**ПРЕДМЕТ: РАЗРАБОТКА НА СОФТУЕР**

**ДЕЙНОСТ: УЧЕНИЧЕСКИ ПРАКТИКИ 2**

***2022-2023***

**Тема: „Приложение за резервиране на обяд”**

**Курсов проект**

*Автори:*

*Стоян Стоянов Иванов, клас XI В*

БургасСъдържание

[1. Въведение 3](#_Toc125379883)

[2. Цели и обхват на софтуерното приложение 3](#_Toc125379884)

[3. Анализ на решението 4](#_Toc125379885)

[3.1 Потребителски изисквания и работен процес 4](#_Toc125379886)

[3.1.1 UML Use-Case Диаграма 4](#_Toc125379887)

[3.1.2 UML Activity Диаграма 6](#_Toc125379888)

[3.2 Примерен потребителски интерфейс 8](#_Toc125379889)

[3.3 Диаграми на анализа 10](#_Toc125379890)

[3.3.1 ER диаграма на базата данни 10](#_Toc125379891)

[3.3.2 Клас диаграма на базите данни 11](#_Toc125379892)

[3.3.3 Клас диаграма на последователността 11](#_Toc125379893)

[3.4 Модел на съдържанието / данните 11](#_Toc125379894)

[4. Дизайн 11](#_Toc125379895)

[4.1 Реализация на структура на приложението (3-layer), Разделение на кода според предназначението му 11](#_Toc125379896)

[4.2 Организация и код на заявките към база от данни 11](#_Toc125379897)

[4.3 Наличие и интуитивност на потребителски интерфейс (конзолен, графичен, уеб) 11](#_Toc125379898)

[5. Тестване 12](#_Toc125379899)

[6. Заключение и възможно бъдещо развитие 12](#_Toc125379900)

[7. Използвани литературни източници и Уеб сайтове 12](#_Toc125379901)

[8. Приложения 12](#_Toc125379902)

[9. Критерии и показатели за оценяване 14](#_Toc125379903)

# Въведение

Идеята на проекта е система за запазване на обяд в училищна столова, с цел намаляването на загубата на храна. Ще се състой от мобилно приложение, уеб-сайт и хардуерна постановка.

Проблема, който се опитва да реши моето приложение, е свързан с изхвърлянето на храна. Според информация от Департамента на Земеделието на САЩ, “хранителните отпадъци се оценяват на между 30-40 процента от хранителните доставки” (САЩ, n.d.). Приложението ще се справя с този проблем, като предлага следните функционалности:

* Календарен изглед на седмицата с опция всеки ден да се запази обяд.
* Опция за отстъпване на храна – Ако ученик е запазил храна, но няма да успее да я консумира и е минал крайният срок за анулирането на поръчката, ще има опцията да я отстъпи на някой съученик, който е изпуснал крайния срок.
* Напомняне по e-mail или push нотификация за запазване на обяд.
* Автентикация през Office 365.
* Идентификация чрез бадж или мобилен телефон.
* Админ панел, в който готвачите могат да качват менюто и да гледат броя записани ученици и учители.

# Цели и обхват на софтуерното приложение

Приложението е насочено към предприятия и училища, които имат столови. Работниците / учениците / учителите в даденото предприятие ще имат достъп до седмичното меню и ще могат да си запазват обяд. Потребителите ще имат следните възможности:

* Календарен изглед с техните училищни часове/срещи и в свободното време опция за поръчка на храна.
* Избор от няколко менюта и свободна консумация, зададени от готвачите.
* Анулиране и запазване на обяд, не по-късно от 72 часа преди консумация.
* Ако срокът е минал, ще има опция за отстъпване на храната на други потребители.
* Опция изпращане на имейл или нотификация при наличие на нов обяд.
* Достъп до API-a чрез ключове.
* Мобилно приложение, с което потребители ще могат да се идентифицират пред готвачите.

Готвачите в стола ще могат да видят, кой е запазил обяд и така да поръчат толкова продукти, колкото са им нужни. Целта на проекта е да се улесни поръчването на храна и да се намали изхвърлянето на храна.

# Анализ на решението

## Потребителски изисквания и работен процес

### UML Use-Case Диаграма

Ето основните участници в нашата система:

* **Потребител:** Всеки ученик/служител, който ходи на обяд.
* **Администратор:** Може да проверя броя направени поръчки и да добавя менюта за идната седмица.

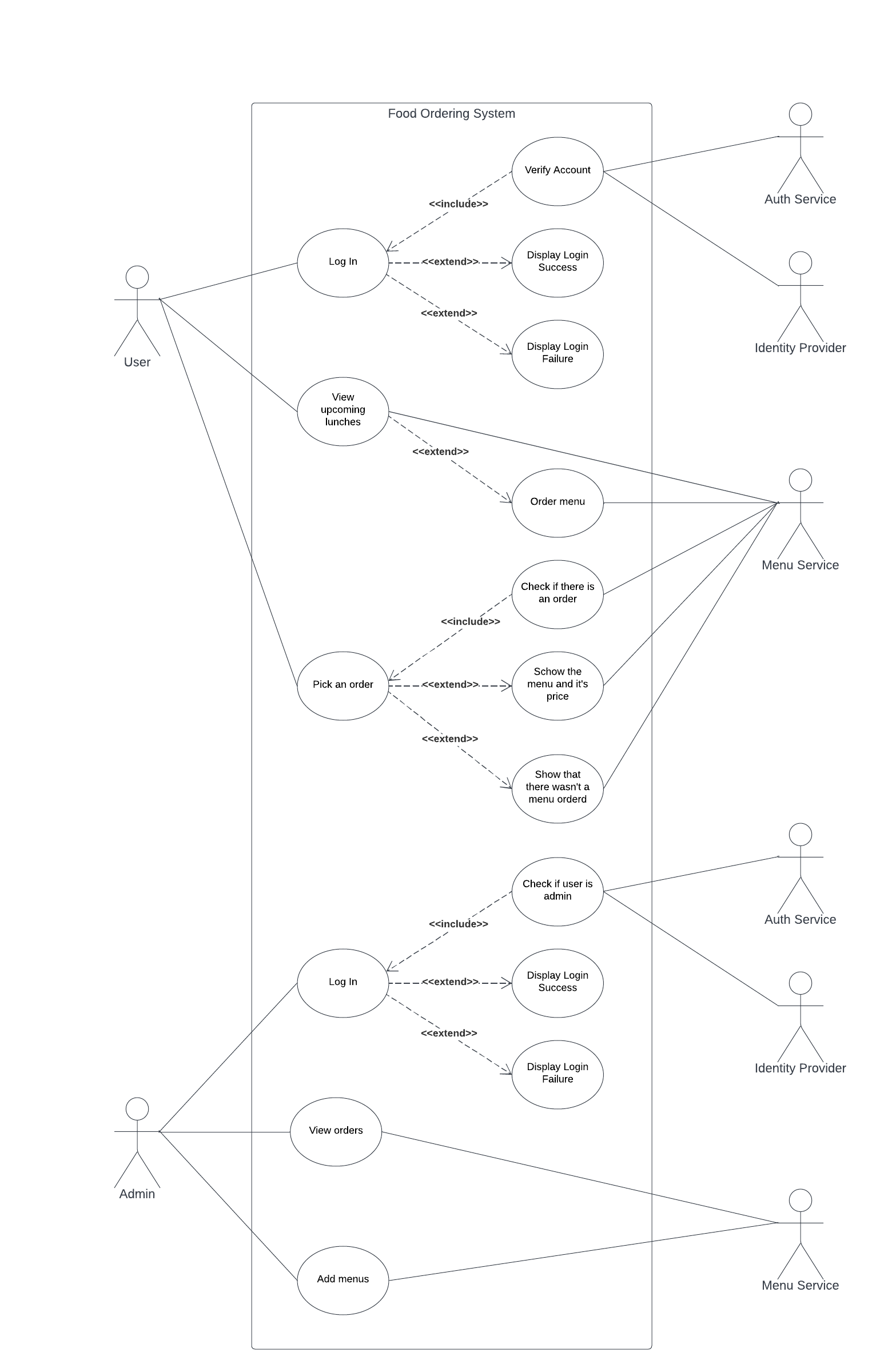
Ето най-популярните случаи на използване на системата за управление на поръчки:

**User:**

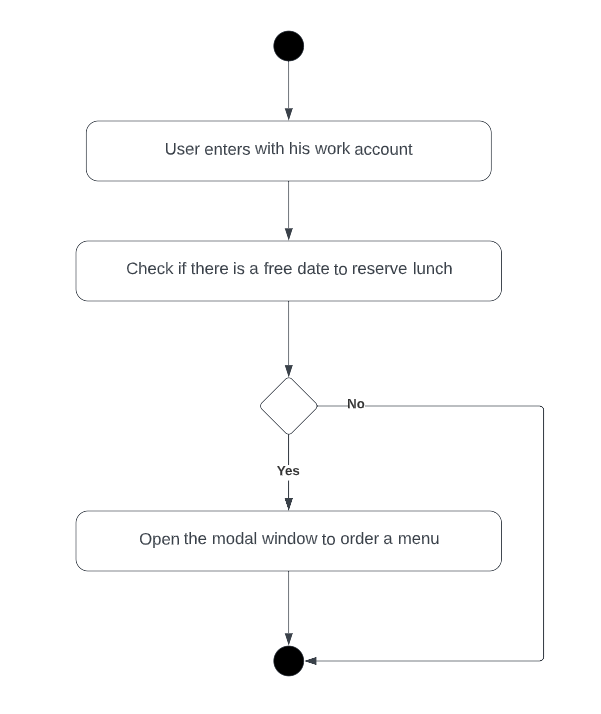
* **Log In:** Влиза в системата с помощта на Single Sign-On (SSO) от Azure AD.
  + **Verify Account:** Упълномощавайте потребителя с помощта на Microsoft Identity Provider.
* **View upcoming lunches:** Има опцията да види менюто за предстоящата седмица.
  + **Order Menu:** Може да направи поръчка за идната седмица
* **Pick an order:** Когато отиде на място, потребителя трябва да се идентифицира чрез бадж или мобилно устройство.

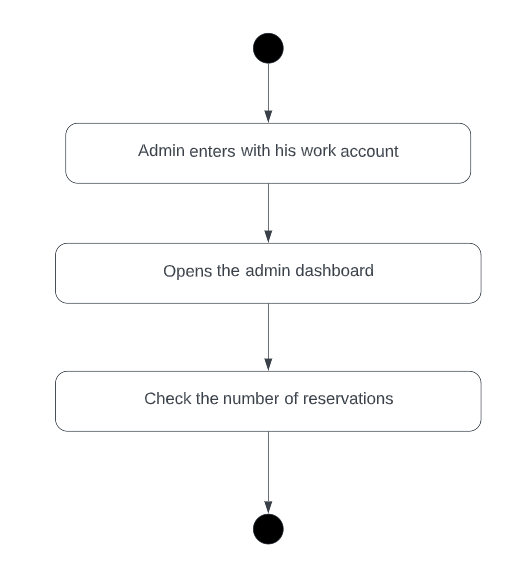
**Admin:**

* **Log In:** Влиза в системата с помощта на Single Sign-On (SSO) от Azure AD.
  + **Check is user is admin:** Проверява дали дадения user има роля admin.
* **View orders:** Статистика с това, кой какво е поръчал и по колко.
* **Add menus:** Добавя меню за следващата седмица.

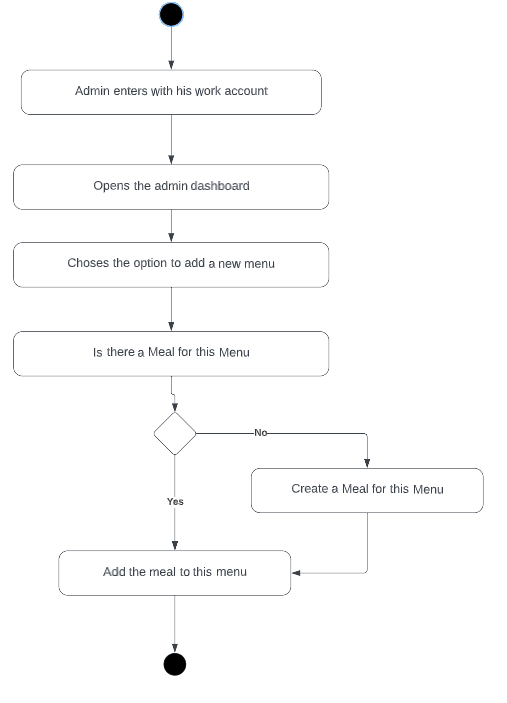


### UML Activity Диаграма

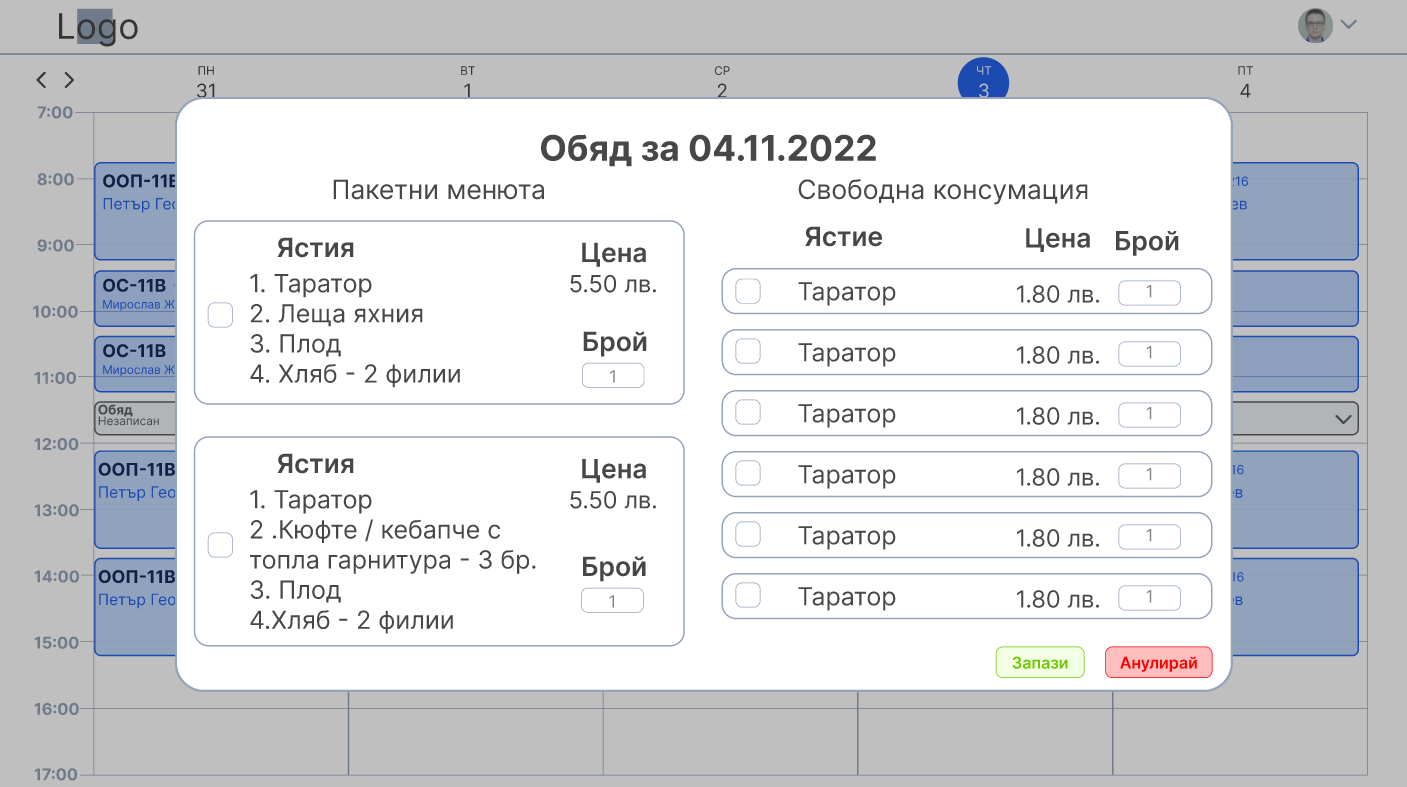
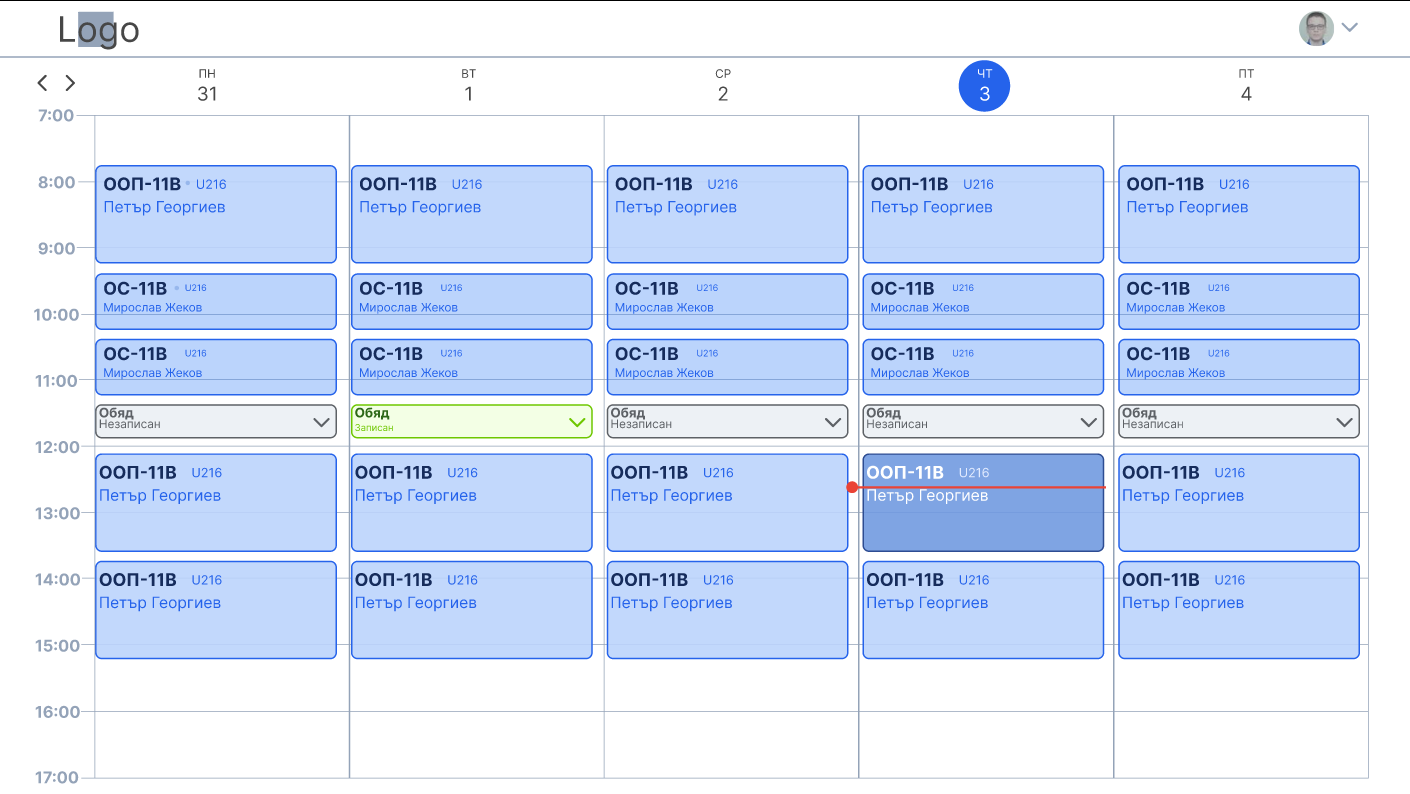
**Запазване на обяд:** Всеки един потребител може да изпълни тази дейност, Това са стъпките, които той трябва да изпълни:

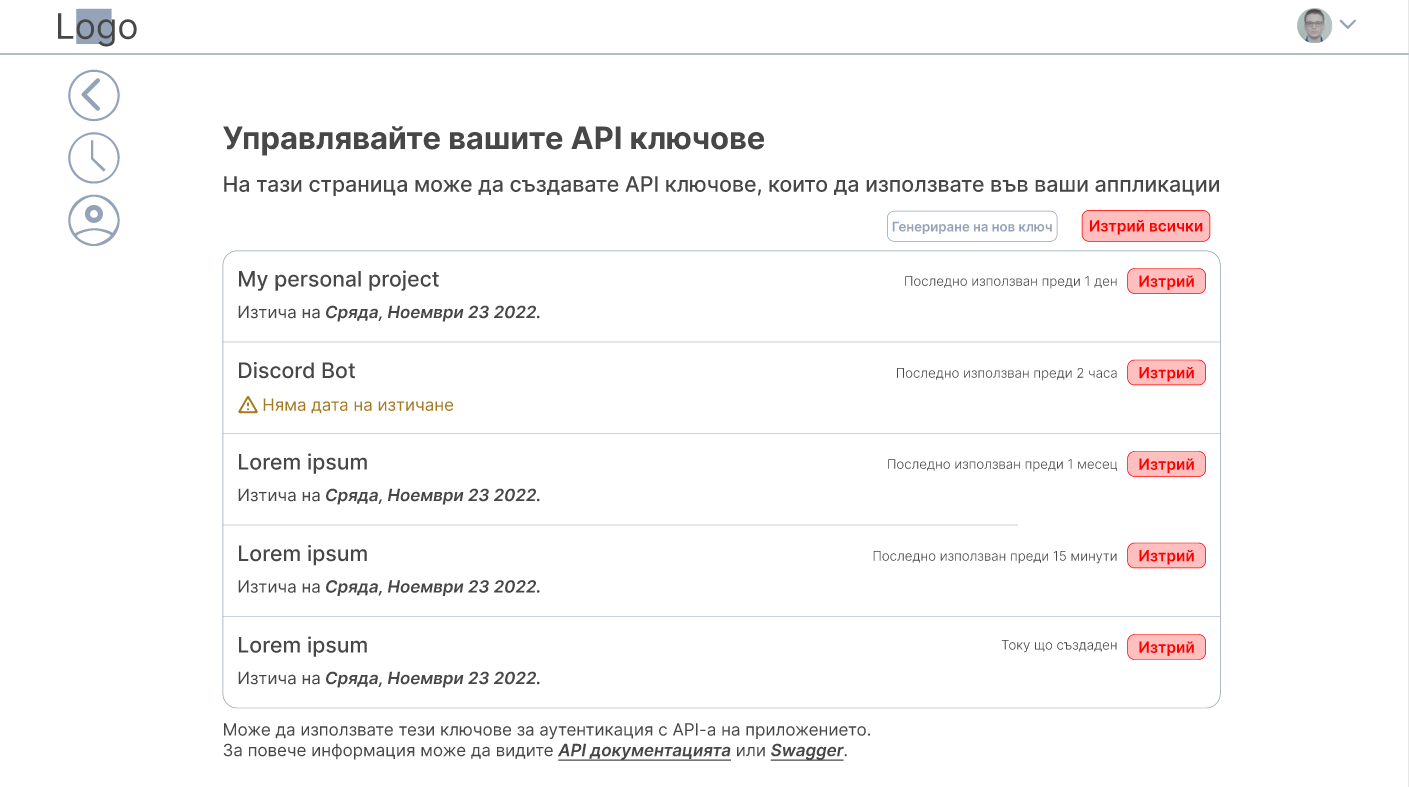
**Проверяване на броя на запазилите обяд:** Само потребителите с административни права могат да изпълняват тази дейност. Това са стъпките, които те трябва да изпълни:

**Добавяне на обяд:** Само потребителите с административни права могат да изпълняват тази дейност. Това са стъпките, които те трябва да изпълни:

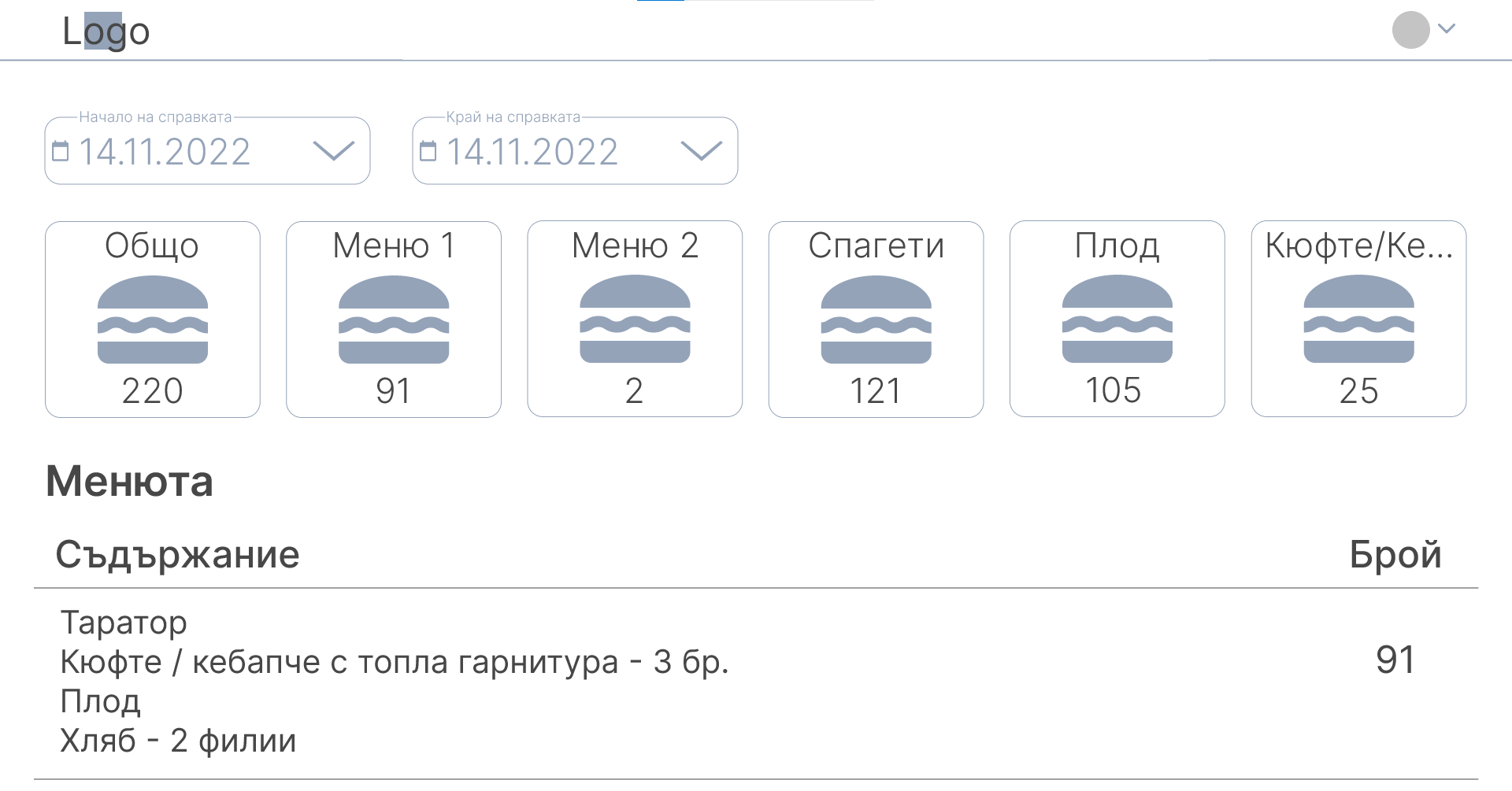


## Примерен потребителски интерфейс

Примерен потребителски интерфейс на потребителя:

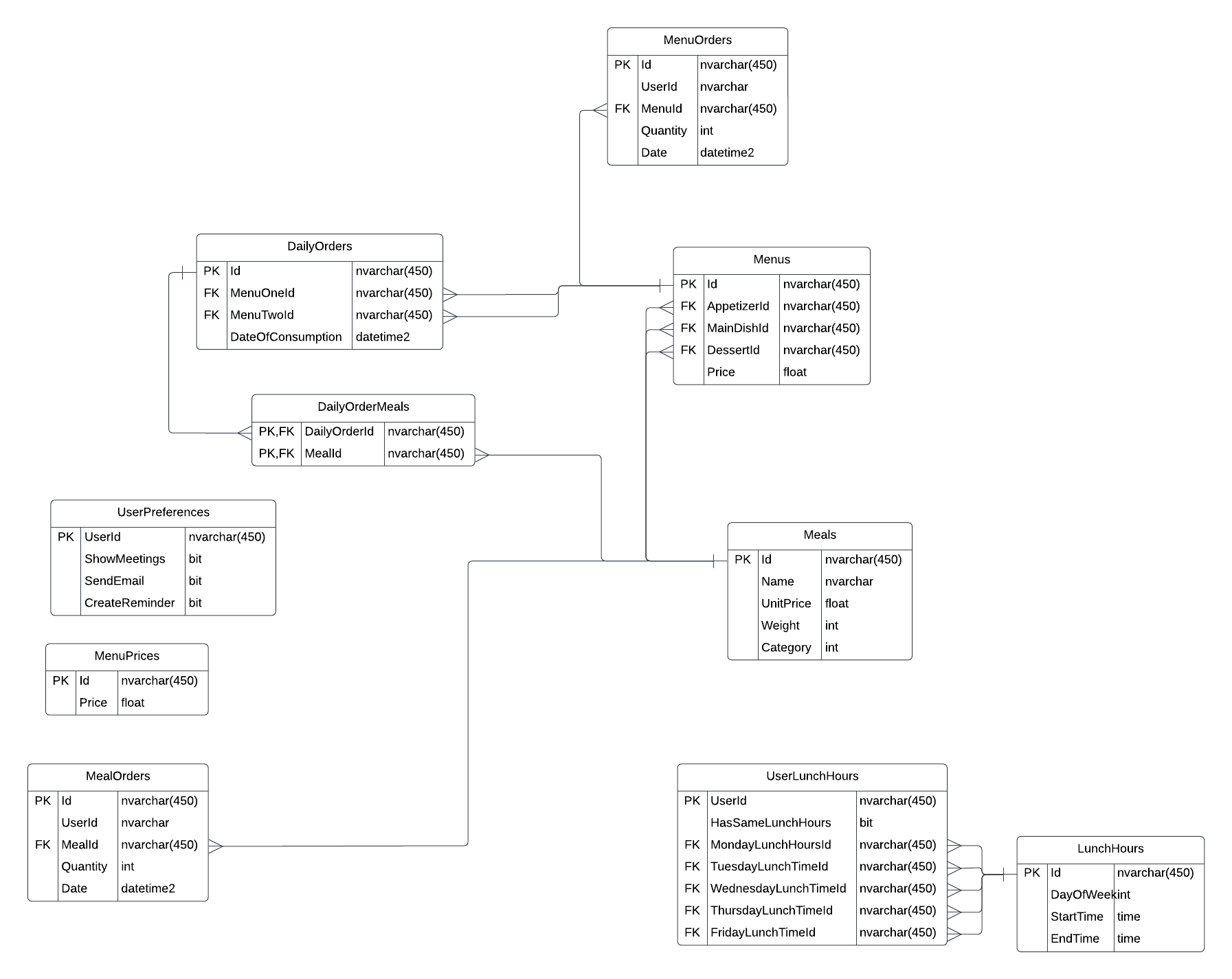


Примерен потребителски интерфейс на административния панела:



## Диаграми на анализа

### ER диаграма на базата данни



## Модел на съдържанието / данните

Моделът на данните/съдържанието, който ще представим в нашия проект, е базиран на следните ресурси:

* Потребители - това са ученици, учители или администратори, които могат да се регистрират и да влизат в платформата за резервация на обяд към училищна столова. Всеки потребител се удостоверява и упълномощаване с акаунта си от Azure Active Directory.
* Менюта - това са списъци от ястия, които се предлагат в кантината за определен ден от седмицата. Всеки ден има различно меню, което се състои от предястие, основно ястие и десерт. Всеки елемент от менюто има име, цена и хранителна стойност.
* Поръчки - това са заявки за резервация на ястия от менюто за определен ден и час. Всяка поръчка има потребител, който я е направил, дата и час на доставка, избрано меню и обща сума за плащане.
* Часове за обяд - това са интервали от време, през които кантината приема поръчки и доставя храна. Всеки интервал има начало и край, максимален брой поръчки, които може да обслужи, и статус (отворен или затворен).

В проекта ни ползваме текстово и графично съдържание от различни типове, които представят информация за потребителите, менютата, ястията и поръчките. Текстовото съдържание се състои от имена, описания, имейли, пароли, предпочитания, цени и хранителни стойности. Графичното съдържание се състои от снимки на потребителите.

# Дизайн

За решението на проекта сме избрали софтуерна платформа .NET 7, която ни предоставя модерни и мощни технологии за разработка на уеб приложения, като ASP.NET Core за API, EF Core за ORM и Blazor WASM за клиент.

## 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 (Three layer architecture), 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 нашия проект този слой се състои от няколко проекта:
  + CBCanteen.Client.ComponentLibrary - това е библиотека от Razor компоненти, която съдържа страници, лейъри и компоненти, които се използват от клиентските приложения.
  + CBCanteen.Client.Mobile - това е MAUI Blazor Hybrid мобилно приложение, което позволява на потребителите да резервират ястия от своите мобилни устройства.
  + CBCanteen.Client.Teams - това е Blazor Teams приложение, което позволява на потребителите да резервират ястия от своята Teams платформа.
  + CBCanteen.Client.Web - това е основното Blazor WASM клиентско приложение, което позволява на потребителите да резервират ястия от своя браузър.
  + CBCanteen.Server.WebHost - това е презентационният слой на API-то, който използва ASP.NET Core за изграждане на RESTful уеб услуги. Той обработва заявките от клиентите и ги препраща към домейн слоя чрез контролери и модели за изглед (view models).
* 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 нашия проект този слой се състои от следните проекта:
  + CBCanteen.Server.Services - това са услугите за бекенда, където се намира бизнес логиката за работа с потребителите, менютата, поръчките и часовете за обяд. Те използват интерфейси и абстракции за взаимодействие с другите слоеве.
  + CBCanteen.Client.Services - това са услуги, които се използват за комуникация между клиента и API-то. Те използват HttpClient за изпращане и получаване на заявки и отговори.
* Слой за даните - представен от следния проект:
  + CBCanteen.Server.Data - това е слоят за данни, където се намират моделите и контекстът за базата данни. Той използва EF Core за ORM и SQL Server за релационна база данни.
* В приложението има още един допълнителен проект
  + CBCanteen.Shared - това са модели за вход (input models) и изход (output models), които се споделят между API-то и клиента. Те представят данните/съдържанието в сериализуем формат (JSON).

## Организация и код на заявките към база от данни

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

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

За да управляваме връзката с базата данни и операциите с данните, трябва да дефинираме контекст за базата данни като клас, който наследява от базовия клас DbContext. В контекста трябва да декларираме 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 извършваме заявки към базата данни, използваме LINQ (Language Integrated Query) синтаксис, който ни позволява да пишем 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 добавяме или изтриваме обекти в базата данни, използваме 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 запазим промените в базата данни, трябва да използваме метода SaveChanges или SaveChangesAsync на контекста.

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

Приложението ни има уеб потребителски интерфейс, който е базиран на Blazor WASM технологията. Blazor WASM ни позволява да създаваме интерактивни и динамични уеб приложения, които се изпълняват в браузъра на потребителя. Интерфейсът е съставен от Razor компоненти, които са HTML елементи с C# код за логика и поведение.

Основните функционалности на интерфейса на приложението са:

* Вход - потребителите могат да влизат в платформата за резервация с техните акаунти от AAD (Azure Active Directory). За тази цел има страницата - Authentiaction
* Преглед на менюто - потребителите могат да преглеждат менюто за текущия или бъдещ ден. За тази цел има страница Menu, която показва списък от ястия, разделени по категории (предястие, основно ястие и десерт). Всяко ястие има име, цена и хранителна стойност. Потребителите могат да избират ястия от менюто чрез чекбоксове.
* Резервация на поръчка - потребителите могат да резервират поръчка за определен ден и час. За тази цел има страница Order, която показва избраното меню, общата сума за плащане и списък от свободни часове за обяд. Потребителите могат да избират час за обяд чрез падащо меню. Потребителите могат да потвърдят или откажат поръчката чрез бутони.
* Преглед на поръчките - потребителите могат да преглеждат своите поръчки за текущия или бъдещ ден. За тази цел има страница Orders, която показва таблица с колони за дата, час, меню и сума. Потребителите могат да филтрират поръчките по дата чрез календар. Потребителите могат да изтриват поръчки чрез бутони.

# 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.

В в апикацията сме създали тестов клас MealServiceTests, който тества методите на услугата IMealService. В конструктора на класа сме инициализирали контекста за базата данни (\_context), мапера (\_mapper) и услугата (\_service), като сме използвали InMemoryDatabase опцията на EF Core за да създадем база данни в паметта. След това сме дефинирали два тестови метода - GetAllMealsAsync\_Returns\_All\_Meals и GetMealByIdAsync\_Returns\_Meal\_With\_Given\_Id - като сме използвали атрибута [Fact] за да ги определим като тестови случаи. Всеки тестов метод се състои от три части - Arrange, Act и Assert. В частта Arrange сме подготвили данните за тестването - например сме добавили няколко ястия в базата данни. В частта Act сме извикали метода, който искаме да тестваме - например GetAllMealsAsync или GetMealByIdAsync. В частта Assert сме проверили резултата от тестването - например сме използвали методите на Assert класа за да сравним очакваните и получените стойности.

# Заключение и възможно бъдещо развитие

В този проект сме разработили уеб приложение за резервация на школска кантина, като сме използвали Blazor WASM технологията за клиентската част и ASP.NET Core за API-то. С това сме постигнали целта си да използваме един и същ език (C#) и платформа (.NET) за сървър и клиент кода. Също така сме използвали EF Core за ORM и SQL Server за база данни.

Използването на Blazor WASM има своите предимства и недостатъци, които трябва да вземем предвид при избора на технология за уеб разработка. Някои от тях са:

* Предимства:
  + По-бърз UI код - тъй като кодът се изпълнява в браузъра, няма нужда от заявки към сървъра за всеки клик или действие на потребителя. Това подобрява отговорността и усещането на потребителя.
  + Когато производителността има значение, използвайте WASM - тъй като WASM работи почти с нативна скорост в браузъра, той може да изпълнява по-сложни и интензивни операции от JavaScript.
  + Офлайн поддръжка - тъй като приложението се сваля в браузъра, то може да работи и без интернет връзка, ако не изисква достъп до външни ресурси или услуги.
  + Може да се разпространява чрез CDN, без нужда от сървър (освен за API) - това означава, че приложението може да се доставя по-бързо и ефективно до потребителите.
  + Може да се изпълнява всеки .NET Standard 2.0 C# код - това означава, че можем да използваме библиотеките, които сме създали, както и голям брой NuGet пакети.
* Недостатъци:
  + Изисква се API слой, ако искаме да достъпваме защитени ресурси - това означава, че трябва да имаме допълнителен проект за API-то и да го хостваме на собствен или облачен сървър.
  + Дебъгването е все още ограничено - тъй като WASM е нова технология, инструментите за дебъгване не са толкова зрели и функционални, колкото при други уеб технологии.

Алтернативата на Blazor WASM е Blazor Server, която използва SignalR за комуникация между клиента и сървъра. Тази версия на Blazor има своите предимства и недостатъци спрямо Blazor WASM, като:

* Предимства:
  + По-малко памет и процесорно натоварване на клиента - това е полезно за слаби устройства или мобилни браузъри.
  + По-добра поддръжка на .NET библиотеките - това е полезно ако искаме да използваме библиотеки, които не са съвместими с .NET Standard 2.0.
  + По-добра поддръжка на стари браузъри - това е полезно ако целим потребители, които използват браузъри без поддръжка на WASM.
* Недостатъци:
  + Повече заявки към сървъра - това може да доведе до по-ниска производителност и по-голяма консумация на ресурси.
  + Зависимост от интернет връзката - ако връзката е прекъсната или забавена, приложението ще спре да работи или ще работи лошо.
  + Сигурността на данните - ако имаме чувствителни данни в кода, трябва да гарантираме, че комуникацията между клиента и сървъра е криптирана и защитена.

Използуемостта на подобни решения в практиката е голяма, тъй като Blazor WASM предлага нов начин за разработка на уеб приложения с C# и .NET. Това може да спести време и усилия на разработчиците, които не трябва да учат нов език или фреймуърк за клиентската част. Също така Blazor WASM може да предложи по-добра производителност и потребителски опит от JavaScript.

# 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 | 04.03.2022 |
| 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 | 18.03.2022 |
| 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 | 16.04.2022 |
| 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 | 30.04.2022 |
| 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 | 31.05.2022 |
| 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 | 15.06.2022 |
| 6. Организация на проекта в система за контрол на изходния код и употреба на добри практики (merge requests, code reviews, branching strategy) | 10 | 25.06.2022 |
| 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) |  | 30.06.2022 |
| Документация на проекта (XML comments, wiki, etc.) | 5 | текущо |
| Презентация на проекта |  | 30.06.2022 |
| Общо | 100 | Финал на първа фаза |