Національний університет «Одеська Політехніка»

Інститут комп’ютерних систем

Кафедра інформаційних систем

Курсова робота з дисципліни «ОБ’ЄКТНО-ОРІЄНТОВАНЕ

ПРОГР АМУВАННЯ»

Система управління автосервісом

Виконав:

студент групи АІ-231

Годун І.О.

Одеса 2025

Вступ……………………………………………………………………………3

Аналіз предметної області…………………………………………………….4

Проєктування програмного забезпечення …………………………………..6

Реалізація програмного продукту……………………………………………..8

Тестування та налагодження…………………………………………………10

Переваги використаної архітектури…………………………………………11

Перспективи розвитку проєкту………………………………………………12

Додаткові можливості системи………………………………………………14

Використані технології та середовища розвитку…………………………..16

Застосування принципів обʼєктно-орієнтованого програмуванння………18

Висновки………………………………………………………………………19

Використані джерела…………………………………………………………20

**Вступ**

Метою курсової роботи є створення інформаційної системи, яка автоматизує основні процеси управління автосервісом.

Ця система дозволяє ефективно керувати клієнтами, їхніми автомобілями, записами на обслуговування, типами послуг та працівниками (майстрами).

Автосервіс — це сучасне підприємство, діяльність якого ґрунтується на наданні послуг з технічного обслуговування, ремонту, діагностики автомобілів. Для якісного функціонування таких закладів потрібне надійне програмне забезпечення, яке дозволяє автоматизувати повсякденні операції, забезпечити збереження історії обслуговувань і оптимізувати комунікацію між працівниками.

У процесі виконання курсової роботи буде використано мову програмування Java та фреймворк Spring Boot. Це забезпечить побудову надійної, масштабованої архітектури з чітким поділом на шари: контролери, сервіси, моделі та репозиторії. Для збереження даних буде використовуватись реляційна база даних PostgreSQL, а для доступу до неї — бібліотека Spring Data JPA.

Система також передбачає реалізацію REST API, що дозволяє зручно взаємодіяти з клієнтською частиною.

Курсова робота сприяє формуванню таких навичок:

* проєктування об’єктно-орієнтованої моделі;
* реалізація CRUD-функціоналу;
* використання інструментів авторизації (JWT);
* модульне тестування і документація API.

**Аналіз предметної області**

Автосервіс — це підприємство, що надає послуги з технічного обслуговування та ремонту транспортних засобів. У сучасному світі, де переважна частина населення користується автомобілями, попит на послуги автосервісів постійно зростає. При цьому значно ускладнюється управління процесами обслуговування: необхідно вести облік клієнтів, даних автомобілів, призначених майстрів, наданих послуг, вартості, тривалості виконаних робіт та історії кожного візиту.

У традиційному підході ці дані зберігаються в журналах або Excel-таблицях, що є незручним, неефективним та не відповідає сучасним вимогам. Автоматизація таких процесів не тільки спрощує роботу персоналу, а й підвищує якість обслуговування клієнтів.

Система, яку передбачено розробити в межах цієї курсової роботи, має реалізовувати такі функції:

* зберігання інформації про клієнтів (ПІБ, контактні дані);
* облік автомобілів, прив’язаних до конкретного клієнта (марка, модель, рік випуску, номер VIN);
* створення та перегляд записів обслуговування (дата, майстер, тип послуги, коментарі);
* призначення відповідальних майстрів;
* зберігання довідника послуг з цінами;
* формування історії обслуговування автомобіля.

Усі ці сутності пов’язані між собою в логічну структуру:

* один клієнт може мати декілька автомобілів;
* одне авто має декілька записів обслуговування;
* один майстер виконує багато робіт;
* один запис пов’язаний із одним типом послуги.

У реальному бізнесі подібна система може бути інтегрована з календарями, СМС-розсилками, електронною оплатою або зовнішніми сервісами. Проте, у межах курсової, реалізовано базову функціональність, що є ядром будь-якої CRM-системи для автосервісу.

**Проєктування програмного забезпечення**

Перед розробкою системи необхідно здійснити проєктування — це включає створення моделей даних, визначення зв’язків між сутностями, проєктування API та логіки взаємодії між компонентами.

**Сутності системи:**

* Client: id, name, phone, email;
* Car: id, brand, model, year, vin, client\_id;
* Mechanic: id, name, experience;
* ServiceType: id, name, price;
* ServiceRecord: id, car\_id, mechanic\_id, service\_type\_id, date, description.

**Зв’язки між таблицями:**

* Client (1) — (M) Car;
* Car (1) — (M) ServiceRecord;
* Mechanic (1) — (M) ServiceRecord;
* ServiceType (1) — (M) ServiceRecord.

**API структура:**

* GET /clients, POST /clients, PUT /clients/{id}, DELETE /clients/{id};
* GET /cars, POST /cars, …;
* GET /records, POST /records, … — аналогічно для решти сутностей.

**Особливості:**

* Використання DTO забезпечує безпеку і зручність передачі даних.
* Валідація полів проводиться через анотації @NotNull, @Size, @Pattern.
* Передбачено обробку помилок через глобальний ExceptionHandler.

**Реалізація програмного продукту**

Процес реалізації програмного забезпечення базувався на сучасному підході до розробки веб-додатків на основі Spring Boot. Архітектура системи структурована за шаблоном MVC та містить чітке розділення на логічні шари:

* Модель (Model): включає сутності Client, Car, Mechanic, ServiceRecord, ServiceType, які описують предметну область. Вони реалізовані як JPA-entity класи з анотаціями @Entity, @ManyToOne, @OneToMany, @JoinColumn тощо.
* DTO (Data Transfer Object): використовується для безпечної передачі даних між клієнтом і сервером, приховуючи зайві поля сутностей.
* Репозиторії (Repository): інтерфейси, що розширюють JpaRepository, дозволяють легко реалізовувати запити до БД без ручного написання SQL.
* Сервіси (Service): містять бізнес-логіку — перевірки, перетворення DTO, взаємодію з базою даних.
* Контролери (Controller): обробляють HTTP-запити та повертають відповіді через REST API.

Основні технології:

* Spring Boot — каркас для швидкого створення застосунку;
* Spring Data JPA — ORM-інструмент для зручної роботи з БД;
* PostgreSQL — реляційна база даних;
* Lombok — для скорочення шаблонного коду (@Getter, @Setter, @Builder);
* JWT — реалізація безпеки (аутентифікація, авторизація);
* Postman — тестування REST API.

REST-контролери:

Наприклад, ClientController дозволяє:

* створювати клієнтів (POST /clients);
* отримувати список (GET /clients);
* оновлювати клієнтів (PUT /clients/{id});
* видаляти клієнтів (DELETE /clients/{id}).

Усі ендпоінти відповідають специфікації REST, а помилки обробляються за допомогою механізмів @ExceptionHandler.

**Тестування та налагодження**

Тестування системи є ключовим етапом розробки, що дозволяє виявити помилки та переконатися в коректній роботі функціоналу. У цьому проєкті основна увага приділялася ручному тестуванню API за допомогою Postman.

Тестові сценарії:

* Реєстрація нового клієнта: POST /clients, очікується статус 201.
* Додавання авто до клієнта: POST /cars, з валідацією імені, року випуску, VIN-коду.
* Запис обслуговування: POST /records, із вказанням дати, майстра та послуги.
* Оновлення оцінки майстра: тест логіки зміни майстра у вже створеному записі.

Інструменти:

* Postman — створення колекцій тестів;
* Swagger UI — автоматична документація API;
* Logback / Console Logging — відслідковування помилок.

Під час тестування було виявлено та усунуто помилки:

* неправильна обробка null-полів у DTO;
* некоректні статуси при виключеннях;
* проблеми з кодуванням JSON-відповідей.

В результаті всі основні сценарії виконуються коректно, API повертає передбачувані відповіді та обробляє помилки згідно з принципами REST.

**Переваги використаної архітектури**

Застосована архітектура відповідає принципам чистого коду і забезпечує такі переваги:

1. Гнучкість — легко додавати нові сутності або змінювати логіку без втручання в інші шари.
2. Повторне використання коду — сервіси й репозиторії можна використовувати у різних модулях.
3. Безпека — реалізовано аутентифікацію за допомогою JWT, обмежено доступ до певних ендпоінтів.
4. Масштабованість — завдяки Spring Boot застосунок готовий до деплою на будь-які хмарні сервіси (Render, Heroku).
5. Тестованість — структура дозволяє легко писати юніт- та інтеграційні тести.
6. Принципи SOLID — кожен клас виконує тільки одну відповідальність, залежності впроваджуються автоматично.

Також окремо варто відзначити використання Lombok, що значно спростив і скоротив код моделі, дозволивши зосередитись на логіці.

**Перспективи розвитку проєкту**

Система, створена у межах курсової роботи, має великий потенціал для розвитку й трансформації у повноцінну комерційну систему. Серед можливих покращень:

**Функціональні розширення:**

* Система запису онлайн з інтерфейсом для клієнтів (наприклад, через Telegram-бота або сайт).
* Інтеграція платіжної системи (наприклад, через LiqPay, Stripe).
* Автоматична генерація рахунків, актів, звітів у PDF.
* Панель керування майстрами — завантаження, графіки, KPI.

**Інтерфейс:**

* Реалізація SPA (Single Page Application) на React/Angular;
* Адаптація під мобільні пристрої;
* Додаток на Flutter для клієнтів (перегляд історії авто, запис онлайн).

**Аналітика:**

* Візуалізація прибутків, витрат, кількості клієнтів;
* Побудова графіків популярності послуг;
* Аналіз продуктивності майстрів.

**Безпека:**

* Ролі та права з більш гнучкими налаштуваннями (RBAC);
* Логування всіх дій (audit trail);
* Двофакторна автентифікація для адміністратора.

Реалізація цих функцій дозволить перетворити проєкт на комплексну CRM-систему для автосервісів малого та середнього бізнесу.

**Додаткові можливості системи**

Хоча основна частина проєкту орієнтована на базові CRUD-операції, система вже закладена як розширювана і масштабована. Ось деякі додаткові можливості, які можуть бути додані при розширенні:

* Фільтрація та пошук: можливість шукати клієнтів, авто або записи за критеріями (дата, майстер, VIN).
* Сортування та пагінація: наприклад, для великої кількості клієнтів або послуг.
* Імпорт / експорт CSV-файлів з історією обслуговувань або клієнтів.
* Вбудована система сповіщень — про майбутній запис, запланований сервіс, нову послугу.
* Логування та журнал подій — для адміністраторів або сервісменів.

Ці функції не лише покращують зручність використання системи, а й роблять її привабливою з комерційної точки зору.

**Використані технології та середовища розвитку**

Під час реалізації курсової роботи було використано сучасний стек технологій, який дозволив створити масштабований, гнучкий і зручний для підтримки серверний застосунок. Нижче наведено основні інструменти та середовища, які використовувалися:

* Java 17 — сучасна об’єктно-орієнтована мова програмування з високим рівнем безпеки, сумісна з великою кількістю бібліотек.
* Spring Boot 3.x — популярний фреймворк для розробки серверних додатків, який спрощує конфігурацію, автоматично створює контексти та дозволяє швидко запускати застосунок.
* Spring Data JPA — модуль для роботи з базою даних через об’єктно-реляційне відображення (ORM).
* PostgreSQL — потужна реляційна база даних з відкритим вихідним кодом, що добре інтегрується з Java-застосунками.
* Lombok — бібліотека, що дозволяє уникати шаблонного коду, наприклад, автоматично генерує гетери, сетери, конструктори.
* JWT (JSON Web Token) — технологія для безпечної передачі інформації між сторонами у вигляді підписаного токена.
* Postman — інструмент для тестування REST API, створення колекцій запитів, перевірки статусів відповідей.
* Maven — система для керування залежностями і збирання Java-проєктів.
* IntelliJ IDEA / Visual Studio Code — середовища розробки з підтримкою Java, Spring Boot, дебагу і тестування.
* Docker (опційно) — для розгортання сервісів у контейнерах під час тестування.

Вибір саме цих технологій дозволив створити систему, що відповідає сучасним вимогам до безпеки, продуктивності й масштабованості. Кожен компонент застосунку чітко виконує свою роль, що підвищує якість програмного продукту.

**Застосування принципів обʼєктно-орієнтованого програмування**

Основною вимогою при реалізації курсової роботи була демонстрація знань принципів ООП (Object-Oriented Programming). Цей підхід дозволяє моделювати складні системи, використовуючи сутності реального світу у вигляді об’єктів.

**Використані принципи ООП:**

* Інкапсуляція — усі поля класів зроблені приватними (private), доступ до них здійснюється через гетери та сетери. Це обмежує прямий доступ до полів і дозволяє перевіряти/обробляти значення перед збереженням.
* Наслідування — у DTO або Response-класах можна створити базовий клас із загальними полями (id, createdAt, updatedAt) і розширювати його у конкретних DTO.
* Поліморфізм — сервіси або контролери можуть працювати з інтерфейсами, а не з конкретними реалізаціями. Це дозволяє змінювати логіку без зміни викликів.
* Абстракція — користувачі взаємодіють із простим API, не знаючи, яка логіка реалізована за цим інтерфейсом. Сервіси, контролери, DTO — все побудовано з абстрактним розмежуванням ролей.

**Висновки**

У процесі виконання курсової роботи з дисципліни «Об’єктно-орієнтоване програмування» було розроблено інформаційну систему для управління автосервісом. Проєкт охопив усі ключові етапи життєвого циклу програмного забезпечення: аналіз предметної області, проєктування структури даних та архітектури, реалізацію серверної частини, тестування та документування функціоналу.

Система дозволяє ефективно керувати такими сутностями, як клієнти, автомобілі, записи обслуговування, типи послуг та працівники. Завдяки чіткому розділенню логіки на шари (модель, сервіс, контролер), реалізація забезпечує високий рівень масштабованості та підтримки. Інструменти, використані в проєкті — зокрема Spring Boot, Spring Data JPA, JWT, PostgreSQL, Lombok — є сучасними стандартами для розробки промислових Java-додатків.

Особливу увагу було приділено реалізації REST API, що дає змогу інтегрувати систему з іншими сервісами або розширити її на фронтенд. Тестування API через Postman продемонструвало стабільність роботи, виявлені помилки були оперативно усунуті.

Отримані результати свідчать про доцільність застосування принципів об’єктно-орієнтованого програмування та шаблонів архітектури у розробці прикладних систем. Створене програмне забезпечення може бути основою для подальшої розробки — включаючи розширення прав доступу, реалізацію вебінтерфейсу, статистичних модулів і підтримки мобільних платформ.

Таким чином, мету курсової роботи досягнуто повністю, а поставлені завдання виконані. Студент продемонстрував здатність аналізувати вимоги, проєктувати структуру ПЗ, писати ефективний код, застосовувати сучасні технології Java-розробки та проводити тестування готового продукту.

**Використані джерела**

1. Документація Spring Boot – <https://spring.io/projects/spring-boot>
2. Документація Spring Data JPA – <https://spring.io/projects/spring-data-jpa>
3. Hibernate ORM – <https://hibernate.org/orm/>