Природо-математическа гимназия

"Васил Друмев"

гр. Велико Търново

ДИПЛОМЕН ПРОЕКТ

за придобиване на трета степен

на професионална квалификация

Тема: Изграждане на комплекс от методи,

средства и подходи при разработване на приложение

„Разпространение на българска и

чуждестранна преса – ЕТ „Вест““

Ученик:

Маргарита Ивова Стайкова

Ръководител консултант:

Мариана Русанова

Професия: „Приложен програмист“

Специалност: „Приложно програмиране“

Велико Търново

2024 г.

Съдържание

Увод

Обект на настоящия дипломен проект е проектирането на уеб-базирано приложение, насочено към по-ефективното функциониране на фирма ЕТ „Вест“. Като основен предмет на дейност е поставено разпространението на българска и чуждестранна преса на територията на град Велико Търново. В сферата на разработката се фокусираме върху изграждането на уеб-приложение с добре структуриран дизайн, предоставящо удобни методи за въвеждане и извеждане на информация, като това ще оптимизира по-голяма част от процесите в дейността на фирмата. В обхвата на разработката са включени следните ключови дейности:

- подаване на заявки към доставчиците;
- отчитане на продажбите и брака;
- обработка на заявки от обектите.

За постигане на ефективност и функционалност системата поддържа три различни вида потребители: администратор, собственик и служител.

Целите на дипломния проект включват изграждането на посочения уебсайт, представяйки подробно всички етапи от процеса на създаването му. Реализирането на поставените цели включва следните етапи:

- Съставяне на план и преглед на различни подходи и технологии в програмирането;
- Анализ на поставените цели;
- Избор на технологии и методи за реализация;
- Изпълнение на плана и постигане на поставените цели.

Структурата на тази дипломна работа е разделена в две части. В първата се извършва анализ на съвременните технологии, използвани за разработка на уеб-базирани бизнес приложения. Това включва преглед на възможностите на програмиране в интернет среда, както и необходимият софтуер за създаване на подобни приложения. Представен е анализ на различните модели и езици за реализация на бази данни, както и системите за управление на тези бази данни. Този анализ е насочен към избора на подходящи инструменти за ефективно управление на данните.

Във втората част е представена архитектурата на уеб-базираното приложение. Тук са представени основните компоненти и тяхното взаимодействие, осигурявайки ясен обзор на структурата на проекта.

Теоретична част

Глава I - Технологии за разработка на уеб-базирани приложения

1. Особености на уеб-базираните приложения

Уеб-базирана информационна система е система, която използва уеб технологии за доставянето на информация и услуги до потребители или други информационни системи (приложения). Чрез тази софтуерна система се публикуват и използват данни посредством хипертекстови принципи.

Информационна уеб система обикновено се състои от един или повече уеб приложения, специфични функции, ориентирани към компоненти, заедно с информационни елементи и други не-уеб компоненти. Web браузър обикновено се използва като **front-end**, докато базата данни като **back-end**.

Въпреки съществуващите прилики между традиционната и уеб-базирани информационни системи, съществуват и значителни различия. Уеб-базирани информационни системи могат да се окажат предизвикателство пред разработчика, който има задачата да се справя с мрежи и потребители, на различни типове данни, както и въпроси, свързани с управление на съдържанието, представянето и използваемостта, които са неизвестни по отношение на условия. Това се дължи на факта, че потребителите могат да бъдат от различни места и да имат различни изисквания.

2. Основни технологии за разработка на уеб-базирани приложения

В рамките на настоящата дипломна работа ще бъдат разгледани ключовите технологии, които оказват значително влияние върху разработката и изграждането на съвременни уеб-базирани приложения. Нужен е обширен набор от програмни езици, обхващайки от back-end езиците, използвани за взаимодействие с базата данни, до уеб езиците, които служат за визуализация на сайта в интернет средата.

Като начало ще бъде направено въведение в основните езици за уеб програмиране, включително **C#**, **SQL**, **HTML** и **CSS**. Особено внимание ще бъде отделено на **ASP.NET Core** библиотеката, която предоставя модерни и изпробвани решения за създаване на уеб приложения.

Допълнително, ще бъде представена същността на архитектурния шаблон **Модел-Изглед-Контролер** (MVC). Този шаблон е широко използван в съвременното програмиране и предоставя структура и организация, които допринасят за яснота и ефективност при проектирането и разработването на уеб-базирани приложения.

2.1. HTML (Hypertext Markup Language)

HyperText Markup Language (HTML) е основният език за описание на уеб страници, който се използва за създаване на графични и текстови елементи на страниците. HTML е стандарт в Интернет, като правилата за него се определят от международния консорциум **W3C**. Най-новата версия на стандарта е **HTML5**.

Описанието на документа става чрез специални елементи или маркери, които се състоят от **етикети** или **тагове** (HTML tags). HTML елементите са градивните блокове на HTML страниците. С HTML конструкции, изображения и други обекти като интерактивни форми могат да бъдат вградени в изобразената страница. HTML предоставя средства за създаване на структурирани документи чрез обозначаване на структурна семантика за текст като заглавия, параграфи, списъци, връзки, цитати и други елементи. HTML елементите са очертани от тагове, написани с ъглови скоби. Тагове като <img /> и <input /> директно въвеждат съдържание в страницата. Други тагове като и обграждат и предоставят информация за текста на документа и могат да включват тагове на поделементи. Браузърите не показват HTML таговете, но ги използват, за да интерпретират съдържанието на страницата.

HTML файловете обикновено се пишат в текстови редактори и се хостват на сървъри, свързани към Интернет. Те съдържат текстово съдържание с маркери, които инструктират браузърите как да показват съдържанието на страницата. Уеб браузърите прочитат HTML документите и ги превръщат в уеб страници, като скриват HTML таговете и показват само съдържанието на страницата.

Едно от основните предимства на HTML е, че документите, създадени по този начин, могат да се разглеждат на различни устройства, включително компютри, таблети и мобилни устройства. Създаването на HTML-базирани уеб страници може да се извършва с помощта на обикновен текстов редактор или специализирани инструменти за създаване на уеб страници, като **Microsoft FrontPage**, **Macromedia Dreamweaver**, **Notepad** и други.

В допълнение към HTML, дизайнерите на уеб страници използват Cascading Style Sheets (CSS), за да задават стилове на съдържанието, като например цветове, шрифтове и разположение. CSS се използва за засилване на дизайна и визуалния аспект на уеб страници.

Пример за използване HTML, част от дипломния проект (Фиг. 1).

```
<h2>Доставчици</h2>
<table class="table">
    <thead>
        <tr>
            <th>Име</th>
            <th>Телефонен номер</th>
            <th>Имейл</th>
            <th>Град</th>
            <th>Действия</th>
        </tr>
    </thead>
</table>
```

Фиг. 1. HTML код

2.2. CSS (Cascading Style Sheets)

Cascading Style Sheets (CSS) е език за стил, използван за определяне на външния вид на документ, написан на език за маркиране като HTML. CSS играе ключова роля в създаването на уеб проекти във **World Wide Web**, заедно с HTML и JavaScript.

Целта на **CSS** е да позволява разделянето на съдържанието и представянето, включително форматиране, цветове и шрифтове. Това разделение може да подобри достъпността на съдържанието, предоставяйки по-голяма гъвкавост и контрол при определяне на характеристиките на представянето. То позволява на множество уеб страници да споделят общи правила за стилове, които се определят в отделен **.css** файл. Този подход намалява сложността и повторенията в структурата на съдържанието и позволява кеширането на активирания **.css** файл в браузъра, подобрявайки скоростта на зареждане на страниците, които споделят този файл и неговото форматиране.

Името "**каскаден**" произлиза от приоритетната схема, която определя кое правило за стил се прилага, ако има повече от едно правило, което съответства на

определен елемент. Тази приоритетна схема е предвидима и осигурява яснота в определянето на стиловете. В допълнение към HTML, CSS се поддържа от други езици за маркиране като **XHTML**, обикновен **XML**, **SVG** и **XUL**.

Синтаксисът на CSS е прост, използвайки ключови думи на английски език, за да уточнява имената на различни стилови свойства. Стиловата таблица се състои от списък с правила, като всяко правило се състои от един или повече селектори и блок за декларации.

Синтаксисът на CSS е съставен от три части: **селектор** (selector), **характеристика** (property) и **стойност** (value). Селекторът е нормален HTML елемент (таг), който трябва да се промени, характеристиката е атрибутът за смяна, като всяка характеристика си има стойност (Фиг.2).

Фиг. 2. CSS синтаксис

Селекторите в CSS служат да дадат информация на браузъра, за кои елементи е предназначено правилото, следващо след селектора. Използват се само за декларациите, които се поставят между тагове **<style></style>** или в отделни файлове. За декларациите, намиращи се в стойностите на атрибута **style** на елементите, селектори не са необходими, защото тези декларации е ясно, че трябва да се приложат само върху съответния елемент.

- **Селектор по тип (име) на таг** - Този селектор съвпада с име на таг.

```
body { background-color: yellow; }
p { text-align: justify; }
```

В примера, първото правило прави цвета на фона на целия документ жълт, а второто правило прави всички абзаци текст да са подравнени двустранно.

- **Селектор по клас** - Клас е понятие от HTML. Клас е група от елементи. Принадлежността на даден елемент към някой клас се посочва с атрибута с име class.

  <h1 class="a">Заглавие от клас "a"</h1>

  ** Абзац от клас "a". **

  ** Абзац от клас "b". **

  ** Абзац, който попада и в клас "a", и в клас "b". **

Имената на класовете се измислят от съставителя на HTML кода. CSS селектор, който трябва да определи клас елементи, за които се отнася правилото, е името на класа с точка пред него.

.a { border: solid 1px; }

h1.a { border-width: 2px; }

.b { border-radius: 5px; }

В този пример: първото правило налага всички елементи от клас a да имат рамка с дебелина 1 пиксел; второто - всички заглавия h1 от клас a да имат дебелина на рамката от 2 пиксела; а третото - всички елементи от клас b да имат заоблени ъгли с радиус 5 пиксела.

- **Селектор по идентификатор** - Идентификаторът е атрибут с име **id**, който се поставя на определени елементи. Целта е съответният елемент да се идентифицира единствено по стойността на своя идентификатор. Правилно е в HTML кода на една страница стойностите на идентификаторите да не се повтарят. (Всеки идентификатор да е уникален!). Имената на идентификаторите се измислят от автора на HTML кода.

** Абзац с идентификатор. **

Абзацът в примера има идентификатор **important**. За да се приложи CSS правило към елемента, носещ съответния идентификатор се използва селектор, който съдържа знак #, следван от стойността на идентификатора.

#important { text-weight:bold; color:red; }

Това правило ще направи абзаца текст с идентификатор important да се изобрази с получер шрифт в червен цвят.

- **Други** - Съществуват и други типове селектори като, селектори по атрибути, групи и комбинации от селектори, псевдокласове и псевдоелементи, и др.

Пример за използване CSS, част от дипломния проект (Фиг. 3).

```
html {
  font-size: 14px;
}

@media (min-width: 768px) {
  html {
    font-size: 16px;
  }
}

.btn:focus,
.btn:active:focus,
.btn-link.nav-link:focus,
.form-control:focus,
.form-check-input:focus {
  box-shadow: 0 0 0 0.1rem white, 0 0 0 0.25rem #258cfb;
}

html {
  position: relative;
  min-height: 100%;
}

body {
  margin-bottom: 60px;
}
```

Фиг. 3. CSS код

2.3. Bootstrap

Bootstrap е библиотека, която помага за проектирането на уеб интерфейси с готови CSS и JavaScript компоненти за общи елементи като навигация, формуляри и бутони. Той също така има отзивчива система за оформление на мрежата, която настройва сайта за устройства с различни размери на екрана, като телефони и таблети. Използвайки системата за оформление Bootstrap, може да се създаде сайт, който

изглежда добре на всяко устройство. Главното предимство е, че е съвместим с всички основни браузъри.

Библиотеката включва няколко **JavaScript** компонента във формата на **jQuery** плъгини, които предоставят допълнителни елементи на потребителския интерфейс, като диалогови прозорци и пояснения, и разширяват функционалността на някои от съществуващите елементи на интерфейса.

2.4. JavaScript

JavaScript е скриптов език за програмиране, чрез който статичната уеб страница може да се „раздвижи“. Раздвижването може да бъде чрез промяна на съдържанието, което се визуализира на уеб страницата след извършване на действия от страна на потребителя, контрол над уеб браузъра и други. JavaScript се използва, освен за създаване на интерактивни уеб страници, но и за разработка на игри и скриптове, които се изпълняват от страна на сървъра.

Програмите създадени с JavaScript се наричат скриптове. Файлът, в който се съдържа кода на скрипта, е с разширение **.js**.

JavaScript скриптовете могат да се вграждат в HTML кода на уеб страницата, директно като код или да се извикват като външен .**js файл**. При зареждането на уеб страницата, външният **.js** файл се сваля локално на компютъра на потребителя и самото изпълнение на скрипта се извършва от уеб браузъра.

За изпълнението на JavaScript скрипт не е нужно преди това кода да бъде компилиран, а директно се изпълнява/интерпретира от уеб браузъра.

Обикновено при работата на даден уеб сайт взимат участие две страни (сървър и клиент) – сървърът, на който се намира сайта, и уеб браузърът на потребителя, през който се разглежда сайта. Уеб браузърът можем да наричаме **клиент**. Някои от елементите на сайта се изпълняват от страна на клиента, като например HTML, CSS, JavaScript, картинки и всички други, които се свалят локално на компютъра на потребителя. От страна на сървъра се изпълняват елементите, които обработват заявките създадени от клиента към сървъра и връщат желаната информация, като например **PHP**, **XML** и други.

JavaScript може да се изпълнява и на сървъра. За целта се използва **Node.js**. В допълнение, може да влияе на почти всяка част от браузъра. Браузъра изпълнява JavaScript кода в цикъла на събития т.е. като резултат от действия на потребителя или събития в браузъра (например document.onLoad).

Възможности на JavaScript:

- Зареждане на данни чрез AJAX;
- Ефекти с изображения и HTML елементи: скриване/показване, пренареждане, влачене, слайд шоу, анимация и много други;
- Управление на прозорци и рамки;
- Разпознаване на възможностите на браузъра;
- Използване на камера и микрофон;
- Създаване на 3D графики WebGL;
- По-добър и гъвкав потребителски интерфейс.

Какво не може да се прави с помощта на JavaScript:

- Не могат да се стартират локални приложения;
- Записването на информация на потребителския компютър или отдалечения сървър е невъзможно;
- Не може да се запазва информация директно в отдалечена база данни.

2.5. ASP.NET CORE

ASP.NET Core е безплатна софтуерна рамка за уеб разработка, с отворен код. Също така тя се явява и следващата стъпка в еволюцията ASP.NET. Тя е разработена съвместно от Microsoft и общността, която е събрала през годините на своето развитие. ASP.NET Core е модуларна софтуерна рамка, която може да върви както на пълната .NET рамка, така и на крос-платформената .NET Core. Въпреки, че е нова софтуерна рамка, изградена върху нов web stack, тя има висока степен на съвместимост с ASP.NET MVC.

Спрямо своите предшественици ASP.NET Core поддържа нова функция – т.нар. „side by side versioning”. При нея различни приложения, които използват една и съща машина, могат да таргетират различни версии на ASP.NET Core, в зависимост от версиите (и нуждите) си. Това не е възможно с по-стари издания на ASP.NET.

Някои от най-основните ѝ предимства са скоростта, широкият спектър от поддържани езици и фактът, че е безплатна. Тя е вградена в познатата Windows сървърна среда, което я прави лесно достъпна, тъй като не са нужни допълнителни инсталация и конфигурация. Също така, ресурси за ASP.NET Core се намират лесно, благодарение на популярността на рамката.

Уеб-сайтовете и апликациите, построени с ASP.NET Core са по-бързи от тези, написани на PHP например, тъй като кодът се компилира, след което се изпълнява. Компилацията се случва само веднъж, а платформата може да го екзекутира отново и отново без да се налага да го чете всеки път. Този метод на компилация също значително подобрява търсенето и идентифицирането на грешки в кода.

Не бива да се пропуска и факта, че въпреки отворения код на рамката, тя все пак се разработва и поддържа от една от най-големите софтуерни компании в света – Microsoft. Това означава, че компанията инвестира в платформата, а техните партньори я използват. Съответно нищо от това, което ще създадете с ASP.NET Core, няма да остарее скоро.

В крайна сметка, ASP.NET Core е прекрасна рамка за създаване на уебсайтове и уеб апликации. Тя е надеждна, бърза, лесна за ползване, безплатна и изключително популярна. Дава пълен контрол над работата и може да се използва както за големи, така и за малки проекти.

2.6. Езикът C#

C# е съвременен обектно-ориентиран език за програмиране с общо предназначение, създаден и развиван от Microsoft като част от .NET платформата. На езика C# и върху .NET платформата се разработва изключително разнообразен софтуер: офис приложения, уеб приложения и уеб сайтове, настолни приложения, мултимедийни приложения, приложения за мобилни телефони и таблети, игри и много други. Всички C# програми са обектно-ориентирани. Те представляват съвкупност от дефиниции на класове, които съдържат в себе си методи, а в методите е разположена програмната логика - инструкциите, които компютърът изпълнява.

В днешно време това е един от най-популярните езици за програмиране. На него пишат милиони разработчици по цял свят. Тъй като C# е разработен от Майкрософт като част от тяхната съвременна платформа за разработка и изпъл¬нение

на приложения .NET Framework, езикът е силно разпростра¬нен сред Microsoft-ориентираните фирми, организации и индивидуални разработчици.

Разпространява се заедно със специална среда, върху която се изпълнява, наречена Common Language Runtime (CLR). Тази среда е част от платформата .NET Framework, която включва CLR, пакет от стандартни библиотеки, предоставящи базова функционалност, компилатори, дебъгери и други средства за разработка. Благодарение на нея CLR програмите са преносими и след като веднъж бъдат написани, могат да работят почти без промени върху различни хардуерни платформи и операционни системи.

Едни от най-съществените предимства на езика са:

- **По-голям акцент върху писането на код**

Изключително важна част от програмирането е оптимизацията на написания софтуер или казано иначе да заделите памет за действията на вашата програма, да я управлявате и все неща в тази насока. При езиците от по-ниско ниво (като Assembly) тези инструкции се задават успоредно с писането на код, което означава, че да програмирате на тях вие трябва да познавате устройството на компютъра, начина по който компонентите му са свързани, колко памет изискват отделните операции и как да ги разпределяте спрямо наличната памет. Тоест вие не можете да се съсредоточите само върху програмната логика, а трябва да съобразявате и куп други неща.При езиците от високо ниво, сред които е и C#, това не е така. Те ви позволяват в началото да се съсредоточите върху логиката на вашата програма, без да се притеснявате за неща като памет и настройки на програмата.

- **Разнообразие от инструменти**

C# е популярен, защото има изключително добра поддръжка. За него са налични множество технологични рамки (frameworks), библиотеки и инструменти. Освен това трябва да се има в предвид, че всички тези неща се поддържат от Microsoft, което означава, че те са надеждни, функционални в следствие на което вие можете да реализирате идеите си много по-бързо от очакваното (разбира се, не без усилие и целеустременост от ваша страна). Всичко това прави C# полезен за изключително широк набор от проблеми.

- **C# е гъвкав**

При другите езици изграждането на едно пълноценно приложение, често пъти може да изисква съчетаването на два или повече езика/технологии. Със C# това остава в миналото. Бидейки език с общо предназначение вие можете да създавате всичко, което ви дойде наум: от уеб сайтове, до десктоп приложения и игри, стигайки дори до мобилни приложения.

Езикът е много широкообхватен и по тази причина може да се срещне в най-различни програми и приложения, но има три области, в които е най-употребяван:

- **Разработване на уеб приложения**

Независимо от платформата, която се използва, C# винаги може да бъде използван за създаването на добри динамични уебсайтове. Примери за уебсайтове, написан на този език, са StackOverflow и Microsoft.

- **Windows приложения**

Microsoft създават C# за себе си. По тази причина никак не е изненадващо, че езикът е перфектен за изграждане на приложения за операционната система Windows. Софтуерните инженери също така могат винаги да разчитат на помощ от огромната общност, както и полезна документация, която да улесни работата им. Едно от най-популярните приложения, създадени със C#, е VisualStudio .

- **Компютърни игри**

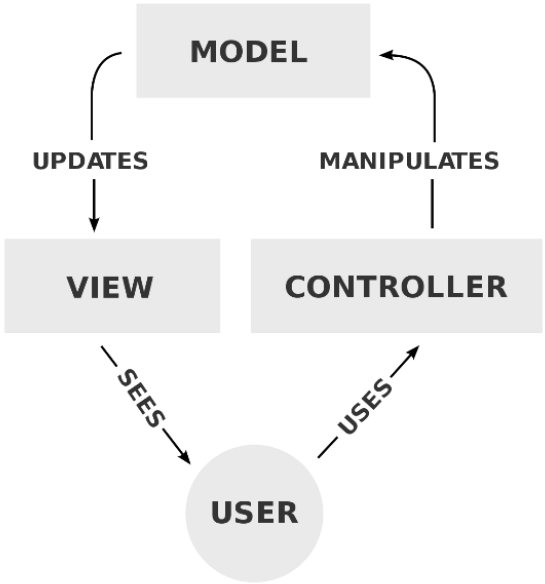
Езикът може да бъде изключително гъвкав и в правилните ръце могат да се създадат уникални и пленяващи игри, както за компютър, така и за мобилно устройство.

Практическа част

Глава I – Концепция

1. Архитектура на приложението

„**Model-view-controller**“ (за по-кратко „MVC”) е архитектурен шаблон, който най-често се използва при създаването на потребителски интерфейс. Той „разделя“ приложението на три взаимосвързани части. Това е направено с цел да се раздели вътрешното представяне на информация от начините по които информацията се представя на и приема от потребителя. MVC шаблонът разделя тези главни компоненти, което позволява на разработчиците да използват отново вече написан код по-ефективно, а също така позволява и паралелна разработка (Фиг. 4).

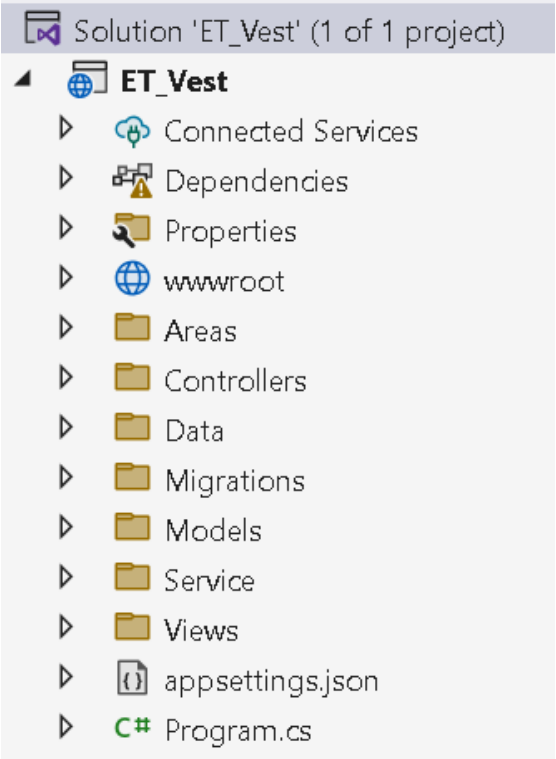


Фиг. 4. Графика на MVC архитектурата

2. Структура

Уеб приложението е проектирано да взаимодейства с всеки потребител, като му предоставя възможност за създаване, обработка и съхранение на данни. На върха на архитектурната структура е потребителят, който има достъп само до потребителския интерфейс на приложението. За него е важно главно бързината и надеждността на приложението, без да се интересува от конкретната база данни или сложността на архитектурата.

Преди да започнем детайлното описание на изграждането на базата данни и модулите с тяхната функционалност е добре да се представи общата структура на проекта според използваният Solution Explorer във Visual Studio (Фиг. 5). Това ще помогне за изграждане на цялостната представа относно проекта.



Фиг. 5. Структура на проекта

3. Относно потребителите

Системата поддържа три вида потребители: администратор, собственик и служител. Всеки регистриран потребител трябва да има:

- потребителско име
- парола
- собствено име
- фамилия
- имейл
- телефонен номер
- роля

Права за регистрация на потребители има само **собственикът**. Той има достъп до цялата база от дани. Може да добавя, премахва и актуализира информация,

регистрира и премахва служител и администратор. Създава заявки и одобрява, подадените от всеки обект заявки, и ги изпраща към доставчиците. Няма права да отбелязва направените продажби.

Администраторът има достъп до цялата база от данни. Може да добавя, премахва и актуализира информация. Няма правя да създава, одобрява и изпраща заявки към доставчици.

Служителят може да подава заявки за доставка на печатни издания към „Собственик“ и да отчита продажбите и брака на обекта си. Актуализира наличностите от продажби и доставки.

4. Относно дейностите

Системата има за цел подобряване дейността на фирмата. Затова достъпът до информация е маскимално улеснен.

При достигане на минимален брой печатни издание или изчерпано количество в конкретен обект, може да се направи заявка(Фиг. 15) за доставка на печатни издания. Наличността се следи в инвентара (Фиг. 16). При изпълнена заявка, количеството наличности се увеличава.

След като „Служителят“ направи заявка, тя се изпраща до „Собственика“ и изчаква одобрение. Ако я одобри, я изпраща към доставчик. При получаване на заявените издания, „Служителят“ отбелязва, че заявката е била изпълнена.

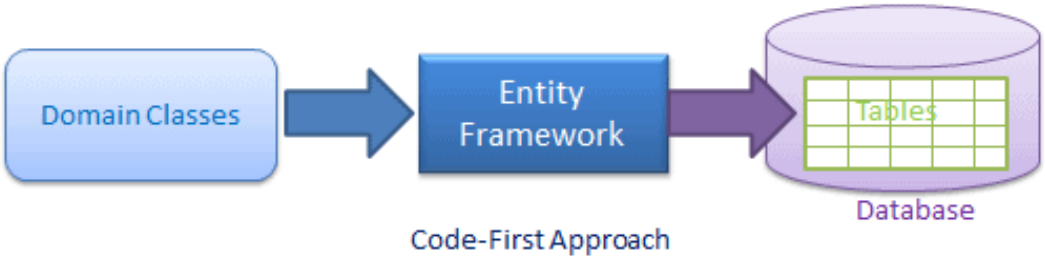
Всеки „Служител“ на обект отбелязва направените продажби (Фиг. 17). Може да вижда дневния си отчет и да бракува вестници и списания. Продажба на издания, чиято наличност е изчерпана, не може да бъде направена. „Собственикът“ може да следи продажбите, реализирани за период, който задава.

Важно е да се уточни, че „Собственикът“ и „Администраторът“ имат достъп до цялата информация в базата, а „Служителят“ имат право да прави заявки, да следи инвентара и да отбелязва продажби само за своите търговски обекти.

Глава II – Програмна реализация

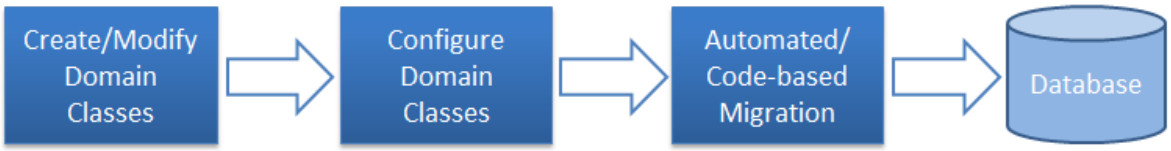
1. Конструиране на базата данни

Подходът, използван за създаването и разработката на базата данни, е Code-First. Това е подход, при който първо се пишат класовете, които представляват модела на данни в код, и след това се използва EF (Entity Framework), за да се генерира схемата на базата данни от тях. Това е полезно за пълен контрол над кода и за избягване на ръчното редактиране на базата данни. Следващата фигура илюстрира този подход (Фиг. 6).



Фиг. 6. Code-First подход

Работният процес на разработка с този подход е: Създаване или модифициране на класове → конфигуриране на тези класове на домейн с помощта на Fluent-API или атрибути за анотация на данни → Създаване или актуализиране на схемата на базата данни с помощта на автоматизирана миграция или миграция, базирана на код. Следващата фигура илюстрира работния процес (Фиг.7).

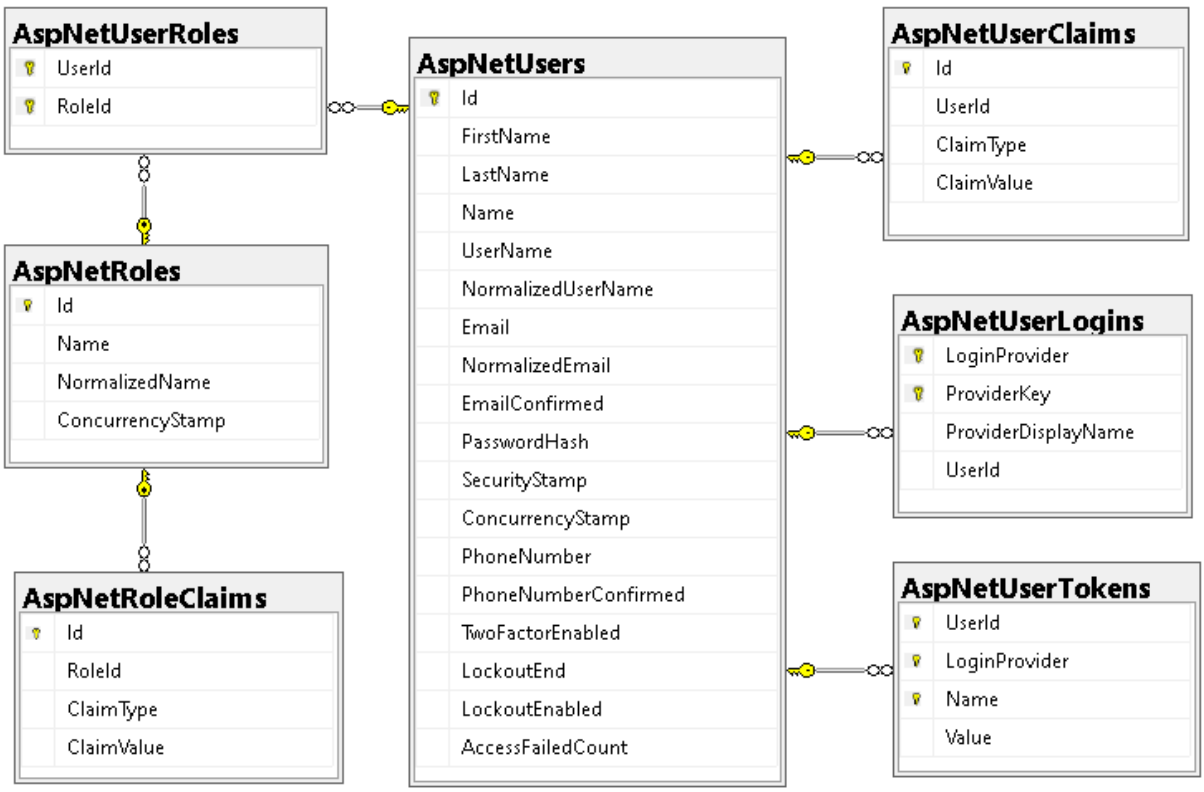


Фиг. 7. Code-First работен процес

2. Модел на автоматично генерираните таблици в базата данни

ASP.NET Core Identity създава няколко таблици в базата данни за управление и съхраняване на информация, свързана с потребителя. Тези таблици са предназначени да работят заедно, за да осигурят цялостна система за удостоверяване

и оторизация на потребителя. По-долу са таблиците на базата данни, генерирани от ASP.NET Core Identity (Фиг. 8).

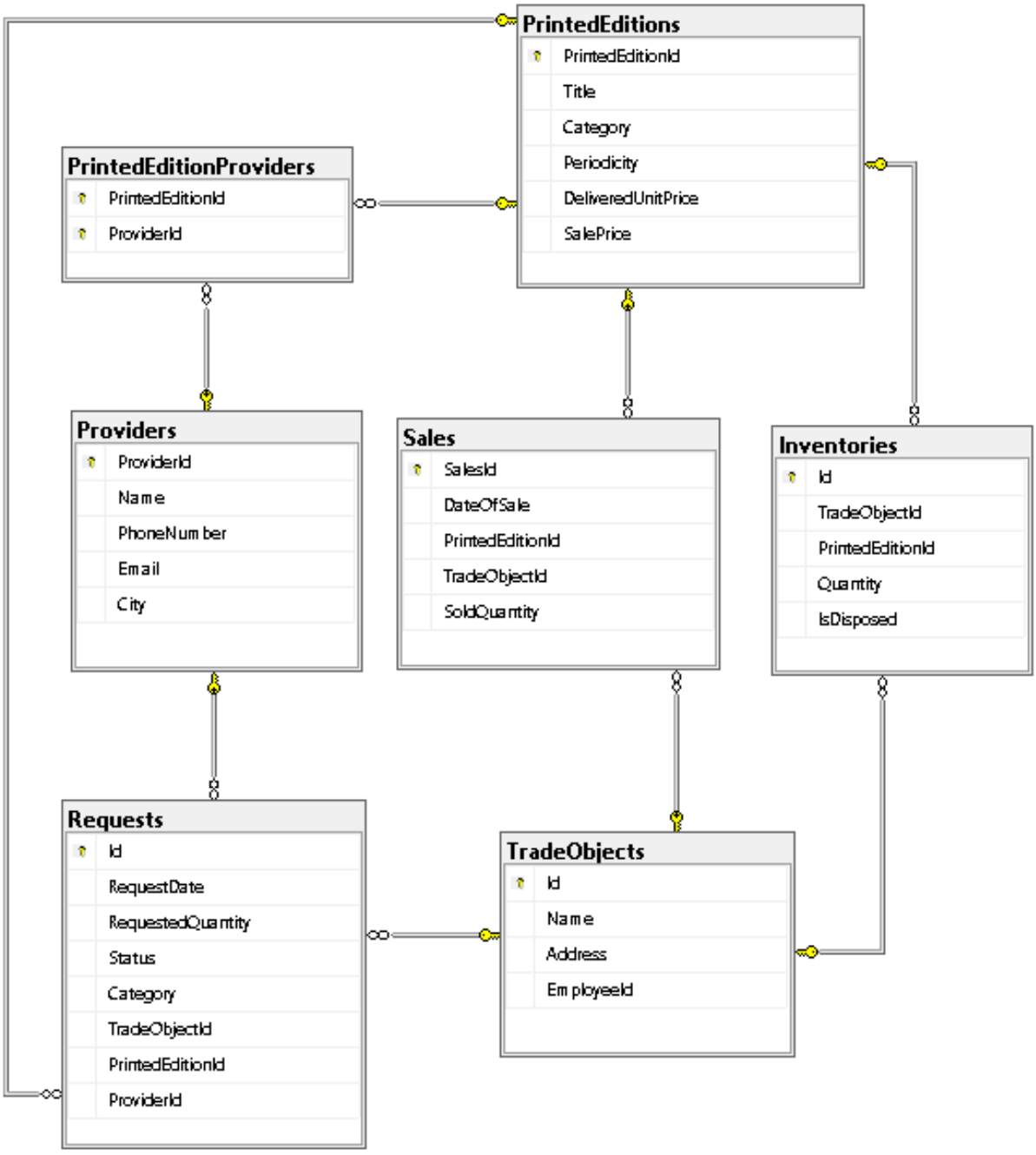


Фиг. 8. Структура на автоматично генерираните таблици

- **AspNetUsers** - Това е основната таблица за съхраняване на потребителска информация. Тя включва основни полета като Id, UserName, Email, PasswordHash и PhoneNumber.
- **AspNetUserRoles** - междинна таблица, показваща връзката “потребител-роля”
- **AspNetRoles** - съхранява информация за ролите
- **AspNetRoleClaims**
- **AspNetUserClaims**
- **AspNetUserLogins** - съхранява информация за потребителските влизания
- **AspNetUserTokens**

3. Таблици в базата данни, създадени в процеса на разработка

Таблиците, които аз създадох, използвайки Entity Framework Core, определят основната функционалност на проекта (Фиг. 9).



Фиг. 9. Структура на таблиците, създадени в процеса на разработка

Класът ApplicationDbContext служи като контекст за връзка с базата данни в приложението. Той има няколко основни функции:

- Управление на модели
- Конфигуриране на модели и връзки между тях
- Извличане на информация за потребителя
- Създаване на база данни

Обобщено казано, класът ApplicationDbContext е централна част от приложението, която обединява моделите, базата данни и конфигурацията за връзка между тях (Фиг. 10). Той служи като посредник между приложението и базата данни, предоставяйки възможност за манипулиране на данни и конфигуриране на връзките между тях. Добавено е и конфигурирация на DeliveredUnitPrice и SalePrice на класа PrintedEdition, като задава типа на колоната в базата данни, с който съответното свойство ще бъде картографирано. В този случай, типът е decimal(18,2), който представлява десетично число с общо 18 цифри, от които 2 са след десетичната запетая. Това се използва за финансови стойности, където точността е важна.

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {

    }
    public DbSet<PrintedEdition> PrintedEditions { get; set; }
    public DbSet<Request> Requests { get; set; }
    public DbSet<TradeObject> TradeObjects { get; set; }
    public DbSet<Provider> Providers { get; set; }
    public DbSet<PrintedEditionProvider> PrintedEditionProviders { get; set; }
    public DbSet<Sale> Sales { get; set; }
    public DbSet<Inventory> Inventories { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        // Define relationships between entities

        modelBuilder.Entity<PrintedEditionProvider>()
            .HasKey(ma => new { ma.PrintedEditionId, ma.ProviderId });

        modelBuilder.Entity<PrintedEditionProvider>()
            .HasOne(ma => ma.PrintedEdition)
            .WithMany(m => m.PrintEditionProviders)
            .HasForeignKey(ma => ma.PrintedEditionId);

        modelBuilder.Entity<PrintedEditionProvider>()
            .HasOne(ma => ma.Provider)
            .WithMany(a => a.PrintEditionProviders)
            .HasForeignKey(ma => ma.ProviderId);

        modelBuilder.Entity<PrintedEdition>()
            .Property(p => p.DeliveredUnitPrice)
            .HasColumnType("decimal(18,2)");

        modelBuilder.Entity<PrintedEdition>()
            .Property(p => p.SalePrice)
            .HasColumnType("decimal(18,2)");
    }
}
```

Фиг. 10. Класът ApplicationDbContext

При изграждането на моделите на приложението са добавени съответните валидации, които следят за валидност на въведените данни. При невалидни данни изписва грешка и не позволява съответното действие да бъде изпълнено.

3.1. Модел на таблицата PrintedEdition

Моделът на PrintedEdition е клас, който служи за запазване на информацията за печатните издания в системата (Фиг. 11).

```
public class PrintedEdition
{
    [Key]
    public int PrintedEditionId { get; set; }

    [Display(Name = "Заглавие")]
    [Required(ErrorMessage = "Заглавието е задължително поле.")]
    public string Title { get; set; }

    [Display(Name = "Категория")]
    [Required(ErrorMessage = "Категорията е задължително поле.")]
    [EnumDataType(typeof(Category))]
    public Category Category { get; set; }

    [Display(Name = "Периодичност")]
    [Required(ErrorMessage = "Периодичността е задължително поле.")]
    [EnumDataType(typeof(Periodicity))]
    public Periodicity Periodicity { get; set; }

    [Display(Name = "Доставна единична цена")]
    [Required(ErrorMessage = "Моля, въведете доставната единична цена.")]
    [RegularExpression(@"^[0-9]+(?:[\.,][0-9]{1,2})?$",
        ErrorMessage = "Моля, въведете валидна цена.")]
    public decimal DeliveredUnitPrice { get; set; }

    [Display(Name = "Продажна цена")]
    [Required(ErrorMessage = "Моля, въведете продажната цена.")]
    [RegularExpression(@"^[0-9]+(?:[\.,][0-9]{1,2})?$",
        ErrorMessage = "Моля, въведете валидна цена.")]
    public decimal SalePrice { get; set; }

    public List<PrintedEditionProvider>? PrintEditionProviders { get; set; }

    public List<Request>? Requests { get; set; }

    public List<Sale>? Sale { get; set; }
}
```

Фиг. 11. Модел на PrintedEdition класа

- **PrintedEditionId** - уникален идентификатор за всяко печатно издание от тип int
- **Title** – свойство от тип string, което описва заглавие на печатно издание
- **Category -** свойство, тип enum, представляващо категорията на печатното издание
- **Periodicity -** свойство, тип enum, представляващо периодичността на печатното издание
- **DeliveredUnitPrice** – свойство от тип decimal, описващно доставна единична цена
- **SalePrice -** свойство от тип decimal, описващно доставна продажната единична цена
- **PrintedEditionProviders** – помощно средство при работа с EF Core, което описва връзката между PrintedEditions и Providers чрез междинната таблица **PrintedEditionProvider**. Връзката е много към много (many to many или many-many), поради което е нужна междинна таблица.
- **Requests, Sales** – връзка много към едно между таблиците: PrintedEditions-Requests PrintedEditions-Sales

3.2. Модел на таблицата Provider

Моделът на Provider е клас, който служи за запазване на информацията за доставчиците в системата и е аналогичен на PrintedEdition (Фиг. 12).

- **ProviderId -** уникален идентификатор за всеки доставчик
- **Name -** име на доставчика
- **PhoneNumber -** телефонен номер на доставчика
- **Email -** имейл на доставчика
- **City -** град
- **PrintedEditionProviders**

```
public class Provider
{
    [Key]
    public int ProviderId { get; set; }

    [Display(Name = "Име")]
    [Required(ErrorMessage = "Името на доставчика е задължително")]
    public string Name { get; set; }

    [Display(Name = "Телефонен номер")]
    [RegularExpression(@"^\+?\d{0,2}\s?\(?\d{3}\)?[-.\s]?\d{3}[-.\s]?\d{4}$",
        ErrorMessage = "Невалиден телефонен номер")]
    public string PhoneNumber { get; set; }

    [Display(Name = "Имейл")]
    [EmailAddress(ErrorMessage = "Невалиден имейл адрес")]
    public string Email { get; set; }

    [Display(Name = "Град")]
    [Required(ErrorMessage = "Градът на доставчика е задължителен")]
    public string City { get; set; }

    public List<PrintedEditionProvider>? PrintEditionProviders { get; set; }
}
```

Фиг. 12. Модел на Provider класа

3.3 Модел на таблицата PrintedEditionProvider

Този модел представлява свързваща таблица между PrintedEdition и Provider (Фиг. 13).

```
public class PrintedEditionProvider
{
    [Key]
        public int PrintedEditionId { get; set; }
        public PrintedEdition PrintedEdition { get; set; }

        public int ProviderId { get; set; }
        public Provider Provider { get; set; }
}
```

Фиг. 13. Модел на PrintedEditionProvider класа

3.4. Модел на таблицата TradeObject

Моделът на TradeObject е клас, който служи за запазване на информацията за търговските обекти в системата (Фиг. 14).

```
public class TradeObject
{
    [Key]
    public int Id { get; set; }

    [Display(Name = "Наименование на обекта")]
    [Required(ErrorMessage = "Наименованието на обекта е задължително")]
    public string Name { get; set; }

    [Display(Name = "Адрес")]
    [Required(ErrorMessage = "Адресът е задължителен")]
    public string Address { get; set; }

    [Display(Name = "Служител")]
    [Required(ErrorMessage = "Полето 'Служител' е задължително")]
    public string EmployeeId { get; set; }

    // Reference to ApplicationUser as the Employee
    public ApplicationUser Employee { get; set; }

    public List<Request>? Requests { get; set; }

    public List<Sale>? Sales { get; set; }
}
```

Фиг. 14. Модел на TradeObject класа

- **Id** - уникален идентификатор за всеки обект
- **Name** – наименование на търговския обект
- **Address** – адрес на обекта
- **EmployeeId и Employee** – foreign key, който служи за свързване на таблицата TradeObject с ApplicationUser с релация едно към едно(one to one)
- **Requests, Sales**

3.5. Модел на таблицата Request

Моделът на таблицата Request (Фиг. 15) отговаря на заявките, които „Служителят“ и „Собственикът“ могат да правят на печатно издание за определен търговски обект при изчерпано количество. Наличността се следи в инвентара.

- **Id** - уникален идентификатор за заявката
- **RequestDate** – свойство, което описва датата на направената заявка, от тип DateTime
- **RequestedQuantity** – заявено количество печатни издани
- **Status** – статус на заявката от тип enum

- **PrintedEditionId и PrintedEdition –** свойства, които описват връзката между Request и PrintedEdition, като типа на връзката е едно към много (one to many).
- **Category-** категория на заявеното печатно издание
- **TradeObjectId и TradeObject** - свойства, които описват връзката между Request и TradeObject, като типа на връзката е едно към много (one to many).
- **ProviderId и Provider** - свойства, които описват връзката между Request и Provider, като типа на връзката е едно към много (one to many).

```
public class Request
{
    [Key]
    public int Id { get; set; }

    [Display(Name = "Дата на заявка")]
    public DateTime RequestDate { get; set; }

    [Display(Name = "Заявено количество")]
    [Required(ErrorMessage =
        "Полето за заявено количество е задължително")]
    [Range(1, int.MaxValue, ErrorMessage =
        "Заявеното количество трябва да бъде положително число")]
    public int RequestedQuantity { get; set; }

    [Display(Name = "Статус")]
    public RequestStatus Status { get; set; }

    [Display(Name = "Категория")]
    public Category Category { get; set; }

    [Display(Name = "Търговски обект")]
    [Required(ErrorMessage =
        "Полето за търговски обект е задължително")]
    public int TradeObjectId { get; set; }
    public TradeObject TradeObject { get; set; }

    [Display(Name = "Печатно издание")]
    [Required(ErrorMessage =
        "Полето за печатно издание е задължително")]
    public int PrintedEditionId { get; set; }
    public PrintedEdition PrintedEdition { get; set; }

    [Display(Name = "Доставчик")]
    [Required(ErrorMessage =
        "Полето за доставчик е задължително")]
    public int ProviderId { get; set; }
    public Provider Provider { get; set; }
}
```

Фиг. 15. Модел на Request класа

3.6. Модел на таблицата Inventory

Моделът на таблицата Inventory(инвентар) (Фиг. 16) е аналогичен на Request. Този клас показва наличностите във всеки обект. При изпълнение на заявките количествата на печатните издания в търговските обекти се увеличава, а при продажба намалява.

- **Id**
- **TradeObjectId и TradeObject**
- **PrintedEditionId и PrintedEdition**
- **Quantity –** свойство от тип int, представляващо броя на наличните печатни издания в търговските обекти
- **IsDisposed –** свойство от тип bool, показващо дали артикулът от инвентара е бракуван.

```
public class Inventory
{
    [Key]
    public int Id { get; set; }

    [Display(Name = "Търговски обект")]
    [Required(ErrorMessage = "Обектът е задължителен")]
    public int TradeObjectId { get; set; }
    public TradeObject TradeObject { get; set; }

    [Display(Name = "Печатно издание")]
    [Required(ErrorMessage = "Изданието е задължително")]
    public int PrintedEditionId { get; set; }
    public PrintedEdition PrintedEdition { get; set; }

    [Display(Name = "Налично количество")]
    [Required(ErrorMessage = "Количеството е задължително")]
    [Range(1, int.MaxValue, ErrorMessage =
        "Количеството трябва да бъде положително число и по-голямо от 0")]
    public int Quantity { get; set; }

    public bool IsDisposed { get; set; }
}
```

Фиг. 16. Модел на Inventory класа

3.7. Модел на таблицата Sale

Моделът на таблицата Sale показва информация за направените от всеки търговски обект продажби (Фиг. 17).

- **SalesId**
- **DateOfSale -** свойство, което описва датата на направената продажба, от тип DateTime
- **TradeObjectId и TradeObject**
- **PrintedEditionId и PrintedEdition**
- **SoldQuantity –** свойтсво от тип int, което описва продаденото количеството на определено печатно издание в един търговски обект
- **Total –** общата сума, която се смята по формулата: продажна цена на печатно издание * броя от продадените печатни издания (SoldQuantity)

```
public class Sale
{
    [Key]
    public int SalesId { get; set; }

    [Display(Name = "Дата на продажба")]
    public DateTime DateOfSale { get; set; }

    [Display(Name = "Печатно издание")]
    [Required(ErrorMessage = "Полето за печатно издание е задължително")]
    public int PrintedEditionId { get; set; }
    public PrintedEdition PrintedEdition { get; set; }

    [Display(Name = "Търговски обект")]
    [Required(ErrorMessage = "Полето за търговски обект е задължително")]
    public int TradeObjectId { get; set; }
    public TradeObject TradeObject { get; set; }

    [Display(Name = "Количество за продажба")]
    [Required(ErrorMessage = "Полето за количество за продажба е задължително")]
    [Range(1, int.MaxValue,
        ErrorMessage = "Количеството за продажба трябва да бъде положително число")]
    public int SoldQuantity { get; set; }

    public decimal Total => PrintedEdition?.SalePrice * SoldQuantity ?? 0;
}
```

Фиг. 17. Модел на Sale класа

4. Реализиране на Контролерите

Контролерът в **MVC** (Model-View-Controller) е един от основните компоненти на този шаблон на проектиране на софтуер. Контролерът е отговорен за приемането на заявки от потребителите, обработката на тези заявки и изпращането на резултатите обратно към потребителите.

Ролята на контролера е да управлява взаимодействието между модела (данните) и изгледа (потребителския интерфейс). Той получава заявки от изгледа, които съдържат информация за действията, които потребителят иска да извърши, и след това извиква методи от модела за да извърши тези действия.

Контролерите имат достъп до различни услуги и източници на данни в приложението чрез инжектиране на зависимости, което им позволява да изпълняват различни задачи като достъп до база данни, валидиране и удостоверяване.

Контролерите могат също да връщат различни типове отговори, като HTML изгледи, JSON данни или изтегляния на файлове, в зависимост от заявката и изискванията на приложението.

За всеки модел от базата данни има по един контролер (PrintedEditionController, ProviderController, RequestController, SaleController, PrintedEditionProviderController, TradeObjectController и InventoryController), всеки от които служи да обработва заявките на потребителя чрез манипулация или извличане на данни от базата.

Ще разгледам само един от контролерите, тъй като всички останали работят на същия принцип. Ще разгледам основните методи в RequestController, тъй като е най-сложен и има най-много функционалности.

- **Add Action** – има два метода: GET метод, който връща изгледа за добавяне на нова заявка. В този метод се подготвят данните за изгледа, като се извличат всички възможни TradeObjects, PrintedEditions и Providers от базата данни и POST метод, който обработва POST заявките за добавяне на нова заявка. Тук се приема обект от тип Request, който представлява данните на новата заявка. След валидацията на модела, заявката се добавя към контекста и се запазват промените в базата данни.

  HTTP GET метода Add() проверява дали текущият потребител е в роля "Служител". Ако е така, тогава се извлича идентификаторът на потребителя (Id). Филтрират се търговските обекти (TradeObjects) в контекста на базата данни (_context), така че само тези, които са свързани с потребителя, да се визуализират на страницата. Ако потребителя е в роля на „Администратор“ или „Собственик“, всички търговски обекти се визуализират (Фиг. 18).

  HTTP POST метода Add() получава параметъра "request", който съдържа данните, въведени от потребителя във формата за добавяне на заявка. Задава

днешната дата като дата на заявката. Ако потребителят е в роля "Собственик", тогава статуса на заявката се задава на "Изпрати до доставчик". В противен случай, се задава на "Изчакваща". Добавя новата заявка в контекста на базата данни (_context) и запазва промените в базата данни (Фиг. 19).

```
public async Task<IActionResult> Add()
{
    if (User.IsInRole("Employee"))
    {
        var user = User.FindFirstValue(ClaimTypes.NameIdentifier);

        ViewBag.TradeObjects = _context.TradeObjects
            .Where(to => to.EmployeeId == user);
        ViewBag.PrintedEditions = _context.PrintedEditions.ToList();
        ViewBag.Providers = _context.Providers.ToList();

        return View();
    }
    else
    {
        ViewBag.TradeObjects = _context.TradeObjects.ToList();
        ViewBag.PrintedEditions = _context.PrintedEditions.ToList();
        ViewBag.Providers = _context.Providers.ToList();

        return View();
    }
}
```

Фиг. 18. GET метода Add() на RequestController

```
[Authorize(Roles = "Owner, Employee")]
[HttpPost]
public IActionResult Add(Request request)
{
    request.RequestDate = DateTime.Today;

    if (User.IsInRole("Owner"))
    {
        request.Status = RequestStatus.SentToProvider;
    }
    else
    {
        request.Status = RequestStatus.Pending;
    }

    _context.Requests.Add(request);
    _context.SaveChanges();
    return RedirectToAction("Index");
}
```

Фиг. 19. POST метода Add() на RequestController

- **Edit Action** – отново има GET и POST. GET метода връща изгледа за редактиране на съществуваща заявка и подготвят данните за изгледа, като се извличат всички възможни TradeObjects, PrintedEditions и Providers от базата данни. POST метода обработва POST заявките за редактиране на съществуваща заявка. Правят се необходимите промени и се запазват в базата данни (Фиг. 20) (Фиг. 21). Допълнителните функционалности са аналогични на тези в Add Action.

```
public async Task<IActionResult> Edit(int id)
{
    var request = _context.Requests
        .Include(m => m.TradeObject)
        .Include(m => m.PrintedEdition)
        .Include(m => m.Provider)
        .FirstOrDefault(m => m.Id == id);
    if (request == null)
    {
        return NotFound();
    }
    ViewBag.TradeObjects = _context.TradeObjects.ToList();
    ViewBag.PrintedEditions = _context.PrintedEditions.ToList();
    ViewBag.Providers = _context.Providers.ToList();

    return View(request);
}
```

Фиг. 20. POST метода Edit() на RequestController

```
[Authorize(Roles = "Owner")]
[HttpPost]
public IActionResult Edit(Request request)
{
    request.RequestDate = DateTime.Today;

    request.Status = RequestStatus.SentToProvider;

    _context.Requests.Update(request);
    _context.SaveChanges();
    return RedirectToAction("Index");
}
```

Фиг. 21. GET метода Edit() на RequestController

- **Delete Action** - Извлича заявката от базата данни, която трябва да бъде изтрита, чрез нейния идентификатор. Проверява дали заявката е намерена в базата

данни и ако заявката съществува, тя се изтрива от контекста на базата данни чрез метода Remove(). Промените в базата данни се запазват чрез извикване на SaveChanges() (Фиг. 22).

```
[HttpPost]
public IActionResult Delete(int id)
{
    var request = _context.Requests.Find(id);

    if (request == null)
    {
        return NotFound();
    }

    _context.Requests.Remove(request);
    _context.SaveChanges();
    return RedirectToAction("Index");
}
```

Фиг. 22. Delete Action на RequestController

5. Реализиране на Изгледите

В MVC (Model-View-Controller) архитектура изгледът е компонентът, отговорен за представянето на данните на потребителя по смислен начин. Изгледът получава данни от контролера и ги изобразява с помощта на различни UI елементи като текст, изображения и формуляри.

Изгледите са от съществено значение за изграждането на потребителски интерфейси. Те предоставят начин за показване на данни на потребителя и му позволяват да взаимодейства с приложението. Чрез добре проектирани изгледи разработчиците могат да създадат по-интуитивно и удобно изживяване за своите потребители.

Като пример ще разгледам изгледа за създаване на заявка (Фиг. 23). Този изглед използва Razor синтаксис за генериране на HTML и формира HTML форма за въвеждане на информация от потребителя. В този конкретен случай, формата съдържа полета за избор на печатно издание, въведение на желаното количество, избор на търговски обект и избор на доставчик. При изпращане на формата, данните се препращат към съответния контролер и действие ("Add" в контролера "Request").

```
<h2>Създаване на заявка</h2>
<form method="post" asp-action="Add" asp-controller="Request">
    <div class="form-group">
        <label asp-for="PrintedEditionId">Изберете печатно издание</label>
        <select asp-for="PrintedEditionId"
        asp-items="@(new SelectList(ViewBag.PrintedEditions, "PrintedEditionId", "Title"))"
        class="form-control">
            <option value="">Изберете печатно издание</option>
        </select>
        <span asp-validation-for="PrintedEditionId" class="text-danger"></span>
    </div>

    <div class="form-group">
        <label asp-for="RequestedQuantity">Количество</label>
        <input asp-for="RequestedQuantity" class="form-control" />
        <span asp-validation-for="RequestedQuantity" class="text-danger"></span>
    </div>

    <div class="form-group">
        <label asp-for="TradeObjectId">Изберете търговски обект</label>
        <select asp-for="TradeObjectId"
        asp-items="@(new SelectList(ViewBag.TradeObjects, "Id", "Name"))"
        class="form-control">
            <option value="">Изберете търговски обект</option>
        </select>
        <span asp-validation-for="TradeObjectId" class="text-danger"></span>
    </div>

    <div class="form-group">
        <label asp-for="ProviderId">Изберете доставчик</label>
        <select asp-for="ProviderId"
        asp-items="@(new SelectList(ViewBag.Providers, "ProviderId", "Name"))"
        class="form-control">
            <option value="">Изберете доставчик</option>
        </select>
        <span asp-validation-for="ProviderId" class="text-danger"></span>
    </div>

    <button type="submit" class="btn btn-primary">Създаване</button>
</form>
```

Фиг. 23. Изглед за създаване на Request

Заключение

В заключението ще направя обзор на поставените цели и тяхното изпълнение.

Приложението е базирано на модел-изглед-контролер (MVC) архитектура, която прави кода лесен за поддръжка и има възможности за бъдещо развитие. Разработено е с използването на C# и .NET Core, което гарантира стабилност и бързина.

Процесът на създаване на приложението е осъществен чрез внимателно планиране, задълбочен анализ на изискванията на проекта и изследване на различни подходи и технологии, които могат да бъдат използвани за изграждането на уебсайта.

Разработката на системата включи използването на съвременни технологии и методи в областта на уеб разработката. Този подход не само осигури ефективно създаване на стабилно и функционално приложение, но и гарантира неговата съвместимост с текущите стандарти и изисквания.

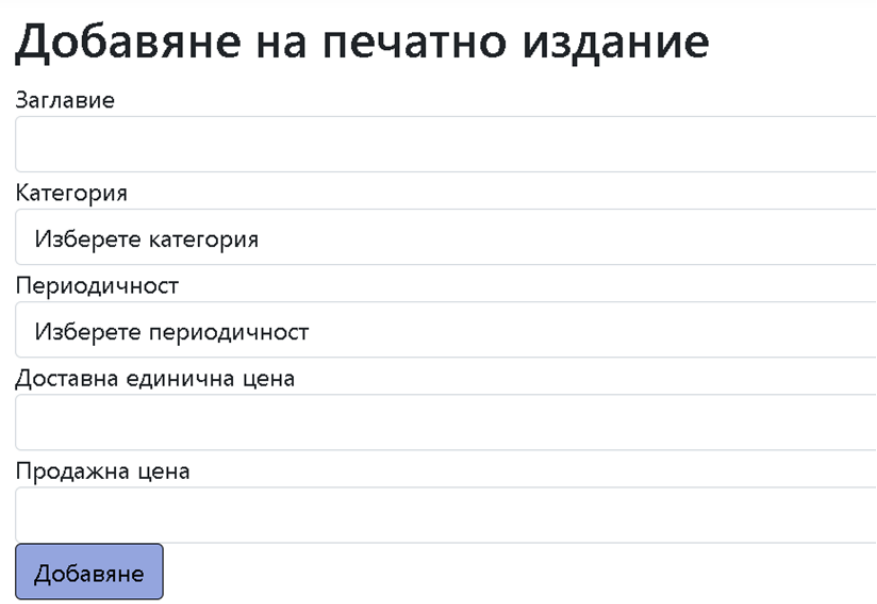
Допълнително, важно е да се отбележи, че разработената система е гъвкава и модулна, което означава, че може да бъде лесно доразвивана и разширявана в бъдеще. Това осигурява възможност за добавяне на нови функционалности, оптимизации и корекции на вече съществуващите, в зависимост от развитието на бизнеса и изискванията на потребителите. Такъв подход позволява системата да остане актуална в дългосрочен план, като по този начин се гарантира непрекъснатото й съответствие със заливащите се технологични и бизнес изисквания.

Дипломният проект успешно постигава своите цели, предоставяйки функционираща и ефективна система за управление на поръчките и продажбитена различни търговски обекти и предлага перспективи за бъдещо развитие и подобрение на функционалностите й.

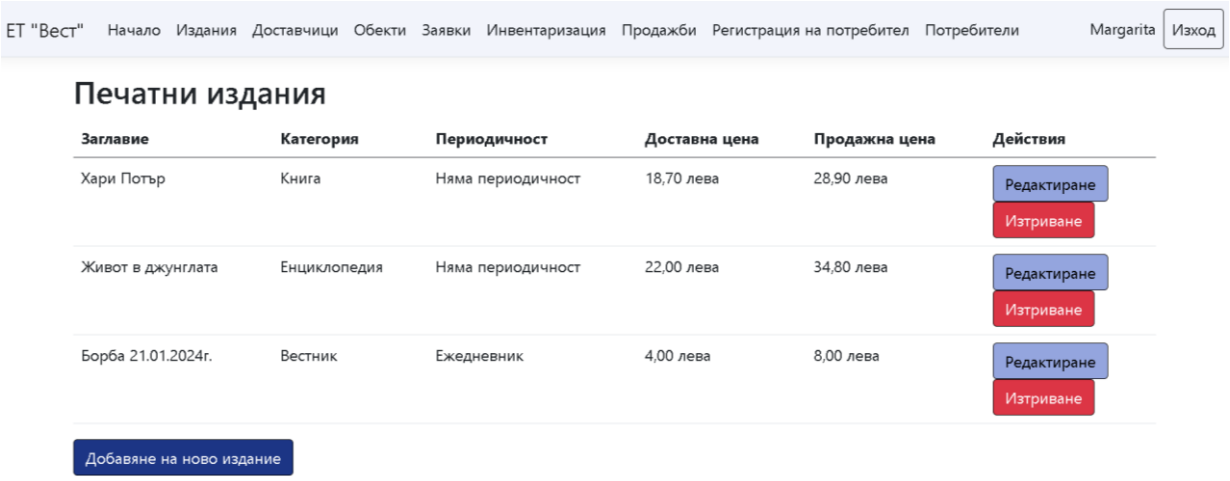
Използвана литература

1. Страница на W3Schools за CSS - https://www.w3schools.com/css/
2. Страница на W3Schools за HTML - https://www.w3schools.com/html/
3. HTML 5 & CSS 3 практическо програмиране за начинаещи - Денис Колисниченки
4. Документация на Microsoft за ASP.NET Core - https://microsoft.fandom.com/wiki/Microsoft_.NET
5. Сайт с информация за MVC архитектурата - https://developer.mozilla.org/en-US/docs/Glossary/MVC
6. Сайт с информация за Code-First подход - https://apidog.com/articles/what-is-code-first/

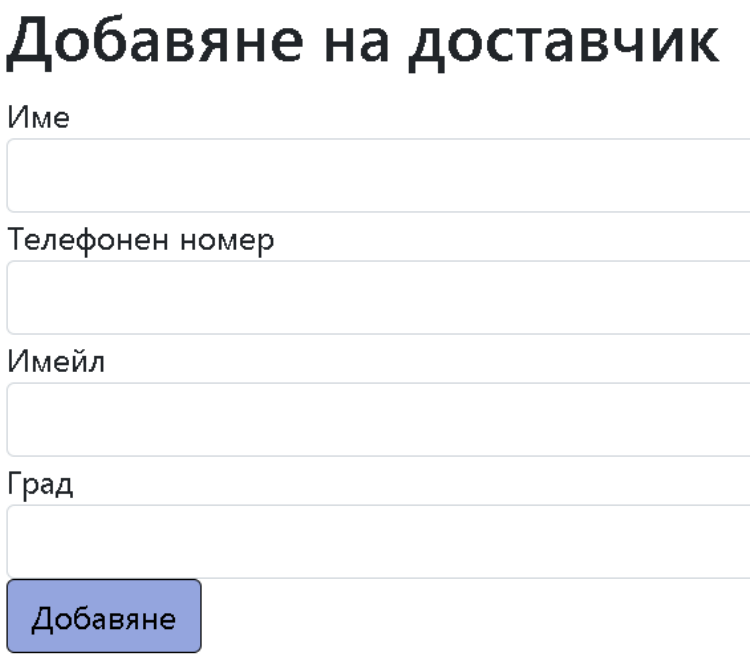
Приложение



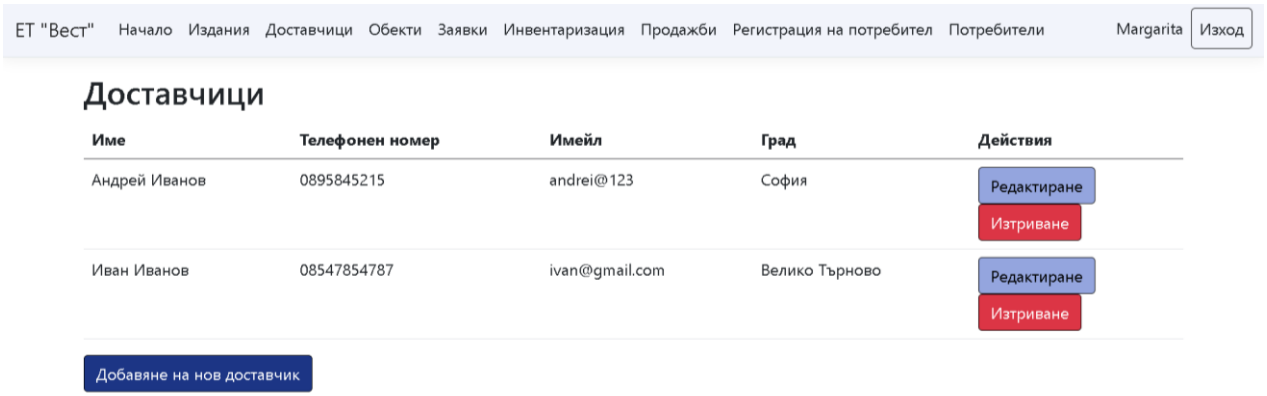
Фиг. 24. Добавяне на печатно издание



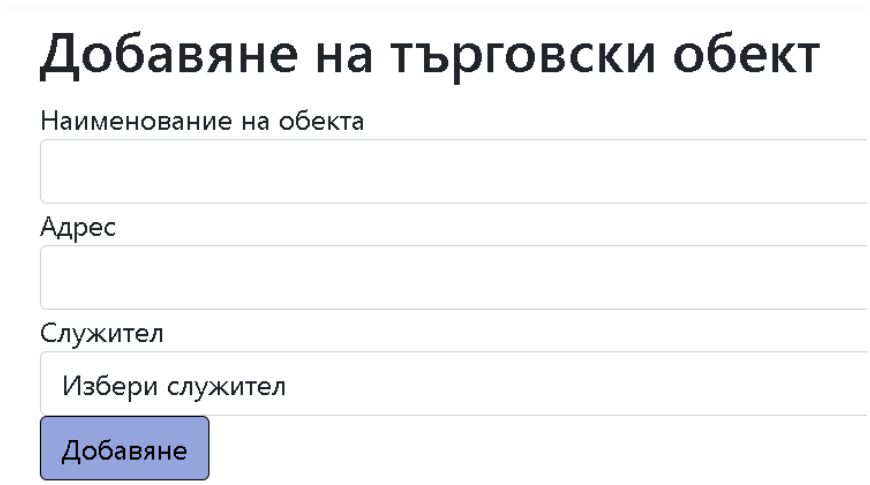
Фиг. 25. Изглед на печатните издания



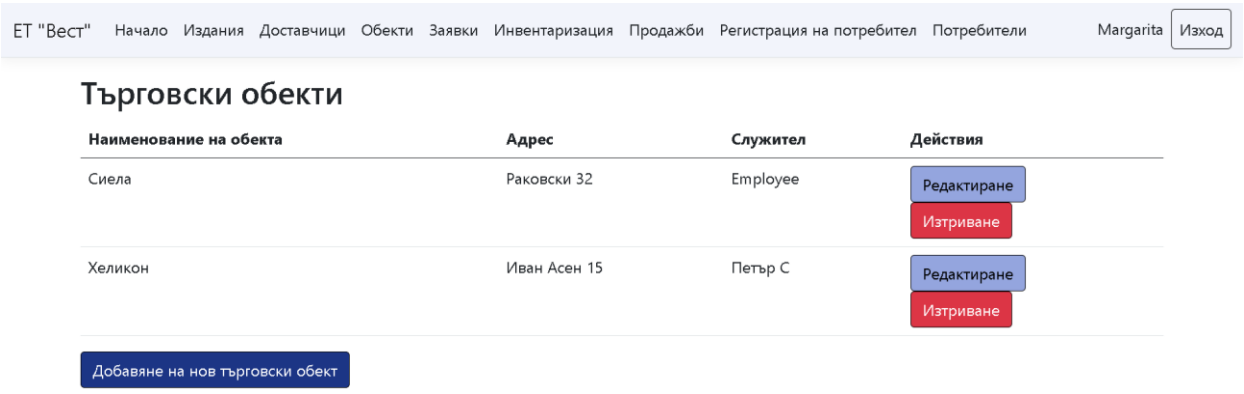
Фиг. 26. Добавяне на доставчик



Фиг. 27. Изглед на доставчиците

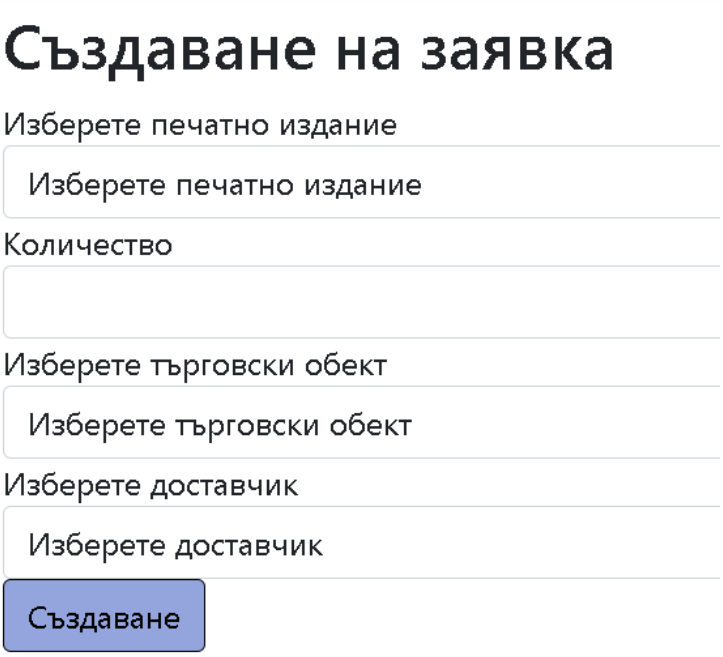


Фиг. 28. Добавяне на търговски обект

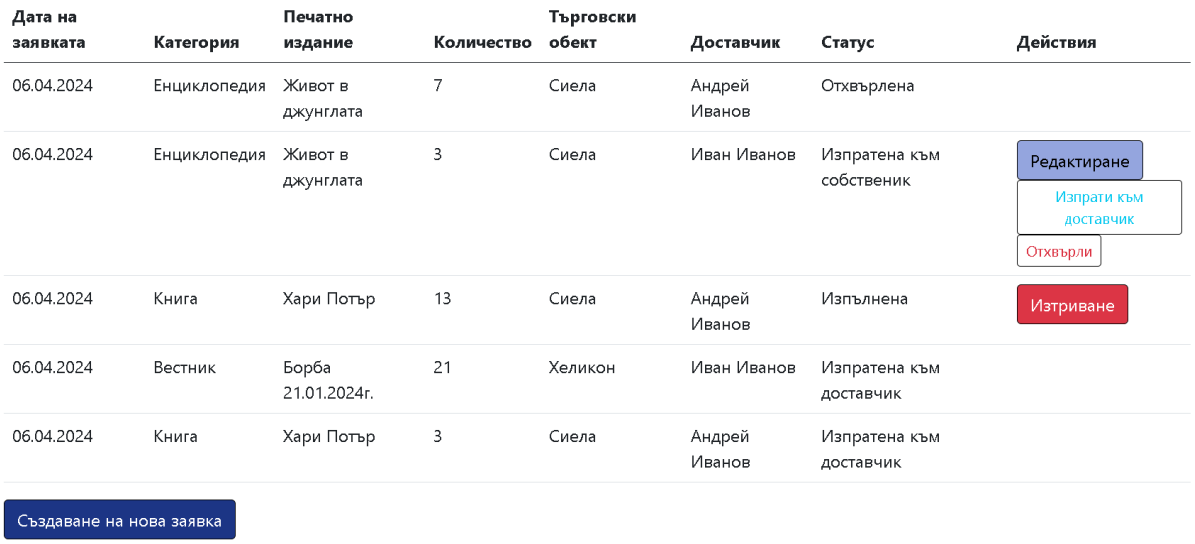


Фиг. 29. Изглед на търговските обект

Достъп до базата с информация за печатните издания, доставчиците и търговските обекти имат само потребителите „Собственик“ и „Администратор“.



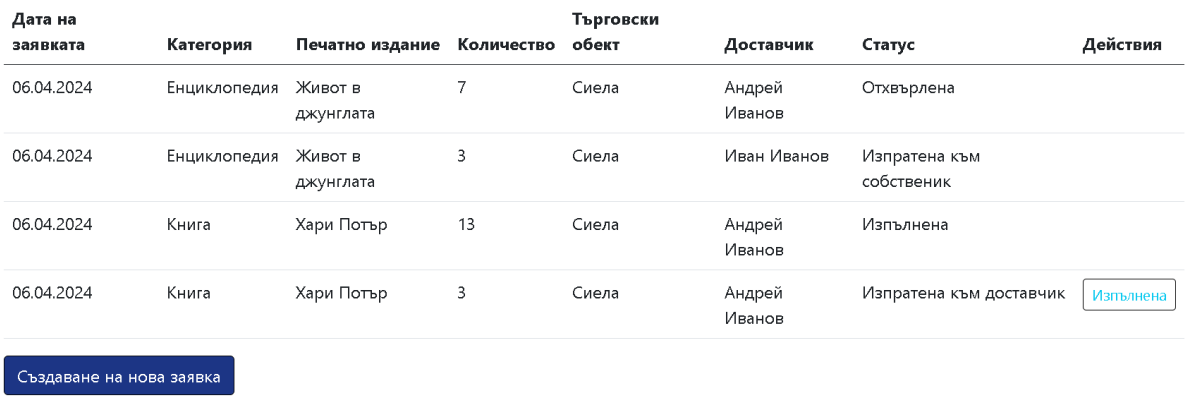
Фиг. 30. Създаване на заявка



Фиг. 31. Лист със заявки при потребител „Собственик“

Когато потребителят е „Собственик“ има право да вижда всички направени заявки на всички търговски обекти. Одобрява и отхвърля заявките. Има право да ги редактира и изтрива.

„Администраторът“ има достъп до информацията за заявките, но не може да прави промени, да добавя или да изтрива.

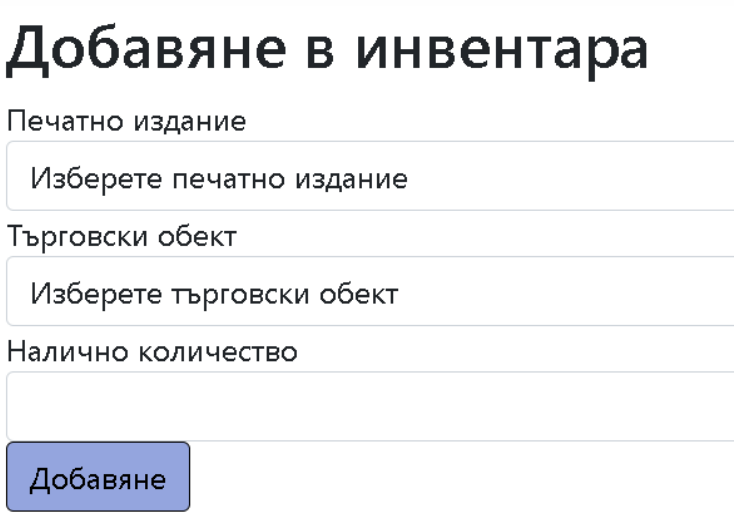


Дата на заявката	Категория	Печатно издание	Количество	Търговски обект	Доставчик	Статус	Действия
06.04.2024	Енциклопедия	Живот в джунглата	7	Сиела	Андрей Иванов	Отхвърлена	
06.04.2024	Енциклопедия	Живот в джунглата	3	Сиела	Иван Иванов	Изпратена към собственик	
06.04.2024	Книга	Хари Потър	13	Сиела	Андрей Иванов	Изпълнена	
06.04.2024	Книга	Хари Потър	3	Сиела	Андрей Иванов	Изпратена към доставчик	Изпълнена

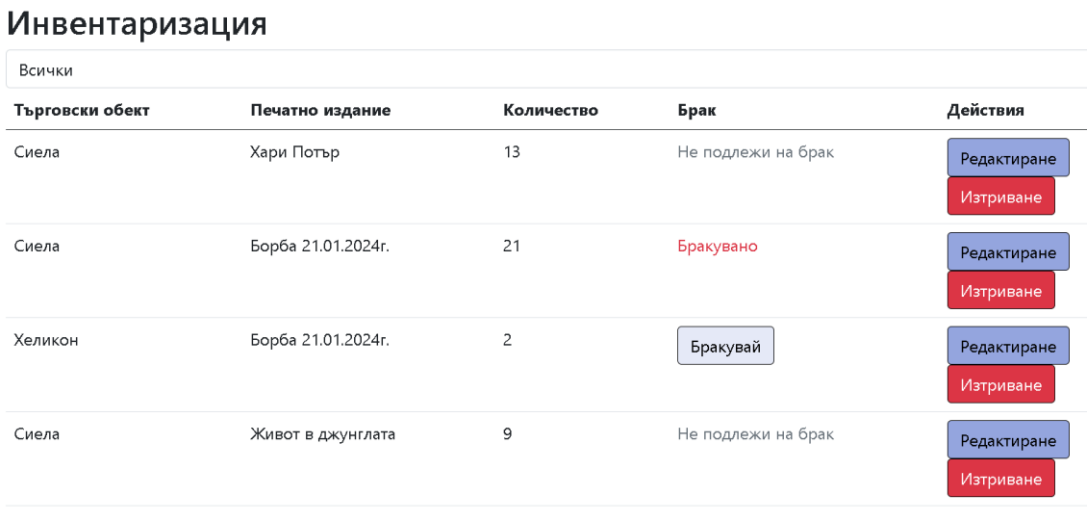
Създаване на нова заявка

Фиг. 32. Лист със заявки при потребител „Служител“

„Служителят“ може да прави заявки само са своите обекти, както и да вижда заявките за тях. Създава заяка и я изпраща към „Собственик“. При получаване на изданията отбелязва, че е изпълнена.



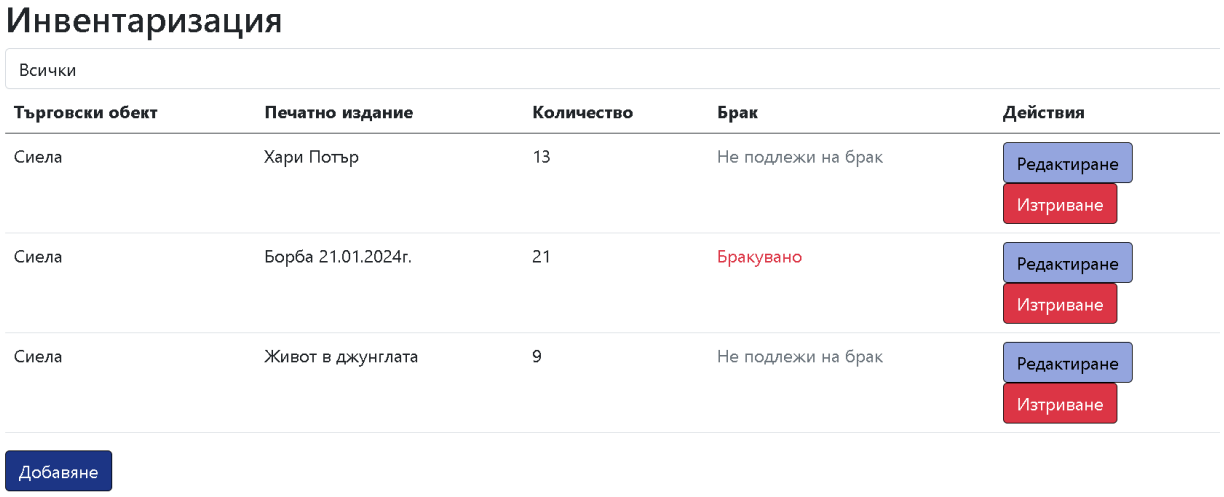
Фиг. 33. Добавяне на издания в инвентара



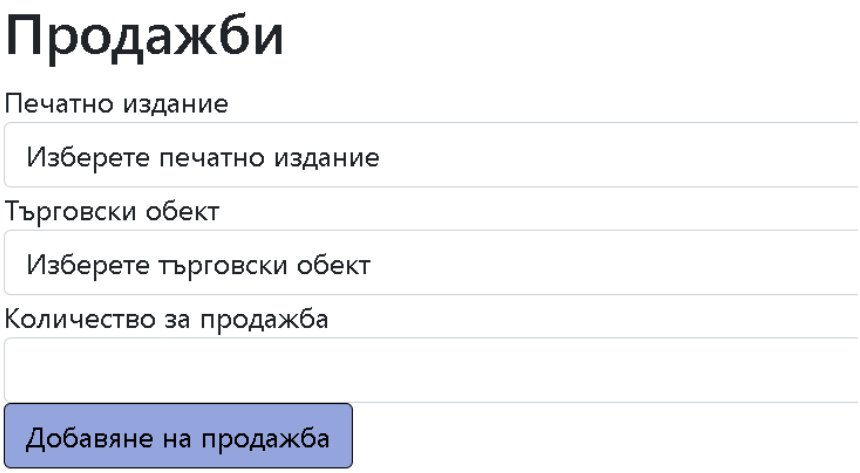
Фиг. 34. Инвентар при потребител „Собственик“

Аналогично на заявките, „Собственикът“ вижда наличностите във всеки търговски обект, докато „Служителят“ само в своите обекти.

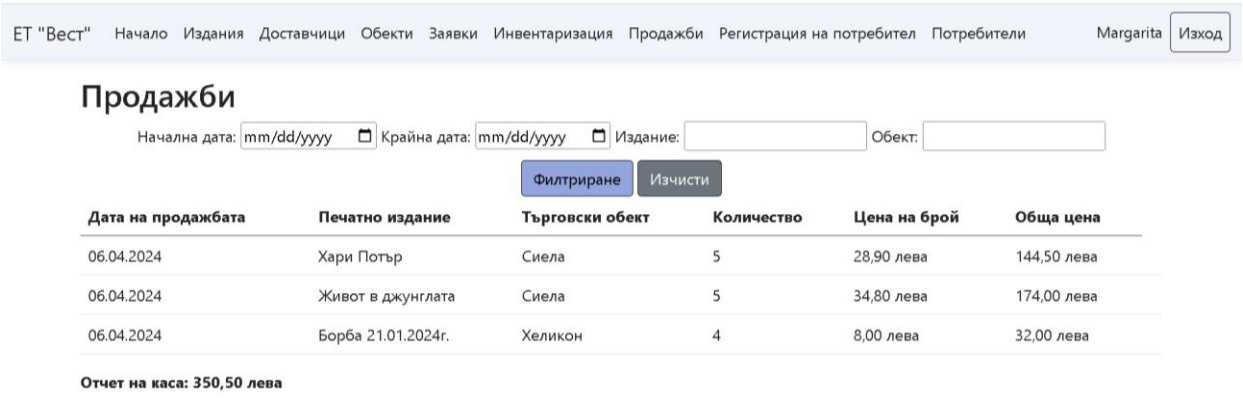
Има опция за филтриране, при която потребителите могат да избират да виждат всички издания, само наличните издания или само бракуваните. Възможност за бракуване имат само вестниците и списанията.



Фиг. 35. Инвентар при потребител „Служител“



Фиг. 36. Добавяне на продажба



Фиг. 37. Изглед на продажбите при потребител „Собственик“

„Собственикът“ не може да отбелязва продажби. Може да следи отчета за всички търговски обекти и списания. Има достъп до справка за продажбите, реализирани за период, който задава, както за всички обекти и издания, така и за конкретни.

Фиг. 38. Изглед на продажбите при потребител „Служител“

„Служителят“ отбелязва продажби на обектите си, като всяка отбелязана продажба е с датата на добавяне. Има право на справка само на дневните продажби, както и отчет на каса.