ДИПЛОМЕН ПРОЕКТ ЗА ДЪРЖАВЕН ЗРЕЛОСТЕН ИЗПИТ

по професия код 481030 „Приложен програмист“

специалност код 4810301 Приложно програмиране“

Тема: „Система за управление на полети на малки летателни апарати“

Автор:

Божидар Димитров , клас XII В

Ръководител:

Красимир Ватев

Бургас

СЪДЪРЖАНИЕ

1 Увод

Уеб приложението има за цел да създаде удобна и ефективна система, която да улеснява работата на геодезистите ползващи фотограметрични технологии за геодезическо производство. Настоящата разработка създава възможност за съвместяване на евтини дрон-технологии за заснемане по фотограметрични методи с прецизни GPS системи конструирани от геодезическа фирма в България. Тази GPS система осигурява 2 см точност при определяне на местоположението на центровете на камерите . Проблем на сглобения хардуер са множеството ненадеждни записи, извличане на метаданни от първичните растерни изображения и ползването им за създаване на файлове с имена на изображения и прецизни координати . При постигане на този резултат, рязко се намалява инвестицията за създаване на високоточна фотограметрична система, в десетки пъти се увеличава производителността на фирмите, които я ползват, както и се увеличава прецизността и надеждността на крайния геодезически продукт. Системата има за цел да категоризира летателните апарати по зони за летене.

Приложението предоставя съвременен, удобен и функционален интерфейс за качване на полети под формата на текстови файлове и точки от тези полети под формата на изображения.

Приложението обработва данни с прецизни координати от полети с дрон и преобразува дадените координати от географски в проекционни. Приложението може да извлича метаданни от изображения и да използва тези данни, за да свърже координатите, получени от предишната прецизна обработка с изображенията, от които извличаме грубите метаданните.

По този начин улесняваме работата на геодезистите, свързана с програмите, които използват, заменяйки ръчната обработка на данните. Системата има голям потенциал да бъде използвана в индустрията на геодезията.

Проектът ще включва анализ на изискванията, дизайн, разработка и тестване на системата. Резултатът ще бъде функционален уеб сайт, който ще елиминира грешните резултати от полетите.

Анализът на изискванията ще включва проучване на сходни системи, определяне на функционалните изисквания и

потребителските нужди. Дизайнът ще включва разработване на интерфейса, структурата и архитектурата на системата.

Разработката ще бъде реализирана със съвременни технологии за уеб разработка. Базата данни ще бъде изградена за оптимално съхранение и извличане на информация. Тестването ще включва тестове за надеждност, производителност, сигурност и потребителски интерфейс.

2 Цели и обхват на софтуерното приложение

Основните цели на приложението са:

- Намаляване на грешката в координатите на снимките. Приложението цели да предостави платформа, която да улеснява свързването на координати на снимки, заснети по време на полет с дрон с тези на полети, записани в текстови файлове, които са по-точни.
- Предоставяне на функционален интерфейс за качване на текстови файлове и изображения, чиито координати ще се свързват. Целта е да се създаде удобен и интуитивен потребителски интерфейс.
- Съхранение и поддръжка на голямо количество полети, изображения и точки от полети. Приложението трябва да може да обработва и съхранява голям брой полети и изображения от различни дронове.

Обхватът на приложението включва:

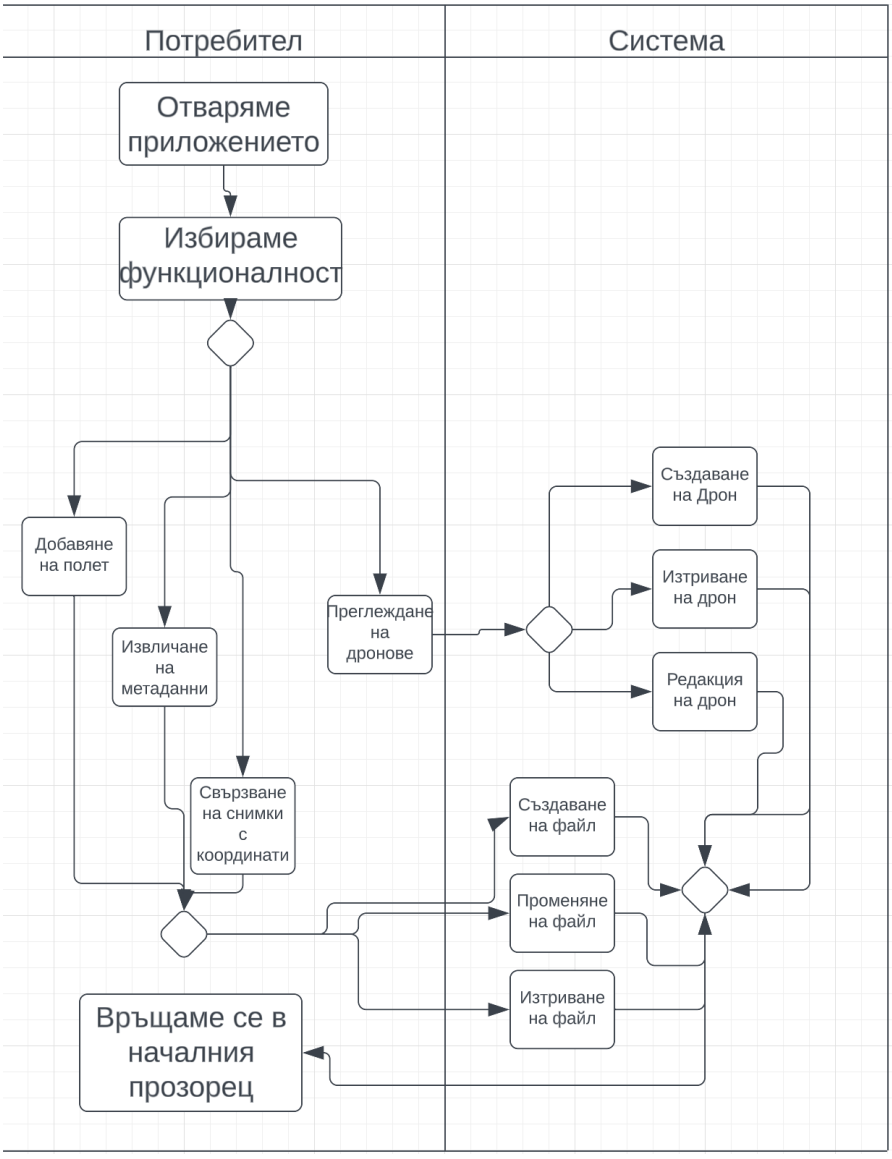
- Възможност за добавяне на дронове чрез имена и категории на дрон.
- Възможност за прикачване на полети, чиито данни са в определено време спрямо съдържанието на текстови файлове.
- Възможност за прикачване на изображения от полети чрез “.jpg” и “.png” файлове.
- Възможност за изобразяване на полетите чрез таблици.
- Възможност за изобразяване на точките от полетите чрез таблици.
- Възможност за изобразяване на метаданните от снимките чрез таблици.
- Възможност за свързване на метаданните от снимките и точките от полетите

- Възможност за избиране на категория на зона на летене на летателен апарат.
- Възможност за автоматично изтриване на ненужни точки от записа на полет.

Проектът няма ограничение за възрастовата граница за използване.

3 Анализ на решението

3.1 Потребителски изисквания и работен процес



Фигура 1 (Диаграма на активността)

Потребителските изисквания са приложението да бъде с лесен за използване графичен интерфейс, който да помогне с по-лесното използване на софтуера.

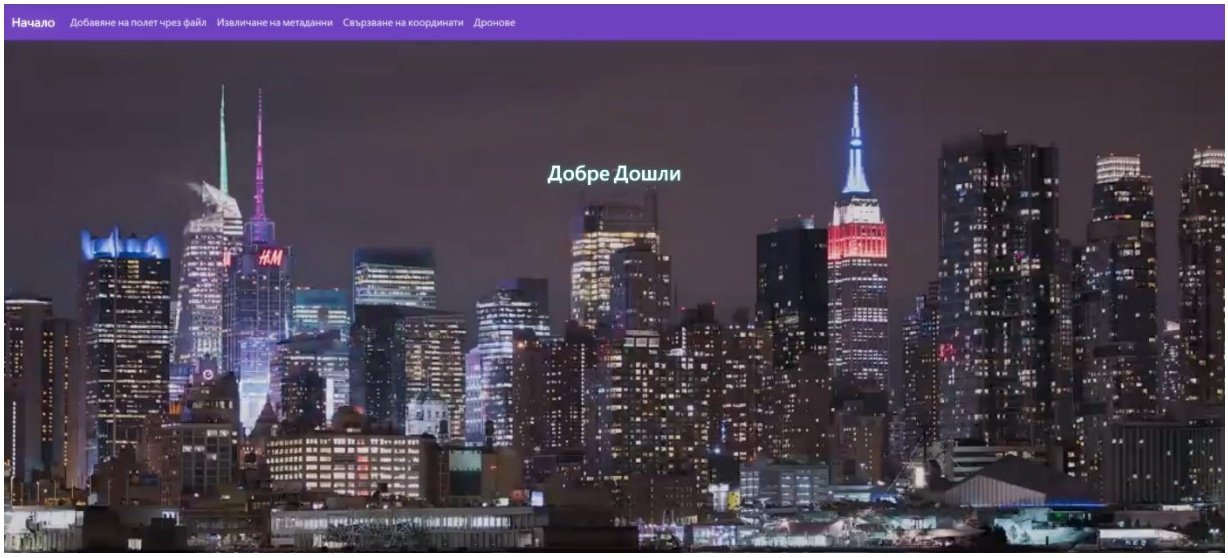
Вход: Данните за вход на системата се получават от добавянето на полети под формата на текстови файлове(.pos, .txt), добавянето на снимки в страницата за извличане на метаданни и добавянето на дронове. Имената на полетите, добавени чрез текстов файл автоматично заемат името на текстовия файл. Имената на полетите, добавени чрез снимки биват записвани ръчно от потребителите.

Обработка: Получените данни се обработват и съхраняват в базата данни, изградена с “MSSQL”. Процесът включва преработването на координатите от географски в проекционни, извличането на метаданните от изображения и верификация на данните. При търсене се извършва сортиране спрямо дрона, от който е записан полета, тип на файловете, в които се пазят полетите и тип на координатите, които ще бъдат върнати в изхода.

Изхода на системата включва:

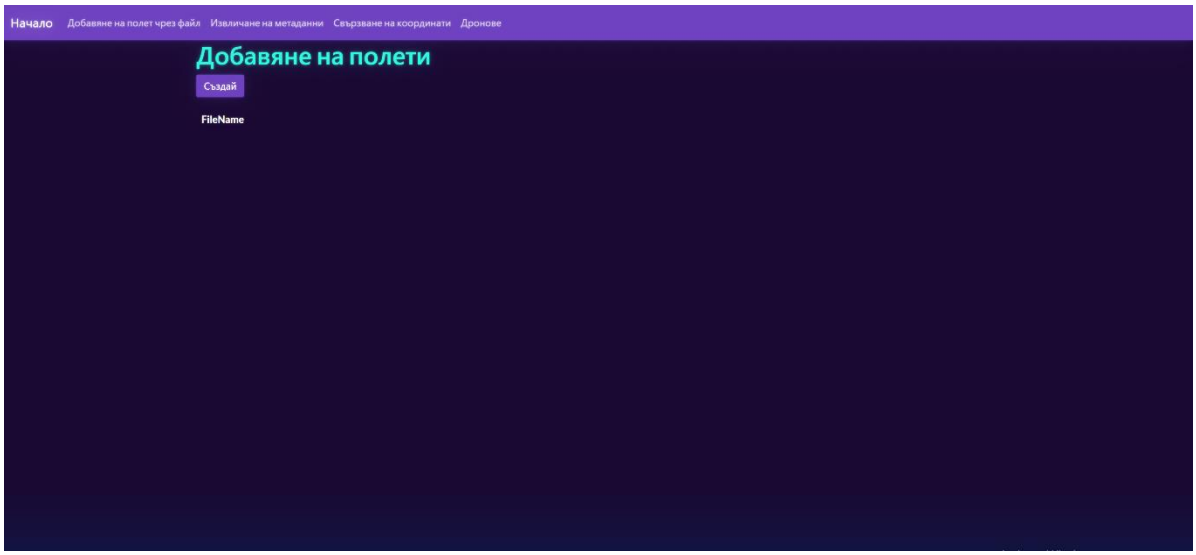
- Предоставяне на списък с полети, които са въведени чрез текстови файлове.
- Предоставяне на списък с полети, които са въведени чрез изображения.
- Предоставянето на списък с координати на точки, които се съдържат в полети, въведени чрез текстови файлове.
- Предоставянето на списък с координати на точки, които се съдържат в полети, въведени чрез изображения.
- Предоставянето на списък с дронове.
- Получените резултати се използват от потребителя за по-добро ориентиране за полетите които иска да свързва.

3.2 Примерен потребителски интерфейс



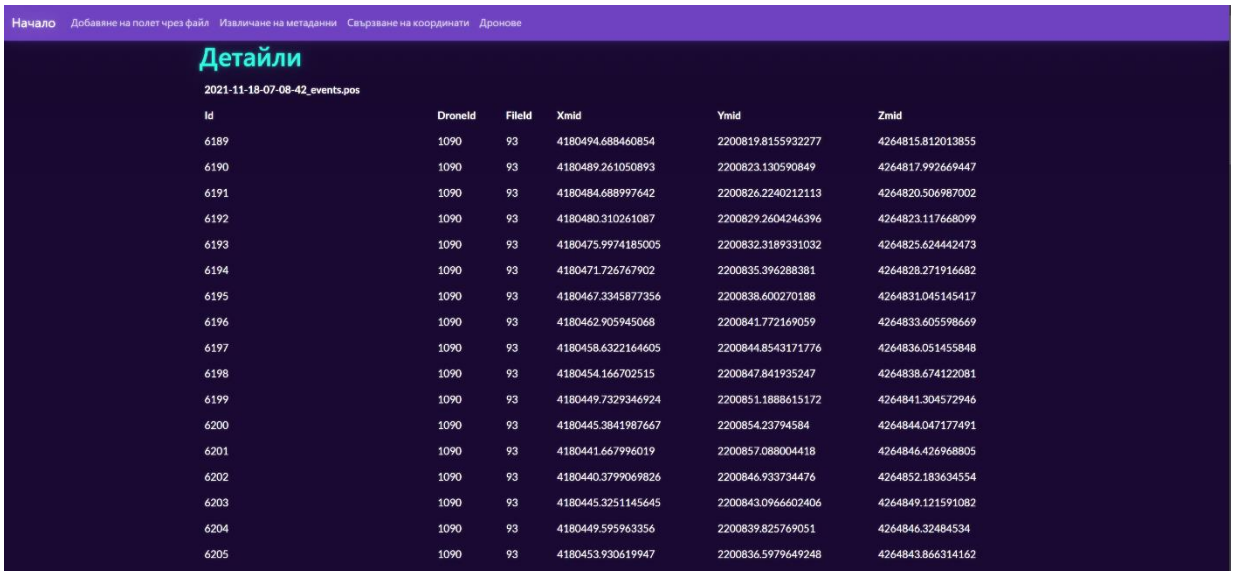
Фигура-2(Начална страница)

Това е началната страница на приложението. За менюто се използва лилава-неонова гама със бял шрифт, за по-добър контраст. Всеки един от бутоните на менюто води до определената страница.



Фигура-3(Страница за добавяне на полети)

Това е интерфейсът на страницата за страницата за добавяне на полети. Бутоните биват лилав и черен цвят с бял текст, за по-добър контраст. Текстът, който показва на коя страница е потребителят, е светлосин, което допълва лилаво-неоновата гама.

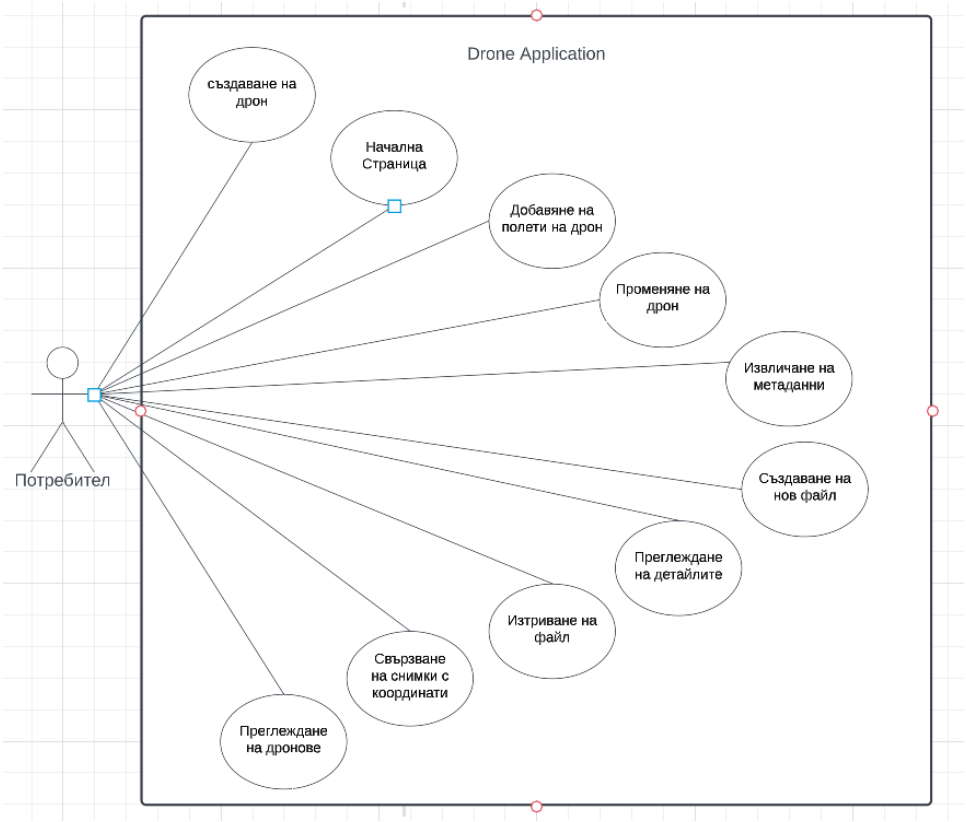


Id	DroneId	FileId	Xmid	Ymid	Zmid
6189	1090	93	4180494.688460854	2200819.8155932277	4264815.812013855
6190	1090	93	4180489.261050893	2200823.130590849	4264817.992669447
6191	1090	93	4180484.688997642	2200826.2240212113	4264820.506987002
6192	1090	93	4180480.310261087	2200829.2604246396	4264823.117668099
6193	1090	93	4180475.9974185005	2200832.3189331032	4264825.624442473
6194	1090	93	4180471.726767902	2200835.396288381	4264828.271916682
6195	1090	93	4180467.3345877356	2200838.600270188	4264831.045145417
6196	1090	93	4180462.905945068	2200841.772169059	4264833.605598669
6197	1090	93	4180458.6322164605	2200844.8543171776	4264836.051455848
6198	1090	93	4180454.166702515	2200847.841935247	4264838.674122081
6199	1090	93	4180449.7329346924	2200851.1888615172	4264841.304572946
6200	1090	93	4180445.3841987667	2200854.23794584	4264844.047177491
6201	1090	93	4180441.667996019	2200857.088004418	4264846.426968805
6202	1090	93	4180440.3799069826	2200846.933734476	4264852.183634554
6203	1090	93	4180445.3251145645	2200843.0966602406	4264849.121591082
6204	1090	93	4180449.595963356	2200839.825769051	4264846.32484534
6205	1090	93	4180453.930619947	2200836.5979649248	4264843.866314162

Фигура-4(страница за детайли)

Гамата на интерфейса е подобна навсякъде. Това е страницата за детайли, която е подстраница на операцията за осредняване.

3.3 Диаграми на анализа



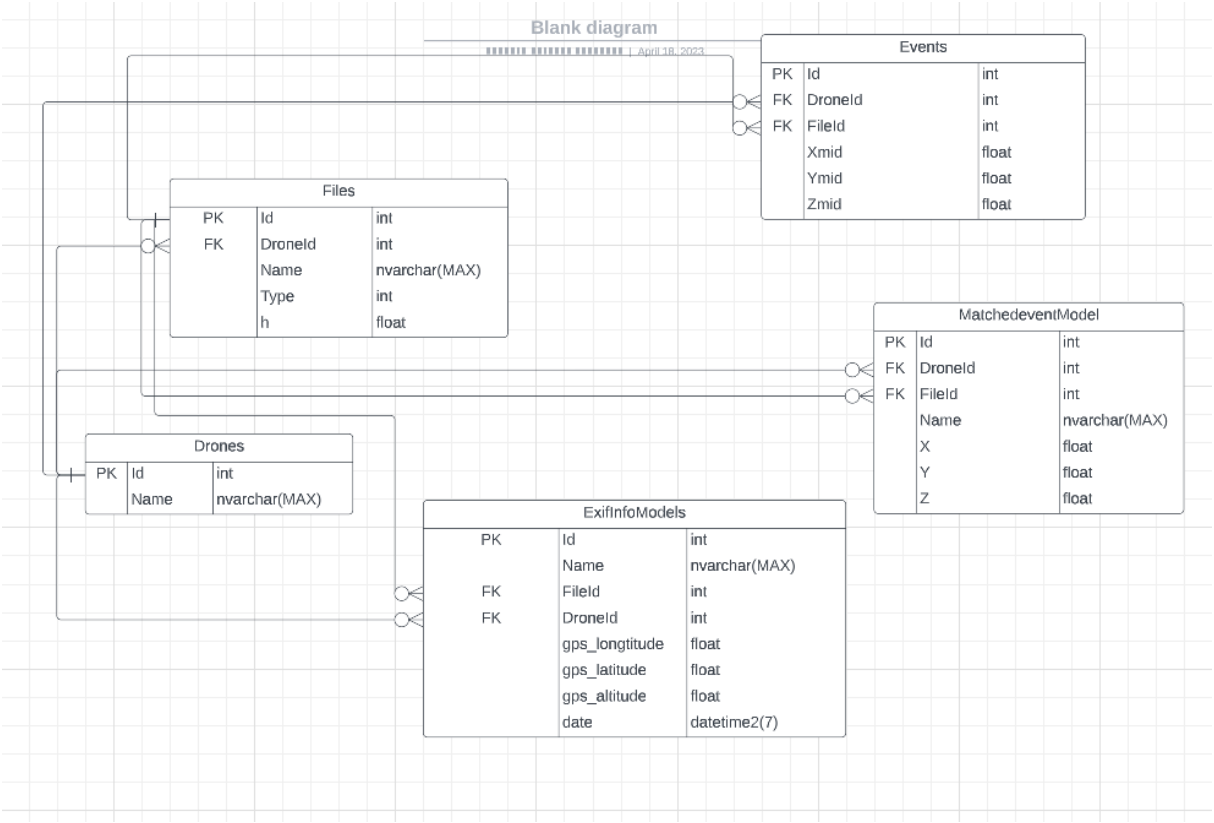
Фигура-5(Диаграма на случаи на употреба)

Потребителят отваря главната страница. За да използва функциите на приложението, потребителят трябва да натисне върху името на задачата, която иска да изпълни, което се намира в менюто. След като си избере функция, се предоставят 4 избора за действие за избраната функция - създаване, детайли, изтриване и променяне. Потребителят винаги има възможността да смени процедурата, която е избра или да се върне в началната страница.

За реализирането на приложението ще бъде използвана база от данни(предварително създадена). Front-End и Back-End частта ще бъдат реализирани на Java, JavaScript, HTML и CSS

3.4 Модел на съдържанието / данни

3.4.1 Данни в базата.



Фигура-6(Моделът на всъщност връзка)

Това е ER диаграмата. Тя описва Таблиците в базата данни и отношенията им помежду си. Базата данни е приведена в трета нормална форма. Таблицата „Drones“ има 2 полета, които са показани на снимката. Полето „id“ служи като „вторичен ключ“ във всички останали таблици. Таблицата „Files“ има „вторичен ключ“, който се свързва с таблицата „Drones“ към „DroneId“. В таблицата „MiddledEventModel“ има 2 „вторични ключа“. Единия е „DroneId“ и се свързва с таблицата „Drones“ към „DroneId“. Вторият е ключът „FileId“ и се свързва с таблицата Files към „FileId“. В таблицата „Events“ има 2 „вторични ключа“. Единия е „DroneId“ и се свързва с таблицата „Drones“ към „DroneId“. Вторият е ключът „FileId“ и се свързва с таблицата Files към „FileId“. В таблицата „ExifInfoModel“ има 2 „вторични ключа“. Единия е „DroneId“ и се свързва с таблицата „Drones“ към „DroneId“. Вторият е ключът „FileId“ и се свързва с таблицата Files към „FileId“.

3.4.2 Данни въведени от потребителя.

Приложението работи и изисква данни в потребителския интерфейс от следните видове: текстови файлове, изображения, текст. Информацията от текстовите файлове се извлича, преработва и запазва в класовете, които са описани в клас диаграмата за моделите. Тази информация се запазва под формата на текст и на числа в класовете. Метаданните от изображенията биват извлечени, преработвани и запазвани в класовете, които са описани в клас диаграмата за моделите. Тази информация се запазва под формата на текст и на числа в класовете. След това информацията бива запазена в базата данни. Графичното съдържание бива запазено в папката “wwwroot/TextFiles”, която се намира в проекта “DroneApplication” в приложението. Текстовото съдържание също. Видеото, което е сложено за фон на началната страница също. Големината на една заявка стига максимално до 1 гигабайт.

Съдържанието в текстовите файлове бива подредено под определен шаблон, за да проработи прочитането. Първите 27 реда са стандартни и се игнорират. След това следват различните полета, като разделителят им е разстояние. От второ до пето поле включително се четат и се запазват от приложението и всичко друго се игнорира. Файловете се прикачват само по единично. Големината на една заявка стига максимално до 1 гигабайт.

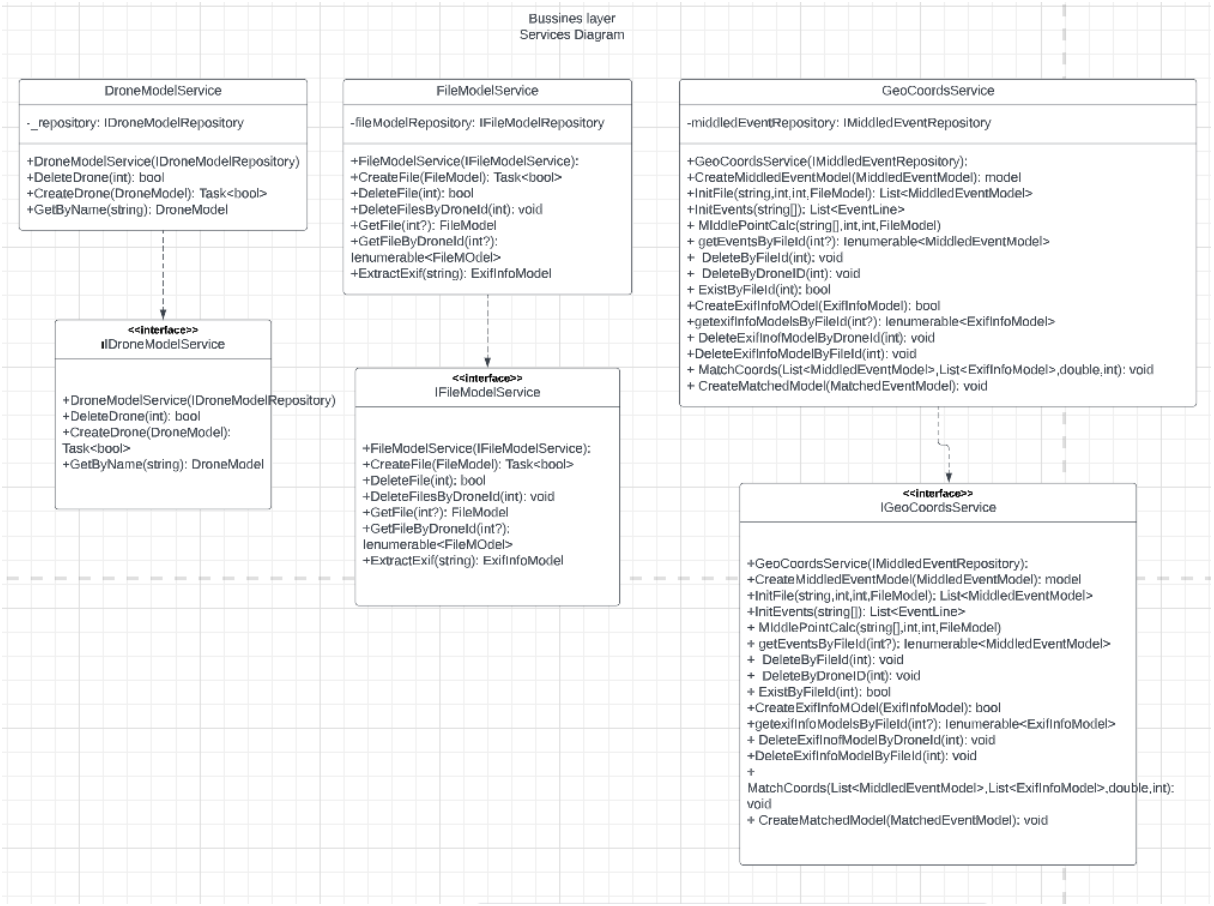
Изображенията, които се качват, трябва да имат метаданни за полетата “GPS-Latitude”, “GPS-Longtitude”, “GPS-Altitude” и “Date”. Снимките могат да се прикачват както по единично, така и по много.

Текстовото съдържание няма ограничения.

4 Дизайн

За направата на проекта като програмен език е избран “C#” i .”Net” като рамка за сървърната система (back-end). За изграждането на сайта е използван HTML, CSS , JS и Bootstrap за изграждането на дизайна на изгледа. Като шаблон за архитектурата е избран MVC (Model-View-Controller) със добавени слой за бизнес логика и слой за работа с базата данни. Използван е шаблонът за хранилища(Repository pattern). Подходът, по който е създадена базата на проекта, е “code first”.

Класовете-услуги от слоя за бизнес логика:



Фигура-7(Диаграма на класовете "Services")

Тази диаграма е за класовете услуги и интерфейсите в бизнес слоя.

Всеки клас има поле, което се използва за запазване на инстанцията на обекта за хранилището, което съответства на съответната услуга.

Класа "DroneModelService" имплементира интерфейса "IDroneModelService". В конструктора на класа бива запазвана инстанцията на класа за хранилището. Останалите методи са стандартни извикват стандартните "CRUD" операции от съответното хранилище.

Класът „DroneModelService" предлага методи за достъп до съответното хранилище:

- DeleteDrone(int id): Извиква метода със същото име от хранилището и връща стойността му.
- CreateDrone(DroneModel model): Извиква метода със същото име от хранилището и връща стойността му.

- GetByName(string name): Извиква метода със същото име от хранилището и връща стойността му.

Класът “FileModelService” имплементира интерфейса “IFileModelService”. И тук методите извикват стандартните “CRUD” операции от съответното хранилище, като има променени методи, които изпълняват “CRUD” операции спрямо определени критерии.

Класът „DroneModelService“ предлага методи за достъп до съответното хранилище, както и метод за извличане на метаданни от снимки:

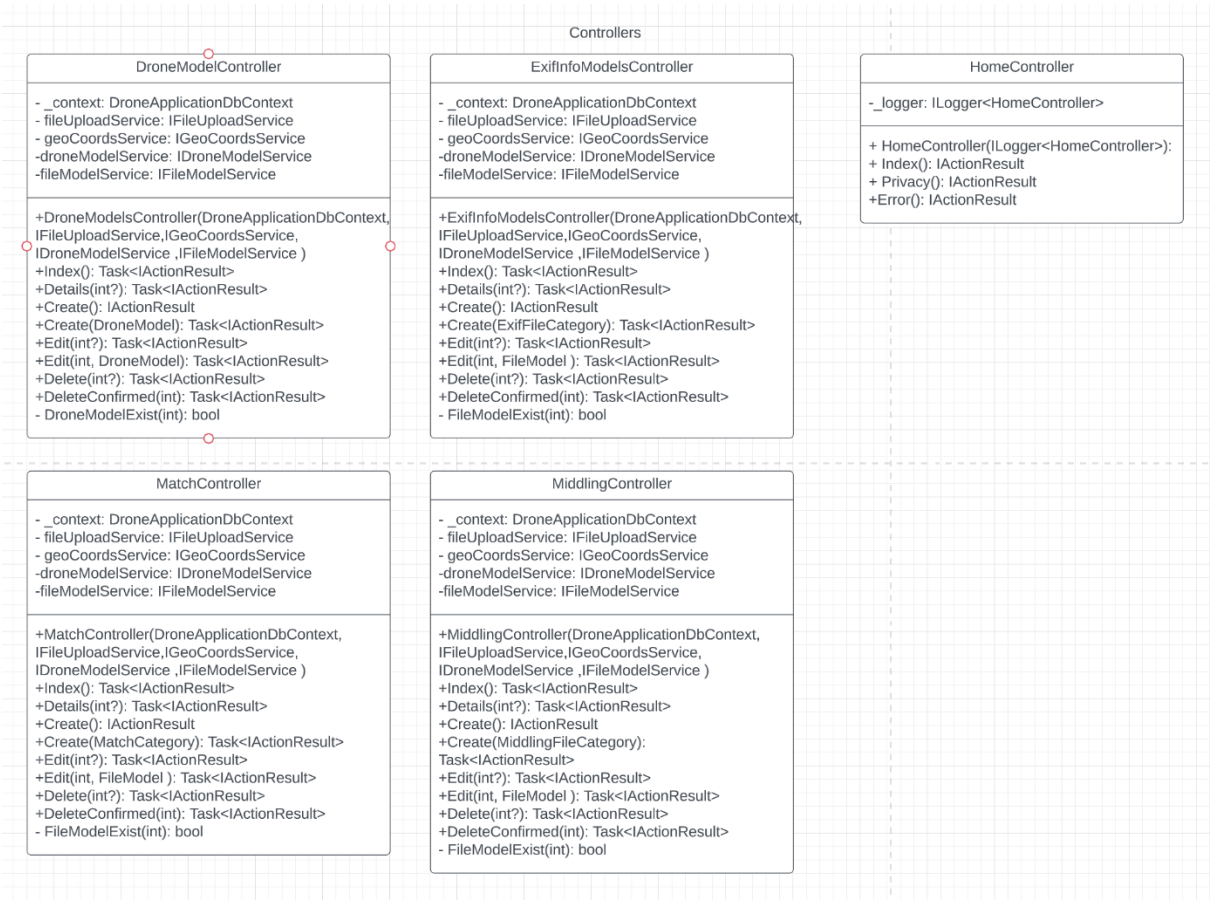
- CreateFile(FileModel model): Извиква метода със същото име от хранилището и връща стойността му.
- GetFile(int? id): Извиква метода със същото име от хранилището и връща стойността му.
- GetFileByDroneId(int? id): Извиква метода със същото име от хранилището и връща стойността му.
- DeleteFilesByDroneId(int id): Извиква метода със същото име от хранилището и връща стойността му.
- ExtractExif(string path): Извлича метаданни от снимки и връща моделен клас “ExifInfoModel”, който ги складира.

Класа “GeoCoordsService” имплементира интерфейса “IGeoCoordsService”. В конструктора на класа бива запазвана инстанцията на класа за хранилището. И тук методите извикват стандартните “CRUD” операции от съответното хранилище, като има променени методи, които изпълняват “CRUD” операции спрямо определени критерии.

- CreateMiddledEventModel(MiddledEventModel model): Извиква метода със същото име от хранилището и връща стойността му.
- initFile(string path, int id, int fileId, FileModel file): Метода “InitFile” проверява дали пътя, който е подаден като аргумент води до съществуващ файл, който може да се отвори и извиква метода “MiddlePointCalc” и връща резултата му.
- InitEvents(string[] input): този метод отваря файла и го прочита ред по ред спрямо шаблона описан в 3.4.2. Връща резултат от тип “List<EventLine>” , който представлява списък от класове, който съдържат координатите от точките на полета.

- MiddlePointCalc(string[] input, int id,int fileId, FileModel file): Изчислява по формула “средната” точка между 2 заснемания на камерата и намира координатите. Извиква метода “CreateMiddledEventModel” от същата услуга за всяка една точка.
- getEventsByFileId(int? id): Извиква метода със същото име от хранилището и връща стойността му.
- DeleteByFileId(int id): Извиква метода със същото име от хранилището и връща стойността му.
- DeleteByDroneId(int id): Извиква метода със същото име от хранилището и връща стойността му.
- ExistByFileId(int id): Извиква метода със същото име от хранилището и връща стойността му.
- CreateExifInfoModel(ExifInfoModel model): Извиква метода със същото име от хранилището и връща стойността му.
- getExifInfoModelsByFileId(int? id): Извиква метода със същото име от хранилището и връща стойността му.
- DeleteExifInfoModelByFileID(int id): Извиква метода със същото име от хранилището и връща стойността му.
- DeleteExifInfoModelByDroneID(int id): Извиква метода със същото име от хранилището и връща стойността му.
- MatchCoords(List<MiddledEventModel> middledEvents, List<ExifInfoModel> exifInfoModels, double h, int FileId): Този метод отговаря за свързването на координатите от полетите добавени с текстов файл, със тези добавени от изображения.
- CreateMatchedModel(MatchedEventModel model): Извиква метода със същото име от хранилището и връща стойността му.

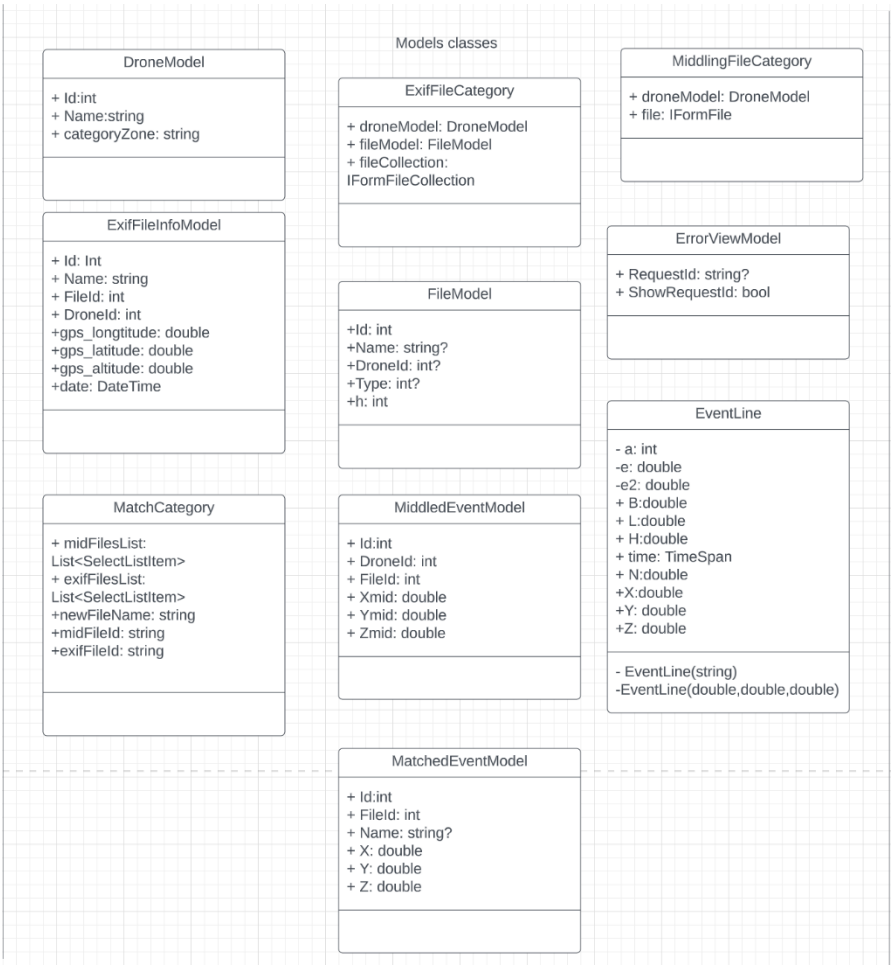
Контролните класове:



Фигура-8(Диаграма на контролните класове)

Това е диаграма за контролните класове. Те взаимодействат пряко с изгледите. Всеки един от тях без класа “HomeController” има “Index”, “Create”, “Delete”, “Details” и “Edit” методи, които отговарят за съответните изгледи. Класа “HomeController” има само “Index” и “Privacy” методи, които съответсват на изгледи.

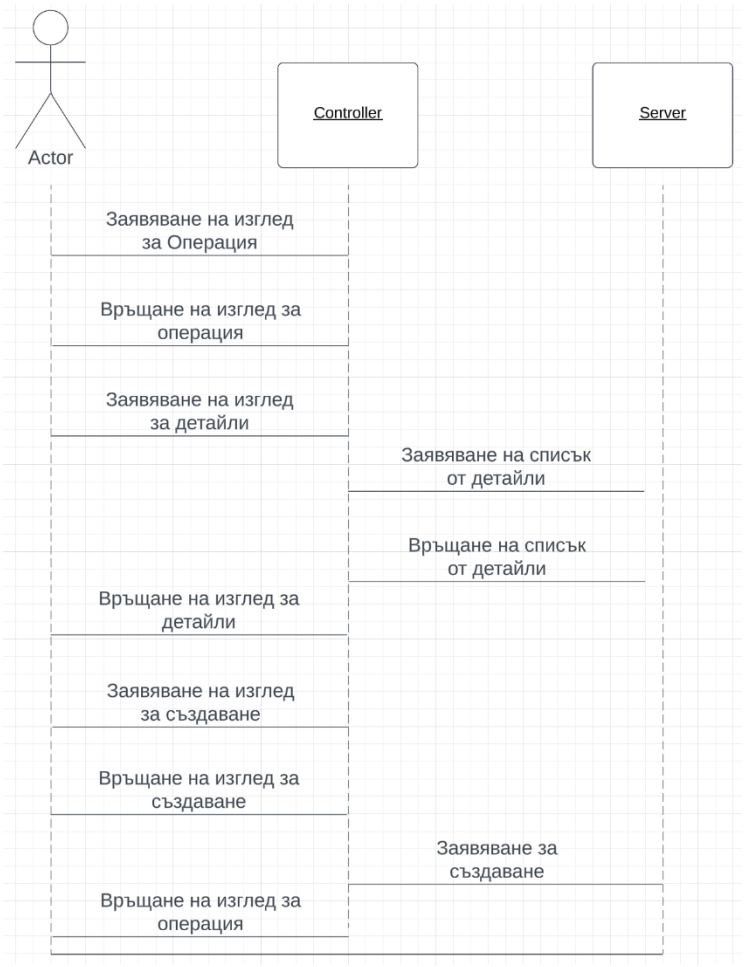
Класове от тип модели:



Фигура-9(Класове от тип модели)

Това е Диаграмата на моделите. Класовете „DroneModel“, „MiddledEventModel“, „ExifFileInfoModel“, „FileModel“ и „MatchedEventModel“ са използвани в базата данни като “Entity Classes”. Класовете, които завършват на „Category“ са използвани като модели, които предаваме в кода за изгледите чрез „HttpPost“ методи. Класа „ErrorViewModel“ е автоматично генериран при инициализиране на връзка с базата данни.

Диаграма на времето:



Фигура-10(Диаграма на последователността)

Диаграмата показва взаимодействието между различните участници в системата:

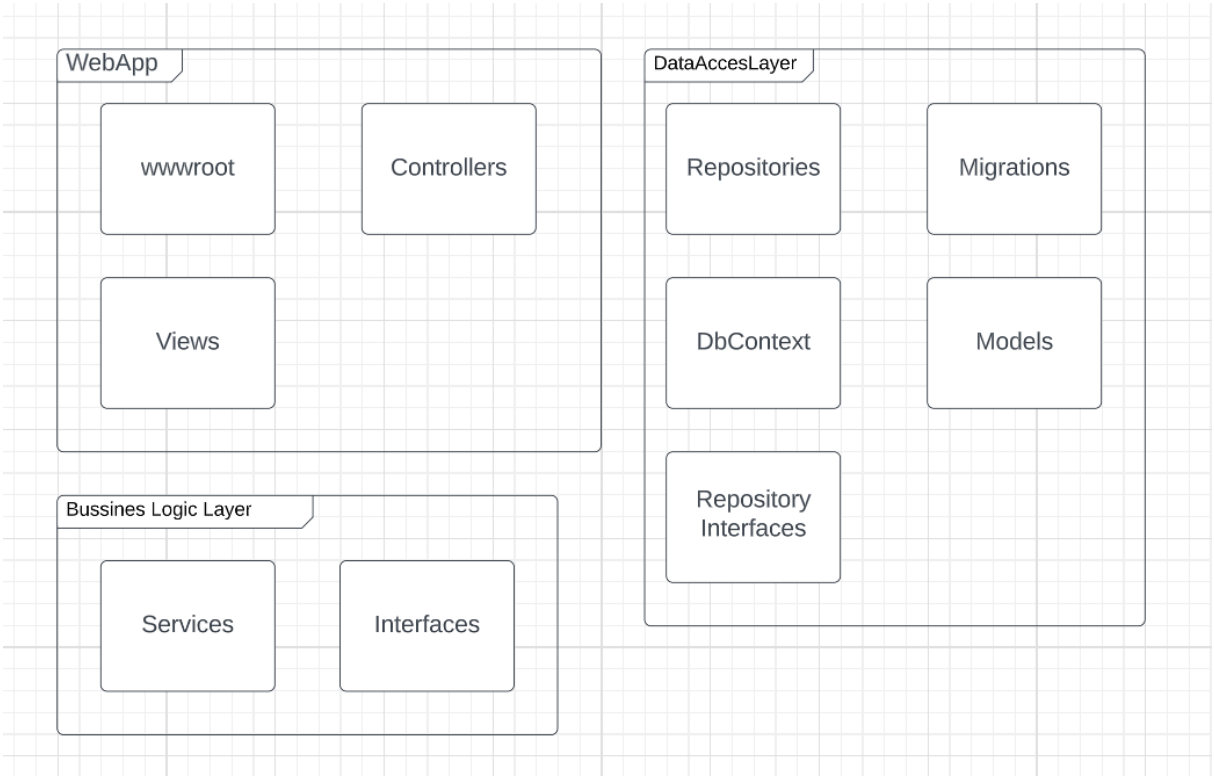
- Actor - Потребителя
- Controller - контролния слой
- Server - Сървъра на базата данни

Взаимодействията между участниците включват:

- Заявяване на изгледа за операцията, която Потребителя иска да отвори. Тези операции могат да бъдат Създаване на полети, извличане на метаданни, свързване на координати от полети със метаданни от изображения.
- Потребител може да избира изглед от детайли.
- Контролера изпраща заявка на сървъра за детайли.

- Сървъра връща информацията за обектите нужни за изгледа за детайли на контролера.
- Контролера визуализира изгледа за детайли на потребителя и потребителя вижда тази визуализация.
- Потребителя може да избира изглед за създаване
- Контролера връща изглед за създаване

4.1 Реализация на архитектурата на приложението



Фигура-11(диаграма на архитектурата на приложението)

Решението е написано на “Asp.Net” като е използвана “MVC” архитектура с добавени слой за бизнес логика и слой за достъп до базата данни. Приложението е използва шаблон за хранилища(Repository pattern). Тази диаграма показва архитектурното разпределение на проектите в решението. Можем да различим три проекта от които „DataAccesLayer“ е слоя за работа с базата данни, а „Bussines Logic Layer“ е слоя за бизнес логиката. Контролния слой се намира в папката „WebApp“ под името „Controllers“. Презентационния слой се намира в същата папка под името „Views“.

Модел-Изглед-Контролер (Model-View-Controller или MVC) е архитектурен шаблон за дизайн в програмирането, основан на разделянето на бизнес логиката от графичния интерфейс и данните в

дадено приложение. За пръв път този шаблон за дизайн е използван в програмния език Smalltalk.

Модел – ядрото на приложението, предопределено от областта, за която се разработва; обикновено това са данните от реалния свят, които се моделират и над които се работи – въвеждане, промяна, показване и т.н. Трябва да се прави разлика между реалния обкръжаващ свят и въображаемият абстрактен моделен свят, който е продукт на разума, който се възприема като твърдения, формули, математическа символика, схеми и други помощни средства

Изглед (англ. View) – тази част от изходния код на приложението, отговорна за показването на данните от модела. Например изгледът може да се състои от PHP шаблонни класове, JSP страници, ASP страници, JFrame наследници в Swing приложение. Зависи от това какъв графичен интерфейс се прави и каква платформа се използва;

Контролер – тази част от сорс кода (клас или библиотека), която взима данните от модела или извиква допълнителни методи върху модела, предварително обработва данните, и чак след това ги дава на изгледа. Също така когато се прави уеб графичен интерфейс това би довело до много лесна модификация на HTML кода дори от човек, който не е програмист – той ще гледа на шаблона просто като на обикновена HTML страница.

4.2 Описание на слоевете, предназначението им, библиотеки и методи включени в съответния слой.

Контролен слой(Controller layer): контролния слой отговаря за директната работа с изгледите и извиква методите от услугите. Той бива разделен на 5 класа: „DroneModelController“, “ExifInfoModelsController”, “HomeController”, “MatchController”, “MiddlingController”. Работата на контролерите е описана в точка 4.4. Пакетите инсталирани в този слой: „MetadataExtractor“, „Microsoft.EntityFrameworkCore.SqlServer“,
“Microsoft.EntityFrameworkCore.”,
“Microsoft.VisualStudio.Web.CodeGeneration.Design”.

Презентационен слой(view layer): този слой съдържа “HTML” код за потребителския интерфейс. Този слой отговаря за изобразяването на “HTML” маркирането, което се изпраща до уеб браузъра на

потребителя. Пакетите инсталирани в този слой: „MetadataExtractor“, „Microsoft.EntityFrameworkCore.SqlServer“,
“Microsoft.EntityFrameworkCore.”,
“Microsoft.VisualStudio.Web.CodeGeneration.Design”.

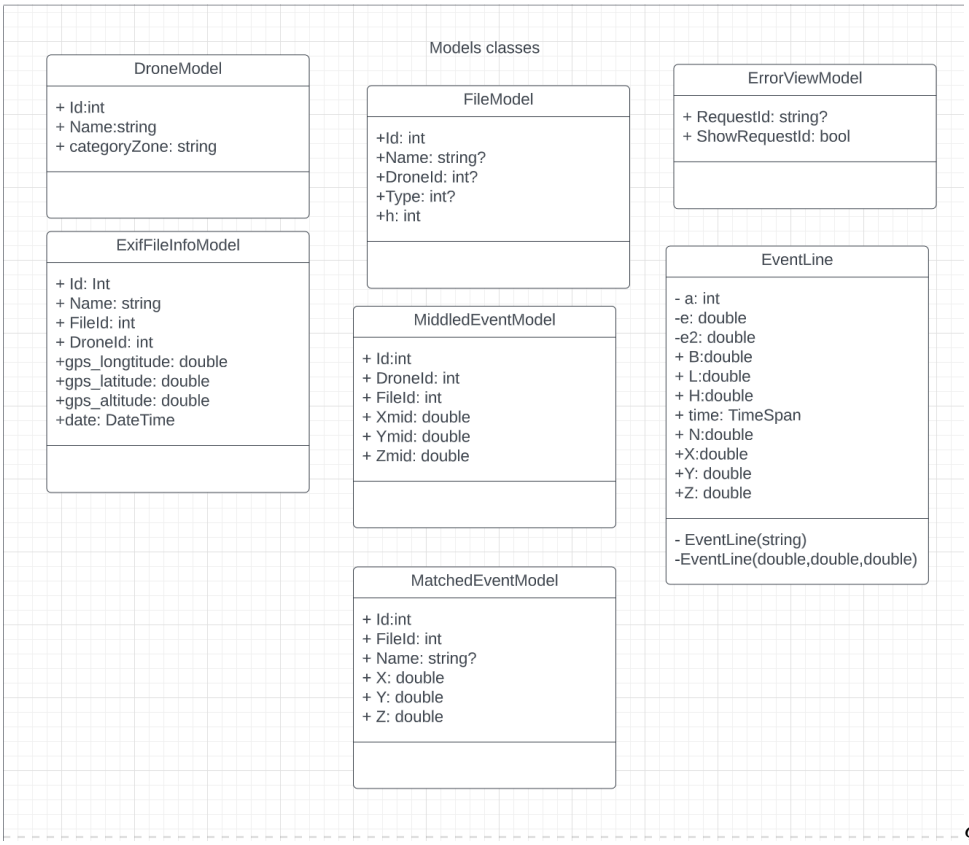
Слой за достъп до базата данни(DataAccesLayer): този слой отговаря за директните „SQL“ заявки, които работят с базата данни. В този слой стоят хранилищата, които отговарят за тези заявки, както и моделите, които се използват за таблици в базата данни. Пакетите инсталирани в този слой: „Microsoft.AspNetCore.Http.Features“, „Microsoft.AspNetCore.Mvc.ViewFeatures“,
„Microsoft.EntityFrameworkCore“,
„Microsoft.EntityFrameworkCore.SqlServer“,
„Microsoft.EntityFrameworkCore.Tools“.

Слой за бизнес логика(Bussines Layer): Идеята на този слой е да раздели бизнес логиката от презентационния слой и слоя за достъп до базата данни. В този слой стоят услугите(Services), които извършват логическите операции в приложението. Пакетите инсталирани в този слой: „ExifLibNet“, „Microsoft.AspNetCore.Http.Features“.

4.3 Организация и код на заявките към база от данни

Заявките на базата се осъществяват в различните хранилища, които биват обяснявани по-подробно в точка 4.3.1. Заявките към базата данни се извършват с помощта на „Entity framework“. “Entity Framework” е ORM технология използвана в „.NET“ приложения за работа с релационни бази данни. Технологията осигурява абстрактен слой между приложението и базата данни, позволявайки ни да работят с обекти на база данни, сякаш са обикновени обекти в приложението. С „Entity Framework“ можем да дефинираме таблици на бази данни като класове в своя код, а колони на база данни като свойства на тези класове. „Entity Framework“ обработва превода на тези класове и свойства в SQL изрази и операции с бази данни. „Entity Framework“ поддържа набор от доставчици на бази данни и предоставя набор от функции за достъп и манипулиране на данни, като LINQ, който ни позволяват да пишем подобни на SQL заявки в кода си и автоматично да проследяваме промените. Като цяло, „Entity Framework“ опростява достъпа до базата данни и манипулирането в .NET приложения, което ни помага да пишем код, който е по-ефективен, поддържаем и мащабен.

4.3.1 Организация на моделите.



Фигура-12(Диаграма на моделите)

Това е диаграма, която изобразява моделите, които присъстват в базата данни. „Entity framework“ работи с модели, като създава таблица с полетата на тези класове като колони в базата данни. Първичните и вторичните ключове както и връзките между тези класове са описани в по-горните диаграми(ER диаграмата, Клас диаграма на моделите).

4.3.2 Хранилища (Repositories).

Шаблона за хранилищата е използван в софтуерното разработване, за да се създаде разделение между бизнес логиката на приложението и слоя за достъп до данни. Той включва създаването на клас „Repository“ за всяка единица в приложението, който предоставя набор от методи за изпълнение на „CRUD“ (Create, Read, Update, Delete) операции върху тази единица.

Приложението има 3 главни хранилища: „DroneModelRepository“, “FileModelRepository”, “MiddledEventRepository”.

Това е кода от „DroneModelRepository“:

```
using DataAccessLayer.Interfaces;
```

```
using DataAccessLayer.Models;
using DataAccessLayer.Repositories.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer.Repositories
{
    public class DroneModelRepository : IDroneModelRepository
    {
        private DroneApplicationDbContext _context;

        public DroneModelRepository(IDroneApplicationDbContext
context)
        {
            this._context = context as DroneApplicationDbContext;
        }
        public async Task<bool> CreateDrone(DroneModel model)
        {
            if (model != null)
            {
                 _context.DroneModel.Add(model);
                _context.SaveChanges();
                return true;
            }
            return false;
        }

        public bool DeleteDrone(int id)
        {
            var droneModel = _context.DroneModel.Find(id);
            if (droneModel != null)
            {
                _context.DroneModel.Remove(droneModel);
                _context.SaveChanges();
                return true;
            }

            return false;
        }

        public DroneModel GetByName(string name) {
            return _context.DroneModel.FirstOrDefault(i => i.Name ==
name);
        }
```

```
    }
}
```

Конструктора в това хранилище запазва инстанцията на контекстния клас на базата данни.

Метода „CreateDrone“ създава запис на дрон в базата данни. Приема моделен клас от типа „DroneModel“.

Метода „DeleteDrone“ изтрива запис на дрон от базата данни. Приема аргумент от тип „int“ който е индекса на дрона в базата данни.

Метода „GetByName“ проверява за наличието на дрон в базата и връща моделен клас от тип „DroneModel“. Приема аргумент то тип „string“ като това е името на дрона.

Това е кода от хранилището “FileModelRepository”:

```
using DataAccessLayer.Interfaces;
using DataAccessLayer.Models;
using DataAccessLayer.Repositories.Interfaces;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer.Repositories
{
    public class FileModelRepository : IFileModelRepository
    {
        private DroneApplicationDbContext _context;

        public FileModelRepository(IDroneApplicationDbContext
context)
        {
            this._context = context as DroneApplicationDbContext;
        }
        public async Task<bool> CreateFile(FileModel model)
        {
            if (model != null)
            {
                _context.FileModel.Add(model);
                _context.SaveChanges();
                return true;
            }
            return false;
```

```
        }

        public bool DeleteFile(int id)
        {
            var fileModel = _context.FileModel.Find(id);
            if (fileModel != null)
            {
                _context.FileModel.Remove(fileModel);
                _context.SaveChanges();
                return true;
            }

            return false;
        }

        public void DeleteFileByDroneId(int id)
        {

_context.FileModel.RemoveRange(_context.FileModel.Where(i =>
i.DroneId == id));
            _context.SaveChanges();
        }

        public FileModel GetFile(int? id)
        {
            return _context.FileModel.FirstOrDefault(d => d.Id ==
id);
        }

        public IEnumerable<FileModel> GetFileByDroneId(int? id)
        {
            return _context.FileModel.Where(i => i.DroneId == id);
        }
    }
}
```

Конструктора в това хранилище запазва инстанцията на контекстния клас на базата данни.

Метода „CreateFile“ създава запис на файл в базата данни. Приема моделен клас от тип „FileModel“.

Метода „DeleteFile“ изтрива запис на файл в базата данни. Приема аргумент от тип “int” който е индекса на файла, който ще бъде изтрит от базата данни.

Метода „GetFile“ проверява дали определен файл е наличен в базата данни и ако е връща моделен клас от тип „FileModel“, който

представлява съответния файл. Приема аргумент от тип „int“, който е индекса на определения файл в базата данни.

Метода „GetFileByDroneId“ Връща тип „IEnumerable<FileModel>“, което представлява списък от файлове, чието поле „DroneId“ в базата данни съответства на подадения аргумент. Аргумента е от тип „int?“ и представлява индекса на дрона, който се съдържа в полето “DroneId” от таблицата с файлове в базата данни.

Метода „DeleteFileByDroneId“ изтрива запис на файл спрямо съдържанието на полето „DroneId“. Аргумента е от тип „int?“ и представлява индекса на дрона, който се съдържа в полето “DroneId” от таблицата с файлове в базата данни.

Това е кода от хранилището „MiddledEventRepository“:

```
using DataAccessLayer.Interfaces;
using DataAccessLayer.Models;
using DataAccessLayer.Repositories.Interfaces;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DataAccessLayer.Repositories
{
    public class MiddledEventRepository : IMiddledEventRepository
    {
        private DroneApplicationDbContext _context;

        public MiddledEventRepository(IDroneApplicationDbContext
context)
        {
            this._context = context as DroneApplicationDbContext;
        }
        public bool CreateMiddledEvent(MiddledEventModel
middledEvent)
        {
            if (middledEvent == null) {
            return false;
            }
            _context.Events.Add(middledEvent);
            _context.SaveChanges();
            return true;
        }
```

```
        public IEnumerable<MiddledEventModel> GetEvents(int DroneId)
        {
            throw new NotImplementedException();
        }

        public IEnumerable<MiddledEventModel> getEventsByFileId(int?
id) {
            return _context.Events.Where(i => i.FileId == id);
        }

        public void DeleteByFileId(int id) {
            _context.Events.RemoveRange(_context.Events.Where(i =>
i.FileId == id));
            _context.SaveChanges();
        }

        public MiddledEventModel GetEventById(int id)
        {
            throw new NotImplementedException();
        }

        public void DeleteByDroneId(int id)
        {
            _context.Events.RemoveRange(_context.Events.Where(i =>
i.DroneId == id));
            _context.SaveChanges();
        }

        public bool ExistByFileId(int id) {
            if (_context.ExifInfoModels.Any(i => i.FileId == id))
            {
                return true;
            }
            return false;
        }

        public bool CreateExifInfoModel(ExifInfoModel model) {
            if (model == null)
            {
                return false;
            }
            _context.ExifInfoModels.Add(model);
            _context.SaveChanges();
            return true;
        }
```

```
        public IEnumerable<ExifInfoModel>
getExifInfoModelsByFileId(int? id)
        {
            return _context.ExifInfoModels.Where(i => i.FileId ==
id);
        }

        public void DeleteExifInfoModelByFileID(int id) {

_context.ExifInfoModels.RemoveRange(_context.ExifInfoModels.Where(i
=> i.FileId == id));
            _context.SaveChanges();
        }

        public void DeleteExifInfoModelByDroneID(int id) {

_context.ExifInfoModels.RemoveRange(_context.ExifInfoModels.Where(i
=> i.DroneId == id));
            _context.SaveChanges();
        }

        public void CreateMatchedModel(MatchedEventModel model) {
            _context.MatchedEventModels.Add(model);
            _context.SaveChanges();
        }


    }
}
```

Конструктора в това хранилище запазва инстанцията на контекстния клас на базата данни.

Метода „CreateMiddledEvent“ създава запис от тип „MiddledEventModel“ в базата данни. Приема аргумент от тип „MiddledEventModel“ който представлява модела, който ще бъде записан в базата.

Метода „GetEventById“ връща тип “MiddledEventModel” , който представлява запис от базата, чийто индекс е равен на аргумента, който е подаден. Аргумента, който е подаден е от тип „int“ и представлява индекса на записа, който търсим.

Метода „GetEvents“ връща тип „IEnumerable<MiddledEventModel>“ , който връща списък от записи от базата, чието поле „DroneId“ е равно на аргумента, който е подаден. Аргумента е от тип „int“ и представлява индекса , който ще бъде търсен.

Метода „DeleteByDroneId“ изтрива запис от базата, чието поле „DroneId“ е равно на аргумента, който е подаден. Аргумента е от тип „int“ и представлява индекса, който ще бъде търсен.

Метода „getEventsByFileId“ връща тип „IEnumerable<MiddledEventModel>“, който представлява списък от точки на полет, чиито поле „FileId“ е равно на подадения аргумент. Аргумента е от тип „int?“ и представлява индекса, който ще бъде търсен.

Метода „DeleteByFileId“ изтрива запис от базата, чието поле „FileId“ е равно на аргумента, който е подаден. Аргумента е от тип „int“ и представлява индекса, който ще бъде търсен.

Метода „ExistByFileId“ връща тип „bool“ спрямо наличието на запис, чиито поле „FileId“ е равно на аргумента. Аргумента е от тип „int“ и представлява индекса, който ще бъде търсен.

Метода „CreateExifInfoModel“ създава запис от тип „ExifInfoModel“ в базата данни. Аргумента, който е подаден е моделен клас „ExifInfoModel“ , който ще бъде създаден в базата данни.

Метода „getExifInfoModelsByFileId“ връща тип „IEnumerable<ExifInfoModel>“, който представлява списък от записи с метаданни, чиито поле „FileId“ е равно на подадения аргумент. Аргумента е от тип „int“ и представлява индекса, който ще бъде търсен.

Метода „DeleteExifInfoModelByFileId“ изтрива всички записи от метаданни в базата, чието поле „FileId“ е равно на подадения аргумент. Аргумента е от тип „int“ и представлява индекса, който ще бъде търсен.

Метода „DeleteExifInfoModelByDroneId“ изтрива всички записи от метаданни в базата, чието поле „DroneId“ е равно на подадения аргумент. Аргумента е от тип „int“ и представлява индекса, който ще бъде търсен.

Метода „CreateMatchedModel“ създава запис от тип „MatchedEventModel“ в базата данни. Приема аргумент от тип „MatchedEventModel“ който представлява модела, който ще бъде записан в базата.

4.4 Наличие на потребителски интерфейс (конзолен, графичен, уеб)

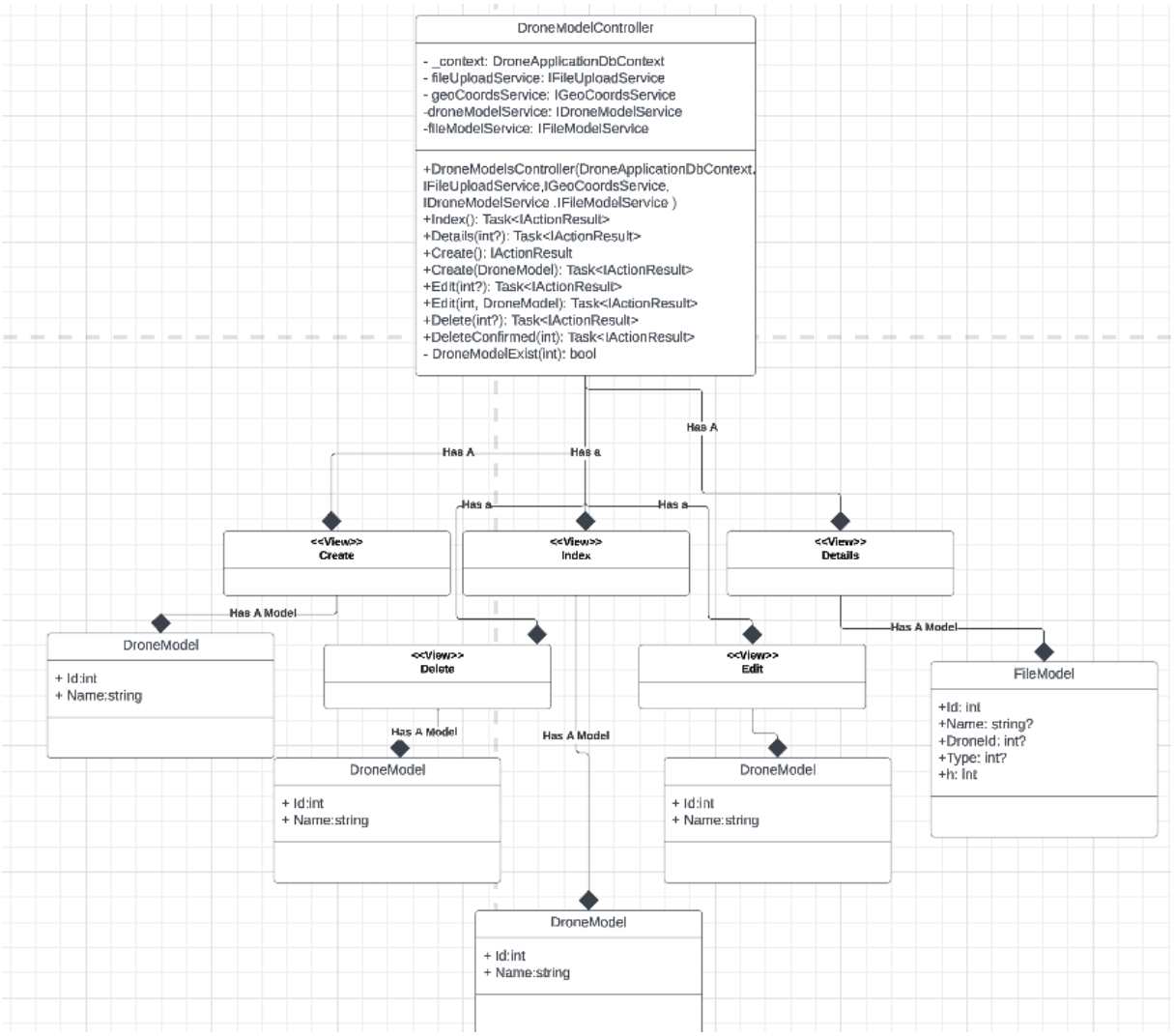
Интерфейсът е разработен с използването на HTML, CSS и JavaScript.

Основните функционалности на интерфейса включват:

- Възможност за добавяне на дронове чрез имена и категории на дрон.
- Възможност за прикачване на полети чрез текстови файлове.
- Възможност за прикачване на изображения от полети чрез “.jpg” и “.png” файлове.
- Възможност за изобразяване на полетите чрез таблици.
- Възможност за изобразяване на точките от полетите чрез таблици.
- Възможност за изобразяване на метаданните от снимките чрез таблици.
- Възможност за свързване на метаданните от снимките и точките от полетите

В приложението има 5 контролера: „DroneModelsController“, “ExifInfoModelsController”, “HomeController”, “MatchController” и “MiddlingController”.

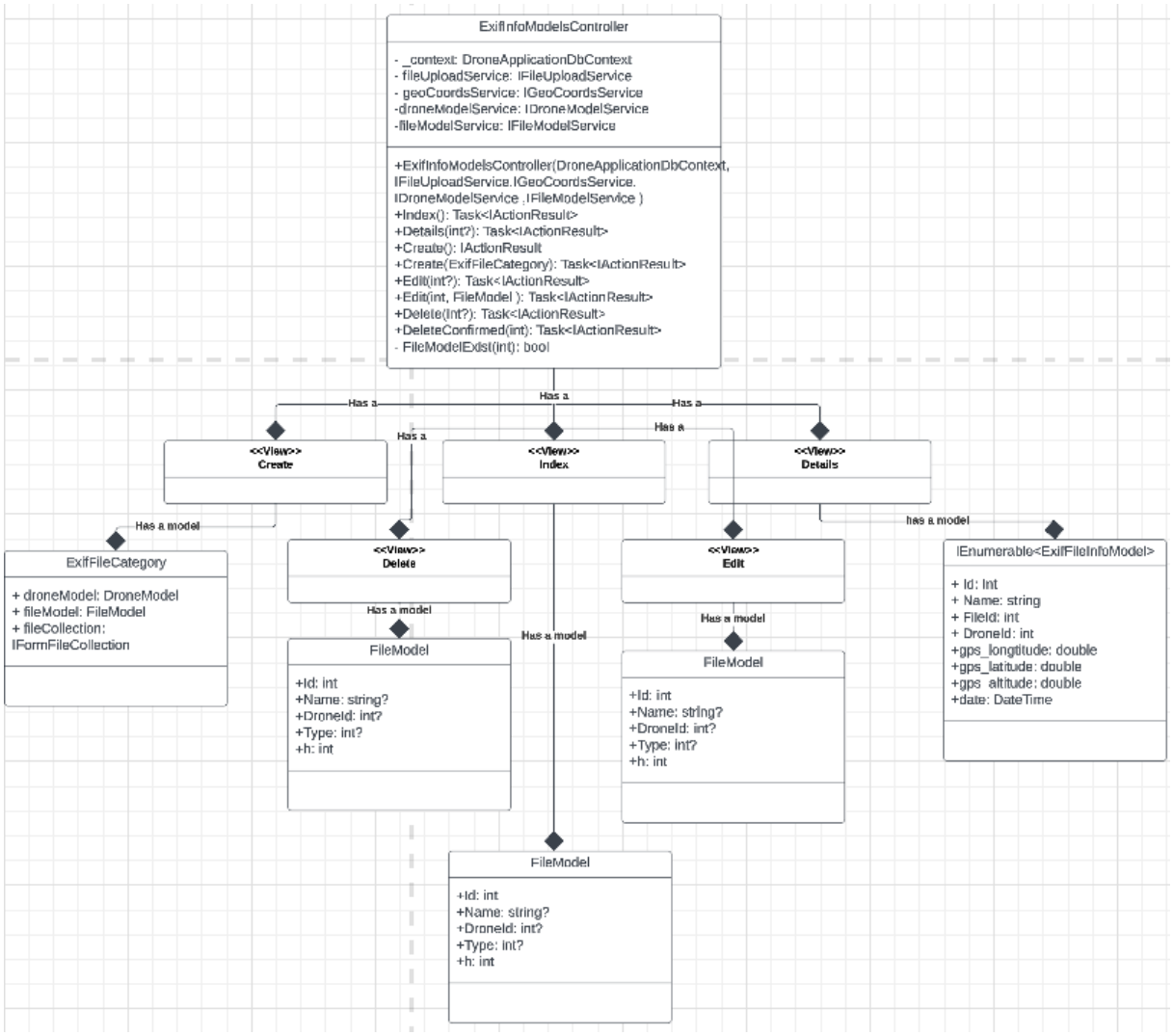
“DroneModelsController”:



Фигура-13(Диаграма на Изгледите в папка ”DroneModels”)

“DroneModelsController” работи с страницата за добавяне на дрон и всичките му изгледни подразделения.

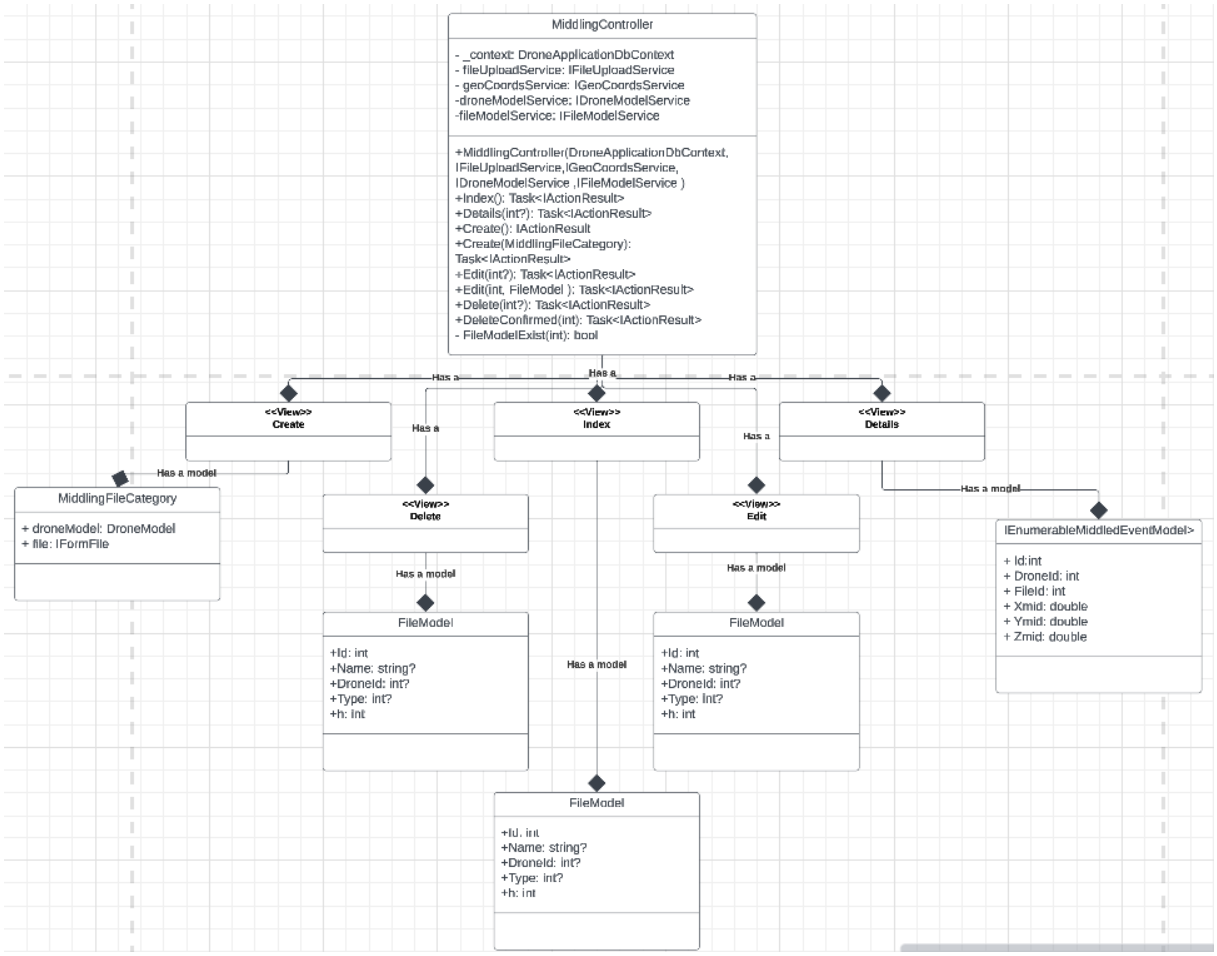
“ExifInfoModelsController”:



Фигура-14(Диаграма на изгледите в папка ”ExifInfoModels”)

“ExifInfoModelsController” работи със извличането на метаданни и всичките му изгледни подразделения.

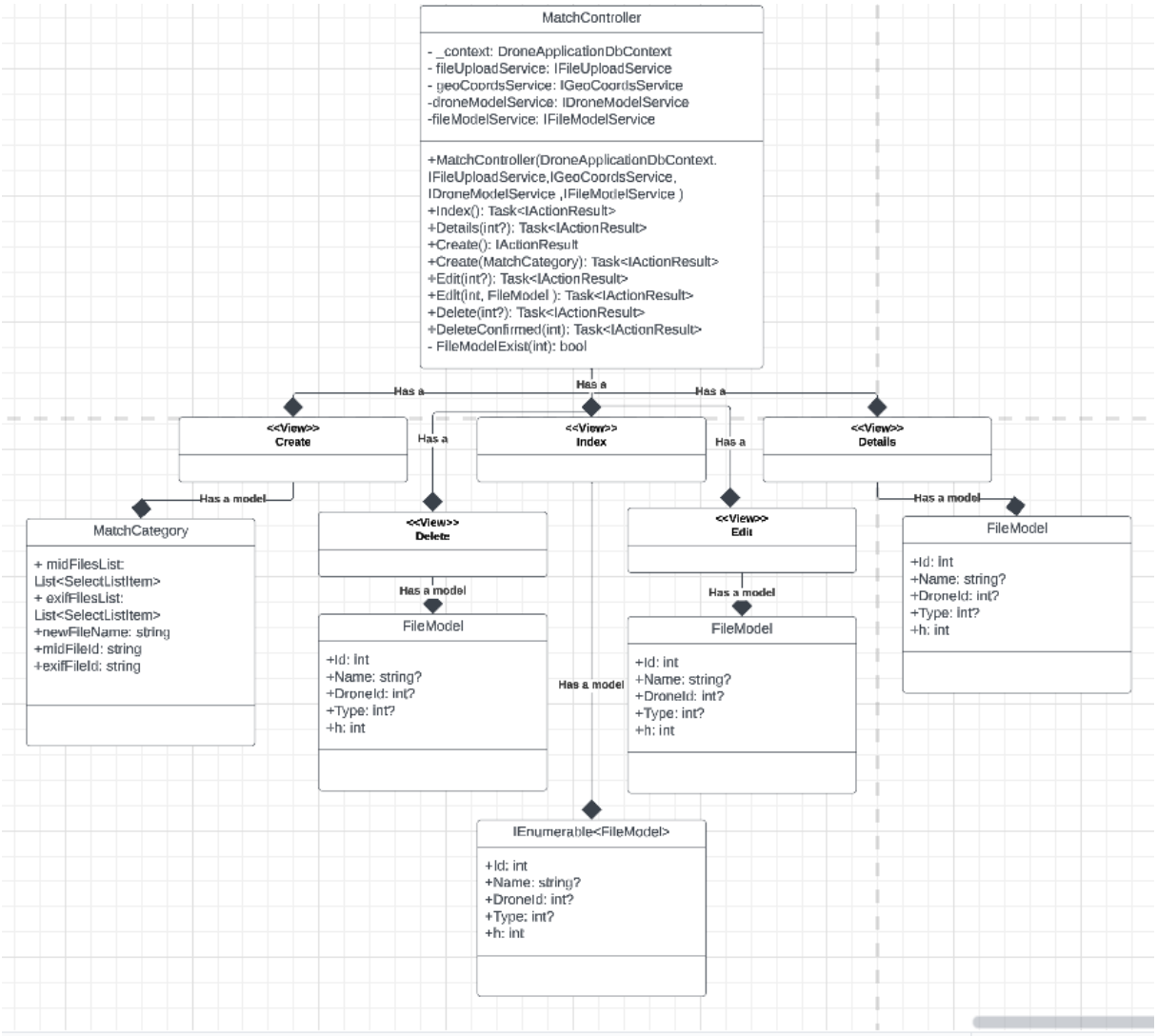
“MiddlingController”:



Фигура-15(Диаграма на изгледите в папка ”Middling”)

“MiddlingController” работи с страницата за добавяне на полети от файл и всичките й изгледни подразделения.

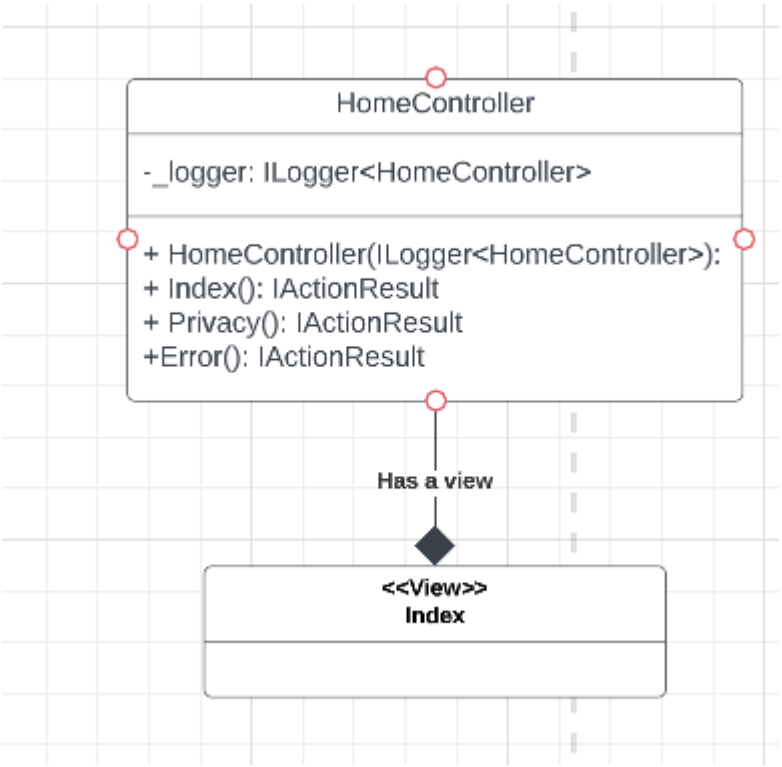
"MatchController":



Фигура-16(Диаграма на изгледите в папка "Match")

„MatchController“ работи със страницата за свързване на снимки с координати и всичките и изгледни подразделения.

“HomeController”:



Фигура-17(Диаграма на изгледите в папка “Home”)

„HomeController“ работи със началната страница.

5 Ефективност и бързодействие на решението

Изчислената ефективност на бизнес слоя е:

„GeoCoordsService“ =O(N^2).

„FileModelService“ = O(N).

„DroneModelService“ = O(N).

Изчислената ефективност на слоя за достъп до базата данни е:

„DroneModelRepository“ = O(N)

„FileModelRepository“ = O(N)

„MiddledEventRepository“ = O(N).

Изчислената ефективност на контролния слой е:

„DroneModelsController“ = O(N)

“ExifInfoModelsController” = O(N^2)

“HomeController” = O(1)

“MatchController” = O(N)

“MiddlingController” = O(N)

6 Тестване

Тестването е съществена част от разработката на софтуер и може да помогне да се гарантира, че проектът отговаря на очакваните стандарти за качество. За да тестваме цялостно проекта, можем да използваме различни видове тестване, включително функционално тестване, тестване на производителността и тестване за сигурност.

Примерни xUnit използвани в приложението:

```
using BussinesLayer.Interfaces;

using DataAccessLayer;

using DataAccessLayer.Models;

using DataAccessLayer.Repositories;

using DroneApplication.FileUploadService;

using FakeItEasy;

using FluentAssertions;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;
```

```
namespace DroneApplication.Test.ControllerTests
{

    public class DroneModelRepositoryTests
    {

        private async Task<DroneApplicationDbContext>
GetDbContext() {
            var options = new
DbContextOptionsBuilder<DroneApplicationDbContext>()
                .UseInMemoryDatabase(databaseName:
Guid.NewGuid().ToString())
                .Options;
            var databaseContext = new
DroneApplicationDbContext(options);
            databaseContext.Database.EnsureCreated();
            if (await
databaseContext.DroneModel.CountAsync() < 0) {
                for (int i = 0; i < 10; i++)
                {

                    databaseContext.DroneModel.Add(
                        new DroneModel()
```

```
                                {
                                    Id = 1,
                                    Name = "stoqn"

                                });
                            await
databaseContext.SaveChangesAsync();
                        }
                    }
                    return databaseContext;
                }
                [Fact]
                public async void
CreateDrone_IsNotNull_ReturnTrue()
                {
                    //Arrange
                    var droneModel = new DroneModel()
                    {
                        Id = 1,
                        Name = "stoqn"

                    };
                    var _context = await GetDbContext();
```

```
            var droneRepository = new
DroneModelRepository(_context);

            //Act

            var result =
droneRepository.CreateDrone(droneModel);

            result.Should().BeTrue();

        }

        [Fact]

        public async void
CreateDrone_IsNull_ReturnFalse()

        {

            //Arrange

            var droneModel = new DroneModel();

            droneModel = null;



            var _context = await GetDbContext();

            var droneRepository = new
DroneModelRepository(_context);

            //Act

            var result =
droneRepository.CreateDrone(droneModel);

            result.Should().BeFalse();

        }

        [Fact]
```

```
            public async void
DeleteDrone_IdDosNotExist_ReturnFalse()
            {
                //Arrange
                var droneModel = new DroneModel()
                {
                    Id = 1,
                    Name = "stoqn"

                };

                var _context = await GetDbContext();
                var droneRepository = new
DroneModelRepository(_context);
                droneRepository.CreateDrone(droneModel);
                //Act
                var result = droneRepository.DeleteDrone(-
3);
                result.Should().BeFalse();
            }
            [Fact]
            public async void
DeleteDrone_IdExist_ReturnTrue()
```

```
        {
            //Arrange
            var droneModel = new DroneModel()
            {
                Id = 1,
                Name = "stoqn"

            };

            var _context = await GetDbContext();
            var droneRepository = new
DroneModelRepository(_context);
            droneRepository.CreateDrone(droneModel);
            //Act
            var result =
droneRepository.DeleteDrone(1);
            result.Should().BeTrue();
        }
    }
}
```

В този клас има няколко метода за тестване на различни функции в “DroneModelRepository”. Веки метод има основни 3 стъпки : подготовка(Assert), действие(Act), и проверка(Assert).

Понеже работи с контекст от базата данни, за да можем да я тестваме без да правим реална инстанция използваме така наречената (MemoryDatabase). Тя се създава в метода GetDbContext().

Метода “CreateDrone_IsNotNull_ReturnTrue()”: прави проверка на коректното функциониране на метода “CreateDrone” в хранилището “DroneModelRepository”. Този метод проверява дали “CreateDrone” работи коректно в случай че аргумента, който се подава няма стойност “null”.

Метода “CreateDrone_IsNull_ReturnFalse()”: прави проверка на коректното функциониране на метода “CreateDrone” в хранилището “DroneModelRepository”. Този метод проверява дали “CreateDrone” работи коректно в случай че аргумента, който се подава има стойност “null”.

Метода “DeleteDrone_IdDoesNotExist_ReturnFalse()”: прави проверка на коректното функциониране на метода “DeleteDrone” в хранилището “DroneModelRepository”. Този метод проверява дали “DeleteDrone” работи коректно в случай в базата няма дрон с поле “Id” което да е равно на аргумента, който е подаден.

Метода “DeleteDrone_IdExist_ReturnTrue()”: прави проверка на коректното функциониране на метода “DeleteDrone” в хранилището “DroneModelRepository”. Този метод проверява дали “DeleteDrone” работи коректно в случай в базата има дрон с поле “Id” което да е равно на аргумента, който е подаден.

7 Заключение и възможно бъдещо развитие

В заключение, работата по проекта за уебсайт е успешна и е завършена в срок. Уебсайтът е разработен с използването на C# .NET, HTML, CSS, JavaScript, Bootstrap и Microsoft SSMS. Уебсайтът е тестван с различни методи, за да се гарантира правилната му работа и функционалност.

Предимствата на използваните технологии включват:

C# е един от най-широко използваните езици за програмиране и предлага голяма гъвкавост и мощност в програмирането на уебсайтове и приложения.

HTML е основен език за създаване на уебстраници и осигурява добра структура и семантика на уебсайта.

CSS осигурява добра визуална привлекателност и форматиране на уебсайта.

JavaScript позволява добавянето на интерактивност и динамика към уебсайта.

Bootstrap предоставя готови компоненти и стилове за уебсайтове, които улесняват работата на програмистите и оптимизират времето за разработка.

Microsoft SSMS е мощна програма за управление на бази данни, която позволява лесното управление на бази данни и манипулиране на данни.

Ограниченията на използваните технологии включват:

Необходимостта от добра практика и опит за създаване на по-сложни уебсайтове и приложения.

Необходимостта от поддръжка на определени версии на различните браузъри.

В геодезията , технологията за която настоящото приложение е разработена , има приложение почти във всички сфери на стопанския живот / кадастър , инвестиционно проектиране , устройствено планиране , картография , 3d модели , облаци от точки и много други /.

8 Инструкции за употреба

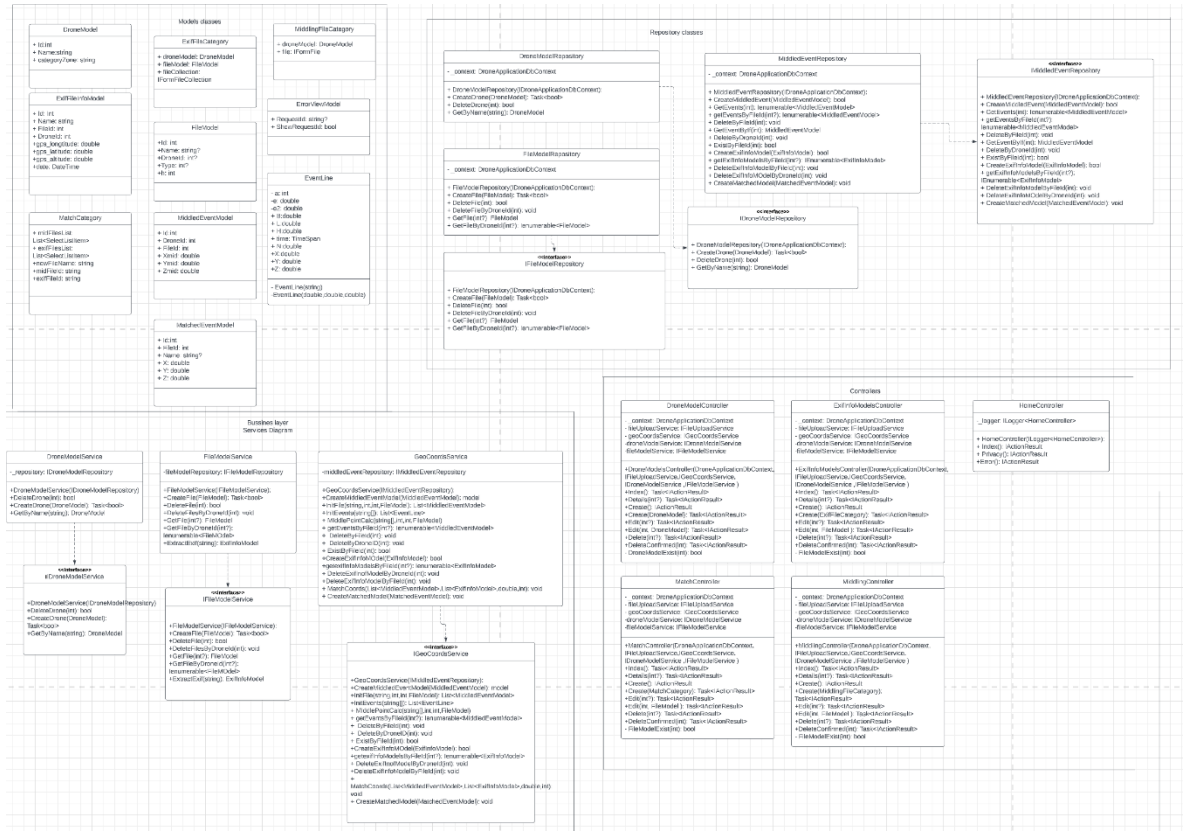
Файловете прикачени в папката “DemoTestFiles”, която се намира както в Github хранилището на проекта, така и в флашката, която е предадена могат да ви послужат при тестване на приложението. Текстовия файл, който се намира в папката представлява данните за полета. Той може да се качи като отворим страницата за качване на полети и след това натиснем бутона създай. В полето за прикачване на файл може да сложите текстовия файл. В под папката на с име “DemoTestFilesImage” може да намерите снимките от съответния полет. Те се прикачват като отворим страницата за извличане на метаданни, след което натиснем създай. В полето за прикачване на файлове можем да качим снимките. Имайте предвид че за да бъдат свързани всички координати, трябва да качите всички налични снимки в папката. За да прикачите повече от 40 мегабайта снимки са нужни администраторски права, за да може да се промени променливата „UploadReadAheadSize“. Засега тази променлва е оставена на стойността си по подразбиране която е около 40 Мегабайта.

9 Използвани литературни източници и Уеб сайтове

1. Речник с терминология английски - български. Извлечено от iate.europa.eu: https://iate.europa.eu/home
2. Сайт за създаване на диаграми. Извлечено от lucidchart: https://www.lucidchart.com/pages/
3. Страницата на Moodle. Извлечено от Moodle: https://moodle.org/
4. Microsoft. ASP.NET Documentation: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0
5. Microsoft. .NET Documentation: https://learn.microsoft.com/en-us/dotnet/

10 Приложения

Приложение 1: Клас Диаграми


Models classes
DroneModel
+ Id:int
+ Name:string
+ category?one: string
ExifFilesYoModel
+ Id: int
+ Name: string
+ Fileid: int
+ Droneid: int
+gps_longitude: double
+gps_latitude: double
+gps_altitude: double
+date: DateTime
MatchCategory
+ matchFiles: int
List<SelectListItem>
+ exifFiles:int
List<SelectListItem>
+newFileName: string
+matFileId: string
+exifFileId: string
FileFilesCategory
+ droneModel: DroneModel
+ fileModel: FileModel
+ fileCollection
IFormFileCollection
FileModel
+Id: int
+Name: string?
+Droneid: int?
+Type: int?
+fc: int
MiddledEventModel
+ id: int
+ Droneid: int
+ Fileid: int
+ Xmid: double
+ Ymid: double
+ Zmid: double
MatchedEventModel
+ id: int
+ Fileid: int
+ Name: string?
+ X: double
+ Y: double
+ Z: double
MiddlingFileCategory
+ droneModel: DroneModel
+ file: IFormFile
ErrorViewModel
+ RequestId: string?
+ ShowRequestId: bool
EventLine
- a: int
-b: double
-c2: double
+ lb:double
+ L:double
+ f:double
+ time: TimeSpan
+ N:double
+X: double
+Y: double
+Z: double
- Event(he)string)
-EventLine(double,double,double)
Repository Classes
DroneModelRepository
- context: DroneApplicationDbContext
- DroneModelRepository(DroneApplicationDbContext)
+ CreateDrone(DroneModel): Task<bool>
+ DeleteDrone(int): bool
+ GetDyName(string): DroneModel
FileModelRepository
- context: DroneApplicationDbContext
- FileYoceRepository(DroneApplicationDbContext)
+ CreateFileFileModels): Task<bool>
+ DeleteFile(int): bool
+ DeleteFileByDroneId(int): void
+ GetFile(int?): FileModel
+ GetFileByDroneId(int?): Ienumerable<FileYocel>
«interface»
IFileModelRepository
- FileYocelRepository(DroneApplicationDbContext);
+ CreateFileFileModel): Task<bool>
+ DeleteFile(int): bool
+ DeleteFileByDroneId(int): void
+ GetFile(int?): FileModel
+ GetFileByDroneId(int?): Ienumerable<FileYocel>
MiddledEventRepostory
- _context: DroneApplicationDbContext
+ MiddledEventRepository(DroneApplicationDbContext);
+ CreateMiddledEvent(MiddledEventYocel): bool
+ GetEventsById(int): Ienumerable<MiddledEventYocel>
+ getEventsByField(int?): Ienumerable<MiddledEventYocel>
+ DeleteByField(int): void
+ GetEventById(int): MiddledEventYocel
+ DeleteByDroneId(int): void
+ ExistById(int): bool
+ getExitInfoModelsByFile(int?): Ienumerable<ExitInfoModel>
+ DeleteExitInfoModelByFile(int?): void
+ DeleteExitInfoModelByDroneId(int): void
+ CreateMatchedModel(MatchedEventModel): void
«interface»
IDroneYocleRepository
+ DroneModelRepository(DroneApplicationDbContext);
+ CreateDrone(DroneModel): Task<bool>
+ DeleteDrone(int): bool
+ GetDyName(string): DroneModel
«interface»
IMiddledEventRepository
+ MiddledEventRepository(DroneApplicationDbContext);
+ CreateMiddledEvent(MiddledEventModel): bool
+ GetEventsById(int): Ienumerable<MiddledEventYocel>
+ getEventsByField(int?): Ienumerable<MiddledEventYocel>
+ DeleteByField(int): void
+ GetEventById(int): MiddledEventYocel
+ DeleteByDroneId(int): void
+ ExistById(int): bool
+ CreateExitInfoModel(ExitInfoModel): bool
+ getExitInfoModelsByFile(int?): Ienumerable<ExitInfoModel>
+ DeleteExitInfoModelByFile(int?): void
+ DeleteExitInfoModelByDroneId(int): void
+ CreateMatchedModel(MatchedEventModel): void
Controllers
DroneModelController
- context: DroneApplicationDbContext
- fileUploadService: IFileUploadService
- geoCoordsService: GeoCoordsService
dronesYocleService: IDroneYocleService
fileModelService: IFileModelService
+DroneModelsController(DroneApplicationDbContext, IFileUploadService,GeoCoordsService, IDroneModelService ,IFileModelService)
+Index(): Task<IActionResult>
+Details(int?): Task<IActionResult>
+Create(): IActionResult
+Create(DroneModel): Task<IActionResult>
+Edit(int?): Task<IActionResult>
+Edit(int, DroneModel): Task<IActionResult>
+Delete(int?): Task<IActionResult>
+DeleteConfirmed(int): Task<IActionResult>
- DroneModelExist(int): bool
ExitFileModelsController
- context: DroneApplicationDbContext
- fileUploadService: IFileUploadService
- geoCoordsService: GeoCoordsService
dronesYocleService: IDroneYocleService
fileModelService: IFileModelService
+ExitFileModelsController(DroneApplicationDbContext, IFileUploadService,GeoCoordsService, IDroneModelService ,IFileModelService)
+Index(): Task<IActionResult>
+Details(int?): Task<IActionResult>
+Create(): IActionResult
+Create(ExitFileCategory): Task<IActionResult>
+Edit(int?): Task<IActionResult>
+Edit(int, FileModel): Task<IActionResult>
+Delete(int?): Task<IActionResult>
+DeleteConfirmed(int): Task<IActionResult>
- FileModelExist(int): bool
HomeController
_logger: ILogger<HomeController>
+ HomeController(Logger<HomeController>)
+ Index(): IActionResult
+ Privacy(): IActionResult
+Error(): IActionResult
MatchController
- context: DroneApplicationDbContext
- fileUploadService: IFileUploadService
- geoCoordsService: GeoCoordsService
-droneYocleService: IDroneYocleService
-fileModelService: IFileModelService
+MatchController(DroneApplicationDbContext, IFileUploadService,GeoCoordsService, IDroneModelService ,IFileModelService)
+Index(): Task<IActionResult>
+Details(int?): Task<IActionResult>
+Create(): IActionResult
+CreateMatchCategory(): Task<IActionResult>
+Edit(int?): Task<IActionResult>
+Edit(int ,IFileModel): Task<IActionResult>
+Delete(int?): Task<IActionResult>
+DeleteConfirmed(int): Task<IActionResult>
- FileModelExist(int): bool
MiddlingController
- context: DroneApplicationDbContext
- fileUploadService: IFileUploadService
- geoCoordsService: GeoCoordsService
-droneYocleService: IDroneYocleService
-fileModelService: IFileModelService
+MiddlingController(DroneApplicationDbContext, IFileUploadService,GeoCoordsService, IDroneModelService ,IFileModelService)
+Index(): Task<IActionResult>
+Details(int?): Task<IActionResult>
+Create(): IActionResult
+Create(MiddlingFileCategory); Task<IActionResult>
+Edit(int?): Task<IActionResult>
+Edit(int ,IFileModel): Task<IActionResult>
+Delete(int?): Task<IActionResult>
+DeleteConfirmed(int): Task<IActionResult>
- FileModelExist(int): bool
Runshes layer
Services Diagrams
DroneModelService
_repository: IDroneModelRepository
+DroneModelService(IDroneModelRepository)
+DeleteDrone(int): bool
+CreateDrone(DroneModel): Task<bool>
+GetByName(string): DroneModel
FileModelService
-fileModelRepository: IFileModelRepository
-FileModelService(IFileModelService);
~CreateFile(IFileModel): Task<bool>
~DeleteFile(int): bool
~DeleteFileByDroneId(int): void
~GetFile(int?): FileModel
~GetFileByDroneId(int?): Ienumerable<FileModel>
+constructor(string): ExitInfoModel
GeoCoordsService
-middledEventRepository: IMiddledEventRepository
+GeoCoordsService(IMiddledEventRepository);
+CreateMiddledEventModel(MiddledEventModel): model
++nitFileString,int,int,FileModel) : List<MiddledEventModel>
+initEvents(string[]): List<EventLine>
+ MiddlePointCalc(string[],int,int,FileModel)
+ getEventsByFileId(int?): Ienumerable<MiddledEventModel>
+ DeleteById(int): void
+ DeleteByDroneId(int): void
+ ExistById(int): bool
+ DeleteExitInfoModelByFile(int?): void
+geoextfileModelsByFile(int?): Ienumerable<ExitInfoModel>
+ UnresolvedInfoModelsByDrone(int?): void
+ DeleteExitInfoModelByFile(int): void
+ MatchCoords(ist<MiddledEventModel>,List<ExitInfoModel>,double,int): void
+ CreateUnmatchedModel(MatchedEventModel): void
«interface»
IDroneModelService
+DroneModelService(IDroneModelRepository)
+DeleteDrone(int): bool
+CreateDrone(DroneModel): Task<bool>
+GetByName(string): DroneModel
«interface»
IFileModelService
+FileModelService(FileModelService)
+CreateFile(FileYocel): Task<bool>
+DeleteFile(int): bool
+DeleteFileByDroneId(int): void
+GetFile(int?): FileModel
+GetFileByDrone.d(int?): Ienumerable<IexModel>
+ExtractExif(string): ExifInfoModel
«interface»
IGeoCoordsService
+GeoCoordsService(IMiddledEventRepository);
+CreateMiddledEventModel(MiddledEventYocel): model
+initFileString,int,int,FileModel) :List<MiddledEventModel>
+initEvents(string[]): List<EventLine>
+ MiddlePointCalc(string[],int,int,FileModel)
+ getEventsByFileId(int?): Ienumerable<MiddledEventModel>
+ DeleteById(int): void
+ DeleteByDroneId(int): void
+ ExistById(int): bool
+CreateExitInfoModel(ExitInfoModel): bool
+getexifFileModelsByFile(int?): Ienumerable<ExitInfoModel>
+ DeleteExitInfoModelByDroneId(int): void
+DeleteExitInfoModelByFileId(int): void
+
MatchCoords(ist<MiddledEventModel>,List<ExitInfoModel>,double,int)
void
+ CreateMatchedModel(MatchedEventModel): void