**ПОИЗВОДСТВЕНА ПРАКТИКА**

**ДЕЙНОСТ: УЧЕНИЧЕСКИ ПРАКТИКИ**

***2023-2024***

**Тема: „Web-базирано приложение за логистика и транспортиране до адрес“**

**Курсов проект**

*Автори:*

*Йоанна Владимирова Симеонова, клас XI В*

БургасСъдържание

[1 Въведение 3](#_Toc164380209)

[2 Цели и обхват на софтуерното приложение 3](#_Toc164380210)

[3 Анализ на решението 5](#_Toc164380211)

[3.1 Потребителски изисквания и работен процес 5](#_Toc164380212)

[3.1.1 UML Use-Case диаграма 5](#_Toc164380213)

[3.1.2 UML Activity диаграма 8](#_Toc164380214)

[3.2 Примерен потребителски интерфейс 11](#_Toc164380215)

[3.3 Диаграми на анализа 15](#_Toc164380216)

[3.4 Модел на съдържанието / данните 15](#_Toc164380217)

[4 Дизайн 16](#_Toc164380218)

[4.1 Реализация на структура на приложението (3-layer), Разделение на кода според предназначението му 16](#_Toc164380219)

[4.2 Организация и код на заявките към база от данни 16](#_Toc164380220)

[4.3 Наличие и интуитивност на потребителски интерфейс (конзолен, графичен, уеб) 16](#_Toc164380221)

[5 Тестване 16](#_Toc164380222)

[6 Заключение и възможно бъдещо развитие 16](#_Toc164380223)

[7 Използвани литературни източници и Уеб сайтове 17](#_Toc164380224)

[8 Приложения 17](#_Toc164380225)

[9 Критерии и показатели за оценяване 19](#_Toc164380226)

# Въведение

Идеята на проекта е система за изчисляване на график за доставки от склад до адрес, като целта на приложението е да подреди доставките по оптимален начин, за пестене на ресурси като гориво. Проектът ще се ситуира в уебсайт.

Проблема, който се опитва да разреши програмата, е голямото разхищение на ресурси и замърсяване на природата осредством количеството парникови газове отделени от превозните средства, използвани за доставка. Според информация от проучвания, проведени от HIT, „между 20% и 30% от емисиите на въглероден диоксид са произведени от доставки на различни по вид пратки“ (Дейвис, 2020). Приложението ще се справя с този проблем, като предлага следните функционалности:

* За служител в магазин:
  + Форма за въвеждане на поръчка с продукти по EAN номер.
  + Изглед със завършена поръчка и резултат на дата и часови диапазон на доставката.
* За доставчик:
  + Изглед с дневен график за доставки.
  + Функционалност за отмятане на пратка като доставена до зададения адрес.
* Админ панел за евентуална редакция на поръчка.

# Цели и обхват на софтуерното приложение

Приложението е насочено към физически магазини, които предлагат опция за доставка до адрес. Служителите ще имат възможността да генерират поръчки, които запълват графика, създаден от програмата. Доставчиците ще могат да виждат своята програма за деня и в коя част на града трябва да доставят пратки.

Понеже всеки град е разделен на квартали, за целите на приложението ще се използва измислен град със зони. В един часови диапазон ще се поместват пратки, поръчани за една съща зона награда, като за следващия откъс от време могат да се доставят пратки до съседна зона. Така горивото, което се изпозва, намаля значително и ефективността се повишава.

Когато се съставя графикът, ще се използва методът на „първия заварил“, т.е. първата направен поръчка отива на първото свободно място, което удовлетворява целите на апликацията.

Като по-нататъчна цел обхватът на приложението може да се разшири и до фирми доставчици и онлайн магазини.

# Анализ на решението

## Потребителски изисквания и работен процес

### UML Use-Case диаграма

*Фиг. 1 Use-Case диаграма*

Основните участници в системата (Фиг. 1) са:

* **Потребител:** Всеки, който използва системата.
* **Служител в магазин:** Може да създава поръчки.
* **Доставчик:** Може да вижда графика и да маркира поръчки като доставени.
* **Администратор:** Може да изпълнява CRUD операции върху поръчките.

Най-популярните случаи на използване системата за логистика са следните:

**User:**

* **Log in:** Влиза в системата с предварително зададени потребителско име и парола**.**
  + **Verify account:** Системата сверява достоверността на въведените данни**.**
    - **Display message:** Показва се съобщение за успешно влизане в системата.
    - **Create access token:** Създава се уникален ключ за сесията на текущия потребител.
  + **Display error message:** Изписва се съобщение с грешка при неправилно въведени данни.

**Retailer:**

* **Register order:** Въвежда нова поръчка.
  + **Display delivery information:** След задаване на поръчката, системата изчислява къде в графика може да бъде поставена тя и връща дата и часови диапазон.
    - **Send email to customer:** Системата изпраща имейл за потвърждение на клиента.

**Deliverer:**

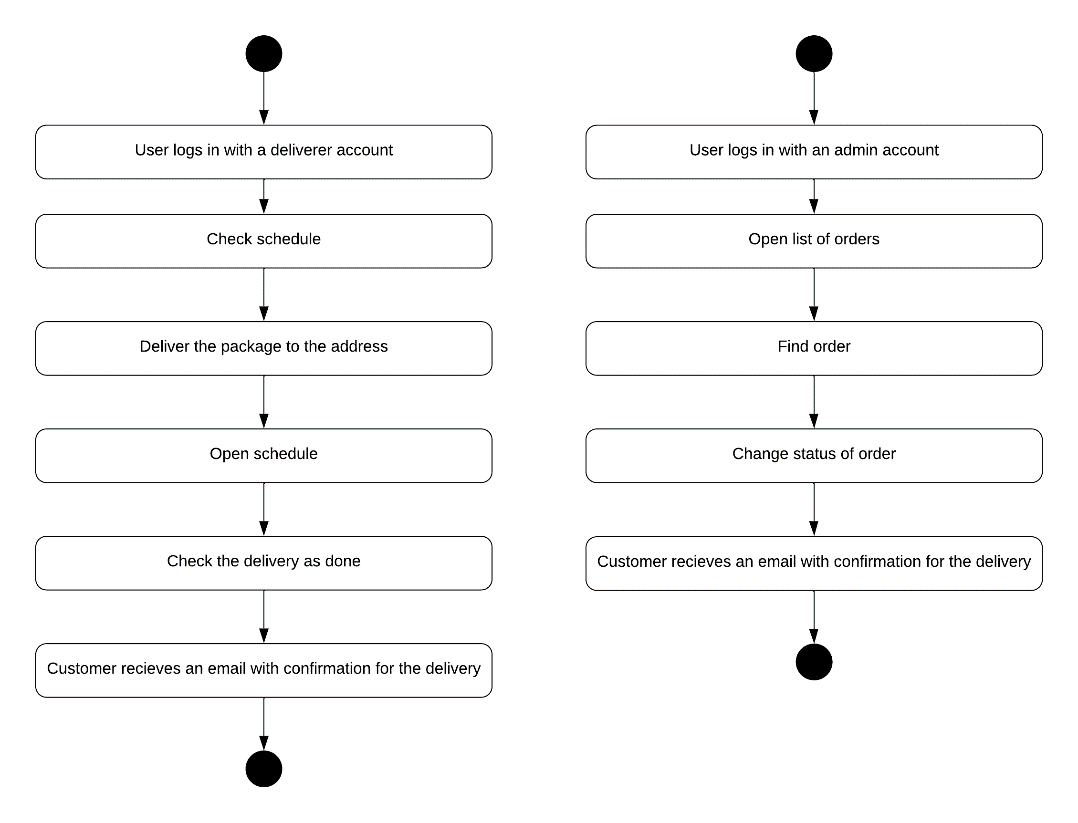
* **View schedule:** Доставчикът може да види дневната си програма за доставки.
* **Check order as delivered:** Задава поръчката като доставена.
  + **Update schedule list:** Премахва поръчката от графика на доставчика.
    - **Send email to customer:** Системата изпраща имейл за потвърждение на доставката на клиента.

**Admin:**

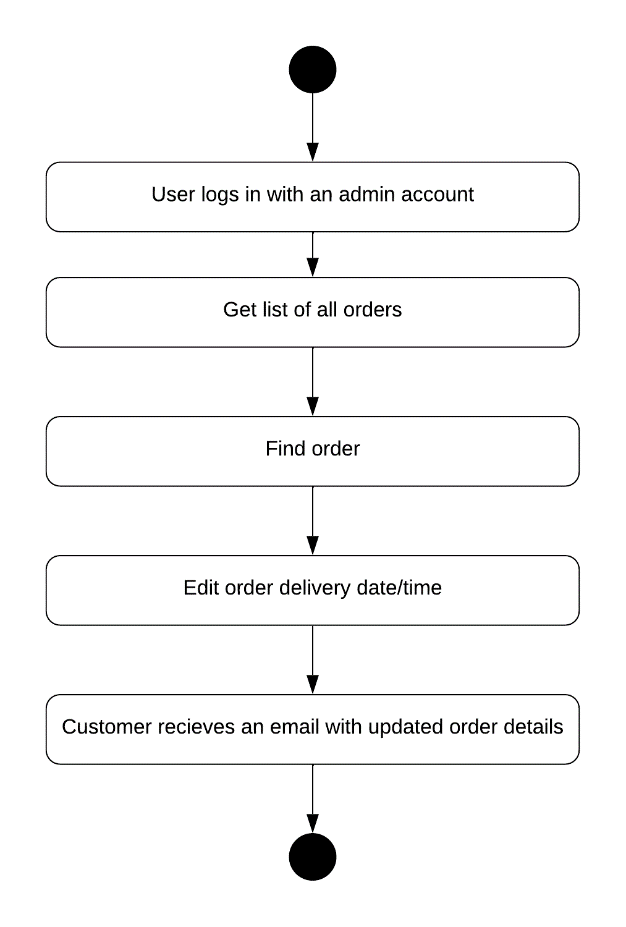
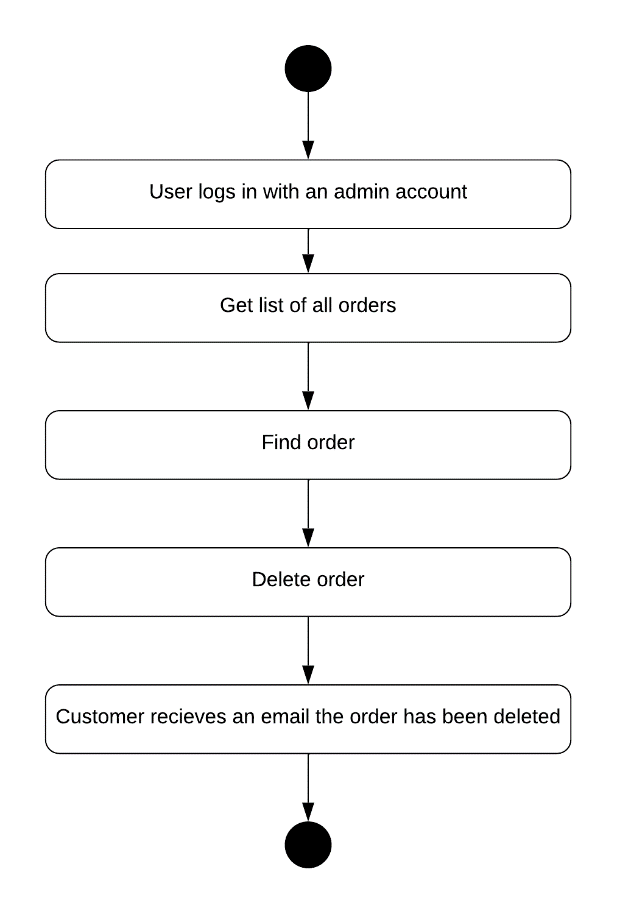
* **Register order:** Въвежда нова поръчка.
  + **Display delivery information:** След задаване на поръчката, системата изчислява къде в графика може да бъде поставена тя и връща дата и часови диапазон.
    - **Send email to customer:** Системата изпраща имейл за потвърждение на клиента.
* **View schedule:** Извежда целия график за доставки по дни.
* **Edit schedule:** Редактира местоположението на поръчка в графика.
  + **Update delivery information:** Обновява данните за поръчката.
    - **Send email to customer:** Системата изпраща имейл с данните за редакцията на клиента.
* **Delete order:** Изтрива поръчка.
  + **Update schedule:** Обновява графика.
    - **Send email to customer:** Системата уведомява клиента за изтриването на поръчката с имейл.
* **Register user:** Регистрира нов потребител.
* **View users:** Вижда списък с всички потребители.
* **Update user:** Редактира потребител.
* **Delete user:** Изтрива потребител.

### UML Activity диаграма

*Фиг. 2 Activity диаграма за създаване на поръчка*



*Фиг. 3 Activity диаграма за манипулация на поръчки*

*Фиг. 4 Activity диаграма за редакция на поръчки*

*Фиг. 5 Activity диаграма за изтриване на поръчка*

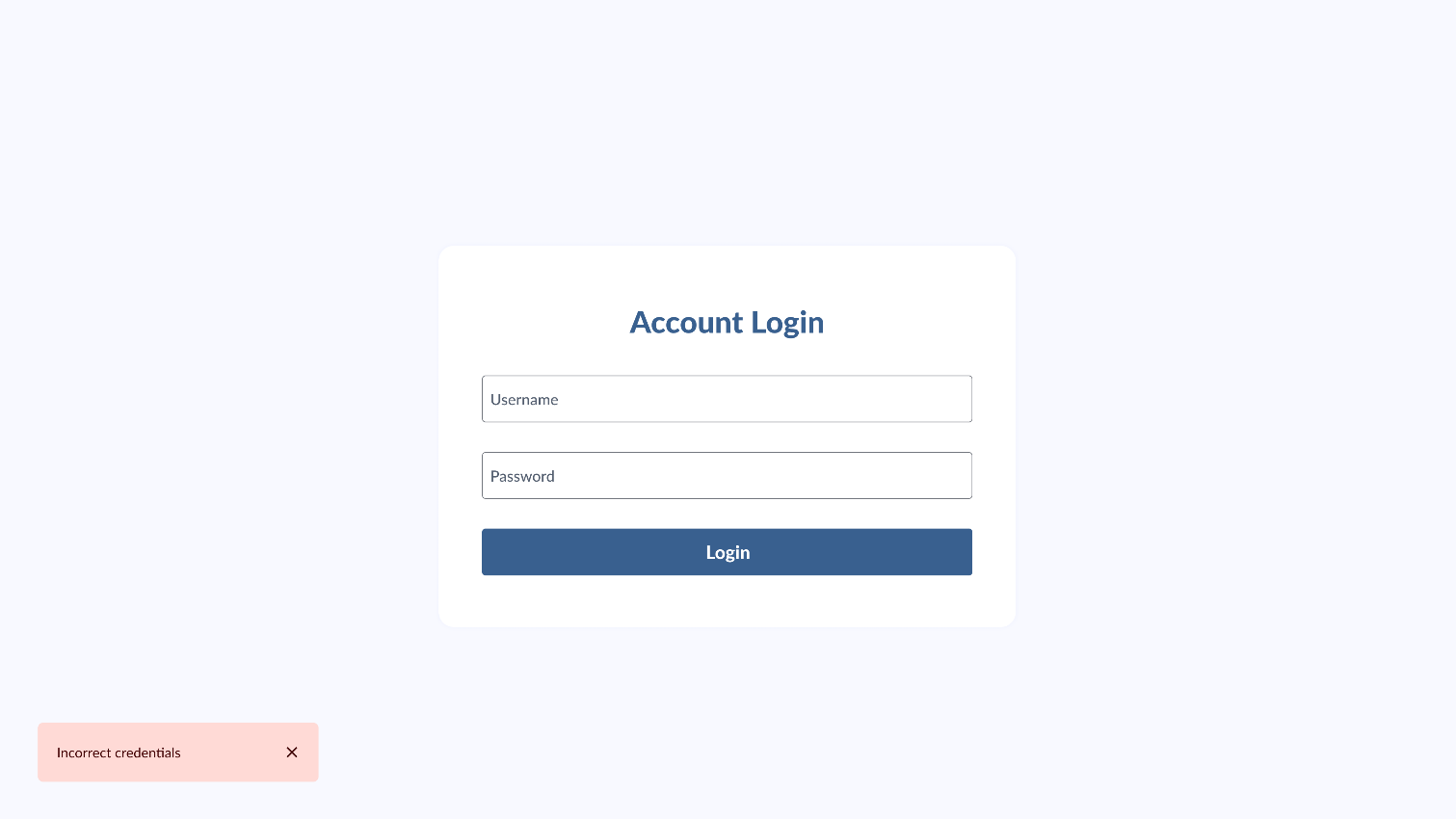
**Създаване на поръчка:** Само администратори и служители на магазина могат да изпълняват тази дейност. Стъпките, които трябва да изпълнят са както са показани на диаграмата (Фиг. 2).

**Отбелязване на поръчка като доставена:** Само администратори и доставчици могат да изпълняват тази дейност. Стъпките, които трябва да изпълнят за двата вида потребители са както са показани на диаграмата (Фиг. 3).

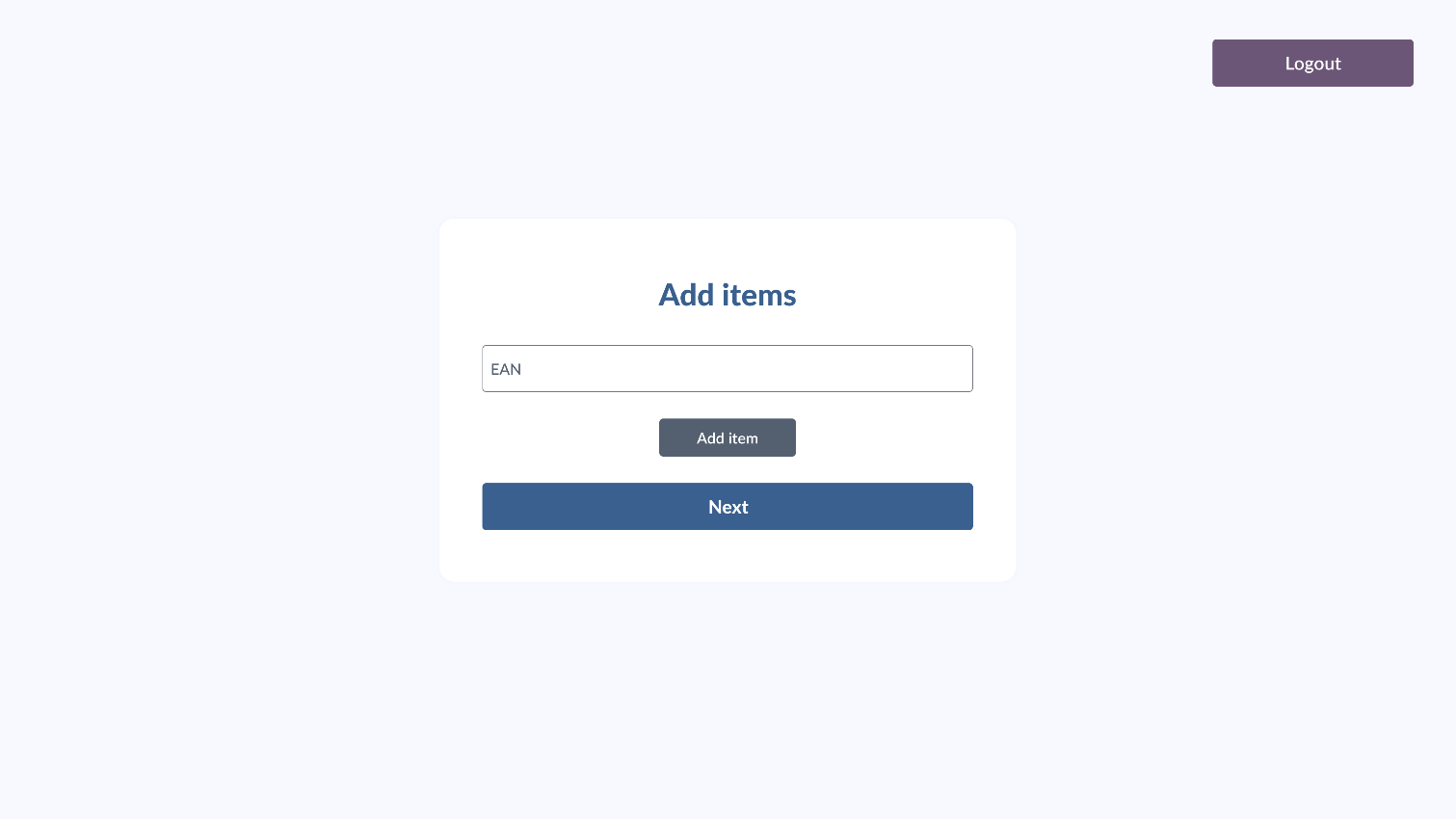
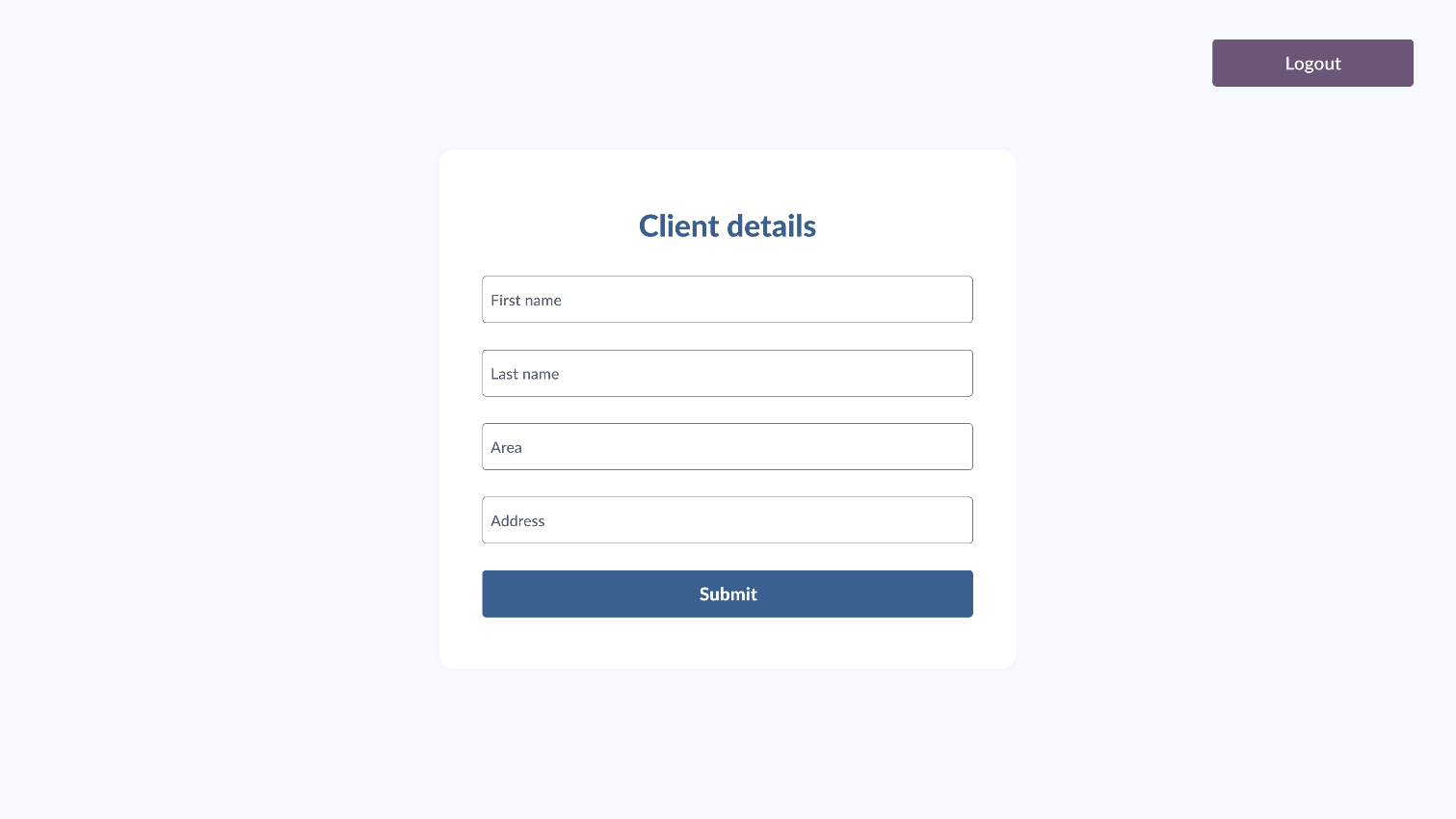
**Редактиране на поръчка:** Само администратори могат да изпълняват тази дейност. Стъпките, които трябва да изпълнят са както са показани на диаграмата (Фиг. 4).

**Изтриване на поръчка:** Само администратори могат да изпълняват тази дейност. Стъпките, които трябва да изпълнят са както са показани на диаграмата (Фиг. 5).

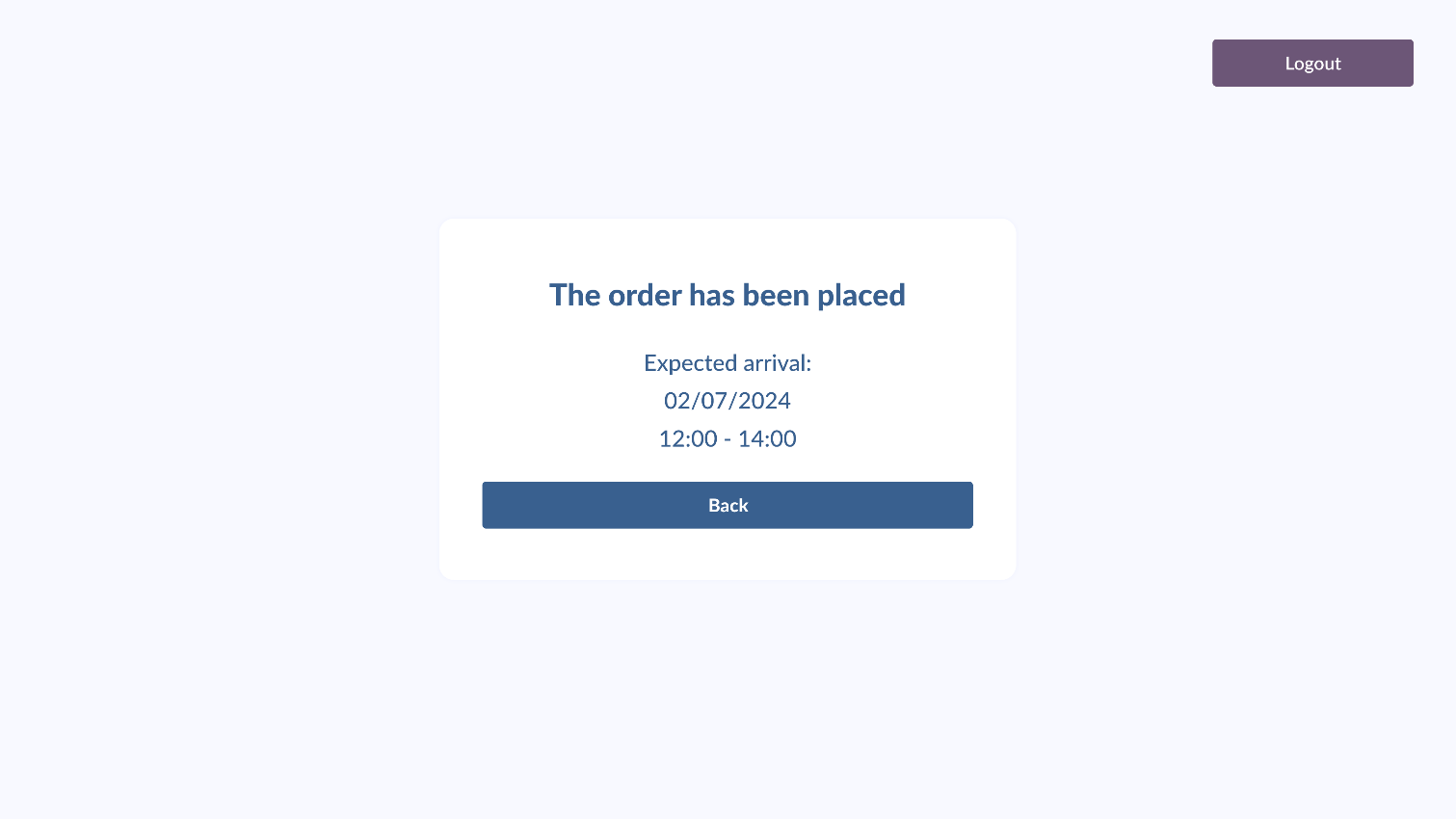
## Примерен потребителски интерфейс

*Фиг. 6 Страница за вход в системата*

*Фиг. 7 Страница за вход в системата със съобщение за грешка*

*Фиг. 8 Страница за добавяне на продукти в поръчка*

*Фиг. 9 Страница за подаване на данни за доставка*

*Фиг. 10 Страница за потвърждение на поръчка*

*Фиг. 11 Страница с дневен график*

Потребителският интерфейс включва изгледи за влизане в системата (Фиг. 6 и Фиг. 7), изгледи за продавач в магазин (Фиг. 8, Фиг. 9, Фиг.10) и такъв за доставчик (Фиг. 11). Целта е, системата да изглежда възможно най-опростено, за да може с нея да се работи бързо и ефективно. Дизайнът е минималистичен, осъществен с Figma, използвайки material theme за добра интуитивност и естетика.

Цветовата палитра също е предоставена от material theme, като схемата е следната:

|  |  |
| --- | --- |
| **Функция** | **HEX код** |
| **Фон** | **#F8F9FF** |
| **Основен акцент** | **#39608F** |
| **Вторичен акцент** | **#545F70** |
| **Третичен акцент** | **#6C5677** |
| **Грешка** | **#FFDAD6** |

## Диаграми на анализа

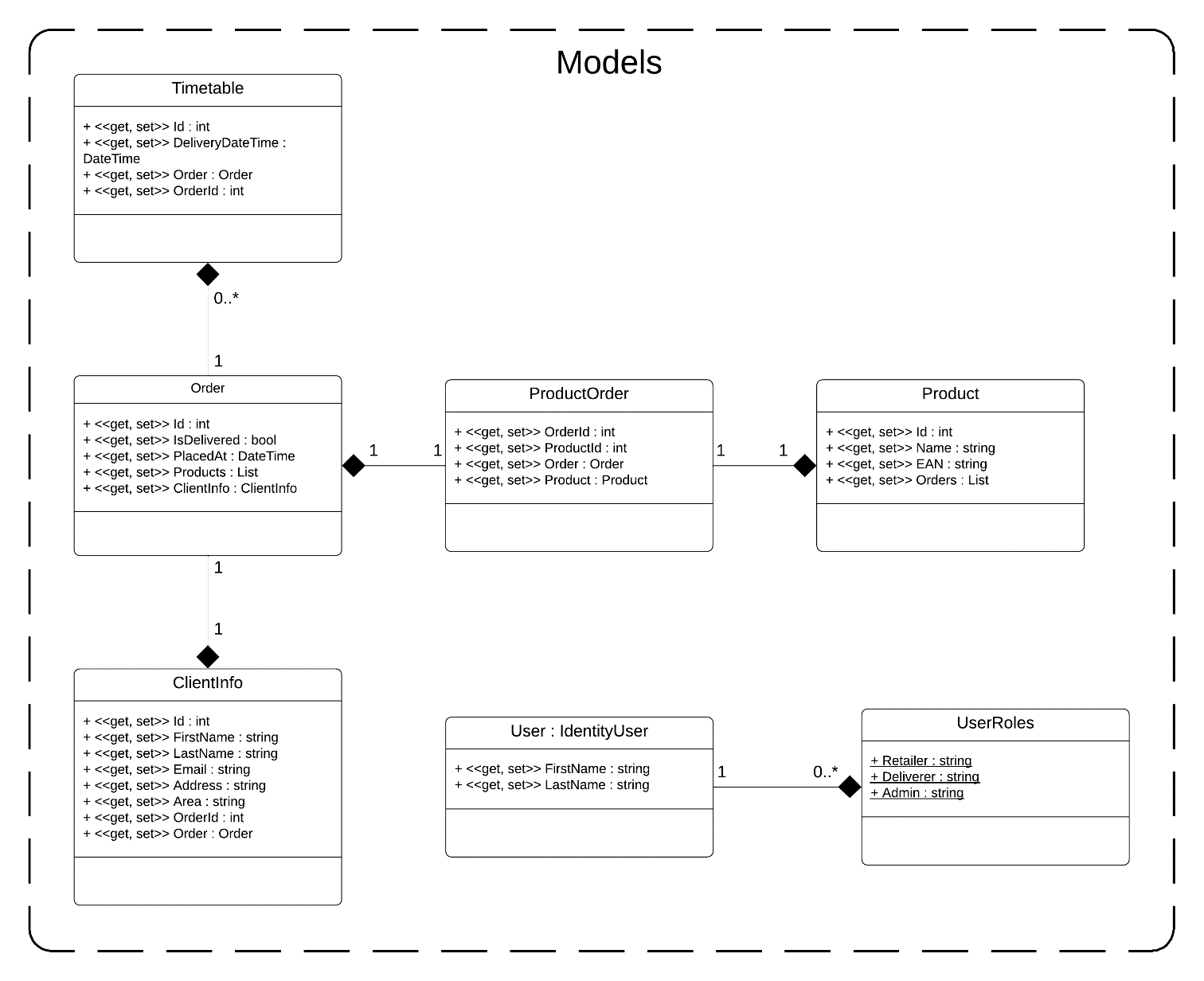
Всички диаграми могат да се намерят на следният електронен адрес:

<https://lucid.app/folder/invitations/accept/inv_d40c8338-64e0-4ce7-9a7d-6d2e01c11b4d>

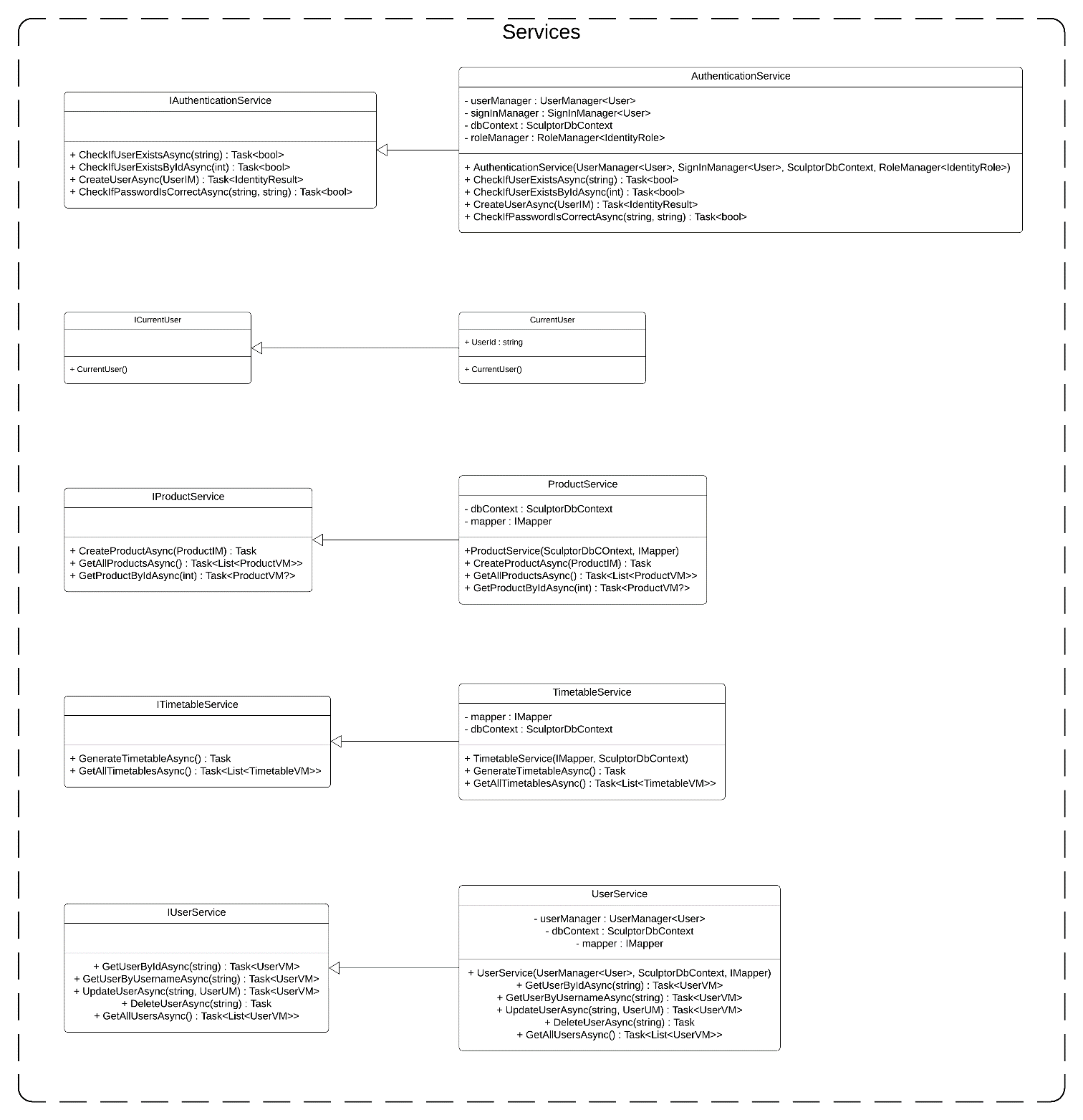
### ER диаграма на базата данни

*Фиг. 12 Диаграма на базата данни*

### Диаграма на класовете



*Фиг. 13 Диаграма на класовете на моделите*

**

*Фиг. 14 Диаграма на класовете на услугите*

## Модел на съдържанието / данните

Моделът на съдържанието/данните, представен в този проект, е базиран на следните ресурси:

* **Потребители** – това са доставчици, работници в магазин или администартори, които могат да извършват различни дейности, спрямо нивото си на достъп в системата. Всеки потребител се удостоверява и упълномощява с предварително зададени в базата данни потребителско име и парола.
* **Поръчки** – това са заявки за доставка на определен брой продукти. Всяка поръчка има адрес на доставка, заедно с останалите данни на клиента, дата и час на доставка и показател за това дали е била доставена.
* **Разписания** – поръчките автоматично се разпределят в разписания по ден и час. Всеки ден има по 40 поръчки, като са сортирани по квартал на доставка.
* **Продукти** – създават се от администраторът и имат име и EAN.

В проекта се използват текстово и графично съдържание от различни типове, които представляват информация за потребителите, поръчките, продуктите и разписанията. Текстовото съдържание се състои от имена, описания, пароли, часове и дати и т.н. Графичното съдържание се състои от представянето на API-я през Swagger.

# Дизайн

За решението е използвана софтуерната платформа .NET 7, която предоставя модерни и мощни технологии за разработка на уеб приложения, като добавка има ASP.NET Core за API, Entity Framework Core за ORM, Swagger за графично представяне за клиента.

## Реализация на структура на приложението (3-layer), Разделение на кода според предназначението му

Схемата на софтуерната архитектура на решението е базирана на принципите на трислойната архитектура, която разделя приложението на слоеве с ниска зависимост и висока съгласуваност. Слоевете са:

* **Презентационен слой** - това е слоят, който комуникира с потребителите и представя данните/съдържанието в удобен и интерактивен формат. В този проект този слой се представя от проекта Sculptor.PL и разделя функционалността на различни крайни точки и ги презентира чрез Swagger.
* **Слой за бизнес логиката** - това е слоят, който съдържа бизнес логиката на приложението. Той е независим от другите слоеве и определя правилата и ограниченията за работа с данните/съдържанието. В този проект този слой се представя от проекта Sculptor.BLL и разделя декларацията и имплементацията на методите и ги разделя по категории.
* **Слой на данните** – този слой се представя от проекта Sculptor.DAL и съдържа моделите и контекста за базата данни. Той използва EF Core за ORM и SQL Server за релационна база данни.
* В приложението има още един допълнителен проект – Sculptor.Common, който съдържа модели за вход и изход, които се споделят между различните слоеве. Те представят данните/съдържанието в сериализуем формат (JSON).

## Организация и код на заявките към база от данни

За достъп до базата данни е използван инструментариума EF Core (Entity Framework Core), който е ORM (Object-Relational Mapping) библиотека за .NET платформата. EF Core позволява работа с данните като с обекти и колекции, без да се налага писане на SQL заявки ръчно. EF Core поддържа различни видове бази данни, като в този проект се използва SQL Server.

За да се използва EF Core, трябва да се дефинират моделите на данните като класове в C# кода. Всеки модел има свойства, които отговарят на колоните в таблицата в базата данни.

За да се управляват връзката с базата данни и операциите с данните, трябва да се дефинира контекст за базата данни като клас, който наследява от базовия клас DbContext. В контекста трябва да се декларират свойства от тип DbSet<T>, където T е моделът на данните. DbSet<T> представлява колекция от обекти, които съответстват на таблица в базата данни.

За да се извършват заявки към базата данни, се използва LINQ (Language Integrated Query) синтаксис, който позволява да писането на заявки като изрази в C# кода. LINQ заявките се превръщат в SQL заявки от EF Core и се изпращат към базата данни. LINQ заявките могат да използват различни методи за филтриране, сортиране, групиране, проекция и агрегация на данните.

За да се добавят или изтрият обекти в базата данни, се използват методите Add, AddRange, Remove или RemoveRange на DbSet<T>. Тези методи променят само локалното състояние на обектите в контекста, без да ги променят в базата данни. За да се запазят промените в базата данни, трябва да се използва метода SaveChanges или SaveChangesAsync на контекста.

## Наличие и интуитивност на потребителски интерфейс (конзолен, графичен, уеб)

Приложението използва Swagger за визуализация на функционалностите. Swagger UI е сървър, чрез който се визуализира разработен API така, че да може да бъде изследван и тестван. Методите са представени като крайни точки, разпеделени по категории.

Основните функционалности на приложението са:

* **Вход** – потребителите могат да се впишат в системата чрез предварително регистрирани потребителско име и парола.
* **Регистриране и моделиране на продукти** – администраторът може да добави, промени и изтрие продукти в и от базата данни.
* **Регистриране и моделиране на поръчка** – продавачът в магазин създава нова поръчка по данни, подадени му от клиент. След това тази поръчка може да бъде отбелязана като доставена от доставчик или администратор.
* **Манипулация на потребители** – администраторския потребител има право да създава, променя и изтрива потребители от системата.
* **Разписания** – Разписанията могат да се създават, като те се разпределят автоматично в рамките на колкото се налага дни. След това доставчик може да види готовите разписания.

# Тестване

Unit testing-a е процес на тестване на отделни методи или класове в изолирана среда, без да зависят от външни ресурси или услуги. Целта на unit testing-a е да провери дали кодът работи правилно и да открие грешки или дефекти в логиката. Unit testing-a се извършва чрез писане на тестови класове и методи, които използват специални библиотеки за тестване, като xUnit за .NET платформата. Тестовите методи използват атрибути като [Fact] или [Theory] за да определят тестовите случаи. Тестовите методи също така използват Assert класа за да проверяват очакваните резултати от тестването.

# Заключение и възможно бъдещо развитие

В този проект е разработен API за автоматична организация на разписания на поръчки от кленти в магазин. Използван е ASP.NET за разработката; това спомага използването на само един език за програмиране – C# и платформа за сървър и клент - .NET Core. Използвани са също EF Core за ORM и SQL Server за база данни.

Entity Framework (EF) е ORM (Object-Relational Mapping) фреймуърк, предоставен от Microsoft, който позволява работата с данни, използвайки концепции от обектно-ориентираното програмиране, вместо директно да се занимават с таблиците в базата данни и SQL заявките. Ето някои ключови характеристики на Entity Framework:

* **Свойства** (Properties): Свойствата на класа на същността обикновено представят колони в съответната таблица в базата данни.
* **Навигационни свойства** (Navigation Properties): Тези свойства представляват връзки между различни същности, често отразявайки външни ключове в схемата на базата данни.
* **Атрибути или анотации** (Attributes or Annotations): Анотациите или атрибутите могат да се използват, за да предоставят допълнителна информация на Entity Framework за това как същността се отразява в базата данни.

Entity Framework позволява извършването на операции за създаване, четене, обновяване и изтриване (CRUD) върху тези същности, като се грижи за превод на тези операции в подходящи SQL заявки за взаимодействие с базата данни.

**Предимства:**

* Лек и крос-платформен:
  + EF Core е създаден специално за .NET Core и последващите версии (като .NET 5, .NET 6 и др.).
  + Може да работи на различни платформи, включително Windows, Linux и macOS.
* Обектно-ориентиран модел:
  + Позволява на .NET работата с база данни, използвайки обектно-ориентирани концепции.
  + Предоставя API за заявки към базата данни и извършване на CRUD операции.
* Нови функционалности:
  + EF Core въвежда подобрена поддръжка на LINQ, подобрени възможности за заявки и поддръжка на NoSQL бази данни (в някои сценарии).

**Недостатъци:**

* Липса на някои функционалности:
  + EF Core все още не поддържа всички функционалности на по-старите версии на Entity Framework (като EF 6).
  + Някои сложни сценарии могат да изискват допълнителни усилия.
* Ограничена поддръжка на някои бази данни:
  + Въпреки че EF Core поддържа много бази данни, все още може да има ограничения за някои специфични системи.

**Някои алтернативи са:**

* NHibernate:
  + NHibernate е зрял, open source ORM за .NET, който предоставя подобни функционалности като Entity Framework.
  + Поддържа различни бази данни и е добре известен сред опитните разработчици.
* Dapper:
  + Dapper е micro ORM, който се фокусира на бързодействие и минимална абстракция.
  + Позволява ви да използвате SQL заявки директно, но с по-лесен начин за мапиране на резултатите към обекти.
* EntityWorker.Core:
  + EntityWorker.Core е нов и бърз ORM, подобен на Entity Framework, но с повече функционалности.
  + Може да бъде по-бърз от Entity Framework и предоставя допълнителни възможности.

# Критерии и показатели за оценяване

|  |  |  |
| --- | --- | --- |
| **Показател** | **точки** | **срок** |
| 2. [Цели и обхват на софтуерното приложение](file:///O:/Admin204.1/Проекти/2021-2022/Ученически%20практики%202/Ученическа%20документация/user_name_Software%20Development.docx#_Цели_и_обхват)  3.1 [Потребителски изисквания и работен процес](file:///O:/Admin204.1/Проекти/2021-2022/Ученически%20практики%202/Ученическа%20документация/user_name_Software%20Development.docx#_Потребителски_изисквания_и) | 5  5 | 15.03.2024 |
| 3.2 [Примерен потребителски интерфейс](file:///O:/Admin204.1/Проекти/2021-2022/Ученически%20практики%202/Ученическа%20документация/user_name_Software%20Development.docx#_Примерен_потребителски_интерфейс)  3.3 [Диаграми на анализа](file:///O:/Admin204.1/Проекти/2021-2022/Ученически%20практики%202/Ученическа%20документация/user_name_Software%20Development.docx#_Диаграми_на_анализа)  3.4 [Модел на съдържанието/данните](file:///O:/Admin204.1/Проекти/2021-2022/Ученически%20практики%202/Ученическа%20документация/user_name_Software%20Development.docx#_Модел_на_съдържанието) | 5  5  5 | 29.03.2024 |
| 4.1 [Реализация на структура на приложението (3-layer),  Разделение на кода според предназначението му.  Допълване на Class диаграми/3.3/](file:///O:/Admin204.1/Проекти/2021-2022/Ученически%20практики%202/Ученическа%20документация/user_name_Software%20Development.docx#_Дизайн). | 10  10 | 19.04.2024 |
| 4.2 [Организация и код на заявките към база от данни](file:///O:/Admin204.1/Проекти/2021-2022/Ученически%20практики%202/Ученическа%20документация/user_name_Software%20Development.docx#_Дизайн). | 15 | 10.05.2024 |
| 4.3 [Наличие и интуитивност на потребителски интерфейс (конзолен, графичен, уеб)](file:///O:/Admin204.1/Проекти/2021-2022/Ученически%20практики%202/Ученическа%20документация/user_name_Software%20Development.docx#_Дизайн). | 10 | 31.05.2024 |
| 5. [Наличие и организация на автоматизирани тестове](file:///O:/Admin204.1/Проекти/2021-2022/Ученически%20практики%202/Ученическа%20документация/user_name_Software%20Development.docx#_Тестване). | 15 | 15.06.2024 |
| 6. Организация на проекта в система за контрол на изходния код и употреба на добри практики (merge requests, code reviews, branching strategy) | 10 | 25.06.2024 |
| 1. [Въведение. Ниво на завършеност на проекта](file:///O:/Admin204.1/Проекти/2021-2022/Ученически%20практики%202/Ученическа%20документация/user_name_Software%20Development.docx#_Въведение) |  | 30.06.2024 |
| Документация на проекта (XML comments, wiki, etc.) | 5 | текущо |
| Презентация на проекта |  | 30.06.2022 |
| Общо | 100 | Финал на първа фаза |