ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО КОМПЮТЪРНО ПРОГРАМИРАНЕ И ИНОВАЦИИ
бул. "Захари Стоянов", жк Меден рудник, 8009 Бургас, office@codingburgas.bg, codingburgas.bg

ДИПЛОМЕН ПРОЕКТ

ЗА ДЪРЖАВЕН ЗРЕЛОСТЕН ИЗПИТ

по професия код 481030 „Приложен програмист“

специалност код 4810301 Приложно програмиране“

ТЕМА: „ПРИЛОЖЕНИЕ ЗА УПРАВЛЕНИЕ НА ЗАДАЧИ И РАБОТА“

Автор: Йоанна Владимирова Симеонова, XII „В“

Ръководител: Виктор Стоев

Бургас

Съдържание

1 Увод ... 3
2 Цели и обхват на софтуерното приложение ... 4
3 Изследване по темата ... 5
3.1 Теоретично изследване: Изследване на методологии за управление на проекти като Agile и Scrum ... 5
3.2 Роля на технологиите: Използване на облачни технологии за синхронизация и съвместна работа в реално време ... 7
4 Анализ на решението ... 9
4.1 Потребителски изисквания и работен процес ... 9
4.2 Примерен потребителски интерфейс ... 9
4.3 Диаграми на анализа ... 12
4.3.1 Диаграма на случаи на употреба ... 12
4.4 Модел на съдържанието / данните ... 13
4.4.1 Диаграма на отношенията между моделите ... 13
4.4.2 Нормализация ... 14
5 Дизайн ... 15
5.1 Реализация на архитектурата на приложението ... 17
5.2 Описание на слоевете, предназначението им, библиотеки и методи включени в съответния слой ... 17
5.2.1 Слой за достъп до данните ... 18
5.2.2 Слой на бизнес логиката ... 20
5.2.3 Презентационен слой ... 23
5.3 Организация и код на заявките към база от данни ... 25
5.4 Наличие на потребителски интерфейс ... 26
6 Ефективност и бързодействие на решението ... 27
7 Тестване ... 29
8 Заключение и възможно бъдещо развитие ... 30
9 Използвани литературни източници и Уеб сайтове ... 31
10 Приложения ... 33

1 Увод

В съвременната динамична работна среда управлението на задачи и проекти е от съществено значение за ефективността и успеха на организациите. С нарастването на дистанционната работа и необходимостта от по-добра координация между екипите, използването на софтуерни решения за управление на задачи става все по-актуално. Това приложение за управление на задачи и работа е разработено с цел да предостави интуитивен и ефективен инструмент за проследяване на задачите, подобряване на организацията и повишаване на продуктивността.

Проектът решава проблема с липсата на централизирана система за управление на задачи, което често води до загуба на време, неефективна комуникация и пропуски в изпълнението на важни задачи. В контекста на съвременните работни процеси, особено в екипите, които работят дистанционно или в хибридна среда, необходимостта от ефективен инструмент за управление на задачи е критична.

За моделиране на архитектурата на приложението е използван езикът UML (Unified Modeling Language) в комбинация със средата за моделиране Lucidchart. Чрез използването на UML диаграми, като диаграми на случаи на употреба, диаграми на класове и диаграми на последователност, е структурирана ясна представа за взаимодействието между различните компоненти на системата. Този подход осигурява лесна разбираемост и поддръжка на проекта.

Останалата част от документацията е структурирана, както следва:

1. Цели и обхват на софтуерното приложение.
2. Анализ на решението:
    a. Потребителски изисквания и работен процес.
    b. Примерен потребителски интерфейс.
    c. Диаграми на анализа.
    d. Модел на съдържанието / данните.
3. Изследване по темата:
    a. Теоретично изследване
    b. Роля на технологиите
4. Дизайн:
    a. Реализация на архитектурата на приложението.

   b. Описание на слоевете, предназначението им, библиотеки и методи, включени в съответния слой.
   c. Организация и код на заявките към база от данни.
   d. Наличие на потребителски интерфейс.
5. Ефективност и бързодействие на решението.
6. Тестване.
7. Заключение и възможно бъдещо развитие.
8. Използвани литературни източници и уеб сайтове.
9. Приложения.

Този документ предоставя цялостен поглед върху процеса на разработка на приложението и неговата роля в подобряването на управлението на задачи и работа.

2 Цели и обхват на софтуерното приложение

Идеята за създаване на софтуерното приложение произтича от необходимостта от централизирана платформа за управление на задачи и работа в различни екипни и бизнес среди. Приложението има за цел да подпомогне потребителите в организирането, проследяването и изпълнението на задачи по ефективен и структуриран начин.

При разработката на приложението са използвани методологиите Agile и Scrum. Agile осигурява гъвкавост в процеса на разработка, като се фокусира върху адаптивността и непрекъснатото подобряване на софтуера въз основа на обратната връзка от потребителите. Scrum, като конкретна методология в рамките на Agile, е използвана за управление на екипната работа чрез разделяне на проекта на кратки, управляеми спринтове с ясно дефинирани задачи и цели. Това позволява бързо идентифициране и решаване на проблеми, както и ефективно разпределение на ресурсите в екипа.

Обхватът на потребителите включва индивидуални потребители, малки и средни екипи, които се нуждаят от систематизирано управление на работния процес. Приложението поддържа следните основни дейности:

- Създаване и управление на задачи.
- Проследяване на напредъка по задачи и проекти.
- Разпределение на задачите между членовете на екипа.

- Настройка на крайни срокове.
- Добавяне на етикети към задачи.
- Канбан дъска за всеки проект.
- Календар за всеки проект.
- Изтегляне на отчети за всеки проект.

На база на обхвата и функционалността, основните цели на приложението са:

- Осигуряване на интуитивен и удобен за ползване интерфейс.
- Подобряване на комуникацията и координацията в екипите.
- Осигуряване на надеждно съхранение и управление на данните.

3 Изследване по темата

3.1 Теоретично изследване: Изследване на методологии за управление на проекти като Agile и Scrum

Управлението на проекти е основен аспект за успешното реализиране на бизнес стратегии, особено в динамични индустрии като софтуерната разработка. Като най-разпространените методологии за управление на проекти се открояват Agile и Scrum, които предлагат адаптивност и ефективност в процеса на разработка. Agile е философия, която залага на итеративен подход към изпълнението на проекти, като основен фокус е поставен върху взаимодействието между членовете на екипа, бързата реакция при промени и непрекъснатото усъвършенстване на продукта. Тези принципи са формулирани в Манифеста за Agile разработка на софтуер (2001), който определя четири основни ценности: предпочитане на хората и взаимодействията пред процесите и инструментите, работещия софтуер пред документацията, сътрудничеството с клиента пред договорните споразумения и адаптивността пред стриктното следване на планове. Тези ценности позволяват на екипите да разработват продукти, които са гъвкави спрямо изискванията на пазара и нуждите на клиентите [1].

Scrum, от своя страна, е конкретна рамка в Agile, която структурира работния процес в кратки цикли, наречени спринтове, всеки с продължителност между една и четири седмици. По време на тези спринтове екипите работят по конкретни задачи, като редовно провеждат срещи за синхронизация и адаптация. В рамките на Scrum

съществуват три ключови роли: Продуктов собственик, който определя визията и приоритетите на продукта; Scrum Master, който подпомага екипа и гарантира спазването на Scrum принципите; и Разработващ екип, който изпълнява задачите по време на спринта. Важен аспект на Scrum е ретроспективният анализ след всеки спринт, който позволява непрекъснато усъвършенстване на работния процес [2].

Сравнението между Agile и традиционните методи като водопадния модел (Waterfall) показва значителни разлики в подхода към изпълнението на проектите. Докато Waterfall следва линейна последователност, Agile позволява паралелно изпълнение на различни фази и непрекъснати корекции в зависимост от обратната връзка на клиентите. Според проучване на Фъргюсън Узума и колегите му (2024), Agile методологиите постигат 82% удовлетвореност на заинтересованите страни и 75% по-бърза доставка на продукти в сравнение с традиционните модели, които осигуряват по-голяма стабилност, но са по-бавни при внедряването на промени [3].

Нарастването на популярността на Agile и Scrum върви ръка за ръка с развитието на технологиите, особено на облачните платформи, които позволяват по-добра координация и сътрудничество в екипите. Облачните технологии играят критична роля в управлението на проекти, като осигуряват достъп до информация в реално време, интеграция с различни инструменти за комуникация и възможности за автоматизация. Инструменти като Smartsheet, Asana и Trello предоставят удобни табла за управление на задачите, като същевременно позволяват гъвкавост в разпределението на работата. Например, Smartsheet интегрира функционалности за автоматизация и връзка с платформи като Microsoft Teams и Slack, докато Trello използва Kanban методологията за управление на работните потоци [4].

Въпреки многото предимства, използването на облачни технологии носи и предизвикателства, свързани със сигурността на данните и зависимостта от интернет връзка. Според публикация в The Guardian (2025), организациите, които разчитат на облачни технологии, трябва да спазват регулаторни изисквания като GDPR и да предприемат мерки за защита срещу киберзаплахи. Докладът подчертава, че въпреки ползите от цифровизацията, организациите трябва внимателно да оценяват риска от потенциални пробиви в сигурността [5].

Комбинацията от Agile методологиите и облачните технологии създава мощен инструментариум за управление на проекти, която позволява по-голяма адаптивност,

бързина и ефективност. Гъвкавостта на Agile, съчетана с възможностите за синхронизация в реално време, прави управлението на проекти по-ефективно и адаптивно към непрекъснато променящите се изисквания на бизнеса. Въпреки това, успешното внедряване на тези методологии изисква внимателно стратегическо планиране и културна адаптация в организациите. В бъдеще, развитието на изкуствения интелект и автоматизираните анализи вероятно ще доведе до още по-ефективни решения за управление на проекти, като допълнително подобри скоростта и качеството на работа в екипите.

3.2 Роля на технологиите: Използване на облачни технологии за синхронизация и съвместна работа в реално време

Облачните технологии играят все по-значима роля в съвременното управление на проекти, предоставяйки на екипите възможност за по-добра синхронизация и сътрудничество в реално време. С разрастването на глобалния бизнес и увеличаването на отдалечената работа, използването на облачни платформи като Microsoft Azure, Google Workspace и AWS се превръща в стандарт за много организации. Тези технологии позволяват не само по-добро управление на задачите, но и осигуряват гъвкавост, подобрена комуникация и достъпност до критична информация отвсякъде и по всяко време [6].

Едно от ключовите предимства на облачните платформи е възможността за синхронизация в реално време, която позволява на екипите да работят съвместно върху документи, да актуализират статуса на задачите си и да проследяват напредъка на проектите без забавяне. Инструменти като Asana и Trello използват облачни технологии, за да осигурят визуални табла и автоматизирани напомняния, което улеснява проследяването на задачите и подобрява ефективността на работните процеси. Например, Trello прилага Kanban принципите за управление на задачите, като позволява на потребителите лесно да преместват елементи между различни етапи на развитие, докато Asana предлага подробни диаграми за проследяване на напредъка и зависимостите между задачите [4].

Автоматизацията на процесите също е съществена част от облачното управление на проекти. Софтуери като Zapier и Power Automate позволяват автоматично синхронизиране на данни между различни приложения, премахвайки необходимостта

от ръчно въвеждане на информация. Това значително намалява риска от грешки и спестява време на екипите, което е особено полезно при управление на мащабни проекти с множество заинтересовани страни [7].

Друг важен аспект е сигурността на данните, тъй като облачните платформи работят с огромни количества чувствителна информация. Платформи като AWS и Google Cloud предлагат многослойни мерки за защита, включително криптиране на данни, многофакторна автентикация и механизми за откриване на заплахи. Въпреки това, организациите трябва да прилагат собствени политики за управление на достъпа и защита на информацията, за да минимизират риска от кибератаки [8].

Освен сигурността, предизвикателство пред облачните технологии е зависимостта от интернет връзката. Въпреки че облачните платформи предлагат несравнима гъвкавост, липсата на стабилен достъп до интернет може да доведе до затруднения при достъпа до важни данни и координация на екипите. За да се справят с този проблем, някои доставчици на облачни услуги като Google Drive и Dropbox предлагат офлайн режими, които позволяват на потребителите да работят върху документи и задачи дори без интернет връзка, като синхронизацията се извършва автоматично при възстановяване на връзката [9].

Въпреки някои предизвикателства, интеграцията на облачните технологии в управлението на проекти предлага множество предимства, които надхвърлят традиционните методи. Гъвкавостта, автоматизацията и възможността за работа в реално време правят облачните платформи незаменими в съвременната бизнес среда. В бъдеще, с развитието на изкуствения интелект и машинното обучение, облачните технологии ще продължат да се усъвършенстват, предлагайки още по-интелигентни решения за прогнозиране на рискове, автоматизиране на процеси и подобряване на производителността на екипите.

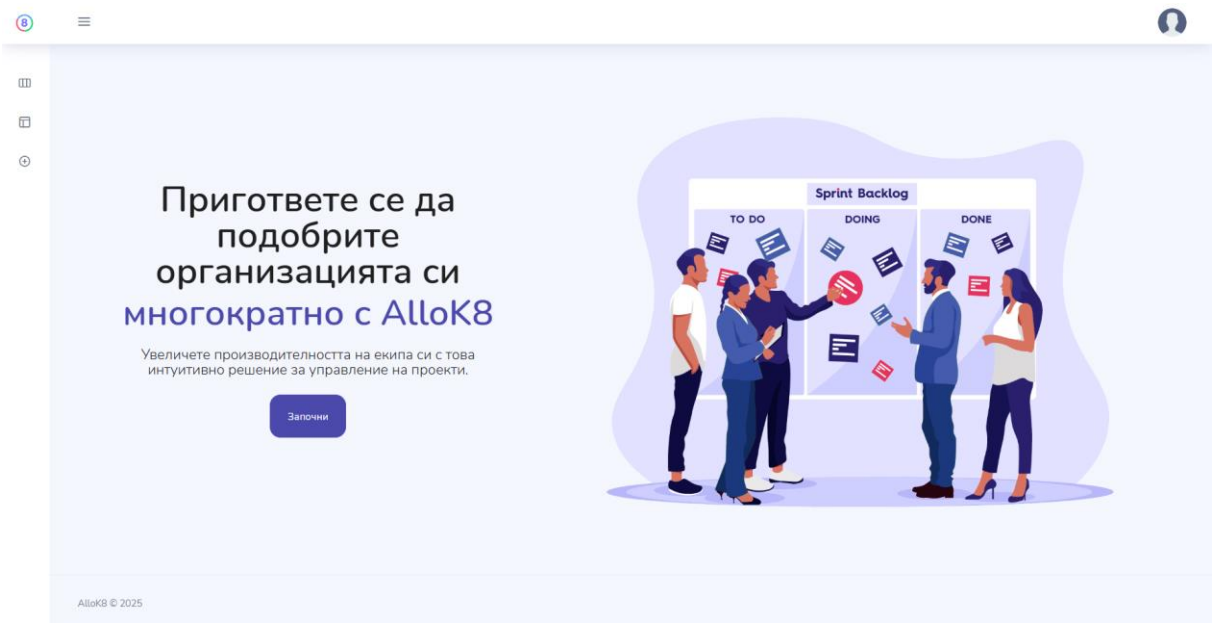
4 Анализ на решението

4.1 Потребителски изисквания и работен процес

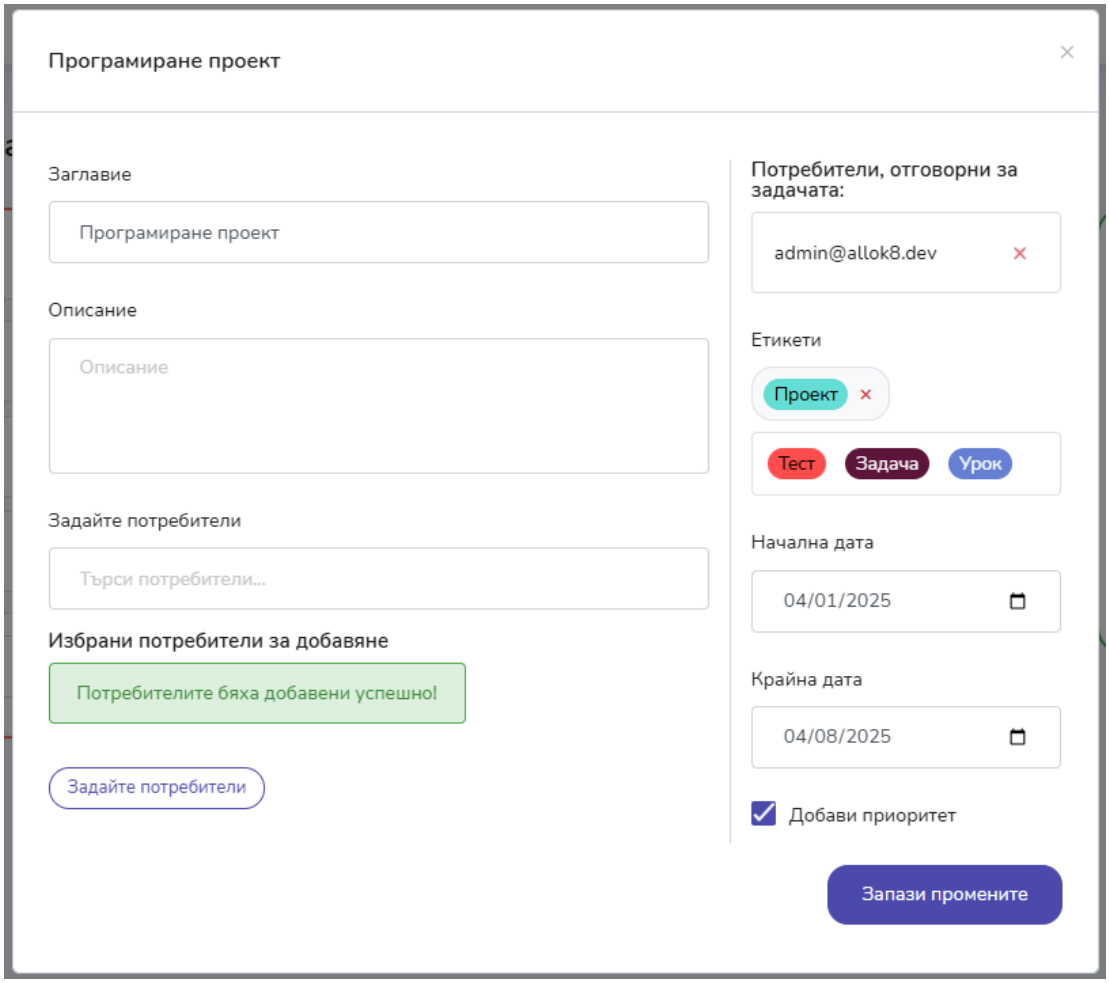
Системата може да се използва от различни предприятия или индивидуални лица и екипи. Те използват системата, за да организират работния си процес ефективно и да следят какво са свършили.

Първо те влизат в системата със своите имейл и парола. След това могат да виждат и да създават проекти, да проверяват крайните срокове на своите задачи в календара, да създават и назначават нови задачи. Също така могат да генерират отчети и да редактират личните си профили.

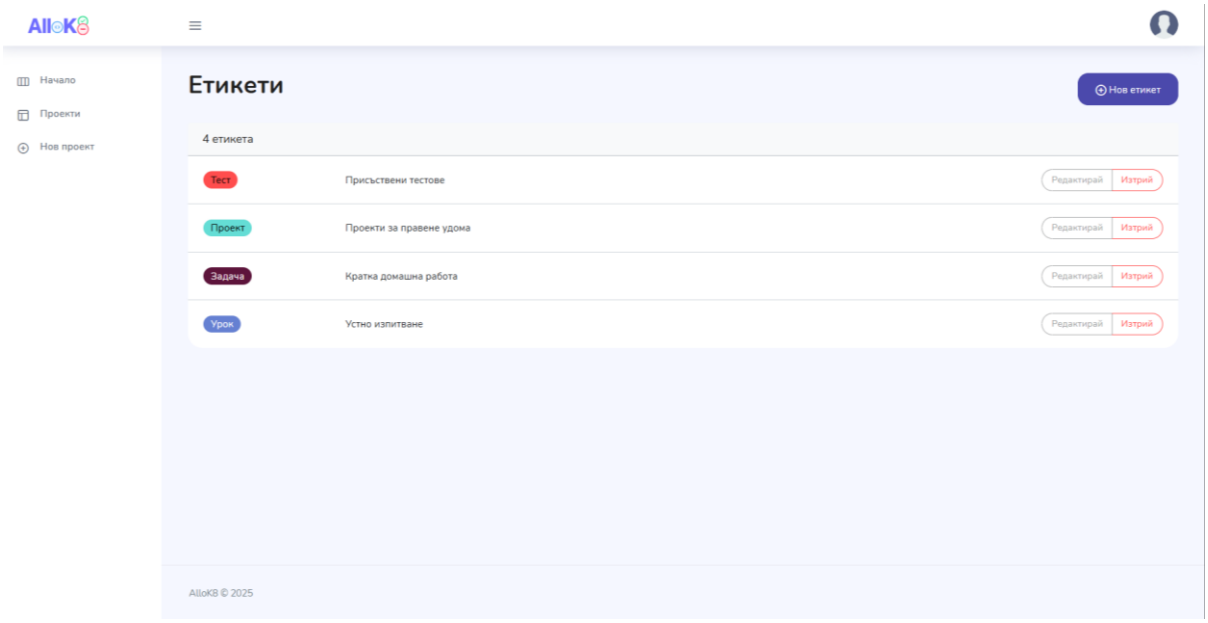
4.2 Примерен потребителски интерфейс



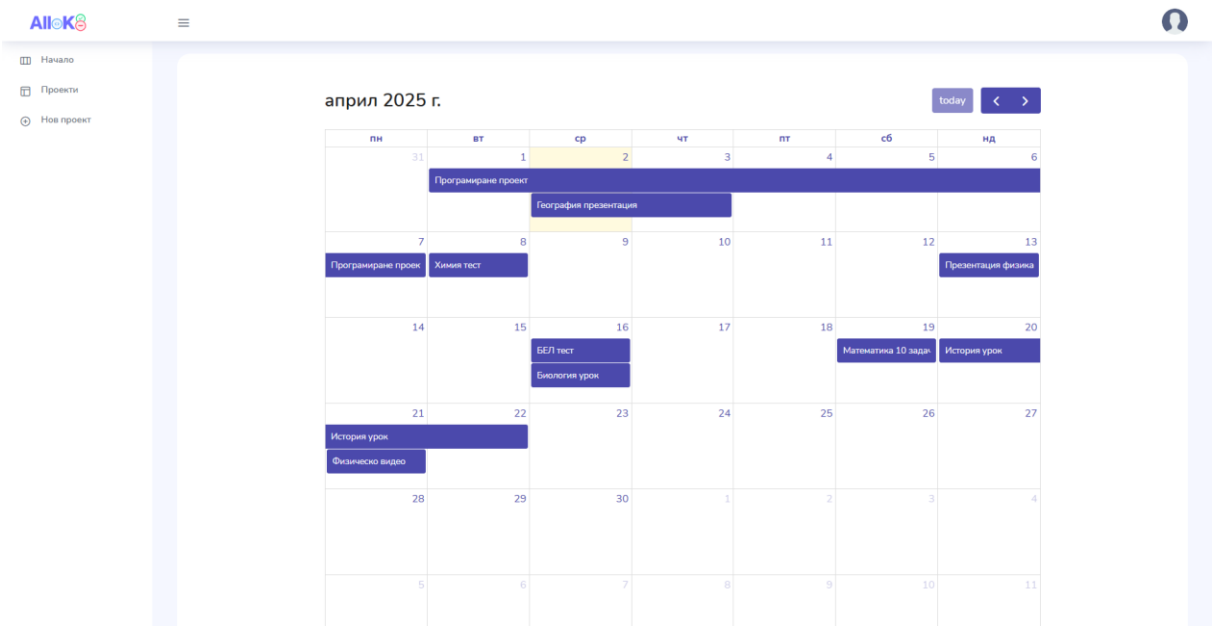
Фиг. 1 Начална страница



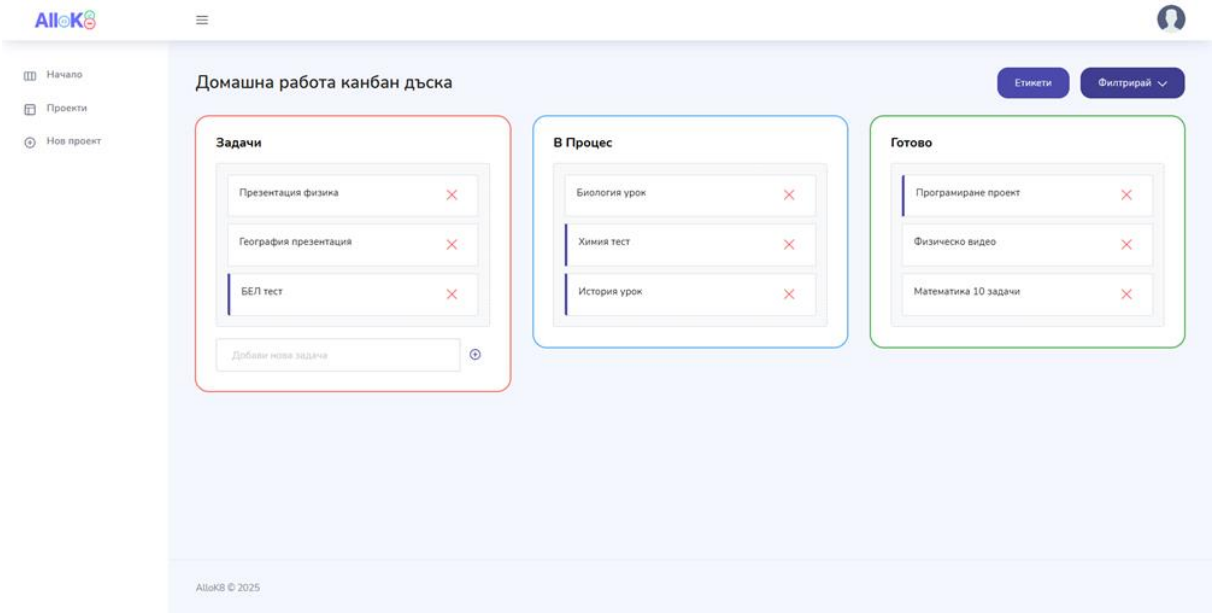
Фиг. 2 Модален прозорец за редактиране на задача



Фиг. 3 Страница етикети за определен проект



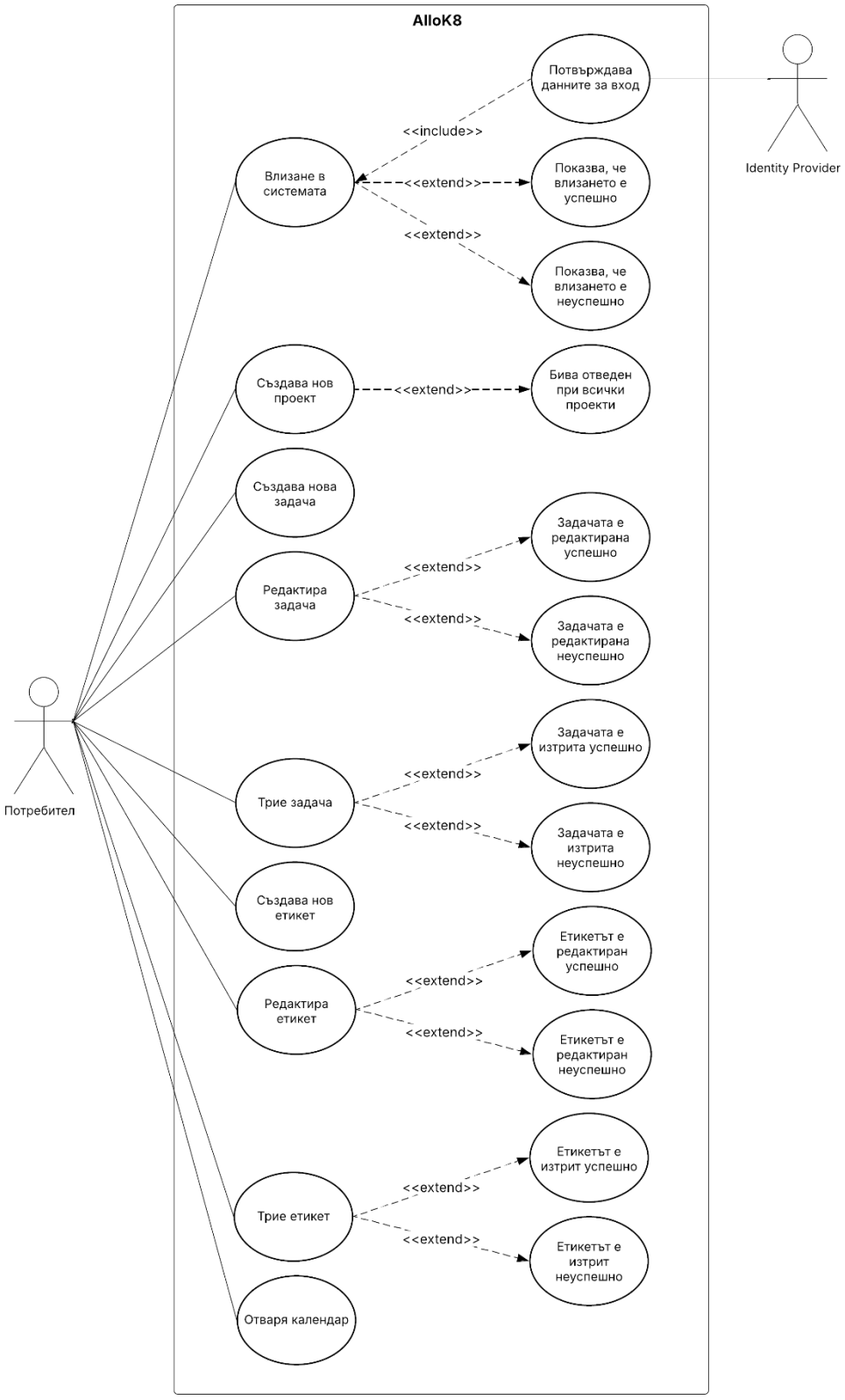
Фиг. 4 Страница с календар за определен проект



Фиг. 5 Страница с канбан дъска за определен проект

4.3 Диаграми на анализа

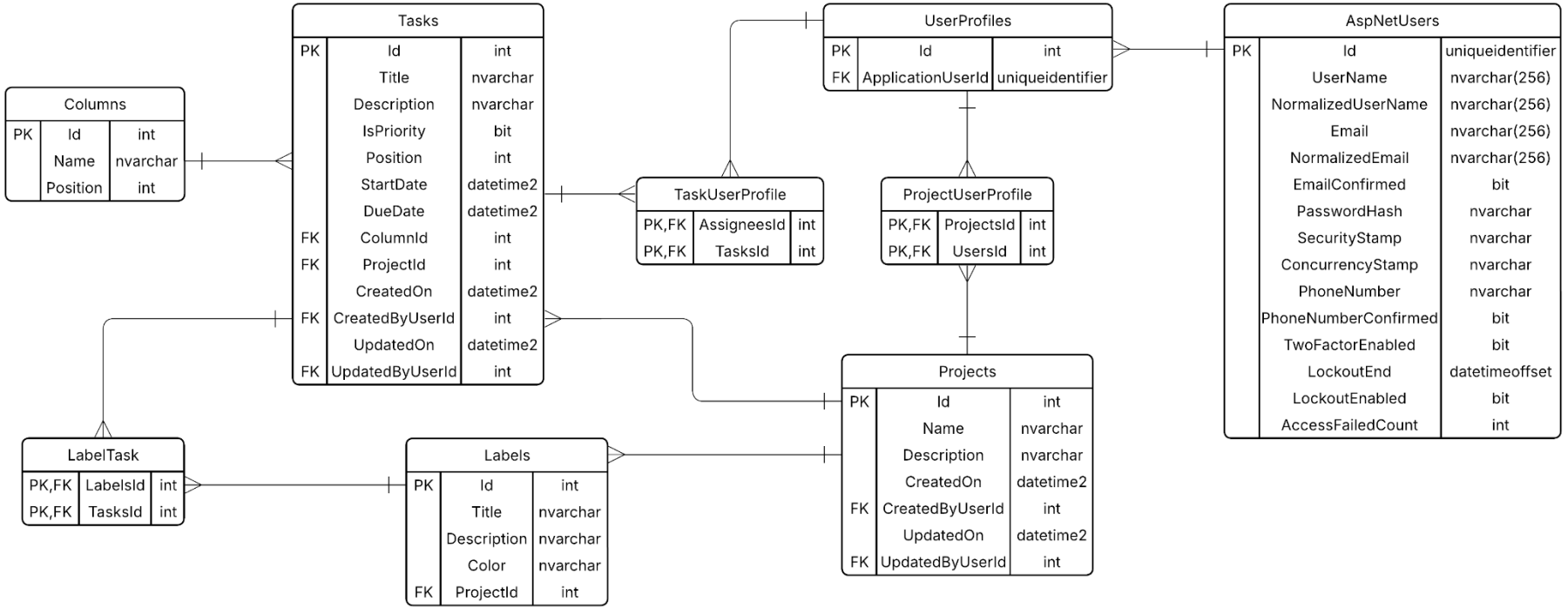
4.3.1 Диаграма на случаи на употреба



Фиг. 6 Диаграма на случаите за употреба (Use-Case)

4.4 Модел на съдържанието / данните

4.4.1 Диаграма на отношенията между моделите



Фиг. 7 Диаграма на отношенията между моделите (Entity-Relationship)

4.4.2 Нормализация

Базата данни представлява съвкупност от таблици, свързани релационно помежду си, чрез множество кардинални връзки. Базата е нормализирана в трета нормална форма, нека демонстрирам защо и как. Нормализацията е начин на подредба на релационни бази от данни, където се цели запазването на целостта данните и намаляването на тяхното повтаряне – един запис не трябва да се повтаря и съответна трябва да бъдат построени отношения между таблиците, за да се осъществи това. Макар че съществуват повече от 3 нормални форми, проекта е нормализиран до трета, защото тя е стандарта и въвеждането на повече форми значително би усложнило процесът по извличане на данни. За осъществяването на всяка нормална форма е нужно условията не предишната нормална форма да бъдат изпълнени. Така се получава инкрементален процес на зависимости при всяка нормална форма. [10]

Първа нормална форма гласи, че таблицата, която разглеждаме трябва да има уникален първичен ключ, както и че стойностите във всеки ред да са атомични, т.е. напълно единични, без няколко записа в една клетка. Затова разделяме полета като име и описание, както и изваждаме информация, която може да има връзка едно към много или много към много в отделни таблици.

Втората нормална форма изисква първоначална първа нормална форма. Това условие е изпълнено. Втората форма гласи, че всеки атрибут на таблица трябва да е функционално зависим само от първичния ключ. Това означава, че ако има полета, които се отнасят за нещо друго освен предмета на таблицаа, те трябва да се изведат в отделна таблица. Например ако имаме таблица Tasks, в нея не може да има поле LabelTitle – Трябва да се отдели в отделна таблица Labels.

Втората нормална форма е постигната. Изискванията за трета нормална форма са, че трябва информацията да е нормализирана във втора норма и колоните на таблицата да са устроени по такъв начин, че промяната на запис в една колона не би се отразил на верността на информацията в друга колона, или така наречената транзитивна зависимост. Нека приемем, че има колона А, която има референтна връзка към колона Б, но колона Б, от своя страна, има референтна връзка към колона В. Следователно, колона А реферира, или е свързана с колона В транзитивно.

Следвайки тези правила, таблиците са нормализирани в трета нормална форма.

5 Дизайн

Проектът е написан на C#, което означава, че е основан върху .NET Core платформата. Използва се версия 8 както за основната платформа, така и за другите библиотеки, произведени от Microsoft и други трети страни. Разделен е на 3 слоя – слой на бизнес логиката, слой за достъп до базата данни и презентационен слой, и 7 проекта:

- Презентационен слой - това е слоят, който комуникира с потребителите и представя данните/съдържанието в удобен и интерактивен формат.
    - AlloK8.PL – MVC проект, в който се задават както пътищата, така и изгледите на приложението.
- Слой за бизнес логиката - това е слоят, който съдържа бизнес логиката на приложението. Той е независим от другите слоеве и определя правилата и ограниченията за работа с данните/съдържанието.
    - AlloK8.BLL – Библиотека от класове, която използва интерфейси и абстракции за взаимодействие с другите слоеве. В папката Identity се съдържат сървисите за потребителите, а в Common за останалите функционалности като задачи, проекти, календар и отчети.
- Слой за данните - представен от следния проект:
    - AlloK8.DAL – Това е слоят за данни, където се намират моделите и контекстът за базата данни. Той използва Entity Framework Core за ORM и SQL Server за релационна база данни.
- В приложението има още три проекта:
    - AlloK8.Common – Това са модели за вход (input models) и изход (output models), които се споделят между контролера и бизнес логиката. Те представят данните/съдържанието в сериализиран формат (JSON).
    - AlloK8.BLL.Tests – Това е проектът, отговарящ за тестването на слоя за бизнес логиката на приложението.
    - AlloK8.Tests.Integration – Това е проектът, отговарящ за тестването на съвместната работа и синхронизацията в приложението.

Приложението използва набор от зависимости, за да постигне своите цели, а това са:

- Entity Framework Core 8 (EF Core 8) – Рамка, която улеснява достъпа до базата данни. Проекта е направен, чрез така наречения начин на първоначалната кодова структура (Code First Approach). Това означава, че първо са писани моделите, с които работи базата данни, а после са преобразувани в таблици със съответните им връзки, отношения и типове на колони.
- AspNetCore 8 – Рамка, която осигурява по-лесно управление и автентикация на потребители и пазенето им в базата от данни.
- EF Core 8 InMemoryDatabase – Тази зависимост е изградена върху EF Core 8 и представлява лесен начин да се създаде фалшива база от данни, съществуваща само в паметта на приложението, а не в някой външен сървър. Използва се при тестването на слоевете и е предпочитана пред рамки, като Moq и други, които позволяват по – лесно заместване на зависимости с изкуствени обекти, защото методите на EF Core 8 имат специфични имплементации, които са трудно заменими, или фалшифицирани от упоменатите библиотеки / рамки. Това е поради естеството на LINQ (Language Integrated Queries (Заявки, вградени в езика)) и неговата интеграция с работната рамка и начина за извличане на записи от библиотеката. Microsoft препоръчват този подход, вместо фалшифицирането.
- SendGrid – Библиотека, която позволява изпращането на имейли от приложението към потребителите.
- xUnit – Рамката, с която е реализирано единичното тестване на слоевете за бизнес логика и достъп до базата данни.
- Stylecop Analyzers – Рамка, която анализира C# кода, за да наложи набор от правила за стил и последователност.
- QuestPDF – Библиотека, която осигурява лесно създаване на PDF документи, като поддържа килизирани шрифтове.
- Essentials – Библиотека с помощни функции и класове.

Приложението използва набор от шаблони за дизайн (Design Patterns), някои от които са:

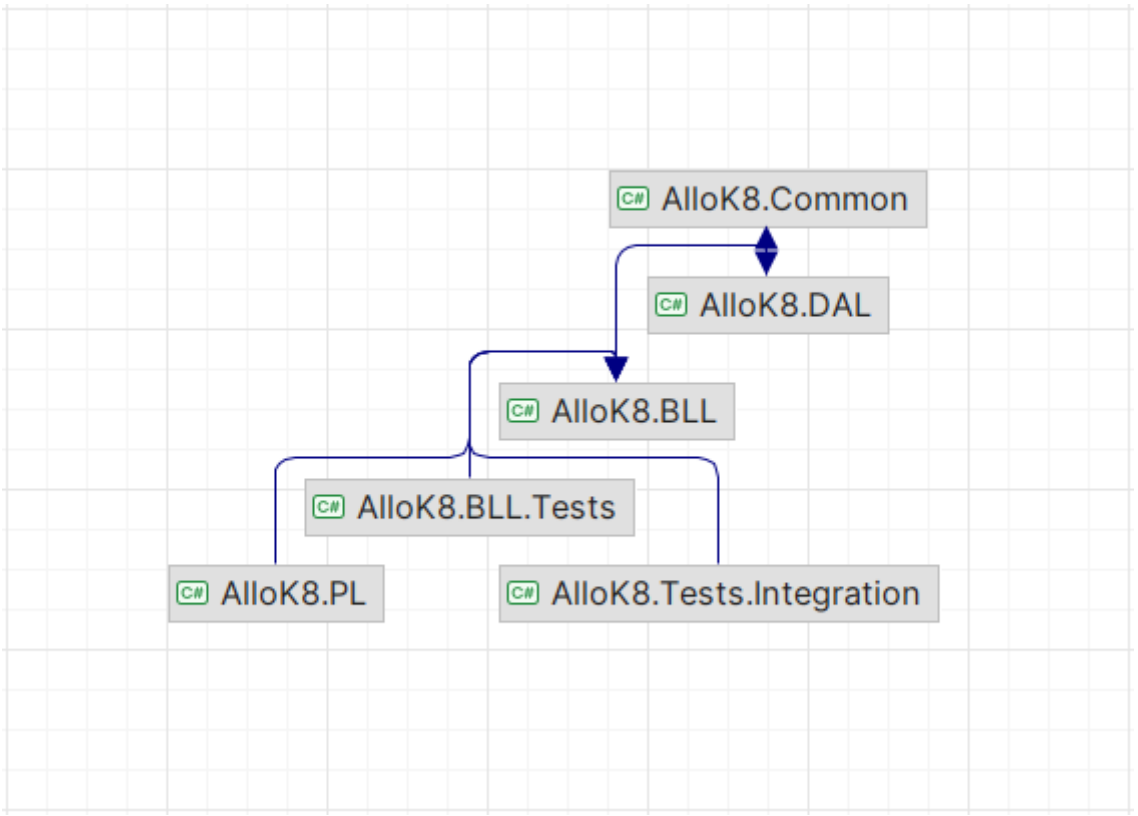
- Инжектиране на зависимости / Обръщане на контрол (Dependency Injection / Inversion of control) – този шаблон се използва главно при уеб приложението,

когато трябва да се инжектират класовете от бизнес слоя като услуги. Намира приложение и при тестването, когато трябва да се заменя същинските имплементации на външни методи със собствени.

- Единица работа (Unit Of Work Pattern) – този шаблон надгражда шаблона на хранилищата, като обединява хранилище от всеки тип в един клас и се грижи всяко едно от тях да използва един и същ контекст към базата данни, като по този начин се предотвратяват конкурентни проблеми и променянето на еднакви записи от няколко хранилища наведнъж. [11]

Тук е представена и диаграма, която описва седемте проекта и техните зависимости.

5.1 Реализация на архитектурата на приложението

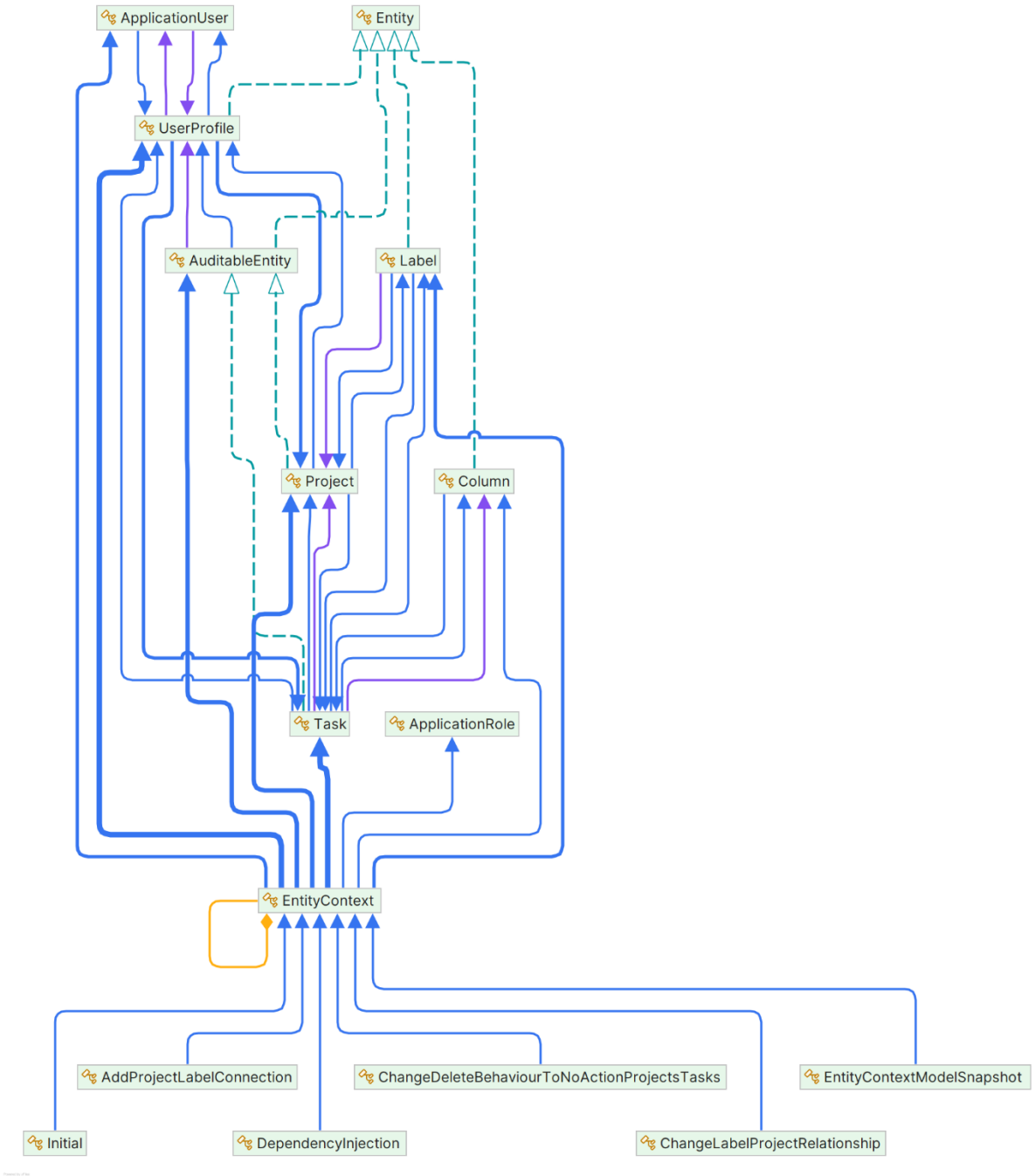


Фиг. 8 Диаграма на отношенията между проектите и техните зависимости

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

Приложението е изградено от следните слоеве:

5.2.1 Слой за достъп до данните

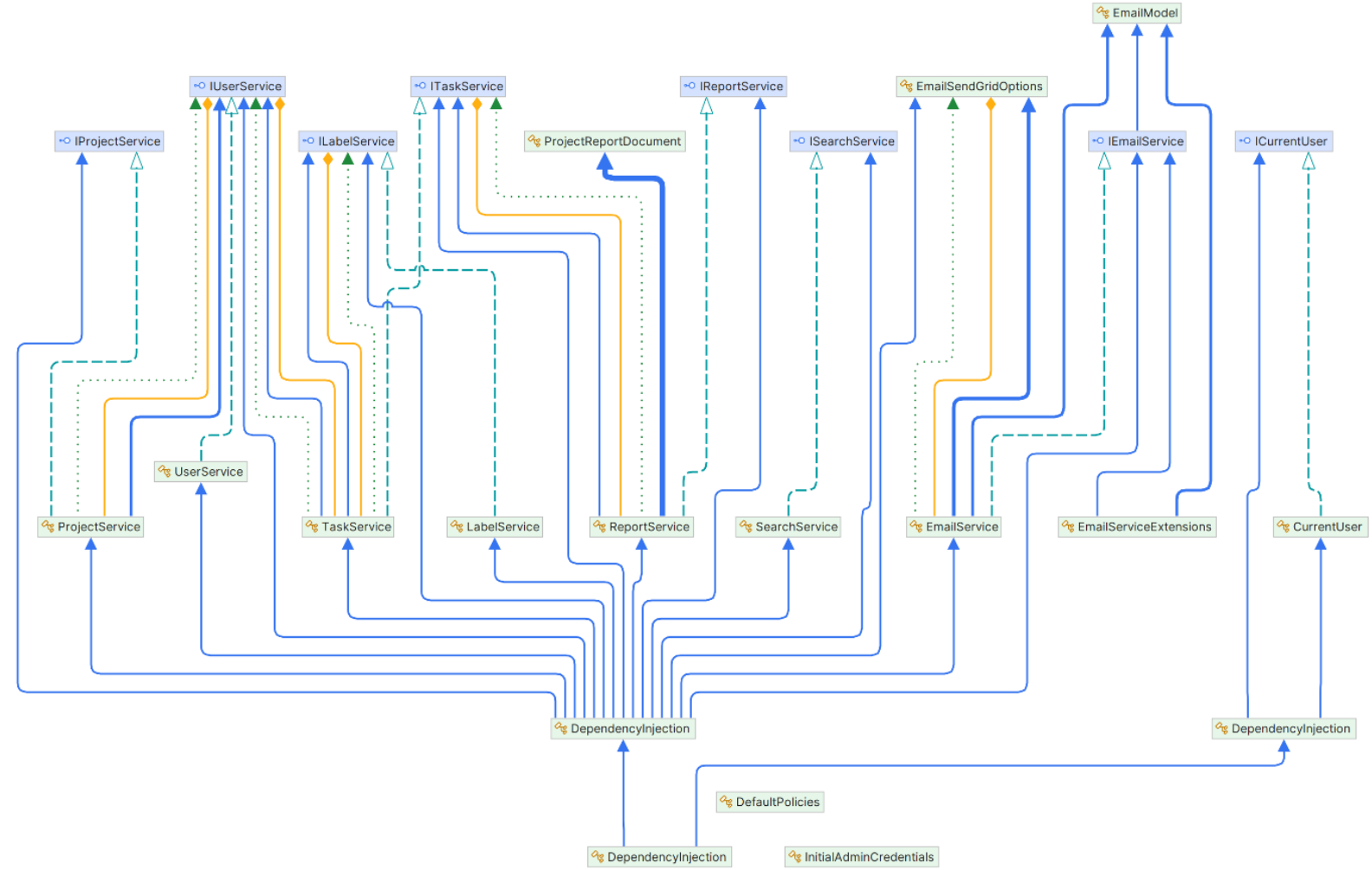


Фиг. 9 Диаграма на слоя за връзка с базите данни

Слоят за достъп до базите данни е представен от проекта AlloK8.DAL. В него се съхраняват самите модели, с които работи приложението, както и класът за контекста на базата данни, с който работи рамката Entity Framework Core. Разделен е на няколко имеви просранства (или папки).

- Models – Тук са моделите за всички таблици в базата данни, които включват модели за ASP.net, модели за абстракция и модели за останалите таблици.
    - ApplicationUser и ApplicationRole – Таблиците, свързани с ASP.net, чрез които се управляват потребителите.
    - Entity и AuditableEntity – Абстракциите на полетата, общи за повече от един модел.
    - Column – Колоните в канбан дъската.
    - Label – Етикети за задачи.
    - Project – Проекти.
    - Task – Задачи.
    - UserProfile – Чрез тази таблица се управляват връзките на потребителя с останалите таблици; тя е свързана 1:1 с ApplicationUser.
- Migrations – В това имено пространство се съхраняват миграциите на EF Core 8.
- В основното именно пространство се намират контекстът за базата данни и инжектирането на зависимости.
    - EntityContext – Контекстът за базата данни.
    - DependencyInjection – Използва низа за свързване с базата данни (БД) от кофигурацията, след което регистрира контекста за БД.

5.2.2 Слой на бизнес логиката



Фиг. 10 Диаграма на слоя на бизнес логиката

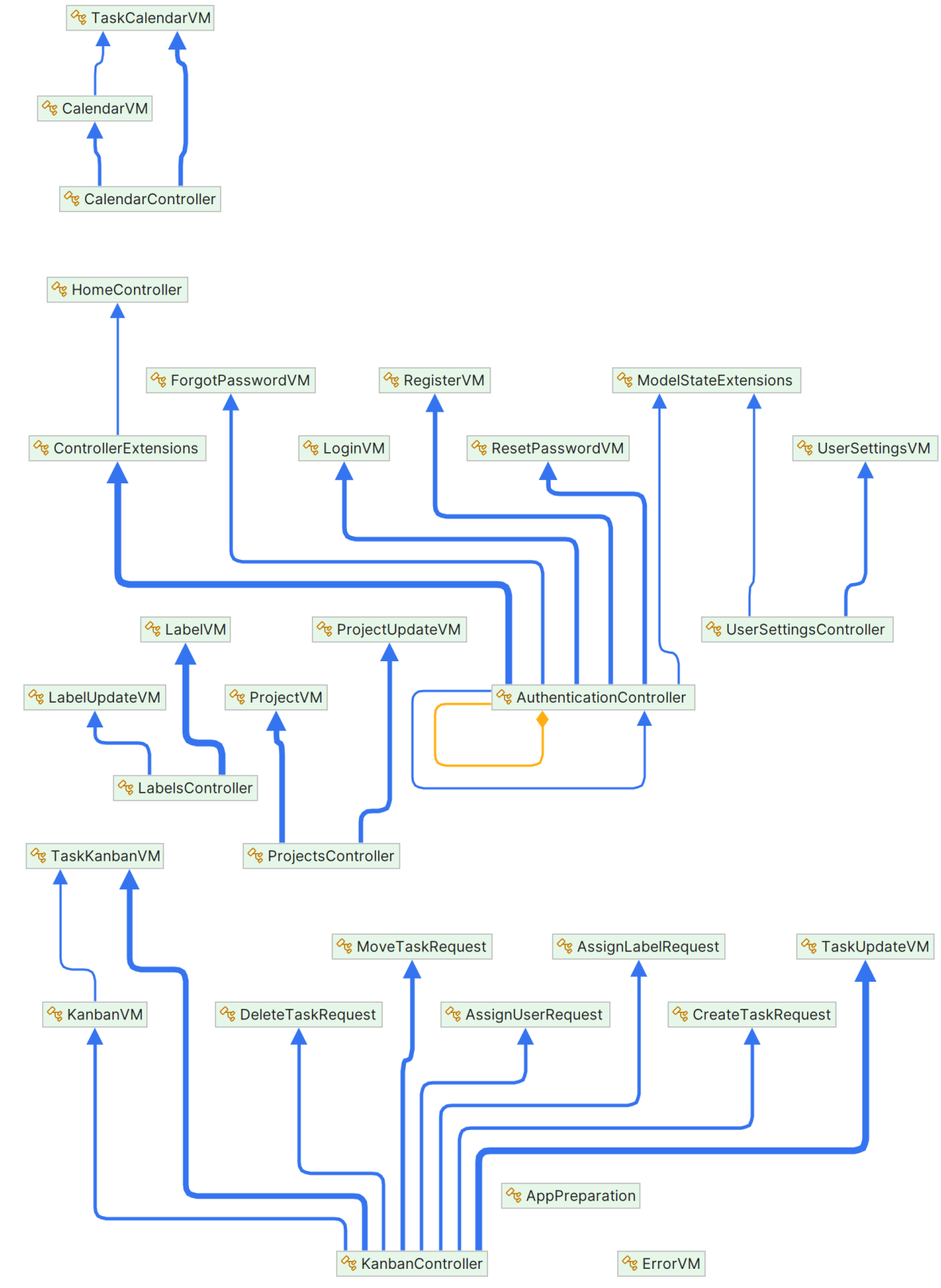
Слоят на бизнес логиката е представен от проекта AlloK8.BLL. Той съдържа бизнес функционалностите на проекта. Резделен е на няколко имеви пространства (или папки).

- Common – Тук се съдържат интерфейси и класове за всеки тип услуги.
  - Папка EmailSending
    - IEmailService/EmailService – Тук са методите за изпращане на имейл чрез SendGrid.
    - EmailSendGridOptions – Това са настройките за имейлите, които се взимат от конфигурацията.
    - EmailModel – Това е моделът с полетата за попълване при изпращане на имейл.
  - Папка Labels – ILabelService/LabelService – Тук са методите за управляване на етикети за задачи.
  - Папка Projects – IProjectService/ProjectService – Тук са методите за управляване на проекти.
  - Папка Reports – IReportService/ReportService – Тук са методите за създаване на отчети под формата на PDF документи.
  - Папка Search – ISearchService/SearchService – Тук са методите за търсене на потребители, които да се подадат на търсачката в презентационния слой.
  - Папка Tasks – ITaskService/TaskService – Тук са методите за управляване на задачи.
  - Папка Users – IUserService/UserService – Тук са методите за управление на UserProfile-и.
  - DependencyInjection – Тук се регистрират сървисите чрез колекция и кофигурацията.
- Identity – Тук се съдържа всичко, свързано с потребителя.
  - Папка Constants – Тук се задават константите за това именно пространство. В DefaultPolicies се задават политиките за достъп на потребителите. В InitialAdminCredentials се задава първоначален профил, с който може да се оперира в системата веднага, когато се създаде нейна инстанция.
  - Папка Contracts – Тук се намира интерфейсът ICurrentUser, който се състои от две полета.

- Папка Internals – Тук се намира калсът CurrentUser, който се използва за задаване на стойности на съвпадащия му интерфейс. Тук също има метод, който се занимава с това да намира идентификатора на потребителя, който е вписан към момента на извикване.
- Папка Extensions – Тук се намира статичният клас EmailServiceExtensions, който се занимава и изпращането на имейли.
- DependencyInjection – Тук се регистрират сървисите чрез колекция.

- В основното именно пространство на този проект се намират 2 файла.
  - Assembly – Този файл позволява на класовете, я които се съдържат сървисите, да са видими за проектите за тестване (юнит и интеграционни тестове).
  - DependencyInjection – Тук се регистрират сървисите от двете под-имеви пространства на проекта чрез колекция и кофигурацията.

5.2.3 Презентационен слой



Фиг. 11 Диаграма на презентационния слой

Презентационният слой се състои от проекта AlloK8.PL. Той е представен от MVC проект, като в него се намират контролерите, изгледите и съответстващите им модели, както и всички файлове за моделиране на изгледите. Разделен е на няколко имеви пространства (или папки).

- Controllers – Тук са контролерите за управление на съответните им изгледи. Те викат методите от слоя за бизнес логиката, като инжектират интефейсите на сървисите, от които се нуждаят.
    - AuthenticationController
    - CalendarController
    - HomeController
    - KanbanController
    - LabelsController
    - ProjectsController
    - UserSettingsController
- Models – Тук са всички модели, от които се нуждаят изгледите, за да си комуникират ефективно с контролерите. Те могат да се разделят на няколко категории
    - Модели за автентикация и моделиране на парола: ForgotPasswordVM, LoginVM, RegisterVM, ResetPasswordVM, UserSettingsVM
    - Модели за календар: CalendarVM, TaskCalendarVM
    - Модели за проекти: ProjectVM, ProjectUpdateVM
    - Модели за канбан дъска: KanbanVM, TaskUpdateVM
    - Модели за етикети: LabelVM, LabelUpdateVM
    - Requests – Това са модели, които се използват, за да подадат информация към контролера, която директно се използва, вместо само да се пренася.
        - AssignLabelRequest
        - AssignUserRequest
        - CreateTaskRequest
        - DeleteTastRequest
        - MoveTaskRequest
- Extensions – Тук се намират два статични класа, чиято работа е да пренасочват потребителя, в зависимост дали е вписан в системата.

- Views – Тук са изгледите на приложението. Те са разделени на подпапки, като всяка отговаря за различна логическа част от него.
    - Authentication – Тук е всичко, свързано с автентикацията на потребителите.
    - Calendar – Тук се показва календар на избрания от потребителя проект.
    - Home – Това е началната страница.
    - Kanban – Тук се показва канбан дъска на избрания от потребителя проект. Също се съдържат модални прозорци за редактиране и изтриване на задачи от съответния проект.
    - Labels – Тук се съдържа всичко за етикетите – изглед с всички етикети, изглед за създаване на етикет, модални прозорци за редактиране и изтриване.
    - Projects - Тук се съдържа всичко за проектите – изглед с всички проекти, изглед за създаване на проект, модални прозорци за редактиране и изтриване.
    - Shared – Тук са всички общи неща за изгледите.
    - UserSettings – Тук вписал се потребител може да промени паролата си.
- AppPreparation – В този статичен клас се създава автоматично базата от данни и се попълва ако е празна.

5.3 Организация и код на заявките към база от данни

За достъп до базата данни използвам инструментариума EF Core (Entity Framework Core), който е ORM (Обектно-Релационно картографиране) библиотека за .NET платформата. EF Core позволява за работа с данните като с обекти и колекции, без да се пише SQL заявки ръчно. EF Core поддържа различни видове бази данни, като в моя проект използвам SQL Server.

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

За да се управляват връзката с базата данни и операциите с данните, трябва да се дефинира контекст за базата данни като клас, който наследява от базовия клас

DbContext. В контекстът трябва да се декларира свойства от тип DbSet<T>, където T е моделът на данните. DbSet<T> представлява колекция от обекти, които съответстват на таблица в базата данни.

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

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

5.4 Наличие на потребителски интерфейс

Приложението има уеб потребителски интефейс, който е базиран на MVC технологията. MVC (Model-View-Controller) позволява изграждането на добре структурирани и мащабируеми уеб приложения, като разделя логиката, изгледите и управлението на заявките. Интерфейсът е разработен с помощта на Razor изгледи, които комбинират HTML и C# код за динамично показване на съдържанието.

Приложението използва Bootstrap 4 и CSS за стилизиране и оформяне на потребителския интерфейс, които осигуряват изчистен дизайн. За взаимодействие с бекенда се използва fetch от JavaScript, което позволява асинхронно извличане и изпращане на данни без необходимост от презареждане на страницата.

Важна част от функционалността е канбан дъската, която е интегрирана в приложението. Всеки проект разполага със своя собствена дъска, където задачите могат да бъдат премествани или редактирани в реално време. Тя също има филтри – за приоритет и „Само мои“.

Благодарение на архитектурата на MVC и използването на Razor изгледи, различните части на приложението могат да споделят общи компоненти, което улеснява поддръжката и разширяването на функционалността.

Основните функционалности на интерфейса на приложението са:

- Вход и регистрация – Потребителите използват имейл и парола, за да се автентикират.
- Преглед на проекти – Потребителите могат да разглеждат всички проекти, които са създали, или в които са добавени. За тази цели има страница „Проекти“, която ги показва като списък с бутони за навигация из всички функционалности, свързани с тях.
- Преглед на канбан дъска – Всеки проект разполага със своя собствена дъска, където задачите могат да бъдат премествани или редактирани в реално време. Тя също има филтри – за приоритет и „Само мои“.
- Преглед на календар – Всеки проект има свой спобствен календар, където могат да се видят всички задачи.
- Допълнители функционалности са:
    - Смяна на паролата – Всеки вписан потребител може да сменя паролата си.
    - Добавяне на потребители към проекти и задаването на задачи на потребителите във всеки проект.
    - Цветни текстови етикети за задачите.
    - Сортиране на задачите по приоритет.
    - Сортиране на задачите, така че да се показват само тези, зададени на вписания потребител.

6 Ефективност и бързодействие на решението

Проектът не работи с алгоритми от висока сложност. По – голямата част от логиката на приложението е свързана с обработката на данните на потребителите, чрез базата данни. Това се случва, през многобройните хранилищата, които използват методите на LINQ (Language Integrated Queries) за да обслужи заявките на потребителите. Друг алгоритъм, който се използва, е приоритетна опашка за приоритизиране на задачите. Поради тази причина, равносметката за бързодействието и ефективността може да се направи, като се разгледа сложността на алгоритмите, използвани от LINQ и от приоритетната опашка.

При LINQ повечето методи са със сложност O(n), защото просто обхождат колекцията. Примери за методи с така сложност са Select(), Single(), First(). Приложението употребява тези методи в имплементацията на методи за четене от базата данни, като GetDetails(), GetAllRecords() В тези случаи сложността на методите е O(n), но има и други методи, като OrderBy(), които имат логаритмична сложност (O(n logn)). Този метод се вика от хранилищата при търсенето и страницирането (Paging) на записи. [12]

Методът Where() е по – особен, защото няма точна сложност. Приложението го използва многократно в методи, като GetWhere(), GetDetails(), IsUserInCompany(). Истината е, че неговата сложност зависи от метода, който кара изброимата колекция да се изпълни, което най – често е .ToList(). Това придава на изброените методи сложност O(n). [12]

Priority Queue в C# е структура от данни, която позволява обработка на елементи според техния приоритет, като винаги извлича този с най-висок или най-нисък приоритет. От .NET 6 нататък е наличен класът PriorityQueue<TElement, TPriority>, който е оптимизиран за ефективност и използва бинарен куп (binary heap) като основна структура. [13]

Основните операции имат следните времеви характеристики:

- Добавяне на елемент (Enqueue) – извършва се за O(log n) чрез добавяне на новия елемент в края и последващо преорганизиране на купчината (heapify-up).
- Извличане на елемента с най-висок приоритет (Dequeue) – също отнема O(log n), тъй като премахването на кореновия елемент изисква реорганизация (heapify-down).
- Преглед на елемента с най-висок приоритет (Peek) – става за O(1), тъй като най-важният елемент винаги се намира на върха на купчината.

Priority Queue в C# осигурява значително по-добра производителност в сравнение с неподредени списъци, които имат O(n) сложност за извличане на минимален/максимален елемент. В сравнение с SortedList или SortedDictionary, които поддържат сортиран ред на елементите и предоставят O(logn) сложност за добавяне и премахване, PriorityQueue е по-ефективна за приложения, където е необходимо бързо

извличане на най-високия приоритет, без необходимост от поддържане на напълно сортирана структура. [13]

Обобщено, сложността на методите, извикани от приложението е променлива, спрямо обстоятелствата, в които са повикани. Най-чести обаче са сложностите O(n logn) и O(n).

7 Тестване

Проектът е тестван чрез рамката xUnit. Има два проекта, отговарящи за тестването: AlloK8.BLL.Tests за тестване на бизнес логиката и AlloK8.Tests.Integration за тестване на синхронизацията между потребителите и екипната работа. Конвенцията за наименоване на тестовете е: ИмеНаМетод_Сценарий_ТрябваДаИзвършиДаденоДействие(). [14]

Тестовете следват подхода на трите А-та (Arrange Act Assert) или подреждане, действане и твърдение. Във всеки един метод първо се случва етапа на на подреждането, където контекста на базата в паметта се зарежда с примерни данни. След се случва етапа на действие, където се вика тествания метод, а накрая се завършва с етапа на твърдението, където се сравнява резултата от извиканата функция с очаквания. [15]

Слоят за бизнес логиката се тества чрез 3 класа с юнит тестове за различните сървиси: LabelServiceTests, ProjectServiceTests и TaskServiceTests. Те проверяват валидността на данните, взети и изпратени през методите на услугите. Имитацията на базата данни се осъществява, чрез функционалност на Entity Framework Core 8, наречена InMemoryDatabase, или база данни, състояща се изцяло в паметта на приложението, вместо да се изпълняват операциите върху същинската база. При всички това работи по еднакъв начин – в кноструктора се създава нов контекст за базата данни, който използва паметта вместо истински сървър. Настройва се контекста да отговаря за чисто нова генерирана база със случайно име, взето от метода GUID.NewGuid().ToString(). Това се прави чрез извикването на направения от мен TestHelpers статичен клас. Всеки клас за тестване провежда функционалността си в контекста на случайно генерираната си база. Това е направено с цел изолацията на един тестов клас от данните на други тестови класове. Ако всички тестове използваха една и съща база, то тогава щеше да има постоянно десинхорнизиране между данните на тази обща база, защото тестовете се

изпълняват по различно време всеки път и предизвикват странични ефекти върху записите, с които работят останалите тестове. По този начин се създават непредсказуеми конкурентни проблеми и различни тестове се провалят по различно време и по различен начин. Отделянето на тестовете в нови бази със случайно генерирани имена предотвратява това явление. [16]

Екипната работа и синхронизацията се тестват с интеграционни тестове за ProjectService. Тук също се използва InMemory база данни. При тестване на услуга като ProjectService, която работи с база данни, ключовите аспекти на интеграционните тестове включват проверка на основните операции – създаване, извличане и актуализиране на проекти. Целта е да се потвърди, че данните се обработват коректно в контекста на цялостното приложение, а не само като отделни модули.

8 Заключение и възможно бъдещо развитие

Обобщено, проектът е разработен до такава степен, че да изпълнява всички поставени цели. Архитектурата на приложението е подпомогната от добрите практики, спазени при създаването моделите, изгледите и всички класове и интерфейси в C# и .NET. Създадена е интерактивна платформа, която има за цел да реши проблема с организиране на работа между повече хора. Цветовата гама е приятна и ненатоварваща, а дизайнът е опростен за възможно най-лесна навигация. Бъдеща доразработка на проекта би било добавянето на възможност за още повече персонализиране и добавяне на подробности по задачите и проектите.

9 Използвани литературни източници и Уеб сайтове

[1] К. Бек, М. Бийдъл, А. Ван Бенекъм, А. Кокбърн, У. Кънингам, М. Фаулър, Д. Гренинг, А. Хънт, Р. Джефрис, Д. Кърн, Б. Марик, Р. С. Мартин, С. Мелър, К. Швабър, Д. Съдърланд и Д. Томъс, „Manifesto for Agile Software Development,“ 2001. [Онлайн]. Available: https://agilemanifesto.org/.

[2] К. Швабър и Д. Съдърланд, „The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game,“ *Scrum Alliance,* 2020.

[3] Ф. Узума, Д. Агбана и А. Д. Алу, „A Comparative Analysis of Agile and Traditional Project Management Methodologies,“ *Educational Administration: Theory and Practice,* p. 5737–5746, 2024.

[4] Atlassian, „What is Trello?,“ 2024. [Онлайн]. Available: https://support.atlassian.com/trello/docs/what-is-trello/.

[5] Джефрис, Дънкън, „From deadlines to budgets, project management skills have become vital for everyone - here's how to make it all easier,“ 31 Януари 2025. [Онлайн]. Available: https://www.theguardian.com/it-starts-with-smartsheet/2025/jan/31/from-deadlines-to-budgets-project-management-skills-have-become-vital-for-everyone-heres-how-to-make-it-all-easier.

[6] М. Армбръст, А. Фокс, Р. Грифит, А. Д. Джосеф, Р. Катз, А. Конвински, Г. Лий, Д. Патерсън, А. Рабкин, И. Стоика и М. Захариа, „A View of Cloud Computing: Trends and Opportunities,“ *Communications of the ACM,* pp. 50-65, 2024.

[7] Microsoft, „Power Automate: Automate your workflows,“ 2024. [Онлайн]. Available: https://powerautomate.microsoft.com.

[8] CheckPoint, „Top Cloud Security Trends in 2025,“ 2025. [Онлайн]. Available: https://www.checkpoint.com/cyber-hub/cloud-security/what-is-code-security/top-cloud-security-trends-in-2025/.

[9] Google, „Google Drive Offline Mode: Work anywhere, anytime,“ 2024. [Онлайн]. Available: https://drive.google.com.

[10] Microsoft, „Description of the database normalization basics,“ 6 Юни 2024. [Онлайн]. Available: https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description.

[11] Microsoft и Т. Дийкстра, „Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application,“ 1 Юли 2022. [Онлайн]. Available: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application.

[12] M. Банимад, „Understanding Time Complexity of C# List and LINQ Methods,“ 25 Януари 2023. [Онлайн]. Available: https://masoudx.medium.com/understanding-time-complexity-of-c-list-and-linq-methods-6b5e36d66243.

[13] Microsoft, „PriorityQueue<TElement,TPriority> Class,“ 2025. [Онлайн]. Available: https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.priorityqueue-2?view=net-8.0.

[14] Microsoft, „Unit testing best practices for .NET,“ 22 Март 2025. [Онлайн]. Available: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices.

[15] П. Гомез, „Unit Testing and the Arrange, Act and Assert (AAA) Pattern,“ 9 Септември 2017. [Онлайн]. Available: https://medium.com/@pjbgf/title-testing-code-ocd-and-the-aaa-pattern-df453975ab80.

[16] qujck, „Dependency injection in unit of work pattern using repositories,“ 18 Април 2013. [Онлайн]. Available: https://stackoverflow.com/questions/16064902/dependency-injection-in-unit-of-work-pattern-using-repositories.

10 Приложения

Фиг. 1 Начална страница

Фиг. 2 Модален прозорец за редактиране на задача

Фиг. 3 Страница етикети за определен проект

Фиг. 4 Страница с календар за определен проект

Фиг. 5 Страница с канбан дъска за определен проект

Фиг. 6 Диаграма на случаите за употреба (Use-Case)

Фиг. 7 Диаграма на отношенията между моделите (Entity-Relationship)

Фиг. 8 Диаграма на отношенията между проектите и техните зависимости

Фиг. 9 Диаграма на слоя за връзка с базите данни

Фиг. 10 Диаграма на слоя на бизнес логиката

Фиг. 11 Диаграма на презентационния слой