

**ДИПЛОМЕН ПРОЕКТ**

**Тема: „Разработка на WEB сайт на фирмата BulBroker“**

**Разработил: Ивайло Ивайлов Тончев ……………**

*(име, презиме, фамилия) (подпис)*

**професия код 481030 „Приложен програмист“**

**специалност код 4810301 „Приложно програмиране“**

**Ръководител-консултант: инж. Николай Христов ………………**

*(име, фамилия) (подпис)*

**2022/2023 г.**

**ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО МЕХАНОТЕХНИКА, ЕЛЕКТРОНИКА, ТЕЛЕКОМУНИКАЦИИ И ТРАНСПОРТ „ХРИСТО БОТЕВ“ – ШУМЕН**

Съдържание

1.    Предназначение на приложението и проблеми, които то решава.

2. Функционалности и правата на достъп които приложението предлага

3.    Използвани технологии при разработката

3.1. Обща информация за приложението

3.2. Програмни продукти използвани при разработката на приложението

3.3. Програмни езици използвани при разработката на приложението

3.4. Библиотеки използвани при разработка на приложението и тяхното

предназначение.

4. Структура на приложението и организация на кода

5. База данни

6. Фронт-енд визуализация и контролери.

7. Функционалност на приложението която, следва да се

имплементира, надграждане.

8. Използвана литература.

I. Предназначение на приложението и проблеми, които то

решава.

1.Предназначение:

Приложението има за цел да предостави възможност на

фирма занимаваща се с брокерство да представи на

клиентите актуалните имоти както и информация за тях.

2.Проблеми, които приложението решава:

Софтуерната разработка предоставя възможност на клиента

(възложител), да публикува съдържание с цел реклама на

продукта предмет на дейността, като това е в допълнение

и/или към плакати, билборди, листовки и други средства

за реклама. По този начин се постига повишение в

посещаемостта на фирмата, а също и се спестяват разходи.

II. Функционалности и правата на достъп които

приложението предлага:

1. Нива на достъп и правомощия в приложението:

1.1. Гост GUEST – това са всички потребители, които достъпват

приложението без да имат регистрация в него. Тяхната анонимност е

гарантирана. Има права да разглежда наличните имоти както и

детайлите в тях. Може да вижда секцията контакти.

1.2. Потребител USER – представлява регистриран потребител,

притежава всички права на гост потребител като има възможност да

запази лични данни (имейл и парола) като и да ги променя в

последствие.

1.3. Администратор ADMIN – Регистриран потребител със

специфична роля. Притежава всички права на User, но има правото да

добавя, редактира и изтрива и.В навигационната лента вижда повече секции

от всички други роли свързани с добавяне,детайли на адресите,изтриване и тн.

III. Използвани технологии при разработката.

1. Обща информация за приложението:

Приложението е разработено посредством Model View

Controller(MVC Framework) на .NET 6. Представлява MPG (Multi

Page Application) като характерно за него е че всеки адрес връща

нов HTML+CSS менажиран от сървъра.

2. Програмни продукти използвани при разработката на приложението:

- MS Word – За документацията към проекта.

- Visual Studio 2022

- Microsoft SQL Management Studio

- Git

- TurtoiseGit

3. Програмни езици използвани при разработката на приложението:

- C# версия .NET 6 – За сървърна логика (back-end)

- MS SQL – За бази данни, като автоматично генериран код

през библиотеката Entity Framework.

- HTML и CSS – За визуалната част (front-end).

- Java Script – За логика при визуалната част (front-end)

4. Библиотеки използвани при разработка на приложението и тяхното

предназначение.

4.1. Библиотеки при сървърната част (Back-End):

- Entity Framework – Представлява ORM предоставящ възможност

за работа с база посредством MS SQL Provider като по този

начин лесно може приложението да се пригоди към друга база,

например Postgre или МySQL. Кодът не е обвързан с

конкретният SQL език пряко.

-Мycrosoft Identity – Представлява набор от класове ,които се

материализират в базата и дават базова функционалност за

управление на потребители, като когато това не е достатъчно

могат да бъдат разширявани както в случая е разширен класът

AppUser.

-LINQ – Набор от Extension методи върху колекции позволяващ

работа с тях. Използва се в съвкупност с Entity Framework като

работата с колекции класове се превежда от ORM до заявки към

базата на съответният език избран чрез Provider.

4.2. Библиотеки при визуалната част (Front-End):

Тези библиотеки се зареждат през CDN връзки (линкове) с цел по-

бързото им сваляне от клиента предвид разположението им на

множество сървъри.

-Bootstrap – Съдържа набор от CSS класове дефиниращи правила

за визуализация на html. Цели да позволи писане на HTML без да

се налага писане на CSS а вместо това се присвояват имената на

класовете от библиотеката на различни HTML елементи.

-Font Awesome – Съдържа набор от CSS класове дефиниращи как

да изглеждат иконки от тип &lt;i class=”…”&gt;&lt;/i&gt;. Като това

позволява да се визуализират иконки в приложението.

IV. Структура на приложението и организация на кода.

1. Структура на приложението:

Приложението е разработено на база основа ASP.NET MVC като е

разширено и преструктуриране за да позволява скалиране и

менажиране на кода и отстраняване и локализиране на бъгове в

процеса на разработка и експлоатация. Кода в приложението е разделен и обособен във отделни папки и файлове със съответните наименования според тяхната дейност и функционалност.

2. Разделеняне на кода, функции и предназначение на всеки раздел:

2.1. DiplomenProekt представлява Console Application. Ролята му е да

стартира приложението и да управлява и използва всички останали

компоненти в целият Solution. В него се съдържат следните

компоненти:

-wwwroot: В него се съдържат статични материали, като

изображения, javascript и CSS файлове, които се достъпват от

Views.

-Areas: Съдържа скафолднати Identity Razor Pages, които са

разширени допълнително за да прилягат на целите на

приложението.

-Controllers: Съдържа класовете контролери които имат за цел да

управляват кои Views да бъдат върнати на потребителя както и

да ги снабдяват с нужните за визуализирането им данни.

-Views: Съдържат организирани в папки за да бъдат откривани по

конвенция \*.cshtml файлове (Razor views) които се достъпват и

връщат през методите в контролерите.

-appsettings.json – Съдържа информация до която има достъп

Program.cs (традиционно тук се съдържа ConnectionString към

съответната база както и друга чувствителна информация)

-Program.cs –Представлява стартируемият клас от който се

стартира приложението в него се съдържа „public static void

Main(string[] args)“ метода. Съдържа конфигурации, как да работи

приложението, в него се регистрират различни класове в

Dependency Injection контейнера, посочва се с кой DbContext да

се работи, какви изисквания има за паролите на потребителите,

дали да се изпозлват Midlewares и Filters.

2.2 Data.Models-в него са енумерациите и всички Entities,

които имат репрезентация в базата с данни.

-Enumerations– Съдържа енумерации които се ползват в класовете, на

практика в тях се съдържа дефиниция (шифър), как да се

интерпретират числовите стойности на отделните колони в

базата със стойности от тип енумерация.

-Еntities – Съдържа всички модел класове нужни на

приложението, тук са поместени базовите абстрактни класове, а

в някои случаи разширяващи функционалност класове например

AppUser.

2.3-Migrations – Директория, в която се съдържат миграциите които

отразяват промените в структурата на базата при началното и

създаване както и в процеса на промяна по време на

експлоатация на приложението.

2.4-Data-– Съдържа ApplicationDbContext, наследник на DbContext.

Указва кои таблици ще бъдат създадени на базата на

съответните класове от Models, също така указва в метода

„protected override void OnModelCreating(ModelBuilder builder)“

чрез fluent API правилата които не могат да се дефинират чрез

конвенционално именуване или атрибути.

-ApplicationDbContext.cs-Клас съдържащ метод който да

връща конфигуриран ApplicationDbContext (в него трябва да е

зададен ConnectionString за да се осъществи връзка с базата

данни.

2.5-DTO-Клас Library съдържащ всички Data Transfer

Objects. Целта на DTO обектите е да излагат малка част от данните

намиращи се в базата към визуализационния слой, както и при

създаване на нови обекти да транспортират само нужната информация

за създаването или редактирането на конкретен обект. Съдържанието

на библиотеката е групирано както следва:

-AddressChoiseDTO.cs-използва се при всички екшъни на Estate,без delete,тъй като

Address си има отделен контролер и се контролира от там.

-CreateEstateDTO.cs-използва се за създаването на имотите с техните екстри

-DetailEstateDTO\_out.cs-използва се за детайлите на имотите с техните екстри

2.6-Services-в него се намира класът DbSeeder.cs:

-DbSeeder-Този клас съдържа логика

проверяваща дали базата е празна или не. Също и методи които

да популират базата първоначално. Това е важно в процеса на

разработка на приложението защото база без записи е трудно да

бъде визуализирана на фронтенд от една страна, и от друга

често се налага да се повтарят операциите по популиране на

база в процеса на разработка, което ако се случва ръчно би

отнело много време и усилия.

2.7-Helpers-в него се съдържа класът GlobalConstants, неговата цел

е да съдържа променливи, които ще улеснят работата във Views,

като например currency= lv или dateFormat = dd-MM-yyyy г.

3. Достъп и зависимости на отделните проекти в Salution, видимост.

3.1. DimplomenProekt – Има право да ползва/вижда:

-Data

-DTO

-Models

-Services

3.2Data– Има право да ползва/вижда:

-Models

-Enumerations

3.3DTO- Има право да ползва/вижда:

-Models

3.4Models- Не ползва функционалност от други

проекти.

3.5Services- Има право да ползва/вижда:

-Data

4. Съображения при избора на връзки между класовете:

При построяването на връзките “dependencies” се съблюдават две

правила.

- Да може да функционира приложението тоест във всяка

библиотека-проект да има достъп до ресурси (класове)

намиращи се в други библиотеки-проекти.

- Да се избегне Circular Reference. Не се допуска две библиотеки

да са видими взаимно пряко или чрез посредник, вместо това

трябва достъпът да бъде иерархичен. По тази причина видно от

зависимостите може да се заключи, че на върха на йерархията на

проектите в Solution се намира Application конзолното

приложение от което се стартира цялото приложение.

V. База данни.

1. Избор на база данни:

При разработката на приложението е избрана релационна база данни

Microsoft SQL Server и приложение за управлението и Microsoft SQL

Management Studio, в което може да се визуализира базата както и да се

пишат Transact SQL заявки към базата.

2. Подход при разработката и внедряване на базата:

2.1. Подходи при създаване на нова база данни:

- DataBase First – При този подход базата се изгражда

посредством скрипт написан и изпълнен на Transact SQL, като

всички правила, Ограничения, процедури и т.н. се дефинират в

SQL. След като базата е готова на всяка таблица от нея трябва да

се създаде отразяващ я клас, който да репрезентира данните от

базата. Създаването на тези класове е възможно да се изпълни и

чрез Scaffold автоматично.

Основно предимство на този подход е, че базата може да се

дефинира по-оптимално и ясно в T-SQL. Могат да се използват

всички “features” на конкретният SQL език вместо само такива

валидни за всички SQL езици.

Недостатък на този подход е че базата дефинирана през T-

SQL е обвързана с конкретният избор на SQL сървър, изисква се

познание в избраният SQL език. По времеемка разработква.

-Code First – При този подход базата се изгражда след и въз

основа на класове в C#. За да се създаде базата, се генерират

миграции които се прилагат към базата. Всяка такава миграция

съдържа автоматично генерирани T-SQL правила. Като могат да се

извършват промени по класовете и да се допълват миграциите в

последствие.

Основно предимство на този подход е, че ORM кореспондира

с избраният SQL сървър, което позволява на програмиста да не

използва чист SQL и предоставя възможност с много малко усилия

да се смени SQL сървъра например с Postgre или MySQL в

последствие.

Недостатъците на този подход са:

Понеже се разчита на Provider клас който да генерира SQL

код, се изисква познание как да се укажат правила при самите

класове които да доведат до правилна интерпретация на класовете.

Пример за това са атрибути, FluentApi правила, или именоване на

свойствата на класовете по конвенция.

Друг недостатък е че ако се изискват функционалности от

базата каквито само конкретната база предлага, то това не е най

удачният вариант защото Provider класовете за всеки тип база

поддържат общи интерфейси тоест само обща функционалност.

2.2. Избран подход и обосновка:

За разработката на приложението е предпочетен подхода Code

First понеже в процеса на разработка ще се променят класовете и

ще се разчита на миграции за да се отразят тези промени, също

така Code First предлага възможност да се разработи приложението

без използване на T-SQL а фокусът се измества в посока на C#

класовете, където може да се възползва програмистът от Intelisense

функциите на Visual Studio 2022.

За неконвенционалните правила се разширява методът

protected override void OnModelCreating(ModelBuilder builder)

Тук се задават правила като композитни ключове и др.

Също така на места са използвани атрибутни обозначения в

класовете дефиниращи таблиците модели\* (Models).

3. Създаване на база данни и реализация в проекта.

Класовете репрезентиращи базата данни се намират в class

Data.Models и могат да се виждат в приложението към

дипломният проект.

3.1. Идентификатори:

Всеки клас ,който репрезентира конкретна таблица в базата

данни, би следвало да има идентификатор (Id), този идентификатор

може да бъде от числов или стрингов тип (GUID). Когато се цели

по-голяма сигурност на данните и невъзможност да се предвиди

идентификатор на отделни записи при зловреден достъп отвън се

назначава string тип за идентификатор, а когато това не е нужно се

използва int.

Това е така понеже ,когато

\*За по нататъчно улеснение ще наричаме класовете съдържащи

Entity типове (или репрезентиращи таблици в база данни) – Модели.

потребителят иска да достъпи своите данни или да види

данни на конкретен потребител ,ако това е позволено (например в

коментари секция ,ако има налична) от GDPR съображения е

необходимо да не може да се достъпват данни на произволни

потребители чрез for-цикъл например.

Някои от таблиците имат така наречените композитни

ключове, вместо идентификатори от тип Int или string. Това се

налага в случаите когато имаме т.нар Mapping таблица съдържаща

„връзка“ между две таблици или 2 Foreign Key със типове

отговарящи на съответната таблица. В този случай ако се цели

уникалност на връзката за да се елиминира повторение на

конкретната връзка се използва композитен ключ генериран на

базата на двата Foreign ключа от връзката, по този начин при

последващ опит да бъде създаден нов запис със същите ключове

това не се позволява от базата тъй като вече има наличен запис с

такъв Id.

3.2. Обща функционалност във (за) всички модели:

За всеки Model съществува базова функционалност, която е

сходна, и по тази причина са създадени два класа -родители, които

да предоставят за наследяване общата функционалност на всички

модели – деца.

[BaseEntityData] както и [BaseEntity&lt;T&gt;:BaseEntityData].

Класовете са дефинирани, като абстрактни\*.

BaseEntityData е родител на класа BaseEntity&lt;T&gt;. По този

начин всички класове „са BaseEntityData” но не всички “са

BaseEntity”.

\*Абстрактните класове не позволяват инстанциране, тяхната цел е

да предоставят функционалност на наследниците им, и да

позволяват наследяващите ги класове да бъдат представяни като

абстрактният тип при дефиниране на данните (например работа с

колекции и др.).

- BaseEntityData.

Тук е поместена функционалност абсолютно необходима за всеки

модел. Пряко този клас се наследява само от модели

репрезентиращи Mapping таблици, понеже идентификаторите в тези

класове са композитни ключове генерирани на базата на

ForeignKeys. Косвено този клас е наследен от всички останали чрез

BaseEntity&lt;T&gt;. Този клас е в основата на всички модели.

Функционалността която този клас поддържа състояние и

поведение, както следва:

- При инстанцирането на кой да е негов наследник се попълва

свойството DateOfCreation от тип DateTime.

- Съдържа се свойство от тип bool IsDeleted което отразява дали

даденият запис е изтрит от базата или не. Това позволява

Soft-Deletion\* на записи от базата. Освен това при промяна на

това свойство сетера е модифициран така, че да запомня датата на

изтриване на обекта, като в същото време позволява изтриването

или задаването на този запис като изтрит само веднъж и при

задаване на обект като изтрит след като той вече е изтрит не се

променя датата на изтриване. По тази причина типа данни на

DateOfDeletion е nullable\* DateTime понеже неизтритите записи

нямат дата на изтриване докато типа DateTime не поддържа null.

Поддържа се функционалност и възтановяване на вече изтрит

запис.

DateOfDeletion както и DateOfCreation се задават автоматично с

текущата дата и час според GMT\*.

\*Soft-Deletion – При базите данни когато има налични Foreign Key

външни ключове, не е лесно даден запис да бъде изтрит ако неговият

Id се явява ForeignKey за друга таблица. Например ако имаме

Потребител и Коментар на потребителя и искаме да изтрием

потребителя, възниква въпроса кой е оставил коментара? Подходите

за справяне с този проблем са няколко. Каскадно, Задаване на Foreign

Keys като null ако това е възможно или Ignore. Всеки един от

подходите по горе, създава множество проблеми, затова решението е

софтуерно да се зададе колона в таблиците от тип bool IsDeleted

(репрезентирана в базата чрез 0-false и 1-true), като вместо да се

изтриват данните просто не се визуализират и борави със данни

имащи зададена стойност true.

\*nullable – По подразбиране типовете имат стойност която не

винаги е null. Например типа int32 ако не бъде указана стойност по

подразбиране е със стойност 0. Както и типа DateTime задава

начална дата при инициализиране. C# позволява декларирането на

типове които да имат възможност да заемат стойност null

умишлено когато това не е тяхното поведение по подразбиране.

GMT – Приложението следва да се качи (deploy) в интернет и не е

гарантирано, че то ще се ползва само от потребители в една и съща

часова зона, затова е редно да се използва универсална общоприета

часова зона, която при нужда да се конвертира към локалното време

което се репрезентира различно за всеки отделен потребител.

BaseEntity<T>.

Представлява Generic абстрактен клас в който се указва типа на

идентификатора и бива наследен пряко от класовете, които искаме

да имат конкретен тип идентификатор. Тоест всички без mapping

моделите където се търси уникалност на връзката. Типа е Generic

защото се търси гъвкаво указване на типа на идентификатора

(string, int) при самото наследяване.

Класът BaseEntityData&lt;T&gt; съдържа свойство Id от тип T като над

свойството с ключа съдържа атрибут указващ на EntityFramework,

че идентификаторите трябва да бъдат задавани от базата, като

поредни а не да се указват от потребителя ръчно (Identity\*).

\*Identity - при ключовете означава записите да бъдат създадени без

да се указват изрично стойностите на техните ключове (Id) в

базата. Съответно базата извършва проверка до кой пореден номер е

достигнато при назначаването на последният ключ и присвоява

идентификатор с пореден номер. Интересното е че при изтриване на

записи и ключове не се обновява броячът и така е възможно да има

липсващи Id-та (да не са поредни). Това е нормално, понеже основна

задача на идентификаторите е те да бъдат уникални а не поредни.

4. Модели- описание на релации и организация на данните във всяка

таблица (модел).

В приложението е използвана библиотеката

Microsoft.AspNetCore.Identity. Тя предоставя базова имплементация

на класа потребител както и на таблици управляващи неговите

правомощия и достъп до приложенията. Тъй като операциите по

регистрация, лог-ин автентикиране и роли са тривиална\* задача

тази библиотека предоставя възможност да се преизползва това

образцово решение и структура като при необходимост тя се

разширява (extend)-ва с допълнителна функционалност.

Библиотеката съдържа както дефиниции на класовете IdentityUser и

IdentityRole, така и съпътстващите работа с базата класове-Services

съответно UserManager, SignInManager и RoleManager.

4.1. AppUser – Представлява клас наследяващ IdentityUser, като му

добавя нови свойства и методи имащи отношение към конкретното

приложение. От библиотеката интерес представляват (и се използват)

класовете AspNetRoles и свързващата таблица AspNetUserRoles

обединяваща конкретен User със съответна роля. Библиотеката предлага

и други класове, но те нямат отношение към функционалността на

приложението.

\*Тривиална задача в програмирането означава задача чието решение е

ясно за имплементация но времеемко и може да се спести

посредством готов код.