналізовано, що механізми автоматизації бізнес процесів у котрих був

[20] загальна назва UNIX-подібних операційних систем на основі однойменного ядра.
[21] узагальнююча назва операційних систем для комп'ютерів, розроблених корпорацією Microsoft.

людський фактор зменшили ризики виникнення помилок. Моніторинг самої системи став зручнішим для потенційного клієнта.

Під час моделювання бізнес процесів виникла потреба у модифікації існуючих та створенні додаткових ендпоїнтів, щоб задовільнити можливі потреби. Так виникла потреба у блокуванні можливості для реєстрації користувачів, що забронювали певне число занять, котрі не відвідали, тобто вони не були підтверджені. Після презентації системи потенційному замовнику виник додатковий бізнес процес для формування автоматичного заняття використовуючи механізм сканування RFID. Тобто розрахування часу заняття відповідно до того, коли користувач увійшов до спортивного залу та вийшов з нього з позначенням, чи було заняття сплачено, чи ні. Додатково виникла потреба у додані до розкладу тренера фіксованих класів занять (TRX, Yoga, Stretching, Box) й визначенні в абонементі, чи має абонемент доступ до обраного класу при формування заняття.

Після аналізу готового програмного проекту було помічено, що для полегшення підтримки, створення нових сервісів варто було б виконати декомпозицію REST API на декілька мікросервісів: сервісу, що буде відповідати за формування та створення занять та сервісу, котрий збиратиме потрібні статистичні дані у котрому можна було б використати технології GraphQL.

Перспективи використання сервісу доволі високі, адже двоє з трьох потенційних замовників підтвердили потребу в такому сервісі, для інтеграції у свою бізнес модель. Сам сервіс працює незалежно, тобто на нього не потрібно витрачати кошти окрім як на підтримки та розміщення на локальних серверах клієнтів, чи у хмарі. Однак для якісного функціонування потрібні доопрацювання та створення додаткових сервісів, що задовільнять нові бізнес потреби замовників. Додатково виникає потреба у покритті тестами коду для виявлення технічних помилок, що не були передбачені розробником, даний процес був частково пропущений під час розробки програмного проекту через обмеженість у часі.

Висновки

Для побудови цілісної, надійної системи потрібно виконати повний та детальний аналіз предметної області для котрої ця система буде використовуватися. Дуже важливо передбачити усі можливі бізнес потреби, котрі будуть виступати фундаментом для майбутнього програмного проекту. У роботі було розглянути процес створення програмного проекту для інформаційної підтримки мережі спортзалів.

Аналіз предметної області допоміг визначитися з технологіями та архітектурою створеного API, перейняти досвід та хороші ідеї аналогічних систем для впровадження та подальшої модифікації, а також передбачити можливі ризики під час експлуатації.

Побудова продуманої архітектури не менш важливий крок, адже відповідно до поставлених задач архітектура й структура програмного проекту може змінюватися і важливо розробити архітектуру так, що кількість цих змін була мінімальною, а якість програми, швидкість, час розробки та ціна відповідали очікуванням. Окрім того розроблена система передбачає можливість інтеграції з додатковими сервісами без великих змін у програмному коді завдяки багаторівневій структурі. Зазначені сервіси можуть комунікувати з REST API через HTTP чи інші протоколи. До кожного рівня представлено аргументацію та пояснення його побудови, котра в більшості випадків залежить від потреб у гнучкості, масштабованості та легкості підртримки.

Для зменшення людського фактору та автоматизації процесів було спочатку змодельовано та пізніше зібрано пристрій на основі Arduino UNO, що має власний програмний код і спілкується з API шляхом HTTP.

Розробка REST API з використанням фреймворку Spring та інших сучасних методів розробки спростила побудову системи, а також додала гнучких засобів для імплементації потрібної бізнес логіки. Важливо зазначити,

що з розвитком технологій продовжують розвиватися та виникати нові архітектурні підходи й засоби для полегшення їх реалізації.

Успішна інтеграція програмного проекту в бізнес модель реальної мережі спортзалів справді дозволить покращити менеджмент системи та поліпшити досвід відвідування відділень мережі клієнтами.

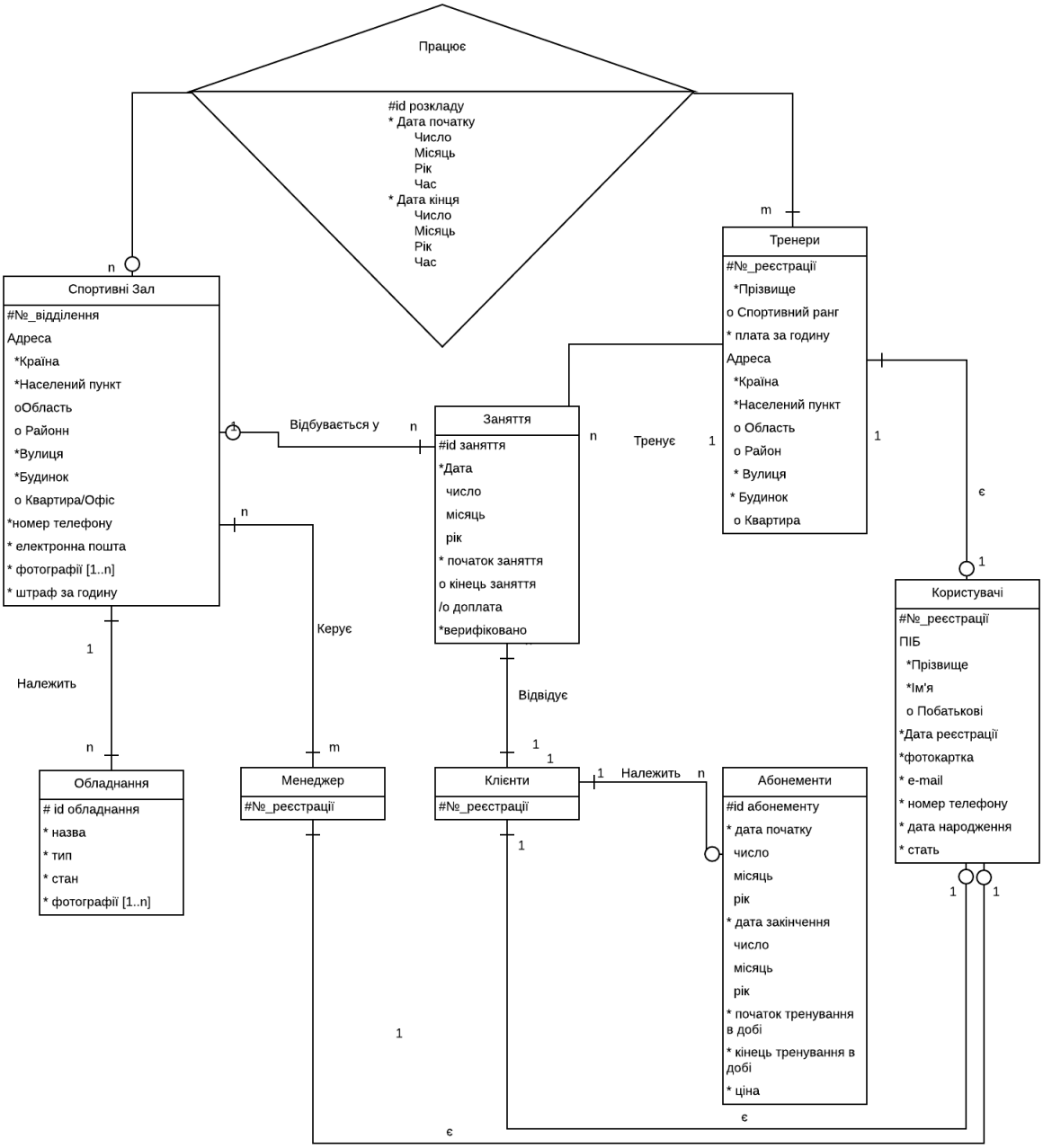
Список літератури

1. Рейтинг мов програмування 2021 [Електронний ресурс] 2021 Автор Руслан Шевченко https://dou.ua/lenta/articles/language-rating-jan-2021/
2. Електронна документація проекту Lombok [Електронний ресурс] 2009-2021 The Project Lombok Authors https://projectlombok.org/features/all
3. Електронна документація проекту Flyway [Електронний ресурс] 2020 Red Gate Software Ltd https://flywaydb.org/documentation/
4. Introduction to databases [Електронний ресурс] 2021 https://www.prisma.io/dataguide/types/relational
5. REST: простою мовою [Електронний ресурс] 8 лютого 2019 автор Андрій Іващенко https://medium.com/@andr.ivas12/rest-%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%8B%D0%BC-%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%BC-90a0bca0bc78
6. Електронна документація проекту MapStruct [Електронний ресурс] MapStruct authors 2020 https://mapstruct.org/documentation/stable/reference/html/
7. JWT handbook Себастьян Пейрот [версія 0.13.0] 5 ст. – What is a JSON WebToken.
8. Електронна документація проекту OpenAPI [Електронний ресурс] 2021 SmartBear Software https://swagger.io/specification/v2/
9. Електронна документація до фреймворку Spring [Електронний ресурс] 2021 автори Phillip Webb, Dave Syer, Josh Long, Stéphane Nicoll, Rob Winch, Andy Wilkinson, Marcel Overdijk, Christian Dupuis, Sébastien Deleuze, Michael Simons, Vedran Pavić, Jay Bryant, Madhura Bhave, Eddú Meléndez, Scott Frederick https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

Додаток А

(обов’язковий)

ER-модель системи



Додаток Б

(обов’язковий)

таблиця «Працює»+ «Працює відповідно ER»

ВІДНОШЕННЯ «ПРАЦЮЄ» (відповідає зв’язку «Працює» у ер-моделі)							
№	**Ключ**	**Ім’я атрибуту**	**Тип**	**NULL / NOT NULL**	**Цілісність посилань (ЦП)**		**Пояснення та опис ЦП**
					ON DELETE	**ON UPDATE**	
1.	PK	timetable_id	Long	NOT NULL			
2.		day	Integer	NOT NULL			Набуває значення від 1 до 7.
3.		start_time	Time	NOT NULL			Час початку робочого дня
4.		end_time	Time	NOT NULL			Час кінця робочого дня. Не може бути раніше ніж start-time
5.	FK	coach_id	Integer	NOT NULL	CASCADE	NO ACTION	
6.	FK	gym_id	Integer	NOT NULL	CASCADE	CASCADE	

Додаток В

(обов’язковий)

Таблиця «Тренери»+ «Тренери відповідно ER»

ВІДНОШЕННЯ «ТРЕНЕРИ» (відповідає сутності «Тренери» у ер-моделі)							
№	Ключ	Ім’я атрибуту	Тип	NULL / NOT NULL	Цілісність посилань (ЦП)		Пояснення та опис ЦП
					ON DELETE	ON UPDATE	
1.	PK	coach_id	Integer	NOT NULL			Табельний номер тренера
2.		lastname	Varchar (255)	NOT NULL			Прізвище тренера. Частина скаладеного атрибуту «ПІБ»
3.		firstname	Varchar (255)	NOT NULL			Ім'я тренера. Частина складеного атрибуту «ПІБ»
4.		middlename	Varchar (255)	NULL			Частина складеного атрибуту «ПІБ». По батькові.
5.		email	Varchar (255)	NULL			Необов’язковий атрибут.
6.	AK	phone	Varchar (255)	NOT NULL			Обов’язковий атрибут. Є альтернативним ключем відношення “Тренер”
7.		country	Varchar (255)	NOT NUL			Країна проживання тренера. Частина «Адреса»
8.		town	Varchar (255)	NOT NULL			Населений пункт, де проживає тренер. Частина складеного атрибуту «Адреса»
9.		area	Varchar (255)	NULL			Область, де проживає тренер. Частина складеного атрибуту «Адреса»
10.		street	Varchar (255)	NOT NULL			Вулиця, де проживає тренер. Частина складеного атрибуту «Адреса»

11.		building	Varchar (255)	NOT NULL			Номер будинку, де проживає тренер. Частина складеного атрибуту «Адреса»
12.		flat	Integer	NULL			Номер квартири, де проживає тренер. Частина складеного атрибуту «Адреса»
13.		sport_rang	Varchar (255)	NULL			Спортивний ранг тренера. Необов’язковий атрибут.
14.		payment	Currency	NOT NULL			Оплата за годину тренеру (в гривнях).
15.		sex	Boolean	NOT NULL			Стать тренера. Чоловік – true, жінка – false.

Додаток Г

(обов’язковий)

Таблиця «Клієнти»+ «Клієнти відповідно ER»

ВІДНОШЕННЯ «КЛІЄНТИ» (відповідає сутності «Клієнти» у ер-моделі)							
№	**Ключ**	**Ім’я атрибуту**	**Тип**	**NULL / NOT NULL**	**Цілісність посилань (ЦП)**		**Пояснення та опис ЦП**
					ON DELETE	**ON UPDATE**	
1.	PK	client_id	Integer	NOT NULL			Ідентифікатор (лічильник) клієнтів.
2.		lastname	Varchar (255)	NOT NULL			Прізвище клієнта. Частина скаладеного атрибуту «ПІБ»
3.		firstname	Varchar (255)	NOT NULL			Ім'я клієнта. Частина складеного атрибуту «ПІБ»
4.		middlename	Varchar (255)	NULL			По батькові клієнта. Частина складеного атрибуту «ПІБ».
5.		photo_url	Varchar(255)	NOT NULL			Фотокарта клієнта. Служить візуальної ідентифікації клієнта.
6.		email	Varchar (255)	NULL			Адреса електронної пошти клієнта.
7.	AK	phone	Varchar (255)	NOT NULL			Номер телефону клієнта. Обов’язковий атрибут. Може використовуватись для ідентифікації клієнта.
8.		birth_date	Date	NOT NULL			

Додаток Ґ

(обов'язковий)

Таблиця № 4 «Заняття»+ «Заняття відповідно ER»

ВІДНОШЕННЯ «ЗАНЯТТЯ» (відповідає сутності «Заняття» у ер-моделі)							
№	**Ключ**	**Ім'я атрибуту**	**Тип**	**NULL / NOT NULL**	**Цілісність посилань (ЦП)**		**Пояснення та опис ЦП**
					ON DELETE	**ON UPDATE**	
1.	PK	workout_id	Integer	NOT NULL			Ідентифікатор (лічильник) занять.
5.		start_time	Time	NOT NULL			Час початку заняття.
6.		end_time	Time	NULL			Час завершення заняття. Може бути порожнім, якщо це заплановане заняття або воно ще не завершилось. Необов'язковий атрибут. Не може бути раніше ніж start_time. Дата завершення заняття повинна збігатись з датою початко.
7.	FR	coach_id	Varchar (255)	NOT NULL	SET NULL	NO ACTION	Зовнішній ключ сутності «Тренер». Необов'язковий атрибут (заняття можуть відбуватись і без тренера). Відповідає зв'язку «тренує на». При видаленні тренера слід встановити null в усіх записах про заняття з ним. Якщо це заплановані заняття з тренером – вони перетворяться на заняття без нього. При оновленні тренера слід оновити усі

							заняття з ним.
8.	FR	client_id	Integer	NOT NULL	CASCADE	CADCADE	Зовнішній ключ сутності «Клієнт». Відповідає зв’язку «відвідує».
9.	FR	gym_id	Integer	NOT NULL	NO ACTION	CASCADE	Зовнішній ключ сутності «Спортзал». Відповідає зв’язку «Відбувається у». При видаленні спортзалу слід встановити спортзал за замовчуванням (екземляр сутності «Спортзали» яка служить для того, щоб зберігати усі заняття клієнтів навіть після того як видалиться спортзал).
ПОХІДНІ АТРИБУТИ							

Додаток Д

(обов'язковий)

Таблиця «Спортзали»+ «Спортзали відповідно ER»

ВІДНОШЕННЯ «СПОРТЗАЛИ» (відповідає сутності «Спортзали» у ер-моделі)							
№	**Ключ**	**Ім'я атрибуту**	**Тип**	**NULL / NOT NULL**	**Цілісність посилань (ЦП)**		**Пояснення та опис ЦП**
					ON DELETE	**ON UPDATE**	
1.	PK	gym_id	Integer	NOT NULL			Номер спортзалу. Служить для ідентифікації спортзалів.
2.		country	Varchar (255)	NOT NUL			Країна, де розміщений спортзал. Частина складеного атрибуту «Адреса»
3.		town	Varchar (255)	NOT NULL			Населений пункт, де розміщений спортзал. Частина складеного атрибуту «Адреса»
4.		area	Varchar (255)	NULL			Область, де розміщений спортзал. Частина складеного атрибуту «Адреса»
5.		street	Varchar (255)	NOT NULL			Вулиця, де розміщенй спортзал. Частина складеного атрибуту «Адреса»
6.		building	Varchar (255)	NOT NULL			Номер будинку, розміщений спортзал. Частина складеного атрибуту «Адреса»
7.		office	Integer	NULL			Номер квартири, де проживає тренер. Частина складеного атрибуту «Адреса»
8.		fine	Currency	NOT NULL			Обов'язковий атрибут «пеня». Служить для обрахунку доплпати клієнта за заняття.

9.		post_index	Integer	NOT NULL			Поштовий індекс спортзалу.
10		email	Varchar (255)	NOT NULL			Адреса електронної пошти спортзалу. Обов’язковий атрибут.

Додаток Е

(обов’язковий)

Таблиця «Фотографії спортзалу»+ «атрибут фотографії спортзалу відповідно ER»

ВІДНОШЕННЯ «ФОТОГРАФІЇ СПОРТЗАЛІВ» (відповідає багатозначному атрибуту «фотографії спортзалу» у ер-моделі)							
№	**Ключ**	**Ім’я атрибуту**	**Тип**	**NULL / NOT NULL**	**Цілісність посилань (ЦП)**		**Пояснення та опис ЦП**
					ON DELETE	**ON UPDATE**	
1.	PK	photo_id	Integer	NOT NULL			Числовий ідентифікатор фотографії.
2.		photo_url	Varchar (255)	NOT NULL			URL фото спортзалу
3.	FK	gym_id	Integer	NOT NULL	cascade	cascade	Зовнішній ключ сутності «Спортзали». Відповідає за реалізацію звязку між спортзалами і їх фотографіями. При видаленні/оновленні спортзалу слід видалити/оновити усі його фотографії.

Додаток Є

(обов'язковий)

Таблиця «Абонементи»+ «Абонементи відповідно ER»

ВІДНОШЕННЯ «АБОНЕМЕНТИ» (відповідає сутності «Абонементи» у ер-моделі)							
№	**Ключ**	**Ім'я атрибуту**	**Тип**	**NULL / NOT NULL**	**Цілісність посилань (ЦП)**		**Пояснення та опис ЦП**
					ON DELETE	**ON UPDATE**	
1.	PK	subscription_id	Integer	NOT NULL			Номер абонементної карти. Ідентифікатор абонементної карти. Тип – числовий.
2.		start_day	Integer	NOT NULL			День початку дії абонементу. Частина складеного атрибуту «дата початку». Набуває значення від 1 до 31.
5.		end_day	Integer	NOT NULL			День закінчення дії абонементу. Частина складеного атрибуту «дата кінця». Набуває значення від 1 до 31.
8.		start_time	Time	NOT NULL			Час початку дії абонементу в добі.
9.		end_time	Time	NOT NULL			Час закінчення дії абонементу в добі. Не може бути раніше ніж start_time
10		price	Currency	NOT NULL			Вартість абонементу в гривнях.
11	FK	client_id	Integer	NULL	cascade	cascade	Зовнішній ключ сутності «Клієнти». Відповідає за реалізацію зв`язку між клієнтами та абонементами. При видаленні клієнта слід видалити всі його абонементи.

Додаток Ж

(обов’язковий)

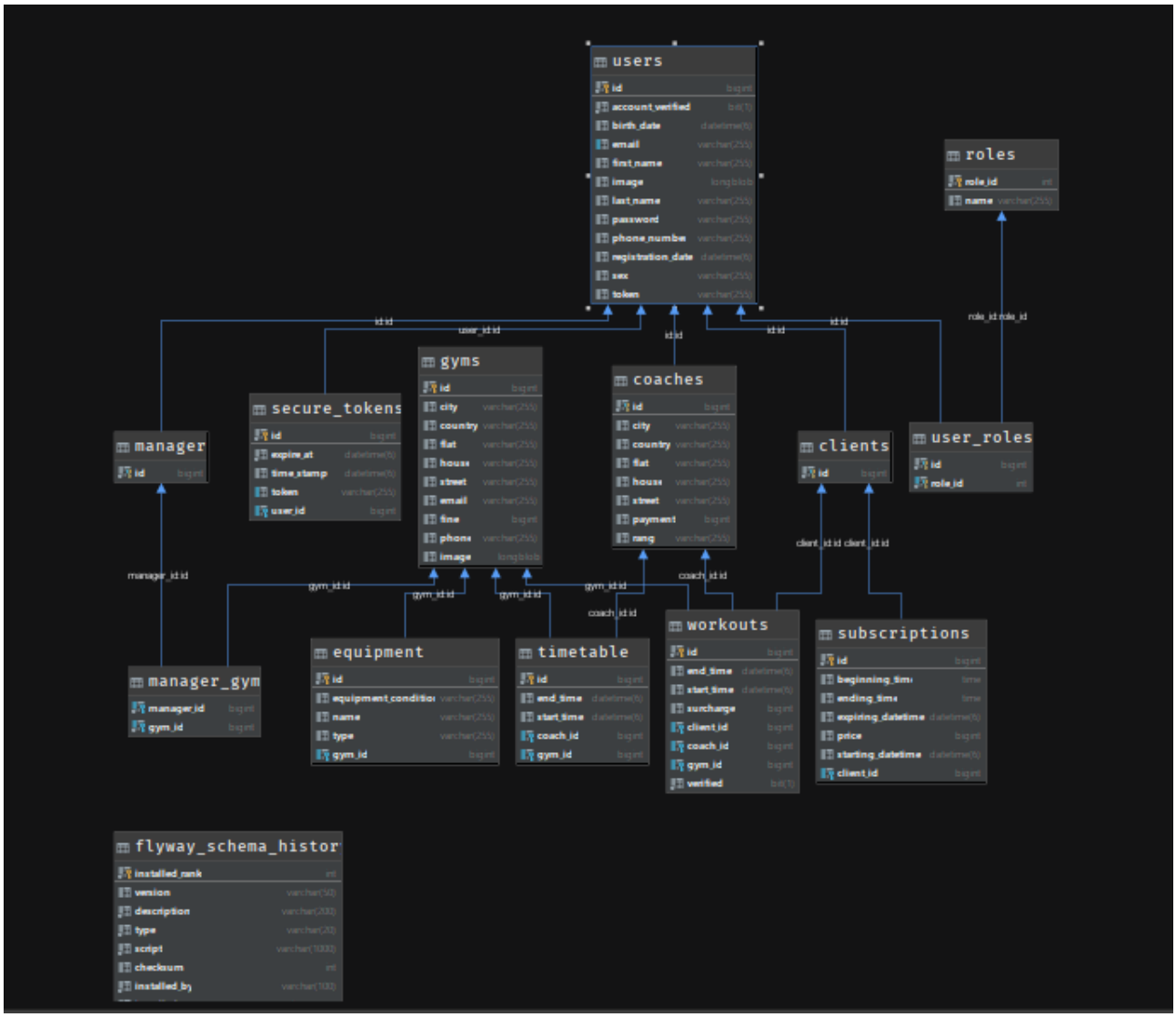
Таблиця «Обладнання»+ «Обладнання відповідно ER»

ВІДНОШЕННЯ «ОБЛАДНАННЯ» (відповідає сутності «Обладнання» у ер-моделі)							
№	**Ключ**	**Ім’я атрибуту**	**Тип**	**NULL / NOT NULL**	**Цілісність посилань (ЦП)**		**Пояснення та опис ЦП**
					ON DELETE	**ON UPDATE**	
1.	PK	equipment_id	Integer	NOT NULL			Номер абонементної карти. Ідентифікатор абонементної карти. Тип – числовий.
2.		name	Varchar (255)	NOT NULL			Назва обладнання.
3.		type	Varchar (255)	NOT NULL			Тип обладнання. Для якої групи м'язів варто використовувати це обладднаня
4.		condition	Varchar (255)	NOT NULL			Стан обладнання. Якщо рядок порожній означає що обладнання справне, інакше вказується причина несправності (наприклад "порвався лівий трос, потребує заміни сидіння")
5.		photo_url	Varchar(255)	NOT NULL			Фотографія обладнання.
6.	FK	jym_id	Integer	NOT NULL	Cascade	Cascade	Зовнішній ключ спортзалу, якому належить обладнання. При видаленні/оновленні спортзалу слід видалити усе обладнання, що йому належало.

Додаток З

(обов’язковий)

Діаграма бази даних MySQL

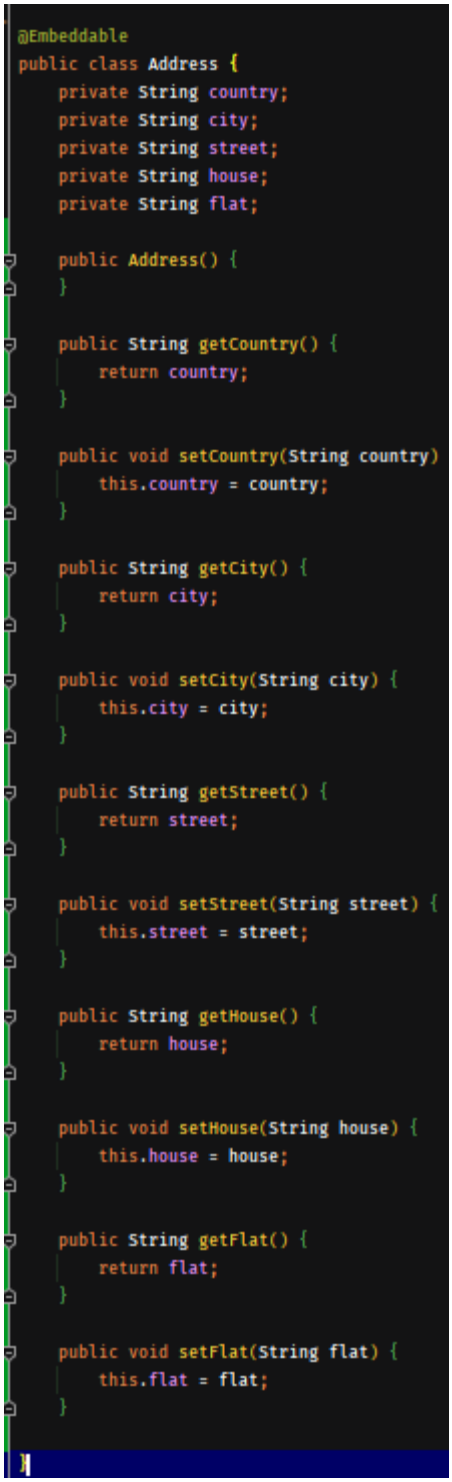


Додаток И

(обов’язковий)

Порівняння класів з використання анотацій Lombok та без їх використання

```
@Getter
@Setter
@NoArgsConstructor
@Embeddable
public class Address {
    private String country;
    private String city;
    private String street;
    private String house;
    private String flat;
}
```



```
@Embeddable
public class Address {
    private String country;
    private String city;
    private String street;
    private String house;
    private String flat;

    public Address() {
    }

    public String getCountry() {
        return country;
    }

    public void setCountry(String country)
        this.country = country;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public String getHouse() {
        return house;
    }

    public void setHouse(String house) {
        this.house = house;
    }

    public String getFlat() {
        return flat;
    }

    public void setFlat(String flat) {
        this.flat = flat;
    }
}
```

Додаток І

(обов’язковий)

Приклад коду, що відповідає за авторизацію користувача у системі

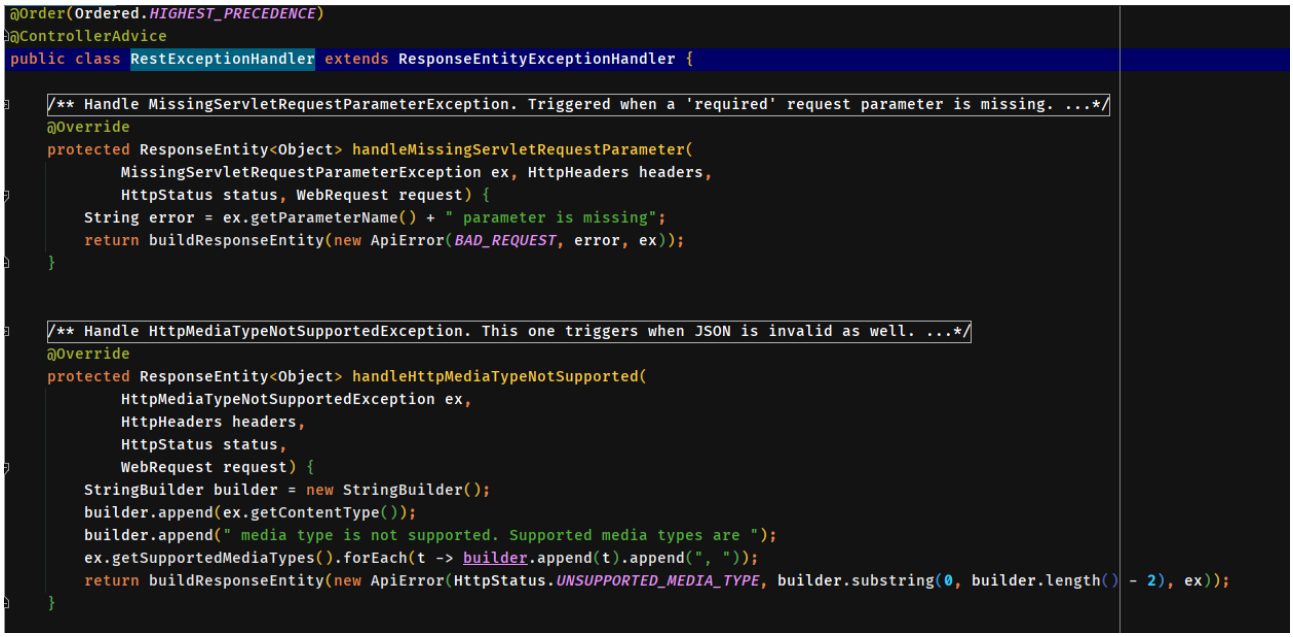
```
@Override
public UserEntity register(SignUpRequest user) throws UserAlreadyExistException {

    //Check if user already registered with us
    if (checkIfUserExist(user.getEmail())) {
        throw new UserAlreadyExistException("User already exists for this email");
    }
    UserEntity userEntity = new UserEntity();
    BeanUtils.copyProperties(user, userEntity);

    userEntity.setBirthDate(Timestamp.valueOf(user.getBirthDate().atStartOfDay()));
    userEntity.setRoles(Stream.of(roleRepository.findByName(ERoles.ROLE_USER),
            roleRepository.findByName(ERoles.ROLE_CLIENT))
            .collect(Collectors.toCollection(HashSet::new)));
    encodePassword(userEntity, user);
    try {
        userEntity.setImage(user.getImage().getBytes());
    } catch (IOException e) {
        log.warn("Exception while user image upload: " + e.getMessage());
    }
    UserEntity result = userRepository.save(userEntity);
    sendRegistrationConfirmationEmail(userEntity);
    return result;
}

@Override
public void sendRegistrationConfirmationEmail(UserEntity user) {
    SecureToken secureToken = secureTokenService.createSecureToken();
    secureToken.setUser(user);
    secureTokenRepository.save(secureToken);
    AccountVerificationEmailContext emailContext
            = new AccountVerificationEmailContext();
    emailContext.init(user);
    emailContext.setToken(secureToken.getToken());
    emailContext.buildVerificationUrl(baseURL, secureToken.getToken());
    try {
        emailService.sendMail(emailContext);
    } catch (MessagingException e) {
        e.printStackTrace();
    }

}
```

```
@Override
public boolean verifyUser(String token) throws InvalidTokenException {
    SecureToken secureToken = secureTokenService.findByToken(token);
    if (Objects.isNull(secureToken) || !Objects.equals(token, secureToken.getToken()) || secureToken.isExpired()) {
        throw new InvalidTokenException("Token is not valid");
    }
    UserEntity user = userRepository.getOne(secureToken.getUser().getId());
    if (Objects.isNull(user)) {
        return false;
    }
    user.setAccountVerified(true);
    userRepository.save(user); // let's same user details

    // we don't need invalid password now
    secureTokenService.removeToken(secureToken);
    return true;
}
```

Додаток Ї

(обов’язковий)

Приклад глобального обробника помилок

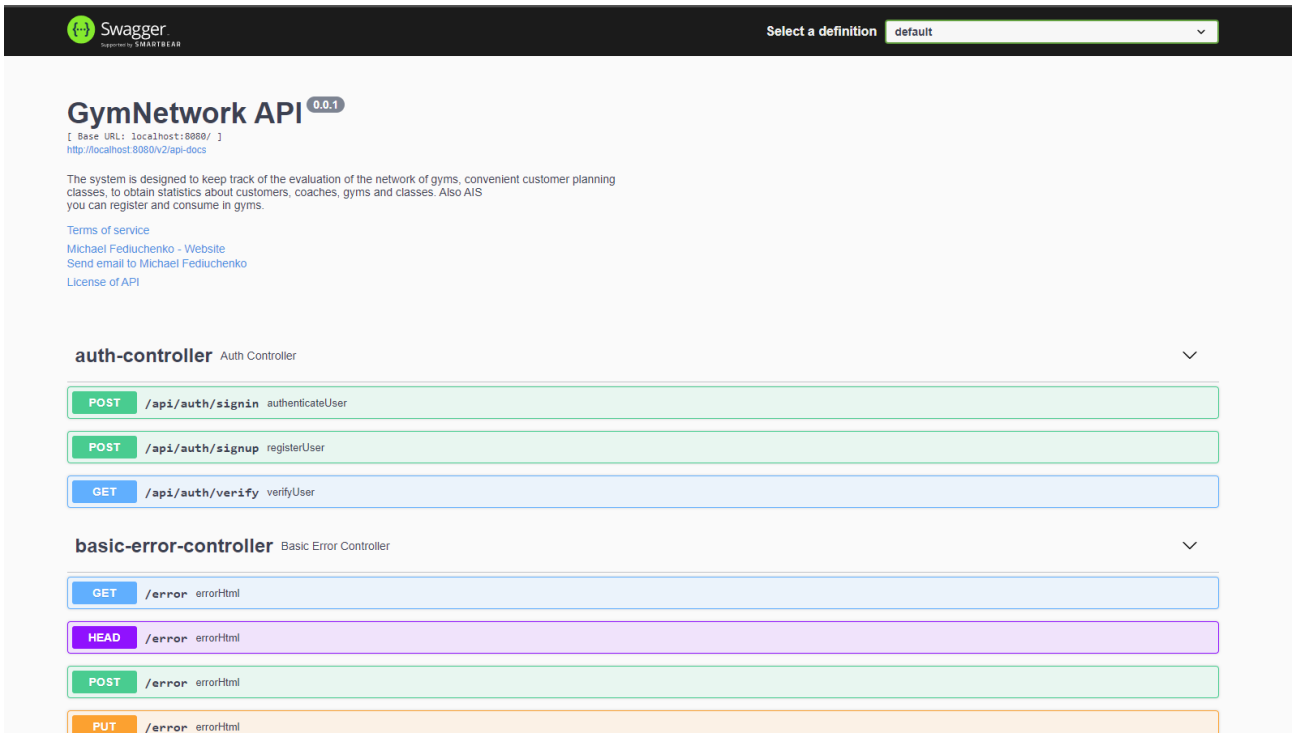


```java
@Order(Ordered.HIGHEST_PRECEDENCE)
@ControllerAdvice
public class RestExceptionHandler extends ResponseEntityExceptionHandler {

    /** Handle MissingServletRequestParameterException. Triggered when a 'required' request parameter is missing. ...*/
    @Override
    protected ResponseEntity<Object> handleMissingServletRequestParameter(
            MissingServletRequestParameterException ex, HttpHeaders headers,
            HttpStatus status, WebRequest request) {
        String error = ex.getParameterName() + " parameter is missing";
        return buildResponseEntity(new ApiError(BAD_REQUEST, error, ex));
    }


    /** Handle HttpMediaTypeNotSupportedException. This one triggers when JSON is invalid as well. ...*/
    @Override
    protected ResponseEntity<Object> handleHttpMediaTypeNotSupported(
            HttpMediaTypeNotSupportedException ex,
            HttpHeaders headers,
            HttpStatus status,
            WebRequest request) {
        StringBuilder builder = new StringBuilder();
        builder.append(ex.getContentType());
        builder.append(" media type is not supported. Supported media types are ");
        ex.getSupportedMediaTypes().forEach(t -> builder.append(t).append(", "));
        return buildResponseEntity(new ApiError(HttpStatus.UNSUPPORTED_MEDIA_TYPE, builder.substring(0, builder.length() - 2), ex));
    }
```

Додаток Й

(обов’язковий)

Приклад користувацького інтерфейсу Swagger 2



Додаток К

(обов’язковий)

Схема моделі підключення мікроконтролера