**ДИПЛОМЕН ПРОЕКТ**

**ЗА ДЪРЖАВЕН ЗРЕЛОСТЕН ИЗПИТ**

по професия код 481030 „Приложен програмист“

специалност код 4810301 „Приложно програмиране“

ТЕМА: „ПРИЛОЖЕНИЕ ЗА УПРАВЛЕНИЕ НА СЕМЕЕН БЮДЖЕТ“

Автор:

Константин Каменов Динев, клас XII В

Ръководител:

Даниела Пендашева

Бургас

СЪДЪРЖАНИЕ

[1. Увод 2](#_Toc194943403)

[2. Цели и обхват на софтуерното приложение 2](#_Toc194943404)

[3. Анализ на решението 2](#_Toc194943405)

[3.1. Потребителски изисквания и работен процес 2](#_Toc194943406)

[3.2. Примерен потребителски интерфейс 2](#_Toc194943407)

[3.3. Диаграми за анализа 2](#_Toc194943408)

[3.3.1. Диаграма на случаи за употреба 2](#_Toc194943409)

[3.3.2. Диаграма на дейността 2](#_Toc194943410)

[3.3.3. Диаграма на последователността 2](#_Toc194943411)

[3.4. Модел на съдържанието/данните 2](#_Toc194943412)

[3.4.1. Диаграма на отношенията между моделите 2](#_Toc194943413)

[3.4.2. Нормализация 2](#_Toc194943414)

[3.4.3. Пояснения за таблиците 2](#_Toc194943415)

[4. Дизайн 2](#_Toc194943416)

[4.1. Реализация на архитектурата на приложението 2](#_Toc194943417)

[4.2. Описание на слоевете 2](#_Toc194943418)

[4.2.1. Слой за достъп до данните 2](#_Toc194943419)

[4.2.2. Слой на бизнес логиката 2](#_Toc194943420)

[4.2.3. Презентационен слой 2](#_Toc194943421)

[4.3. Организация и код на заявките към базата данни 2](#_Toc194943422)

[4.4. Наличие на потребителски интерфейс(консолен, графичен, уеб) 2](#_Toc194943423)

[5. Ефективност и бързодействие на решението 2](#_Toc194943424)

[6. Тестване 2](#_Toc194943425)

[7. Заключение и бъдещо възможно развитие 2](#_Toc194943426)

[8. Използвани литературни източници и Уеб сайтове 2](#_Toc194943427)

[9. Приложения 2](#_Toc194943428)

# **Увод**

В днешно време, управлението на личните финанси е от ключово значение за постигане на финансова стабилност. Много млади хора се затрудняват да проследяват своите приходи и разходи, което води до финансови затруднения. Липсата на ясна представа за разходите може да доведе до вземането на неправилни решения, водещи до затруднения при спестяването за важни цели.

Това уеб приложение позволява на потребителите да въвеждат своите приходи и разходи, да ги категоризират, да получават отчети и анализи, както и да поставят месечни бюджети. Приложението ще помогне на потребителите да получат по-ясна представа за финансовото си състояние и да вземат информирани решения за управлението на приходите си.

Приложението предлага следните функционалности:

* Проследяване на семейните приходи и разходи – записване и категоризиране на всички финансови операции, задаване на дневни и текущи разходи;
* Общ семеен бюджет – възможност за разпределение на средствата по категории (храна, жилищни разходи, образование и др.);
* Графики и анализ на разходите – визуализация на тенденциите в разходите и приходите, най-високите разходи по категории и идеи за тяхното намаляне;
* Задаване на спестовни цели – заделяне на част от даден приход за съответната спестовна цел;
* Напомняния за плащания – автоматични известия за предстоящи сметки, надвишавания на бюджетни лимити;
* Оптимизация на бюджета – идентифициране на най-големите разходи и предложения за тяхното намаляване.

Този документ представя поставените цели, предлаганите решения, тяхната реализация и анализ. Описаната информация е допълнена с UML (Unified Modeling Language) диаграми, тестови сценарии и обосновани разсъждения.

# **Цели и обхват на софтуерното приложение**

Целта на този проект е създаване на приложение за планиране и управление на семейния бюджет, което да помага на хората да управляват по-ефективно финансите си. Приложението е предназначено да помага на домакинствата да планират, проследяват и анализират своите финанси. Основната му задача е да осигури контрол върху семейните финанси, като предоставя инструменти за разпределение на бюджета, анализ на разходите, заделяне от приходите за спестовни цели и известия (напомняния) за важни плащания и сметки.

Основни потребители са:

* Семейни двойки, споделящи общи финанси и разходи;
* Домакинства с деца, чийто бюджет трябва да бъде разпределен между различни нужди(храна, транспорт, услуги, др.);
* Малки домакинства и съквартиранти, използващи го с цел прозрачно управление на общите разходи.

Домакинствата ще имат ясна представа за своите спестявания и разходи, което ще им помогне да управляват финансите си по-ефективно. Целта на приложението е да се улесни управлението на семейния бюджет, като предоставя ясна картина на финансовото състояние на домакинството и помага за намаляването на ненужните разходи.

# **Анализ на решението**

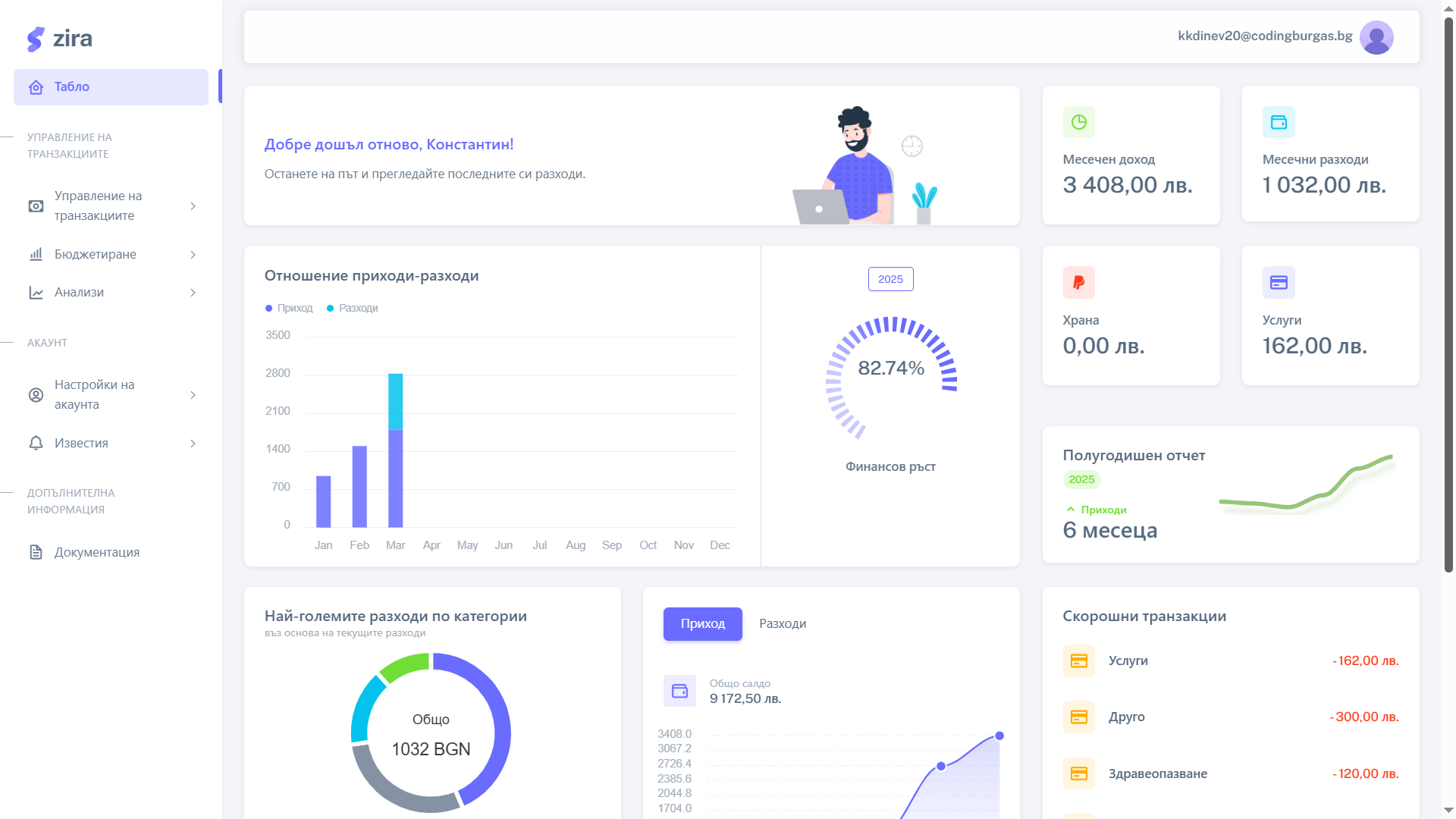
## Потребителски изисквания и работен процес

Приложението е предназначено за семейства и малки домакинства, които желаят да управляват своите финанси ефективно. То предоставя удобни инструменти за планиране и проследяване на бюджети, спестовни цели и лични финанси. Процесът на работа е интуитивен: потребителите добавят приходи и разходи, създават бюджети и проследяват финансовото си състояние чрез графики и статистики.

## Примерен потребителски интерфейс

Приложението разполага с модерен и удобен интерфейс, който улеснява управлението на личните финанси. Основни елементи на интерфейса:

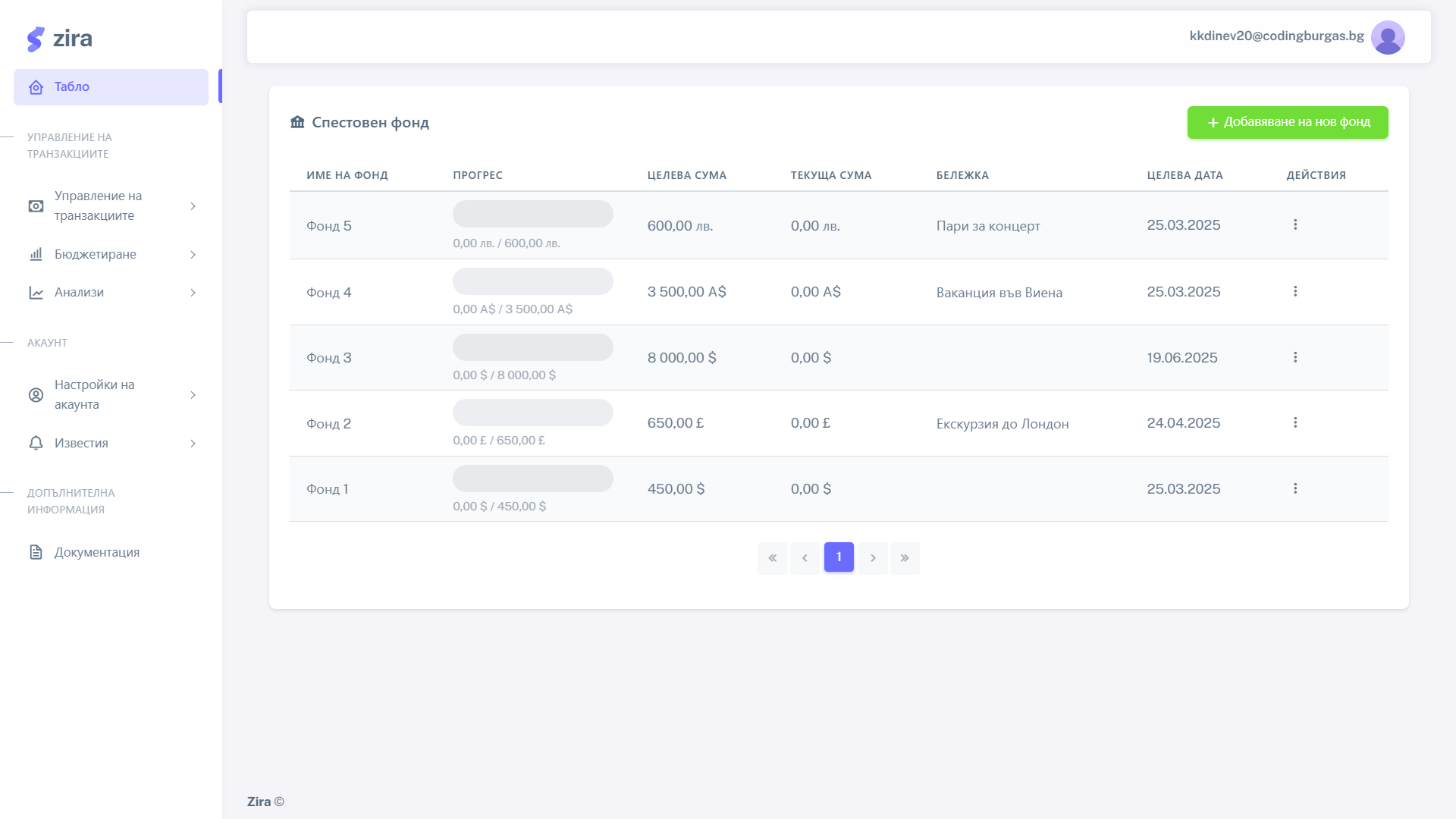
* Панел за управление(Dashboard) – показва финансовото състояние, графики и основни метрики;
* Управление на транзакции – раздел за управление на транзакциите;
* Бюджетиране – раздел за следене на състоянието на бюджетие и спестовните цели;
* Анализи – страница, обобщаваща потребителските финанси под формата на графики и таблици;
* Настройки на акаунта – потребителска страница и обща информация за него;
* Известия – напомняния за важни финансови събития;
* Документация – препратка към хранилището(repository) на проекта



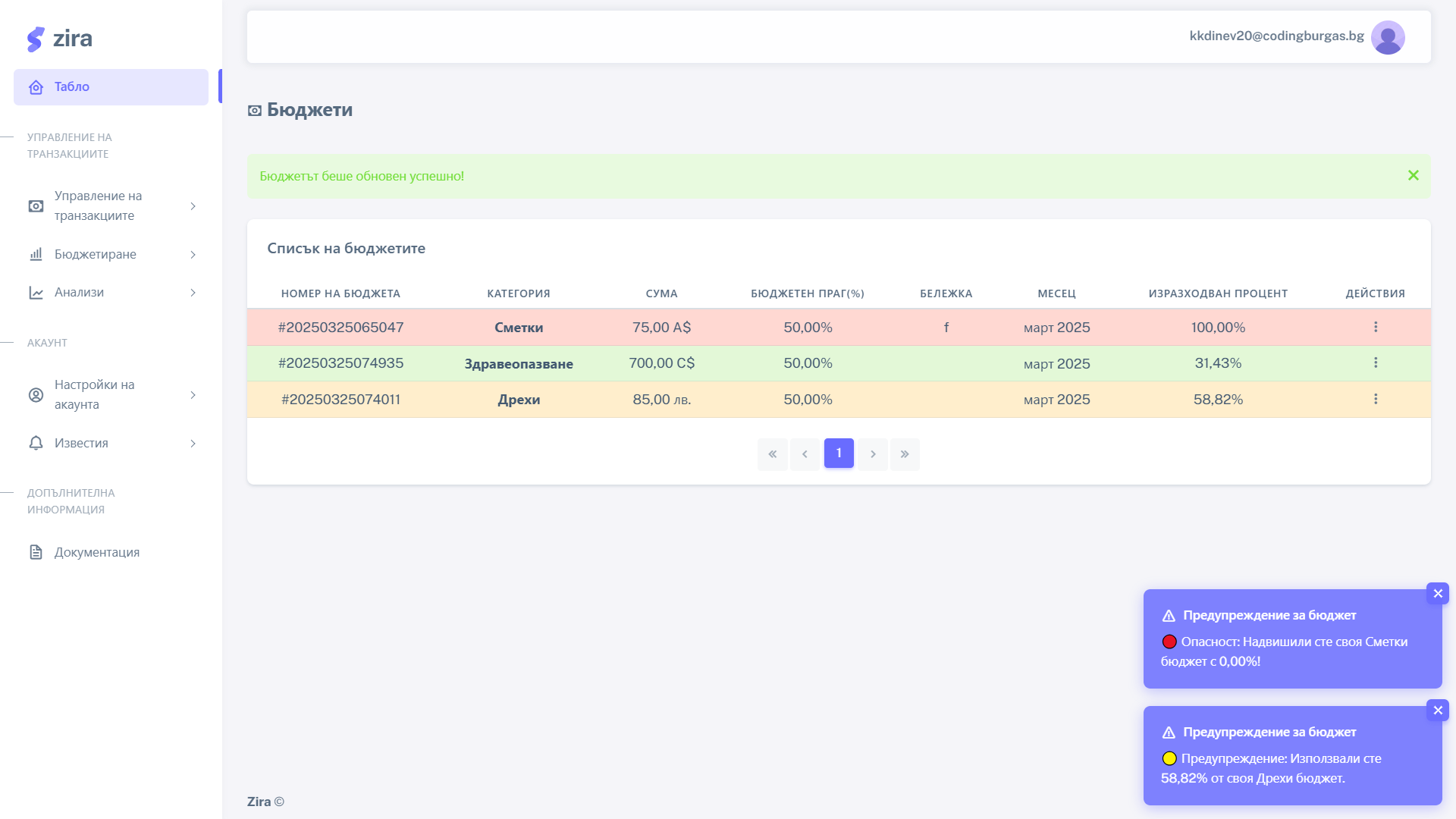
Фигура № 1: Страница, на която потребителят може да види в обобщен вид информация за своите финанси.



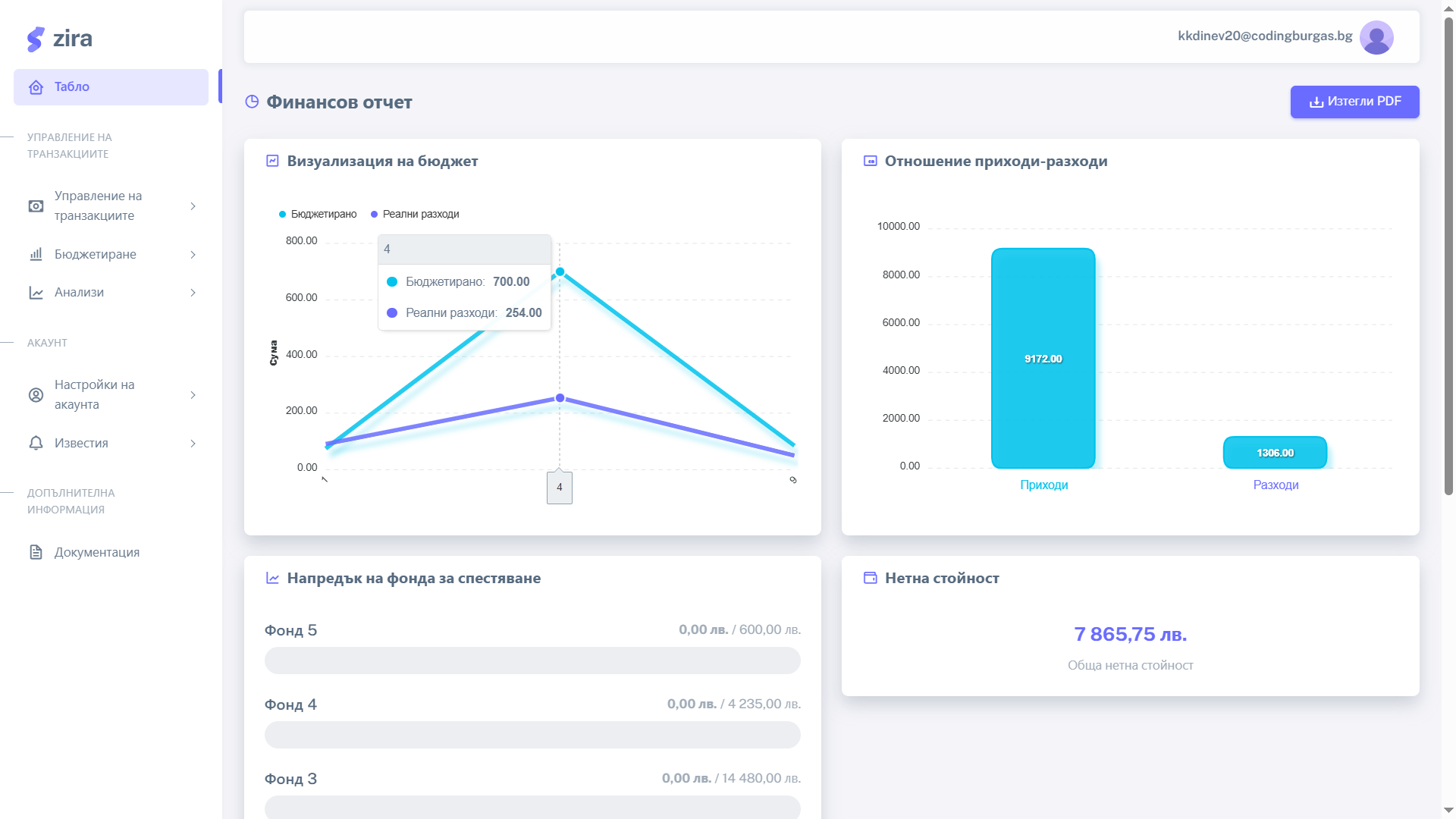
Фигура № 2: Страница, на която потребителят може да види транзакциите си.



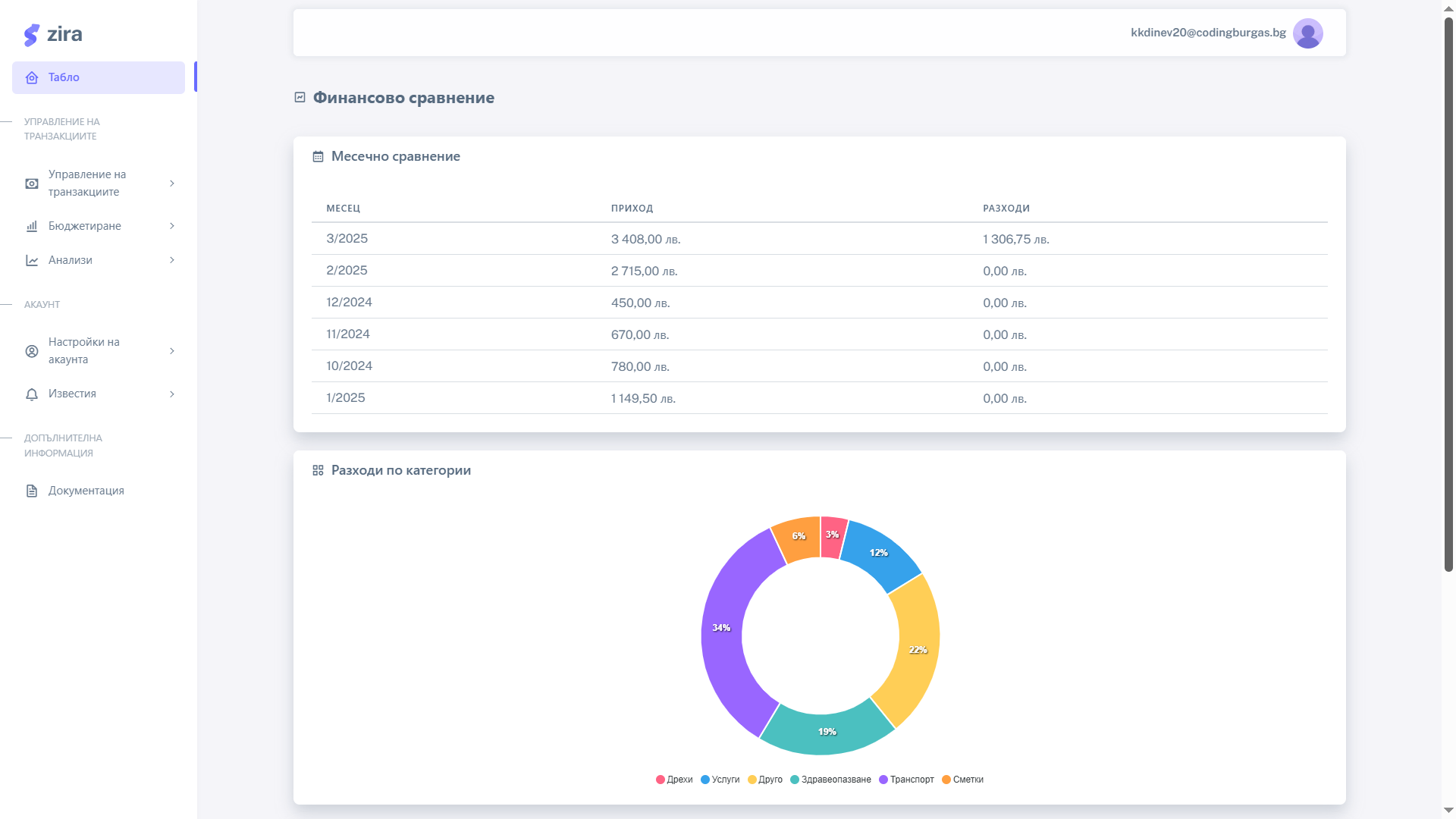
Фигура № 3:Страница, на която потребителят може да проследява спестовните си цели.



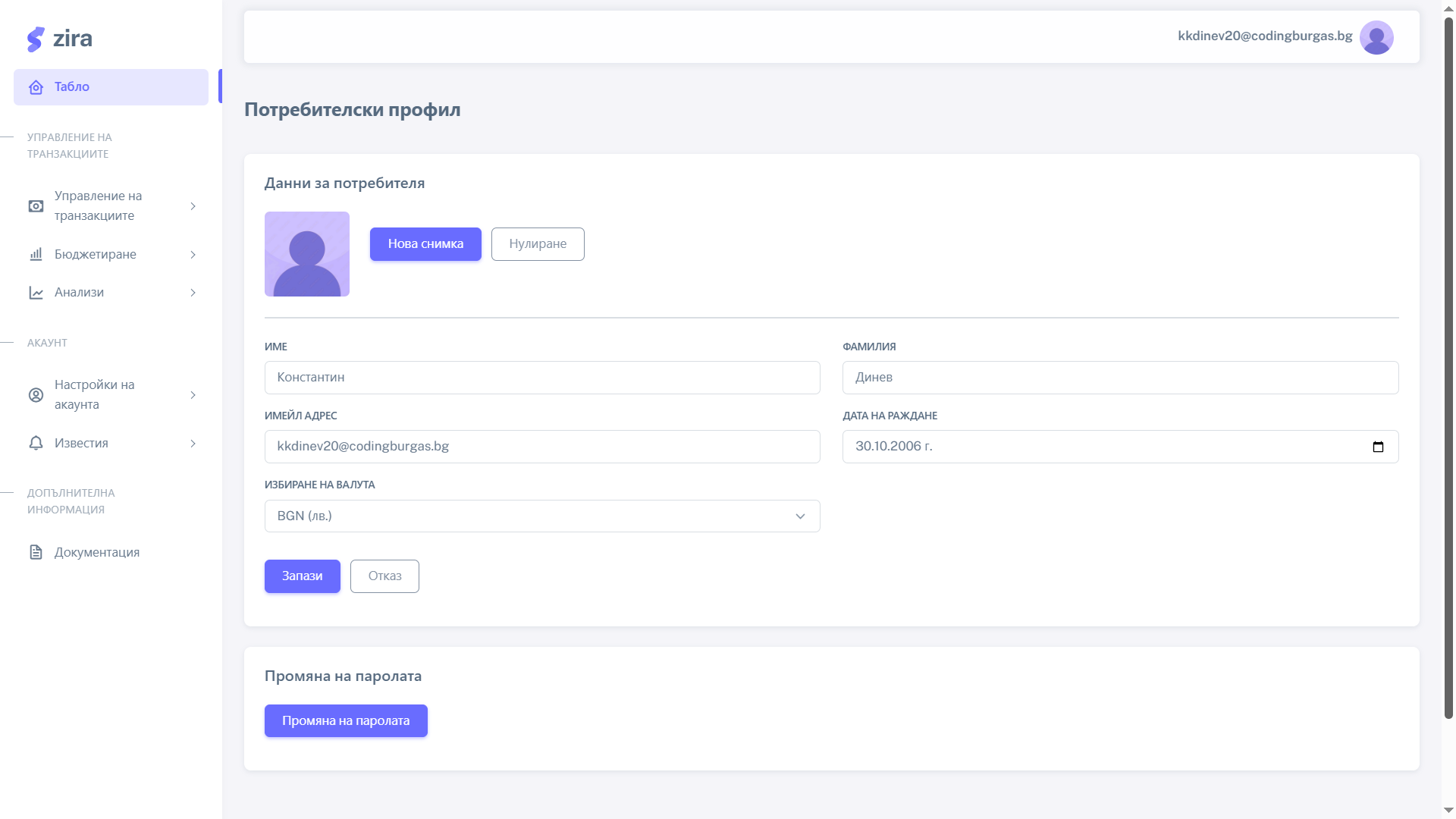
Фигура № 4: Страница, на която потребителят може да проследява състоянието на зададените си бюджети.



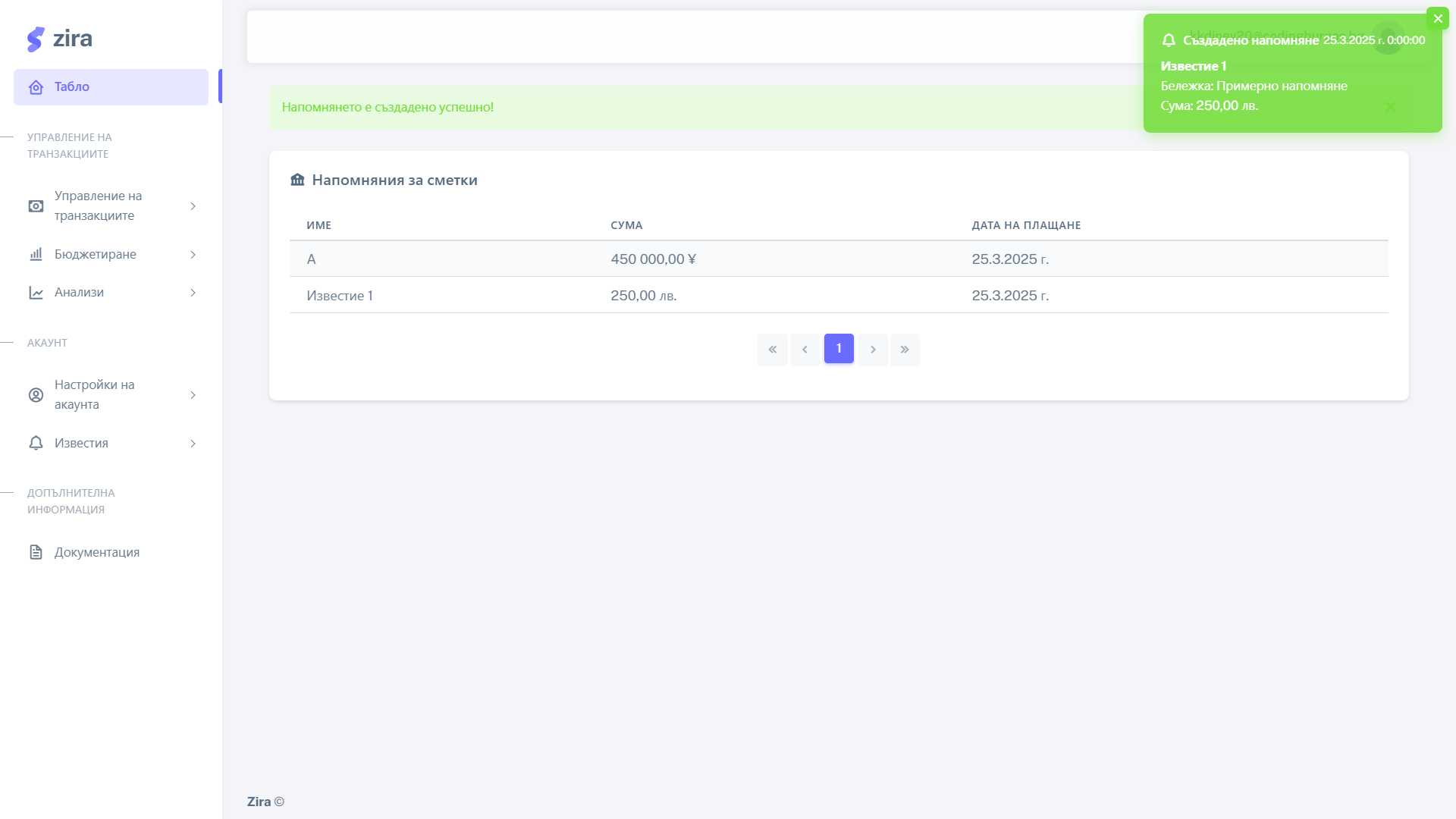
Фигура № 5: Страница, на която потребителят може да следи своето финансово си състояние.



Фигура № 6: Страница, на която потребителят може да види обобщен годишен анализ на своите финанси.



Фигура № 7: Страница за настройките на потребителя.



Фигура № 8: Страница с напомнянията за сметки

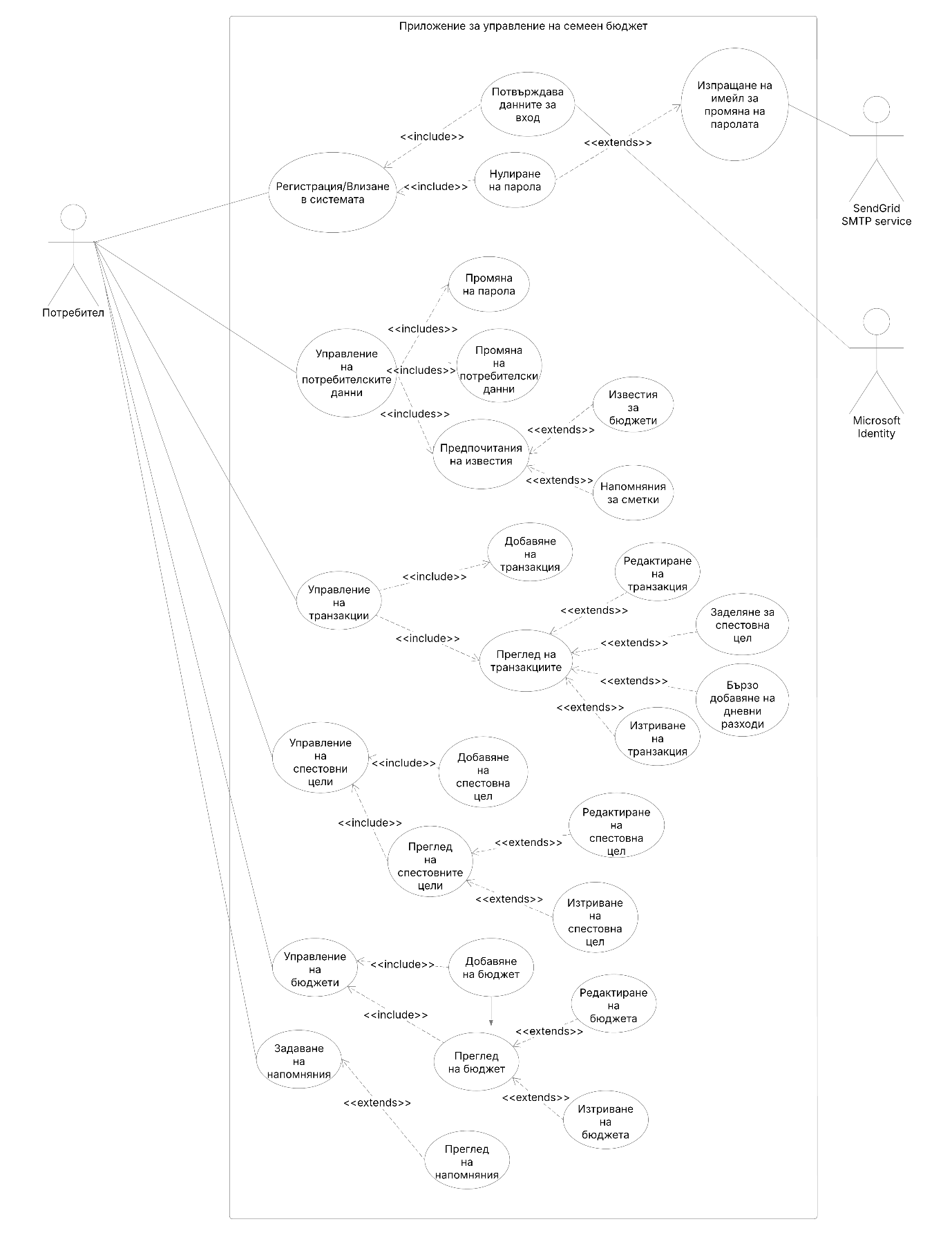
## Диаграми за анализа

UML диаграмите играят ключова роля в процеса на планиране и структуриране на софтуерната система още преди началото на същинската ѝ реализация. Те служат за визуализация на взаимодействията между потребителите и системата, бизнес процесите, както и динамиката на комуникация между различните компоненти. Основните видове диаграми, използвани в проекта, са use case диаграми, activity диаграми и sequence диаграми. Всяка от тях има различна цел и предоставя ценна перспектива към архитектурата на приложението.

Диаграмите на случаите за употреба(Use case) се използват за представяне на основните функции на системата от гледна точка на крайния потребител. Те показват какви действия може да извършва един потребител – например добавяне на транзакция, създаване на бюджет или задаване на спестовна цел – и с кои модули на системата взаимодейства. По този начин се дефинират основните изисквания към системата и се улеснява комуникацията между техническия и нетехническия екип.

Диаграмите на дейноста(Activity Diagrams) илюстрират логическата последователност на действия в рамките на конкретна функционалност. Например, activity диаграмата за създаване на спестовна цел показва стъпките от отваряне на формата до валидиране на входните данни и запис в базата. Те са особено полезни за моделиране на бизнес процесите и улесняват идентифицирането на потенциални проблеми или усложнения.

### **Диаграма на случаи за употреба**



Фигура № 9: Диаграма на случаите за употреба (Use Case)

На представената диаграма на случаите за употреба са изобразени основните взаимодействия между потребителя и приложението. Потребителят има достъп до различни функции, свързани с планиране, проследяване и управление на финансовите ресурси, като системата предоставя удобен начин за въвеждане, категоризиране и анализ на приходи и разходи.

Работният процес включва няколко ключови модула. Потребителят може да управлява своя профил, да добавя и редактира финансови транзакции, да задава бюджетни граници и да следи изпълнението на бюджета чрез различни отчети и анализи. Системата предлага разширени възможности като автоматични известия при достигане на бюджетни лимити, интеграция със SMTP услуга (SendGrid) за изпращане на напомняния и Microsoft Identity за сигурна автентикация.

Диаграмата визуализира цялостния процес на управление на семейния бюджет чрез ясно разделени функционални блокове. Всеки случай на употреба представя конкретно действие, което потребителят може да извърши, а използваните връзки <<includes>> и <<extends>> показват зависимостите между различните функции. Това позволява на системата да осигури гъвкава и ефективна платформа за управление на личните финанси, подобрявайки организацията и финансовата дисциплина на потребителите*.*

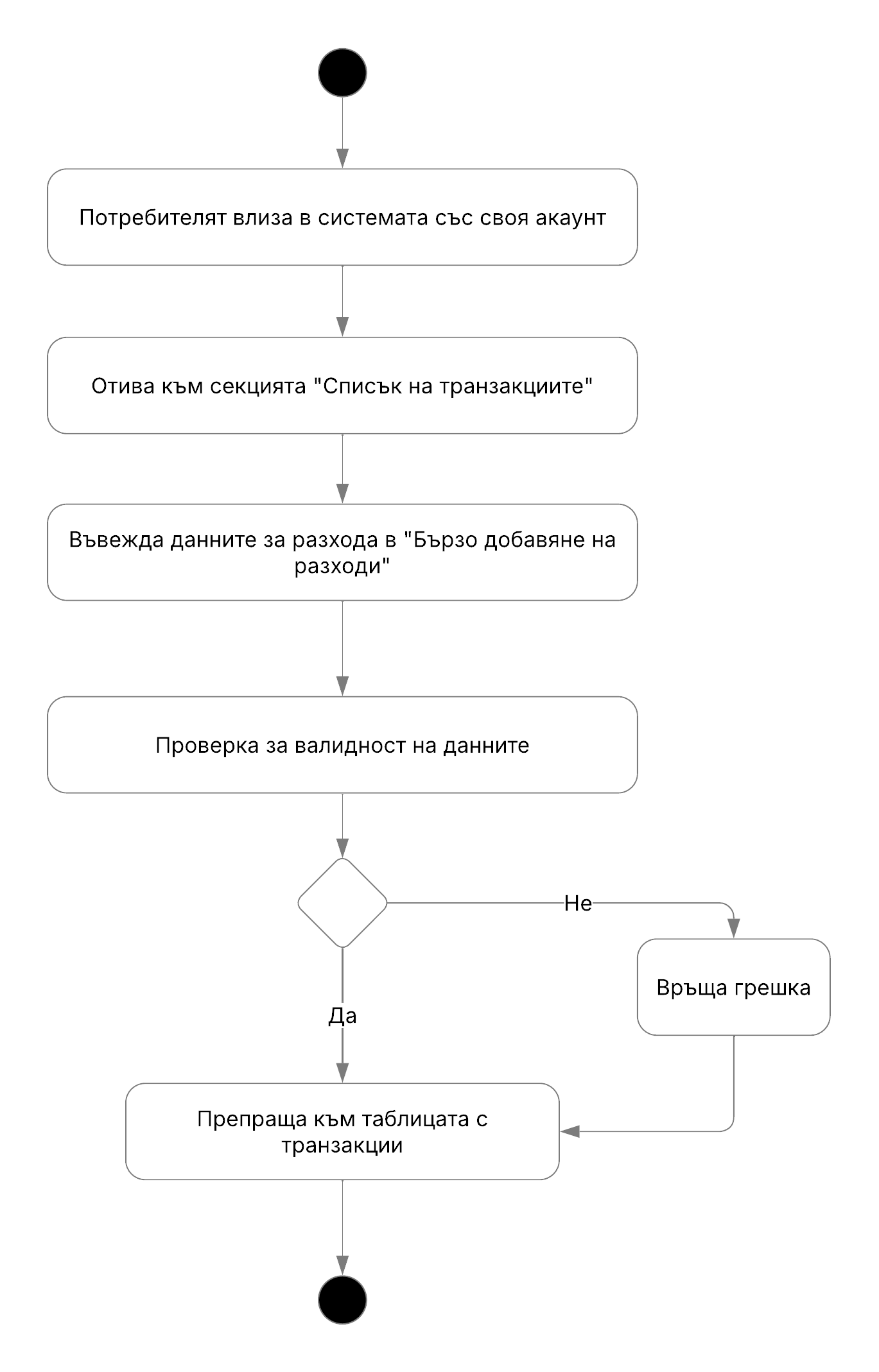
### **Диаграма на дейността**

**Добавяне на транзакция:** Позволява добавяне на приход или разход. Стъпки на изпълнение:



Фигура № 10: Диаграма на дейността №1(Activity diagram)

**Бързо добавяне на разходи:** Опция за директно добавяне на днешен разход. Стъпки на изпълнение:



Фигура № 11: Диаграма на дейността №2(Activity diagram)

**Редактиране на транзакция:** Потребителят може да промени данни по дадена транзакция. Стъпки на изпълнение:



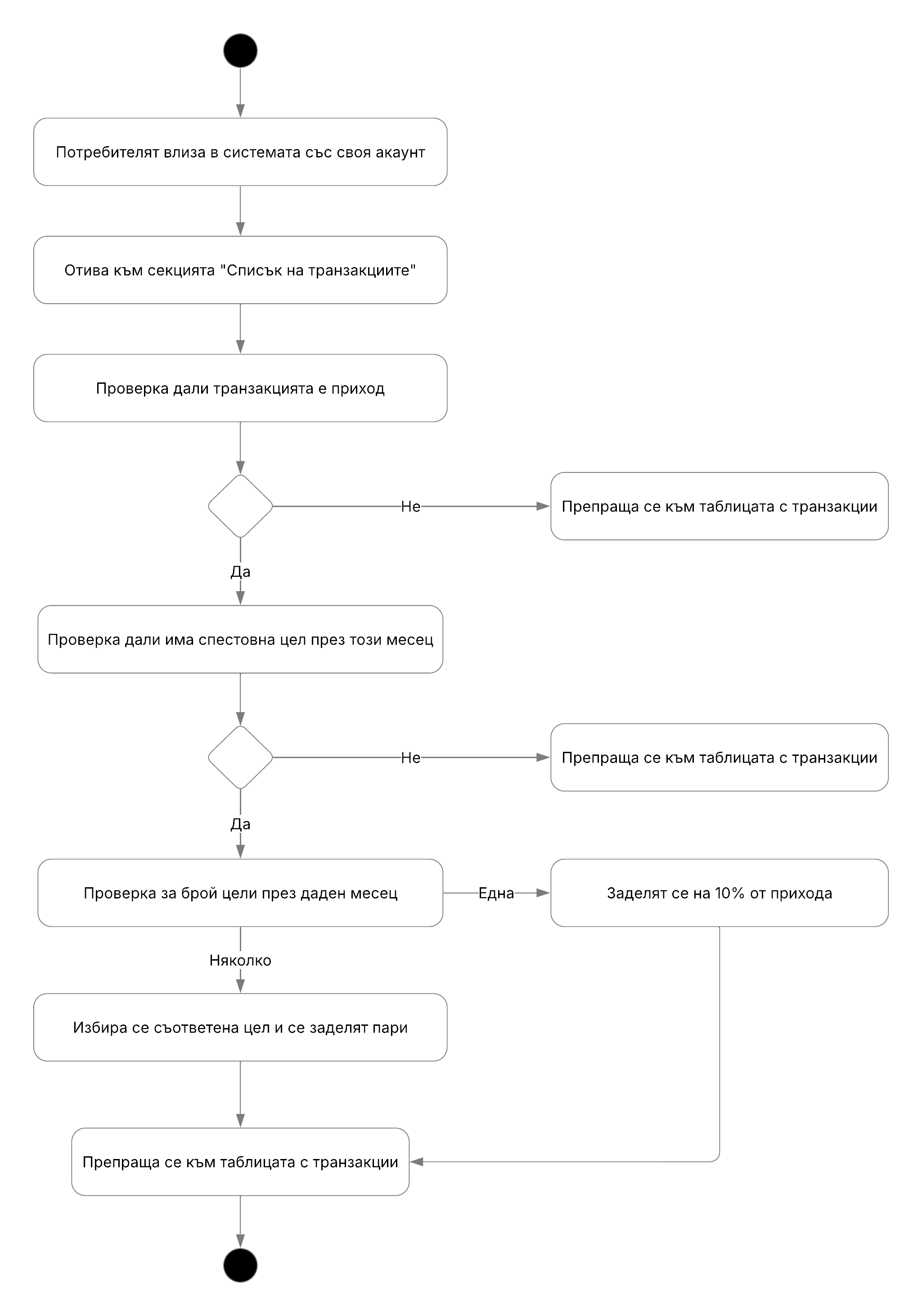
Фигура № 12: Диаграма на дейността №3(Activity diagram)

**Изтриване на транзакция:** Потребителят може да изтрие директно своя транзакция. Стъпки на изпълнение:



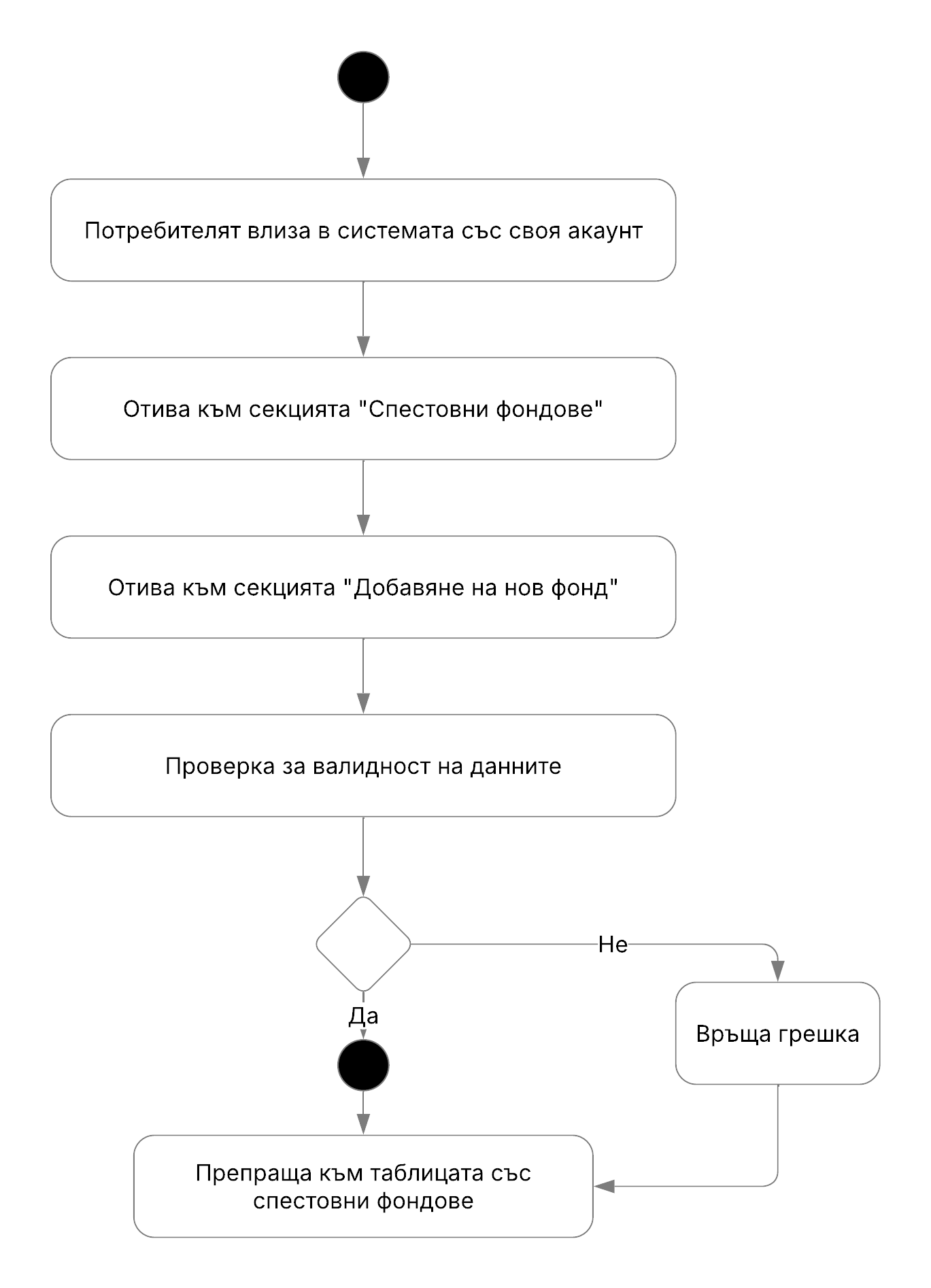
Фигура № 13: Диаграма на дейността №4(Activity diagram)

**Заделяне на приход за спестовна цел:** Ако дадена транзакция е приход, то част от нея може да бъде заделена за спестовен фонд в дадения месец. Стъпки на изпълнение:



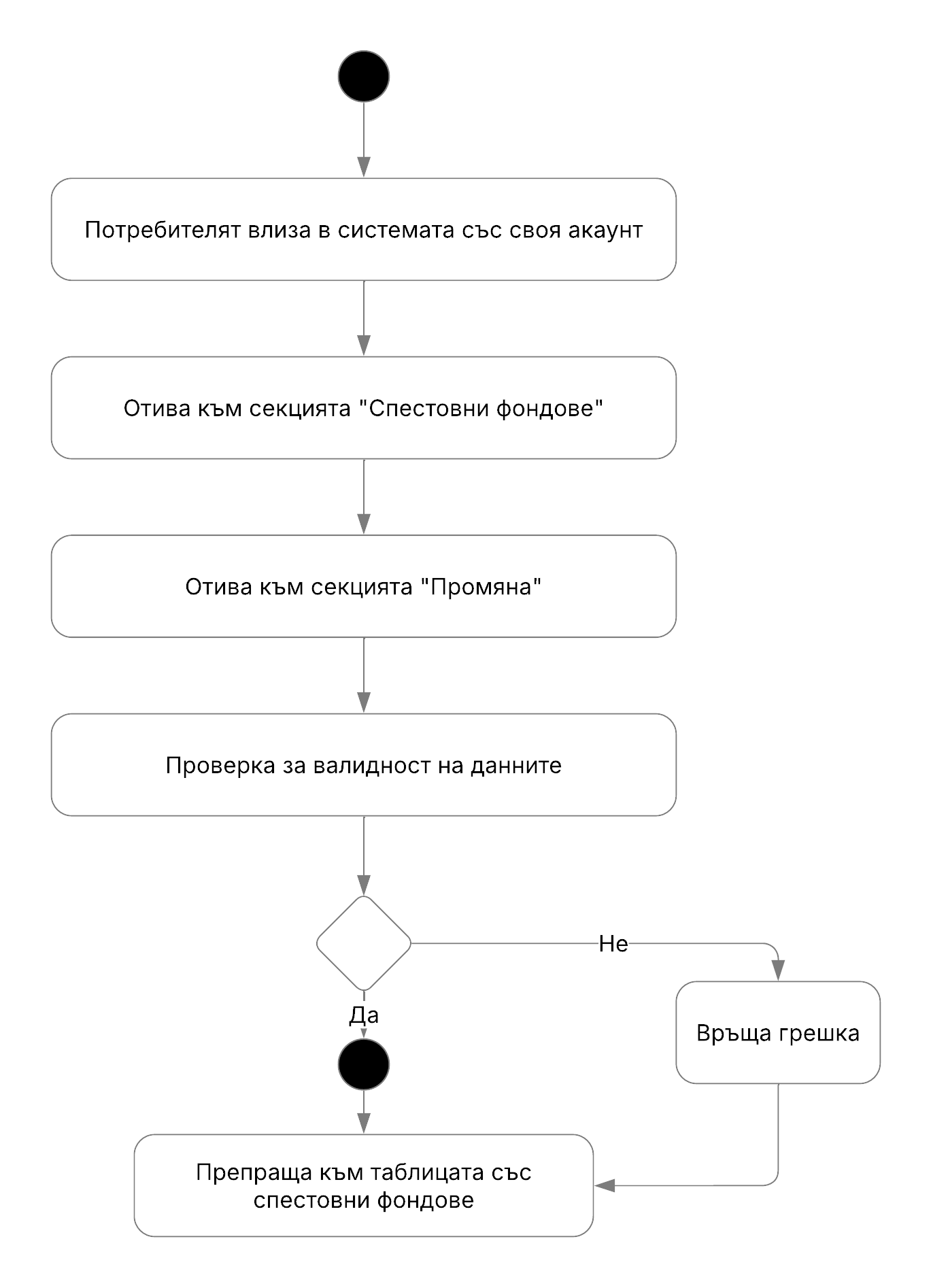
Фигура № 14: Диаграма на дейността №5(Activity diagram)

**Добавяне на спестовни цели:** Позволява добавяне на спестовна цел. Стъпки на изпълнение:



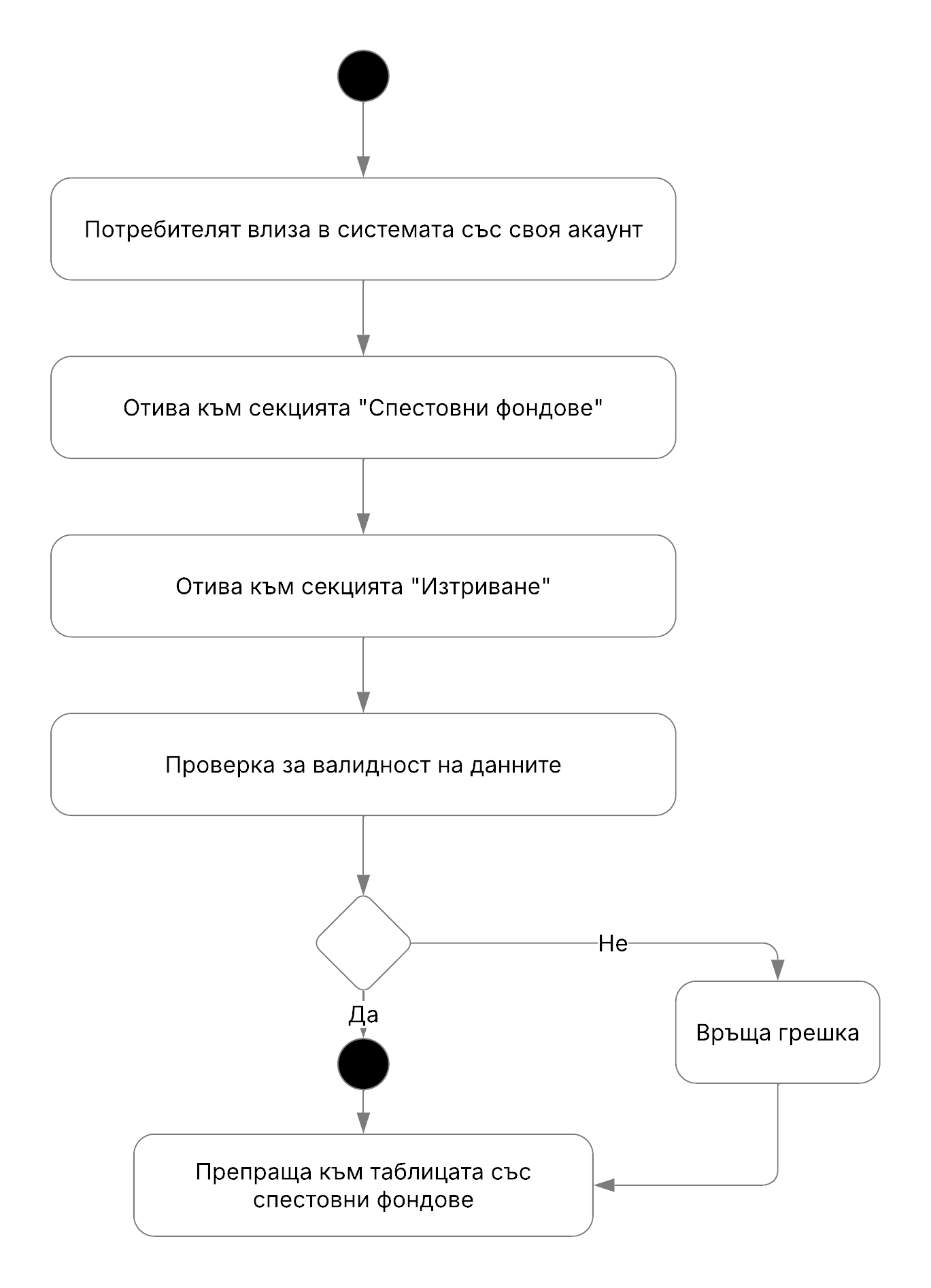
Фигура № 15: Диаграма на дейността №6(Activity diagram)

**Редактиране на спестовна цел:** Потребителят може да промени данни по дадена спестовна цел. Стъпки на изпълнение:



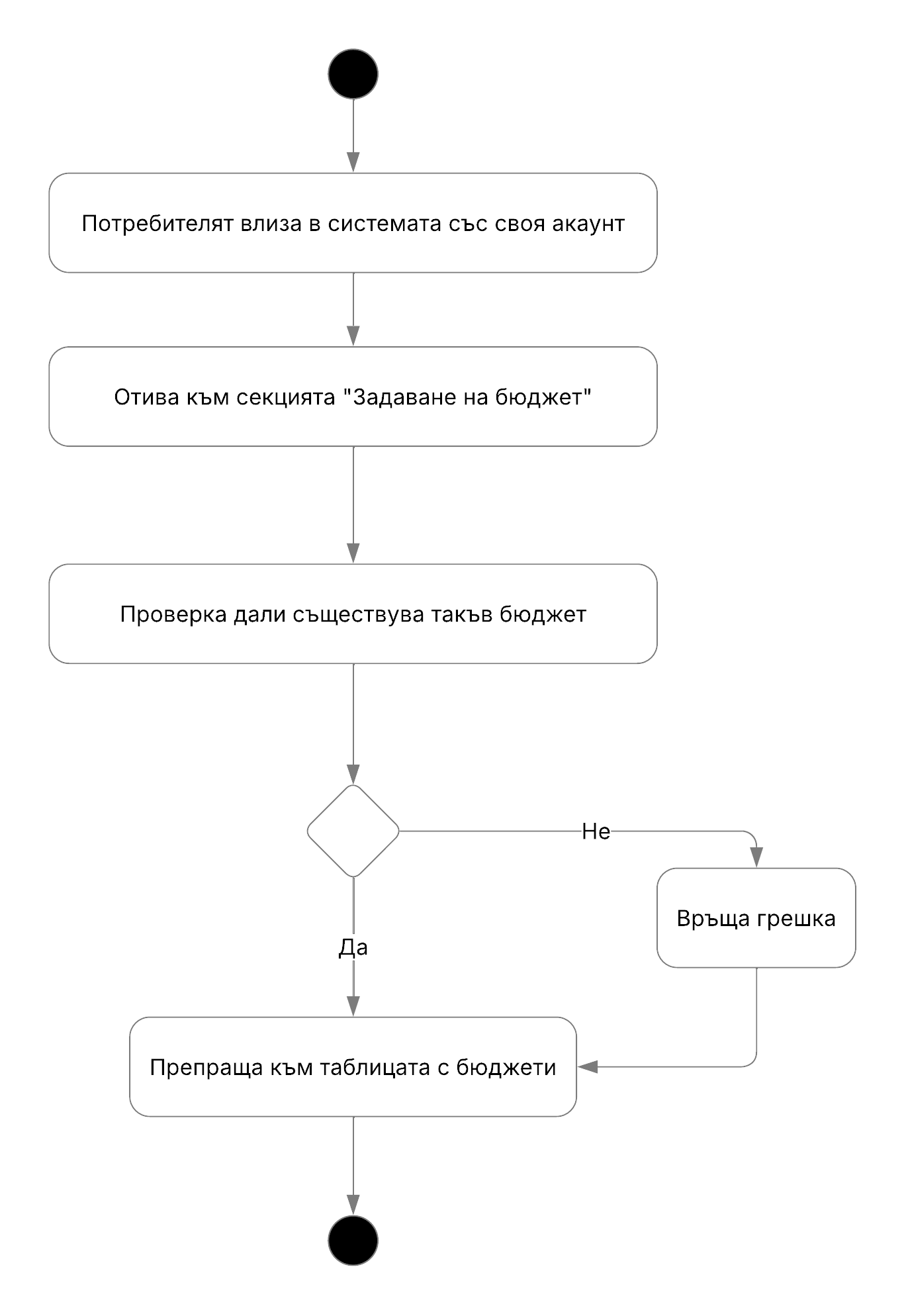
Фигура № 16: Диаграма на дейността №7(Activity diagram)

**Изтриване на спестовна цел:** Позволява премахване на съществуваща спестовна цел. Стъпки на изпълнение:



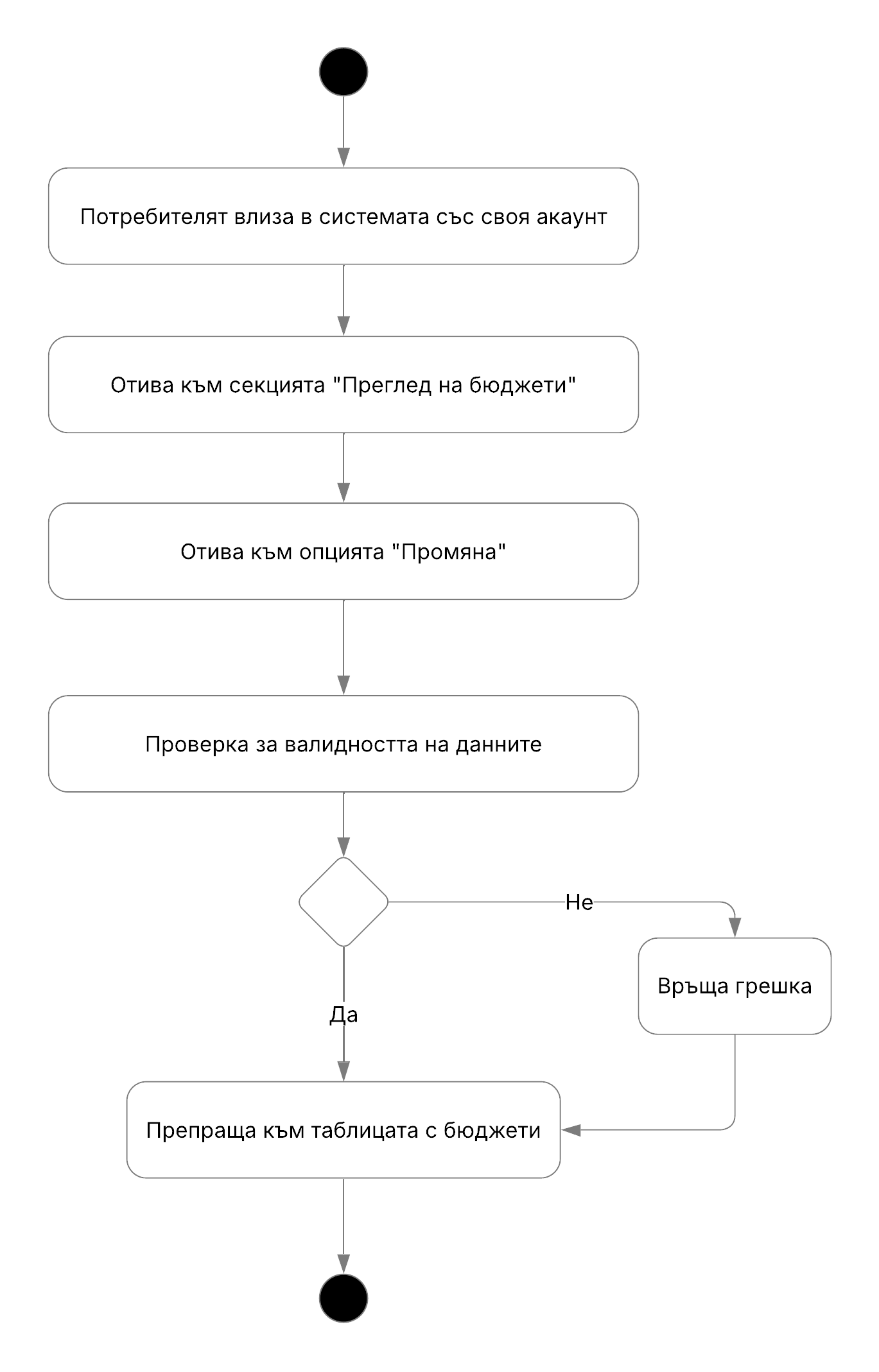
Фигура № 17: Диаграма на дейността №8(Activity diagram)

**Добавяне на бюджет:** Позволява на потребителите да създават нов бюджет за дадена категория. Стъпки на изпълнение:



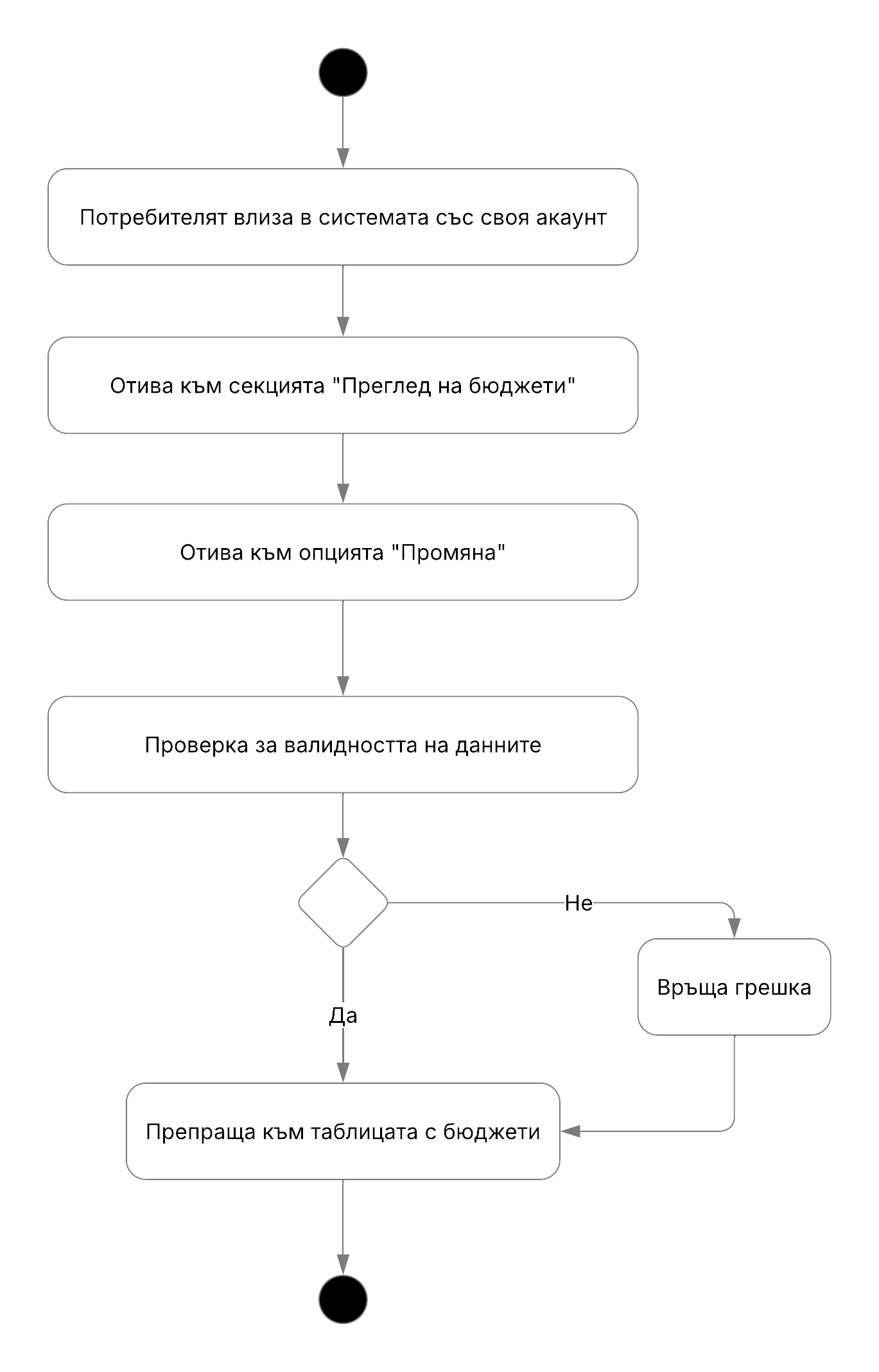
Фигура № 18: Диаграма на дейността №9(Activity diagram)

**Редактиране на бюджет:** Позволява промяна на вече съществуващ бюджет. Стъпки на изпълнение:



Фигура № 19: Диаграма на дейността №10(Activity diagram)

**Изтриване на бюджет:** Потребителят може да изтрие директно своя бюджет. Стъпки на изпълнение:



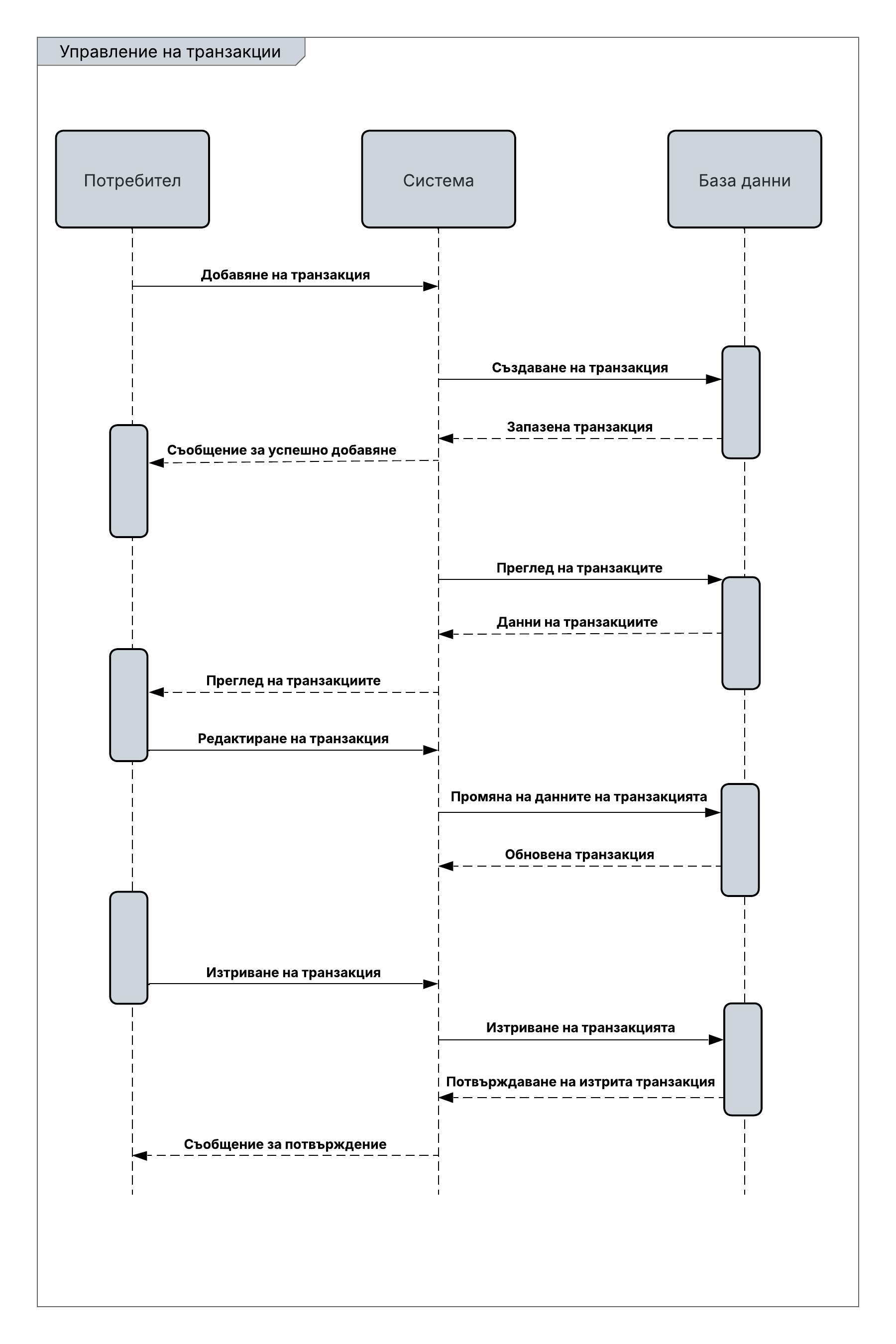
Фигура № 20: Диаграма на дейността №11(Activity diagram)

**Добавяне на напомняния:** Позволява на потребителите да създават напомняния за финансови задачи и бъдещи сметки. Стъпки на изпълнение:

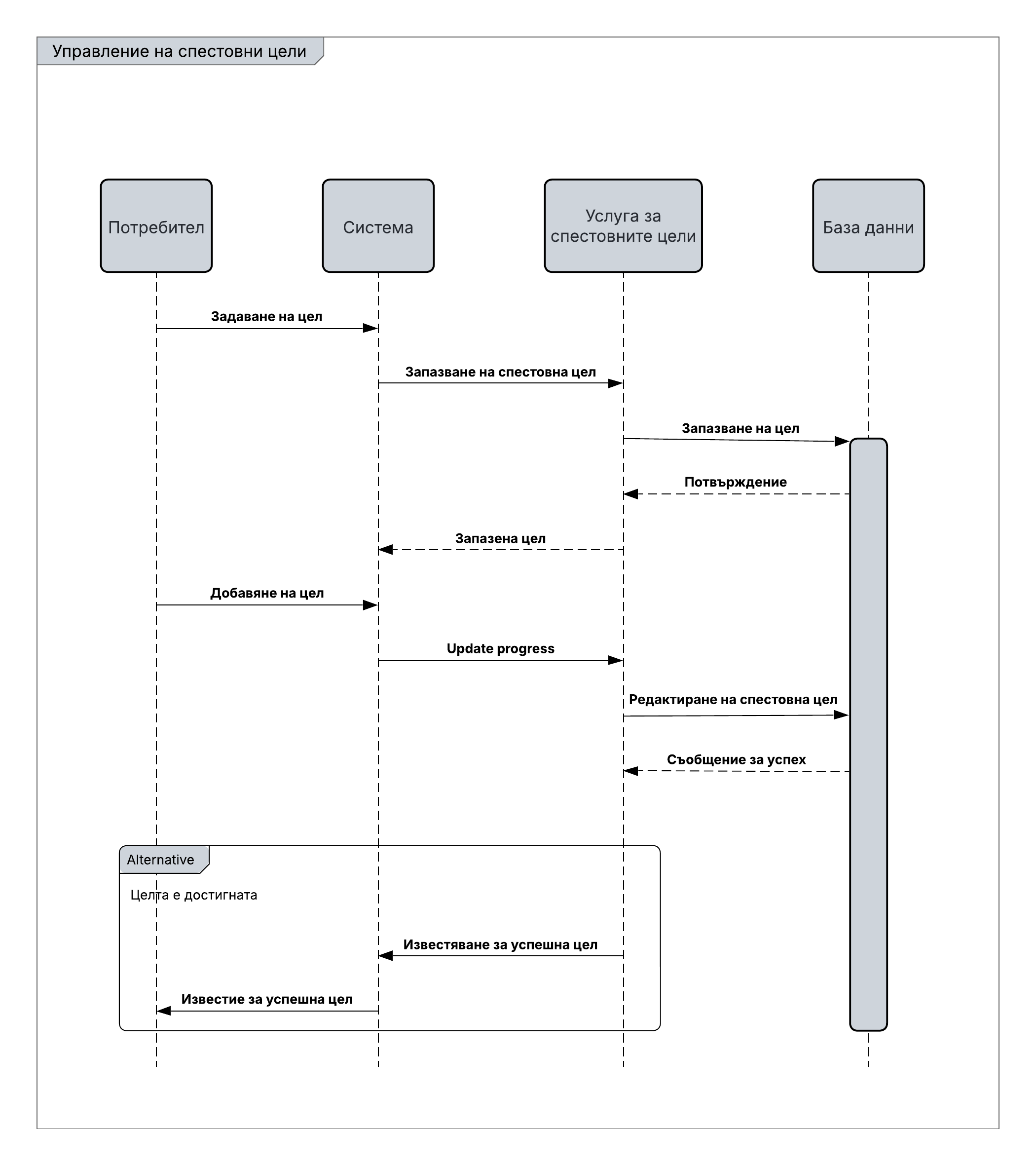


Фигура № 21: Диаграма на дейността №12(Activity diagram)

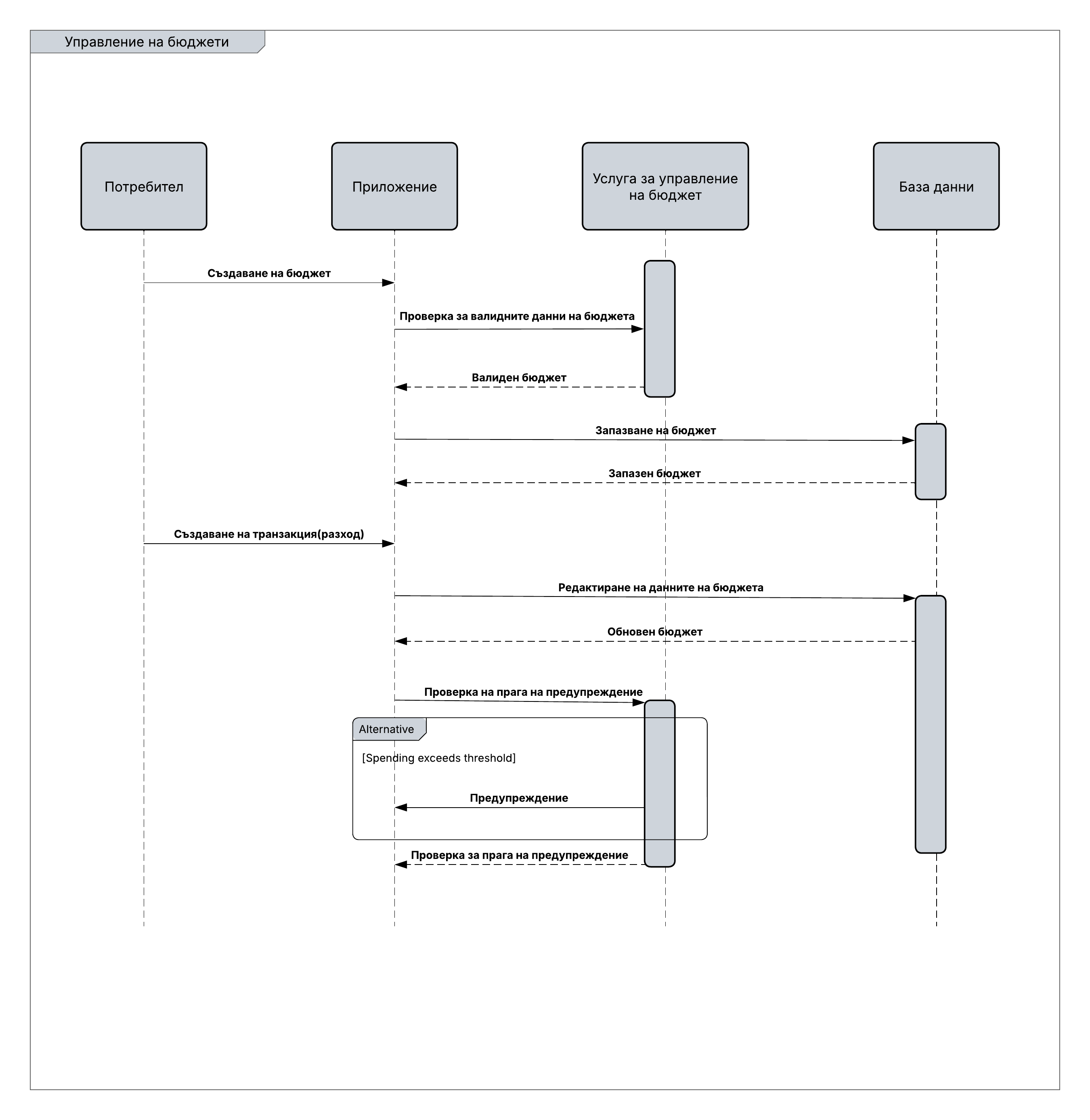
### **Диаграма на последователността**



Фигура № 22: Диаграма на последователността (Sequence diagram)



Фигура № 23: Диаграма на последователността (Sequence diagram)



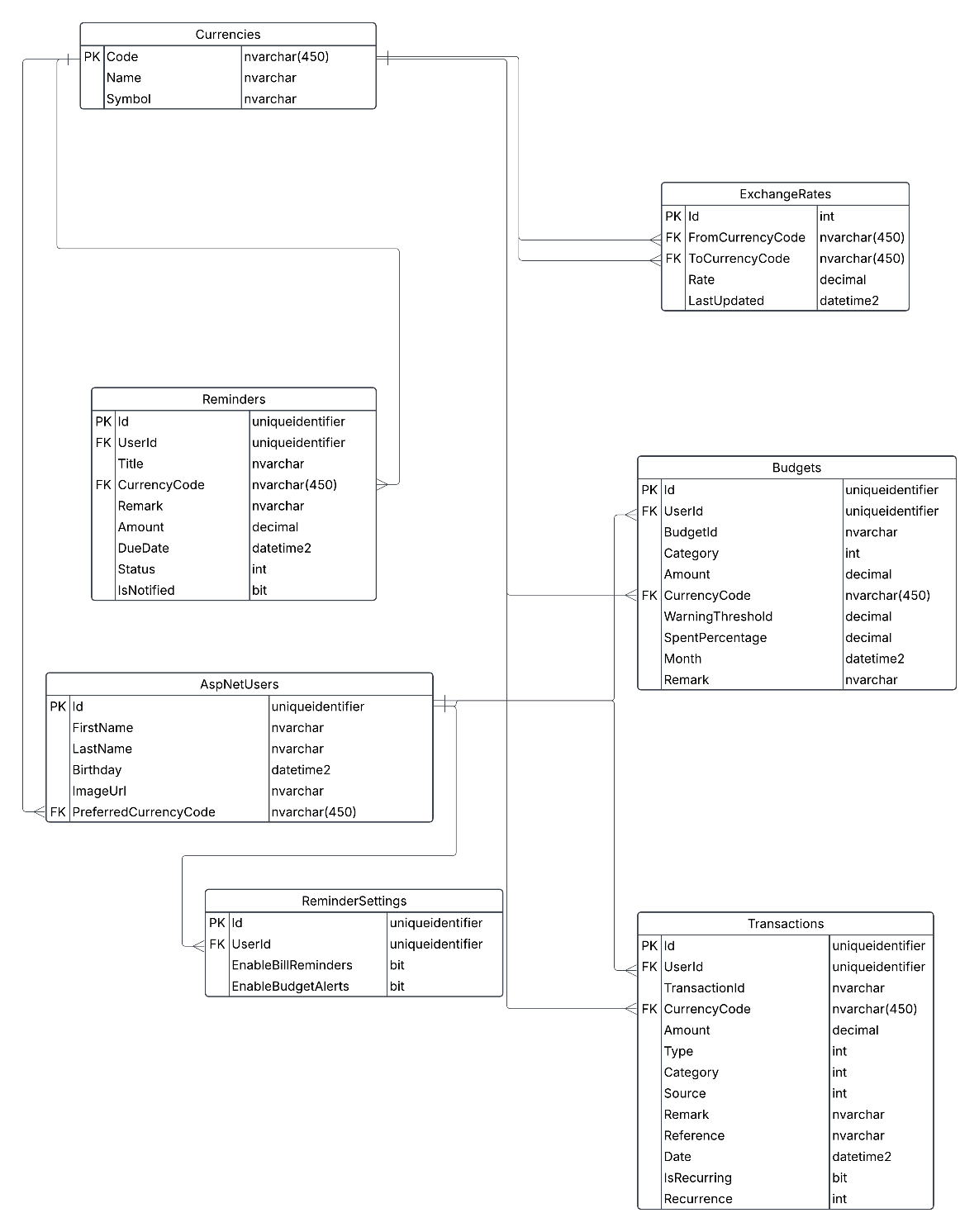
Фигура № 24: Диаграма на последователността (Sequence diagram)

Диаграмите на последователността (Sequence Diagrams) показват взаимодействието между потребителите, системата и базата данни във времето. Те илюстрират как заявките преминават през различни компоненти, като подчертават реда на операциите. [1]

В случая с управлението на спестявания и бюджети, диаграмите описват CRUD операциите (създаване, четене, актуализация и изтриване), като проследяват как потребителят взаимодейства със системата, а тя от своя страна комуникира с базата данни. Това е полезно за визуализиране на логиката на изпълнение и за откриване на потенциални проблеми в работния процес.

## Модел на съдържанието/данните

### **Диаграма на отношенията между моделите**



Фигура № 25: Диаграма на отношенията между моделите (Entity Relation)

### **Нормализация**

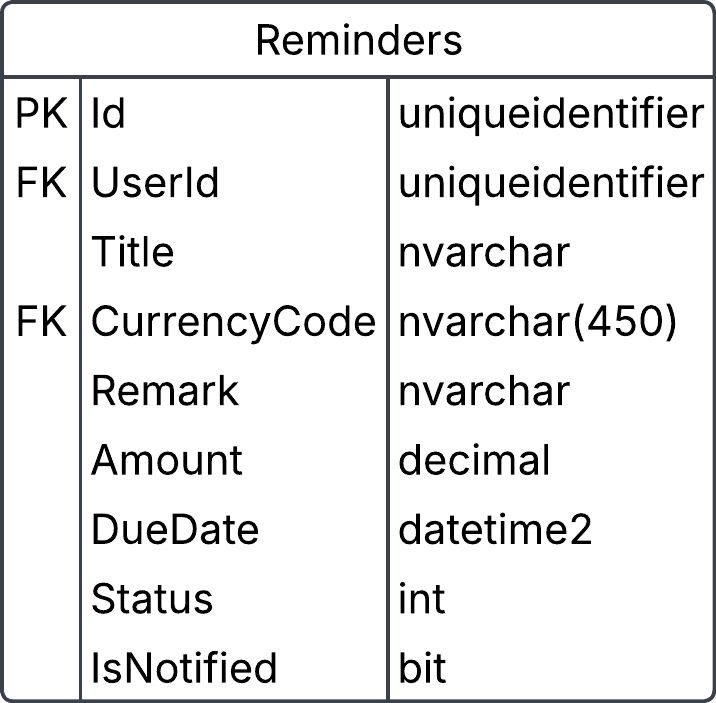
Базата данни представлява съвкупност от таблици, свързани релационно чрез множество кардинални връзки. Тя е нормализирана до трета нормална форма, което ще бъде демонстрирано с обяснение на процеса. Нормализацията е метод за организиране на релационни бази от данни, който има за цел да запази целостта на данните и да намали тяхното дублиране. Един запис не трябва да се повтаря, а отношенията между таблиците се изграждат така, че да предотвратят излишно повторение. [2]

Съществуват повече от три нормални форми, но проектът е нормализиран до трета, тъй като тя се счита за стандартна. Допълнителните нормални форми биха усложнили значително процеса на извличане на данни. Всяка нормална форма изисква изпълнение на условията на предходната, което създава инкрементален процес на зависимости.

За да отговаря на изискванията на първа нормална форма, една таблица трябва да спазва няколко ключови правила. На първо място, тя задължително трябва да притежава първичен ключ, осигуряващ уникалност на всеки запис. Освен това, стойностите в отделните редове трябва да бъдат атомарни – да не се съдържат списъци или множество стойности. Всеки запис в таблицата трябва да бъде уникален, като именно първичния ключ го гарантира.

Например таблицата Reminders съдържа информация за всички известия, които потребителя е задал – напомняния за сметки. Ако се вземе една примитивна версия на таблицата, тя би изглеждала така:

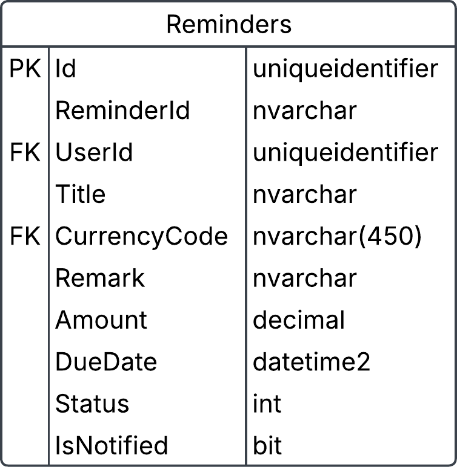
Фигура № 26: Диаграма на не нормализирана таблица



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Иден. номер | Иден. Номер на  Потр. | Заглавие | Валутен код | Бележка | Сума | Дата на плащане | Статус | Дали е известен |
| 0000-0000-0000 | 0000-0000-0000 | Известие | BGN | Бележка 1 | 150 | 08-04-2025 | 1(Pending) | False |

В показаната таблица и примерен запис липсва колона, която да идентифицира еднозначно напомнянията, тъй като те могат да бъдата редактирани и променяни. Колоната, която съдържа информация за напомнянията, включва различни атрибути като заглавие, бележка, статус. Това нарушава принципа на първа нормална форма за атомарност. За да бъде постигнато съответствие с това правило, може да се добави нова колона ReminderId, която да гарантира уникалност на записите. Това поле се генерира на база датата на създаване на записа до секундата, което предотвратява дублирането на данни в таблицата. С тази промяна структурата на таблицата ще бъде подобрена:

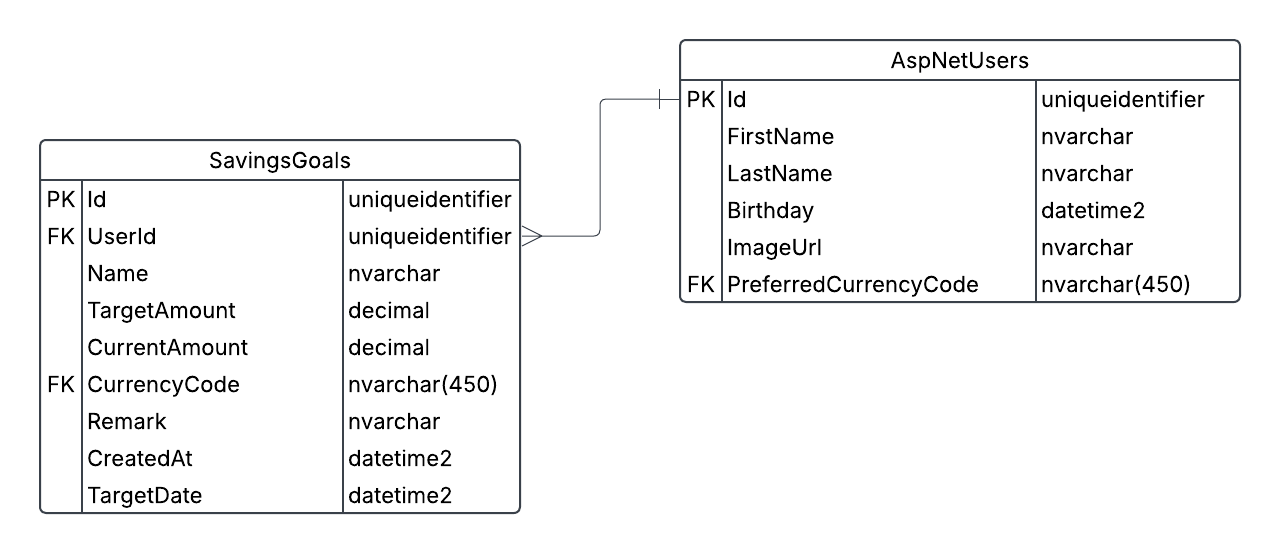
Фигура № 27: Диаграма на таблицата в първа нормална форма



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Иден. номер | Номер на известие | Иден. Номер на  Потр. | Заглавие | Валутен код | Бележка | Сума | Дата на плащане | Статус | Дали е известен |
| 0000-0000-0000 | #202503 | 0000-0000-0000 | Известие | BGN | Бележка 1 | 150 | 08-04-2025 | 1(Pending) | False |

Преминаването към втора нормална форма(2NF) изисква първо изпълнение на условията на първа нормална форма. След като това е постигнато, следващото правило е, че всички атрибути на таблицата трябва да бъдат функционално зависими от първичния ключ. Примерна таблица, която удовлетворява изискванията на 2NF:[]

Фигура № 28: Диаграма на таблица във втора нормална форма (2NF)



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Иден. номер | Иден. Номер на  Потр. | Име | Целева сума | Текуща сума | Валутен код | Целева дата |
| 0000-0000-0000 | 0000-0000-0000 | Цел 1 | 150 | 100 | BGN | 04-03-2025 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Иден. номер | Първо име | Последно име | Рожденна дата | Валутен код |
| 1 | Иван | Иванов | 06-06-2006 | EUR |

След като втора нормална форма е успешно приложена, следваща стъпка е постигането на трета нормална форма. Трета нормална форма гласи следното – таблицата трябва да е във втора нормална форма, както и да бъде организирана така, че промяната на стойност в една колона не влияе на стойността в друга колона. Това означава, че не трябва да съществуват транзитивни връзки между атрибутите.

Транзитивността представлява две точки или, в този случай, две колони, свързани с една обща колона, взаимно да си влияят. Ако се използва пример с три колони – A, B и C, при който A е свързана с B, а B е свързана с C, то тогава A индиректно зависи от C. Това представлява транзитивна зависимост, която трябва да е премахната, за да се гарантира нормализацията на данните. [3]

След като всички такива зависимости са отстранени, може да се заключи, че таблиците са нормализирани в трета нормална форма (3NF). По аналогичен начин, ако същият подход бъде приложен към останалите таблици в базата, ще постигнем пълна нормализация на цялата структура на базата данни.

### **Пояснения за таблиците**

Приложението се състои от 15 таблици, като повечето са нормализирани до трета нормална форма (3NF).

Таблиците, отговарящи за потребителя – AspNetRoleClaims, AspNetRoles, AspNetUserClaims, AspNetUsers -, са генерирани от Identity функционалността на ASP.NET.

Таблицата AspNetUsers съдържа допълнителна информация за потребителя – имена, дата на раждане, имейл, др. Има връзки с почти всички таблици в базата от тип едно-към-много.

Таблицата Currencies има само 7 записа – валутите, които системата поддържа. Всички таблици са свързани с нея, защото всеки запис може да има различен тип валута.

Таблицата ExchangeRates съдържа курсовете за обмени от един тип до друг. Тя е свързана само с Currencies.

Таблицата Transactions записва всички потребителски приходи и разходи, включително сума, дата, категория, бележки и връзки към AspNetUsers и Currencies.

Таблицата Budgets съдържа информация за зададените бюджети на потребителя, като лимит, категория, период и текущо изразходваната сума. Тя е свързана с AspNetUsers и Currencies.

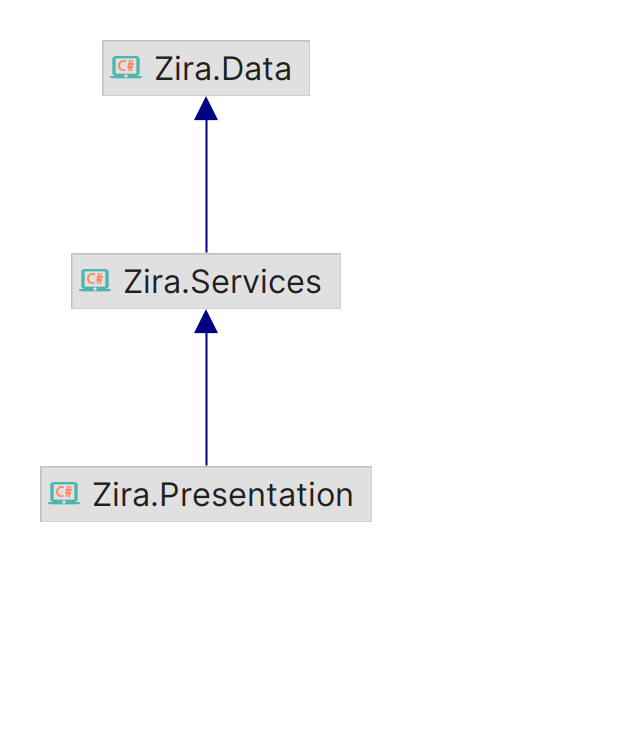
Таблицата SavingsGoals представя спестовните цели, зададени от потребителя. Включва начална сума, целева сума и статус. Свързана е с AspNetUsers и Currencies.

Таблицата Reminder съхранява данни за напомнянията, създадени от потребителя, като заглавие, сума, дата на изпълнение и статус.

Таблицата ReminderSettings съхранява потребителските предпочитания за известия.

# **Дизайн**

Езикът за разработка е C# и проектът е основан върху .NET платформата. Използва се версия 8 за .NET както и за другите библиотеки, използвани в проекта. Разделен е на 3 слоя – слой за бизнес логиката, слой за достъпа до базите данни, презентационен слой и 4 допълнителни проекти – за тестове и общи файлове.



Фигура № 29: Диаграма на отношенията между главните проекти

* Презентационен слой – това е слоят, чрез който потребителя достъпва и работи с другите два слоя. Данните биват представяни в интерактивен и удобен за използване формат. Състои се от един главен проект:
  + Zira.Presentation – Това е основното Razor MVC приложение, което позволява на потребителите да добавят и обработват своите приходи и разходи.
* Слой за бизнес логиката – това е слоят, в който се съдържа бизнес логиката в приложението. Свързан е със слоя за достъп до базите данни, за да може да работи с тях. В него има един проект:
  + Zira.Services – В него се съдържат услуги относно създаването на потребителските акаунти, създаването и разпространението на известия относно достигането и надвишаването на месечните бюджети, както и даването на месечен отчет за разходите. Използва интерфейси и услуги за взаимодействие с другите класове.
* Слой за данните – в слоя за данните се съдържа инициализацията на базите данни, таблиците в нея и техни взаимовръзки. Има следния слой:
  + Zira.Data – слоят за данни на приложението, който съдържа контекстът и моделите. Работи с базата данни чрез Entity Framework Core(ORM библиотека).
* В приложението се съдържат още четири проекта:
  + Zira.Common – Съхраняват се в него локализирани думи, с цел превод на сайта на български език.
  + Zira.Services.Tests – Този проект отговаря за тестването на слоя за бизнес логиката.
  + Zira.Presentation.Tests – Този проект отговаря за тестването на презентационния слой.
  + Zira.EndToEndTests – Този проект отговаря за сигурността на личните данни на потребителя.

За да може приложението да бъде напълно функционално, се използват набор от зависимости(dependencies), които улесняват и подобряват разработката на софтуера:

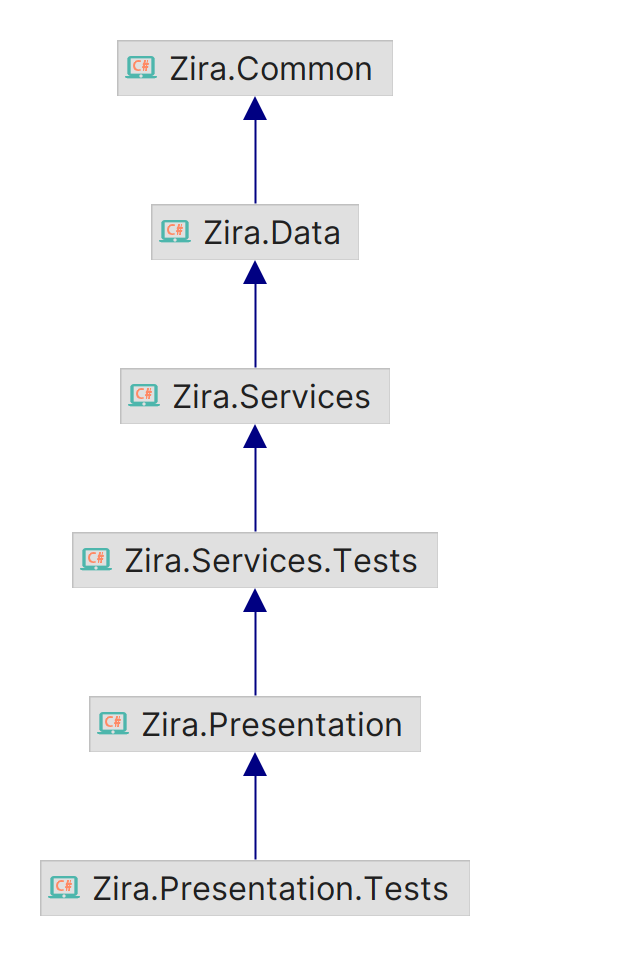
* Entity Framework Core 8(EF Core 8) - ORM технология, която улеснява работата с базата данни. Проектът е изграден по принципа Code First, като първо се дефинират моделите, след което се генерират таблиците с необходимите полета, типове и релации
* EF Core 8 InMemoryDatabase – Разширение на EF Core 8, което позволява използването на временна база данни в оперативната памет, без необходимост от външен сървър. Подходящо е за тестване, тъй като методите на EF Core 8 трудно се заменят чрез стандартни mock библиотеки като Moq. Този метод се препоръчва от Microsoft като по-ефикасно решение вместо използването на фалшиви обекти. [4]
* SendGrid – Инструмент за изпращане на имейли директно от приложението към потребителите.
* xUnit – Тестова рамка за модулно тестване на бизнес логиката и достъпа до базата данни.
* Stylecop Analyzers – Анализатор, който проверява C# кода за съответствие с установени стилови и структурни правила
* Moq – Библиотека, която позволява създаването на изолирани тестове чрез заместване на външни зависимости.
* SignalR – Инструмент за реална комуникация между клиента и сървъра чрез уеб връзки.
* BenchmarkDotNet – Библиотека, използвана за измерване на производителността на различни части от кода.

Използвани шаблони за дизайн (Design Patterns) в приложението:

* Разпределение на отговорностите(Separation of Concerns) – Разграничаване на логическите части на приложението, така че всяка от тях да има ясна и самостоятелна функционалност.
* Инжектиране на зависимости / Обръщане на контрол (Dependency Injection / Inversion of control) – Използва се за управлението на зависимости между компонентите, което улеснява поддръжката и тестването на приложението.
* Стратегия(Strategy Pattern) – позволява избор на различни алгоритми и методи (стратегии) по време на изпълнение. Използва се при методите за преобразуван е от един тип в друг тип валута в бизнес слоя.
* Разделяне на отговорностите при командни заявки(CQRS) – при този шаблон се разделят четенето(Queries) и обработката на данни(Commands) в приложението. [5] Например, в бизнес логиката има методи за четене – GetUserBudgetsAsync(), GetSavingsGoalsAsync() , а също и методите за записване и обработка – AddTransactionAsync(), CreateSavingsGoalAsync().

На следващата страница е представена диаграма, която описва проектите и връзките между тях:

## Реализация на архитектурата на приложението

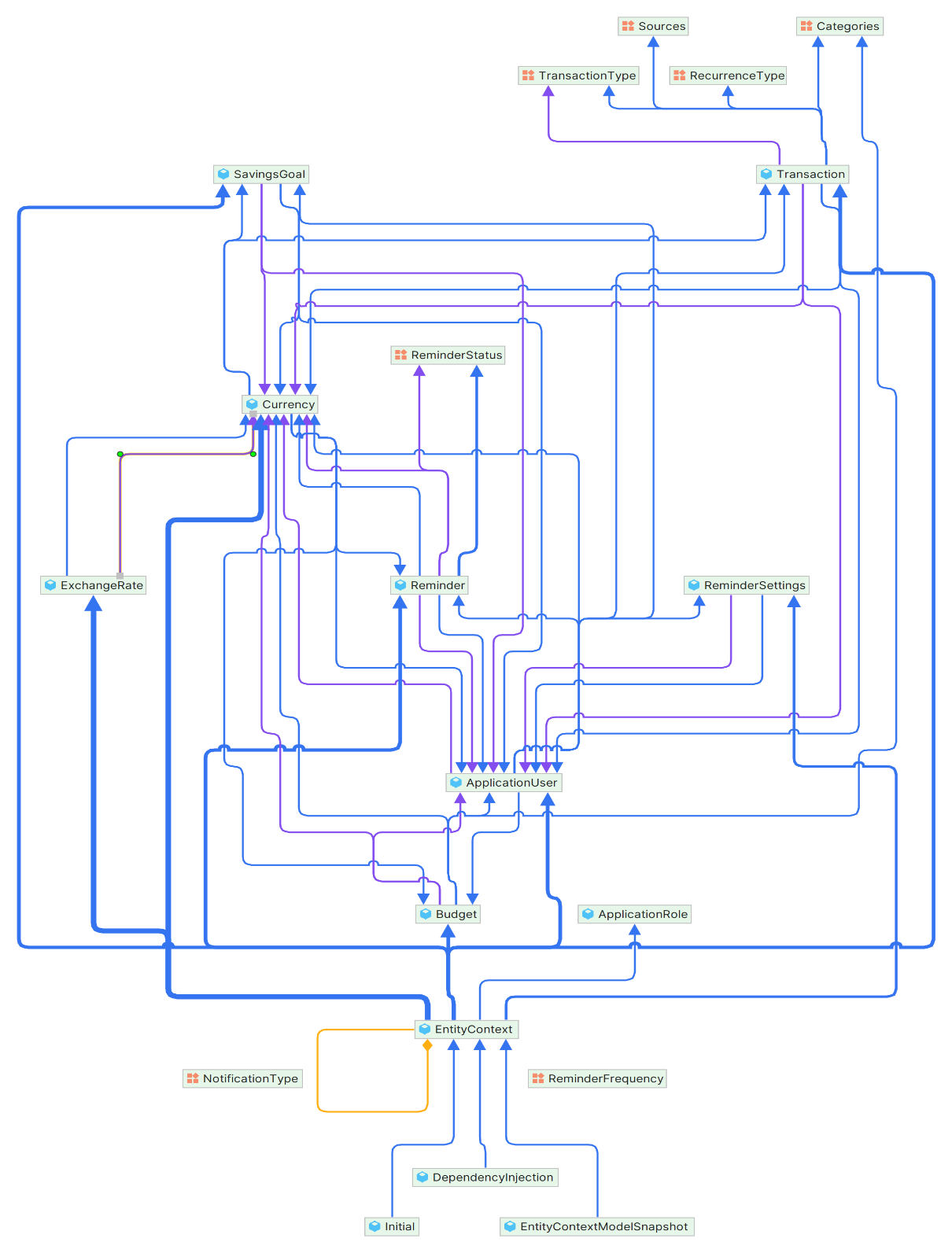


Фигура № 30: Диаграма на отношенията между проектите и техните зависимост

## Описание на слоевете

Приложението е изградено от следните слоя:

### **Слой за достъп до данните**

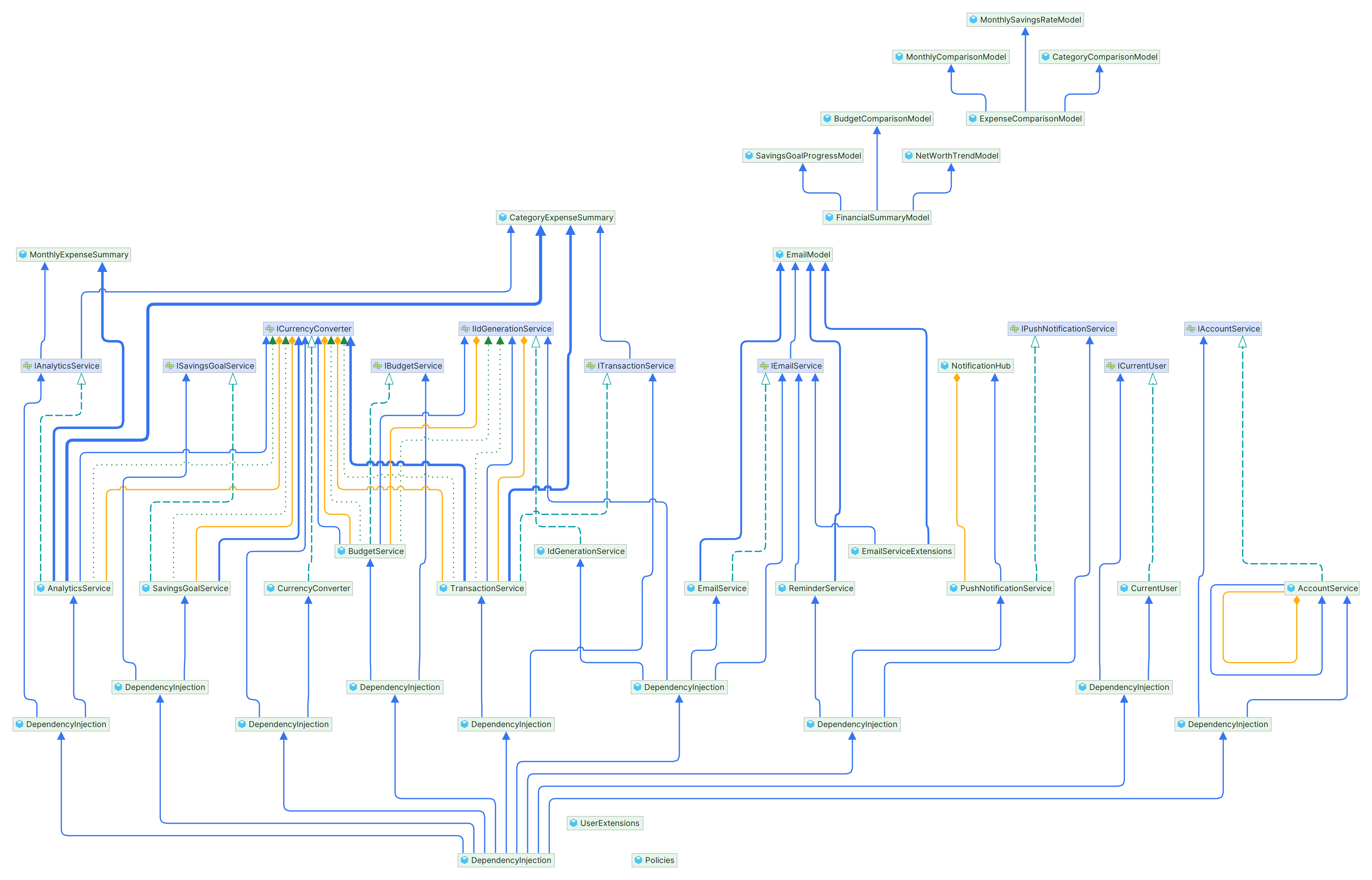
****

Фигура № 31: Диаграма на слоя от данни

Този слой е представен от проекта Zira.Data. Той съдържа основните модели, използвани от приложението, както и класа за контекста на базата данни, с който работи Entity Framework Core. Освен това включва всички класове, които имплементират репозиториен модел и шаблона Unit of Work. Структурата на проекта е организирана в различни пространства от имена и папки за по-добра подредба и управление на кода.

* Enums – В това именно пространство(namespace) се съхраняват енумераторите за приходите, разходите, бюджетите и напомнянията.
  + Categories – категории на даден разход или бюджет
  + Sources – източници на даден приход
  + TransactionTypes –видовете транзакции
  + ReminderStatus – статуса на дадено известие
* Models – В това именно пространство се съхраняват моделите, свързани с приложението.
  + ApplicationUser – данни за потребителя
  + Budget – създадените бюджети на потребителя
  + Currency – обща информация за дадена валута
  + ExchangeRates – курсовете на обмен за всеки вид валута
  + Reminder – създадените напомняния за сметки на потребителя
  + ReminderSettings – настройките на даден потребител за напомняния
  + SavingsGoal – спестовните цели на потребителя
  + Transaction – финансовите транзакции(приходи/разходи) на потребител
* Data – В това именно пространство се съхраняват моделите, свързани с потребителя.
  + ApplicationRole
  + DependencyInjection – този клас вписва всички зависимости от проекта и ги вкарва в главния клас на презентационния слой Program.cs.
  + EntityContext – Класът за контекста на базата данни.
* Migrations – В това имено пространство се съхраняват миграциите на EF Core 8

### **Слой на бизнес логиката**

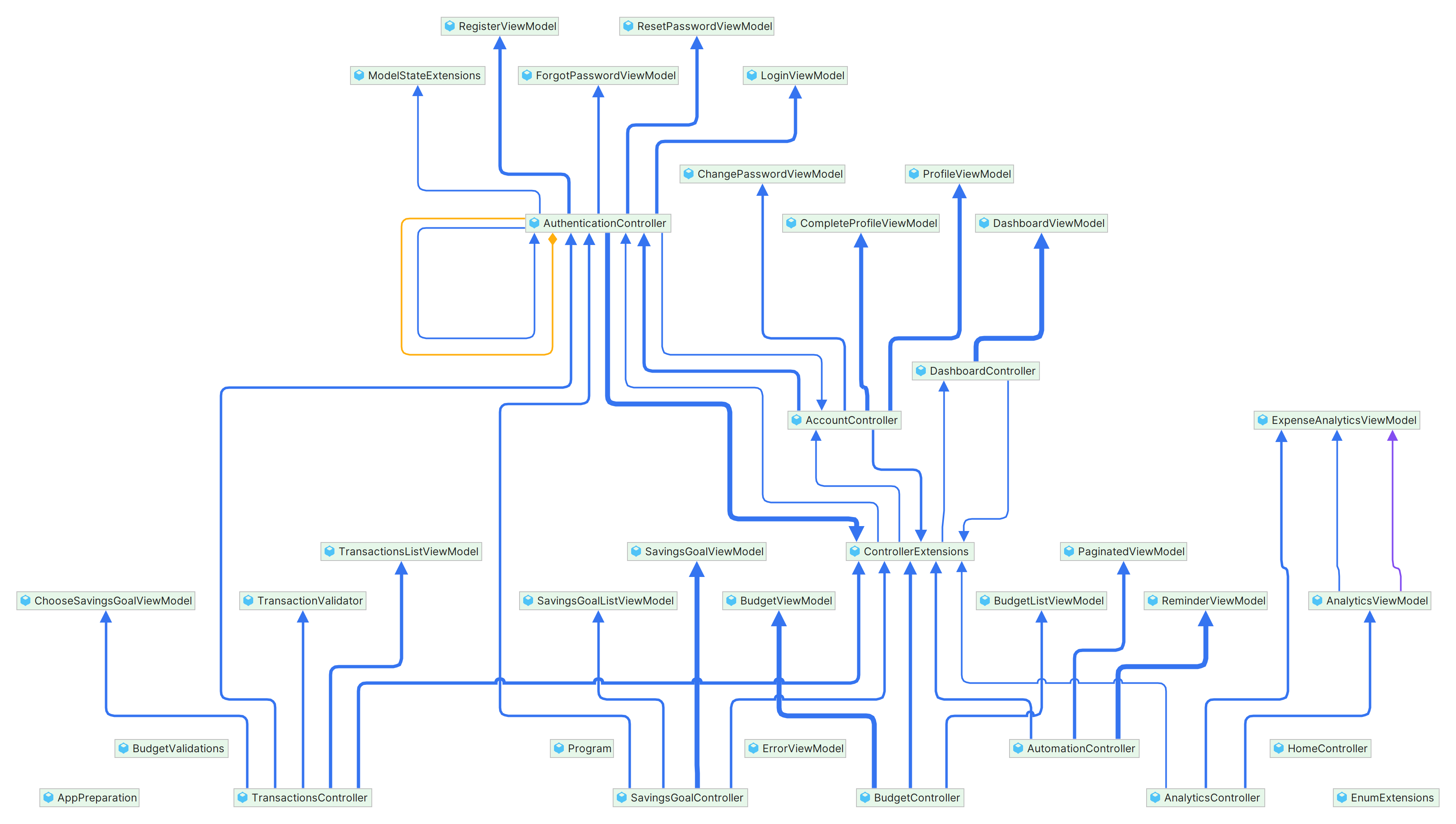


Фигура № 32: Диаграма на бизнес слоя

Този слой е представен от проекта Zira.Services. Той съдържа основните бизнес услуги и дейности на потребителя. Разделен е на няколко именни пространства и папки.

* Constants – В това именно пространство се съдържат константи, използвани в презентационния слой.
  + Policies
* Contracts - В това именно пространство се съдържат интерфейсите на услугите, представени от слоя за използване от презентационния слой.
  + IAccountService
  + IAnalyticsService
  + IBudgetService
  + ICurrencyConverter
  + ICurrentUser
  + IEmailService
  + IIdGenerationService
  + IPushNotificationService
  + ISavingsGoalService
  + ITransactionService
* Extensions - В това именно пространство сa разширения(extensions), използвани от презентационния слой
  + EmailServiceExtensions
  + UserExtensions
* Internals - В това именно пространство са имплементациите на горе-посочените интерфейси.
  + AccountService
  + AnalyticsService
  + BudgetService
  + CurrencyConverter
  + CurrentUser
  + EmailService
  + IdGenerationService
  + NotificationHub
  + PushNotificationService
  + ReminderService
  + SavingsGoalService
  + TransactionService
* Models - В това именно пространство са имплементирани модели за работа в слоя за бизнес
  + BudgetComparisonModel
  + CategoryComparisonModel
  + CategoryExpenseSummary
  + EmailModel
  + ExpenseComparisonModel
  + FinancialSummaryModel
  + MonthlyComparisonModel
  + MonthlyExpenseSummary
  + MonthlySavingsRateModel
  + NetWorthTrendModel
  + SavingsGoalProgressModel
* DependencyInjection - Клас, който реализира инжектиране на зависимости от този слой към останалите.

### **Презентационен слой**



Фигура № 33: Диаграма на презентационния слой

Този слой е представен от проекта Zira.Presentation. Той отговаря за визуалната репрезентация на приложението. Разделен е на няколко имеви пространства и папки.

* Controllers – В това именно пространство се съдържат класовете за управление(controllers), които работят с услуги от бизнес слоя.
* Extensions - В това именно пространство се съдържат разширения, използвани от презентационния слой.
* Models – В това именно пространство се съдържат моделите, които заместват моделите от базата в класовете за управление.
* Views – В това именно пространство се съдържат изгледите/потребителския интерфейс на приложението, които работят с класовете за управление.

## Организация и код на заявките към базата данни

В този проект за работа с базата данни се използва библиотеката за обектно-релационно картографиране (ORM) EF Core, която е част от .NET екосистемата. Тя позволява управлението на данни чрез обекти и колекции, като елиминира необходимостта от ръчно писане на SQL заявки. EF Core предоставя поддръжка за множество релационни бази данни, включително SQL Server, MySQL и MariaDB, а в случая се използва MS SQL Server.

За да се работи ефективно с EF Core, базовите данни се моделират чрез класове, които се намират в слоя за достъп до данни на приложението. Всеки клас отговаря на конкретна таблица в базата и съдържа свойства, които представляват колоните на таблицата.

Основната връзка с базата се реализира чрез клас, който наследява DbContext. В този клас се дефинират свойства от тип DbSet<T>, където T е моделът, който съответства на дадена таблица. DbSet<T> работи като колекция от обекти, които представляват записи в съответната таблица в базата.

За извличане и обработка на данни от базата се използва LINQ (Language Integrated Query). Чрез него могат да се прилагат различни операции като филтриране, сортиране, групиране и агрегация на данни, което улеснява работата със заявки и подобрява четимостта на кода. [6]

LINQ заявките поддържат следните методи: стандартните CRUD(Create, Read, Update, Delete) заявки:

* Add/Read/Update/Delete: Методи за добавяне, прикачване, актуализиране или премахване на обекти от контекста.
* SaveChanges / SaveChangesAsync: Запазване на промените в базата данни.
* FirstOrDefaultAsync / FindAsync: Намиране на същност по нейния първичен ключ.
* Въвеждане: Осигурява достъп до информация за проследяване на промените за конкретен обект.
* OnConfiguring (При конфигуриране): Позволява конфигуриране на опциите на контекста (напр. низ за свързване), когато не се използва инжектиране на зависимости.
* AddRange/ AddRangeAsync: Тази функция добавя колекция от нови същности към DbContext със състояние Added и започва да ги проследява. Новите същности ще бъдат вмъкнати в базата данни, когато се извика SaveChanges() или SaveChangesAsync().

## Наличие на потребителски интерфейс(консолен, графичен, уеб)

Приложението разполага с уеб-базиран потребителски интерфейс, реализиран чрез ASP.NET Core MVC. Този интерфейс предоставя интуитивно и лесно за навигиране уеб приложение, което позволява на потребителите да управляват своите финанси ефективно.

Интерфейсът е създаден на база MVC (Model-View-Controller) архитектура, като разделя логиката, представянето и управлението на заявките. Това осигурява по-добра мащабируемост и лесна поддръжка.

Приложението е достъпно както на настолни компютри, така и на мобилни устройства, благодарение на адаптивния (responsive) дизайн, реализиран чрез Bootstrap и CSS медийни заявки.

Основни функционалности на интерфейса на приложението включват:

* Вход и регистрация - потребителите могат да създават акаунти и да влизат в системата чрез ASP.NET Identity, което гарантира сигурно удостоверяване(authentication) и упълномощяване(authorization) на данните.
* Начално табло (Dashboard) - след влизане в системата, потребителите виждат общ преглед на финансите си, включително текущ бюджет, приходи, разходи и финансови цели.
* Управление на транзакции - потребителите могат да добавят, редактират и изтриват транзакции (приходи и разходи). Всяка транзакция съдържа дата, тип, категория на разход/източник на приход, сума и валута.
* Визуализация на бюджети и финансови цели - възможност за задаване на бюджети и спестовни цели, както и проследяване на напредъка.
* Графики и анализи - интегрирани са интерактивни диаграми за визуализиране на разходите и приходите по категории чрез ApexCharts.js.
* Филтриране и търсене на транзакции - потребителите могат да филтрират транзакциите пo категория, за да намират лесно нужната информация.

Допълнителни функционалности на приложението:

* Управление на потребителските данни – потребителите могат да преглеждат и променят своите данни – имейл, парола, имена, др.
* Предпочитания за известия – потребителите имат възможност да избират какъв вид известия могат да получават.
* Анализи и графики - подробни статистики за разходите и приходите с допълнителни филтри и визуализации.

# **Ефективност и бързодействие на решението**

Системите за управление на личните финанси се развиват значително през последните години, като основният фокус е върху автоматизацията на бюджетирането, проследяването на разходите и подобряването на финансовата култура. Сред най-известните софтуерни решения в тази сфера са YNAB (You Need a Budget), Mint и PocketGuard. Тези приложения използват различни подходи за управление на финансите, но споделят общата цел да помогнат на потребителите да управляват бюджета си по-ефективно.

YNAB (You Need a Budget) е един от най-популярните инструменти за бюджетиране, основан на философията „Кажи на парите си какво да правят“. Приложението използва методология, базирана на разпределение на всеки приход към конкретна категория разходи, което помага на потребителите да планират предварително и да избягват финансови изненади. YNAB насърчава изграждането на финансова стабилност чрез концепции като „да харчиш парите на миналия месец“ и „да бъдеш гъвкав с бюджета си“. Системата се синхронизира с банкови сметки, предлага мобилна и уеб версия и има активна общност с обучителни материали. Основният ѝ недостатък е, че е платено приложение, което може да не е от полза за някои потребители [7].

Други популярни решения като Mint и PocketGuard също предлагат функции за проследяване на разходите и бюджетиране, но се различават по подхода си. Mint, например, автоматично анализира банковите транзакции и създава бюджети, докато PocketGuard има функция за изчисляване на наличните средства, като изважда фиксираните разходи и спестяванията от приходите(https://rockstarfinance.com/mint-app-review). Въпреки че тези приложения предлагат безплатни версии, те често разчитат на реклами или ограничени функции в сравнение с платените платформи като YNAB. [8]

Изборът на подходяща система за управление на личните финанси зависи от индивидуалните предпочитания и нужди на потребителя. Докато YNAB предлага детайлно планиране и изграждане на навици за финансов контрол, Mint и PocketGuard залагат на автоматизирани анализи и удобство.

# **Тестване**

Проектът е тестван с помощта на различни тестови рамки, осигуряващи висока надеждност и качество на кода. Тестването включва модулни (unit), интеграционни (integration), за производителност (performance) и за сигурност (security) тестове. Тестовете на бизнес логиката(Unit tests) проверяват отделни модули и методи от бизнес логиката, като ги изпълняват в изолация от останалата част на системата. В проекта тестовете са реализирани чрез xUnit, а за мокване на зависимости се използва Moq.

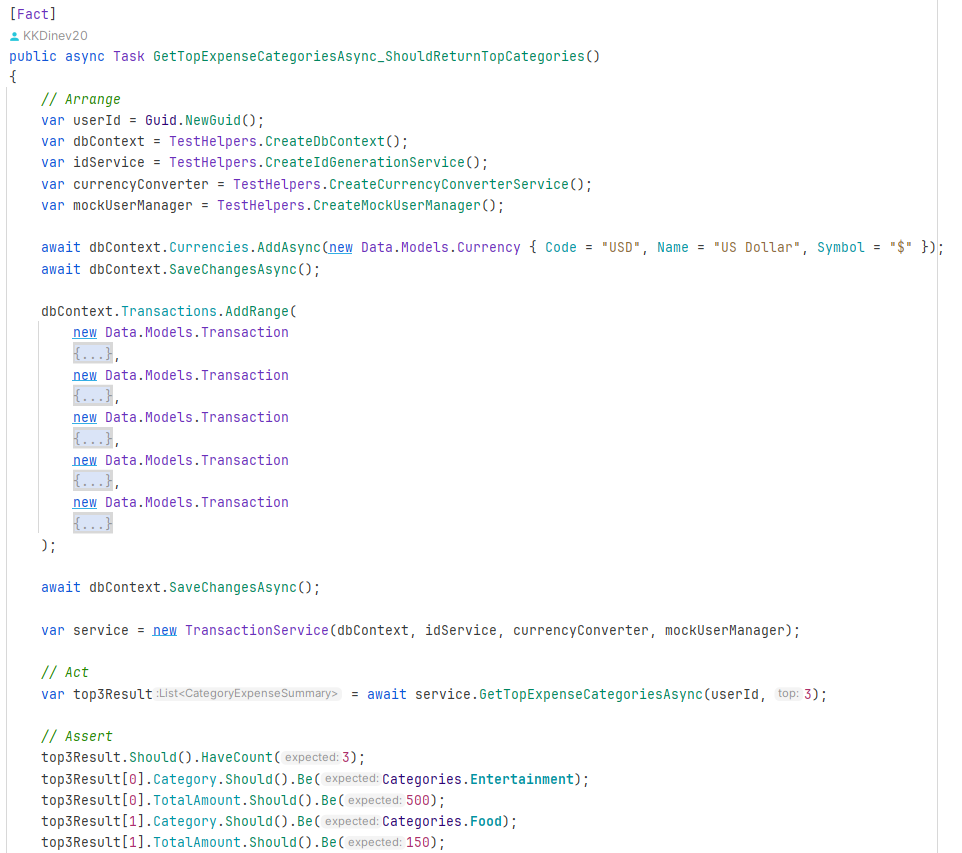
Проектът, отговарящ за тестването на бизнес логиката, следва конвенция за именуване на тестовете:

ИмеНаМетод\_ТрябваДаИзвършиДаденоДействие().

Тестовете следват подхода Arrange – Act – Assert:

* Arrange – Подготвят се данните и зависимостите.
* Act – Изпълнява се тестваният метод.
* Assert – Проверява се дали резултатът отговаря на очакванията.

За съхранение на тестовите данни се използва InMemoryDatabase от Entity Framework Core, което позволява изпълнение на тестовете без реална база данни. Всеки тестов клас използва отделна база със случайно име, създадено чрез GUID.NewGuid().ToString(). Това гарантира независимост между тестовете и елиминира проблеми със споделена база данни. Примерен тестов клас е TransactionServiceTests, тестващ управлението на транзакциите:



Фигура № : Метод от класа за тестване на транзакции

Производителността на ключови операции се тества с BenchmarkDotNet, който измерва времето за изпълнение на различни методи и анализира тяхната ефективност.

Примерен тест за производителност проверява колко бързо се извличат транзакциите на даден потребител от базата данни.

Подходът включва:

* Многократно изпълнение на даден метод и измерване на средното време за изпълнение.
* Сравнение на различни алгоритми за оптимизация на заявки към базата.
* Натоварване на класовете за управление и методите с голям брой заявки, за да се анализира поведението му под голям обем данни.



Фигура № : Примерни тестове за производителност

Сигурността на приложението се проверява чрез Cypress, като основен фокус са атаки от типа SQL инжектиране на данни, XSS (Cross-Site Scripting) и CSRF (Cross-Site Request Forgery).

Примерен тест в Cypress проверява дали входните данни се валидират правилно и дали е възможно да се изпълни SQL инжектиране на данни чрез текстови полета.

Използват се следните подходи:

* Автоматизирани тестове за защита от XSS, чрез въвеждане на JavaScript код в полета за вход.
* Тестване на сесии и жетони(tokens) за защита срещу CSRF атаки.
* Симулиране на атаки чрез изпращане на модифицирани заявки.



Фигура № : Част от тестовете за сигурност на защита на потребителските данни

Интеграционните тестове гарантират, че различните модули на приложението работят правилно заедно. В този проект те проверяват взаимодействието между трите слоя, като използват реални зависимости, но в контролирана тестова среда.

За изпълнението им се използва xUnit, като се създава тестов сървър чрез CustomWebApplicationFactory, който симулира средата на приложението. Базата данни се управлява чрез Entity Framework Core с InMemoryDatabase, за да се гарантира независимост между тестовете.

Примерен интеграционен тест проверява дали при добавяне на нов бюджет, той се съхранява коректно в базата и може да бъде извлечен чрез заявка към даден клас за управление:



Фигура № : Примерен интеграционен тест за бюджетите

# **Заключение и бъдещо възможно развитие**

Проектът успешно покрива изискванията на заданието, като предоставя цялостно решение за управление на личния бюджет, включващо функционалности за проследяване на транзакции, бюджети, спестявания и напомняния. Чрез използването на съвременни технологии и добри софтуерни практики, системата осигурява сигурност, мащабируемост и удобство за потребителя.

В бъдеще могат да бъдат добавени допълнителни функционалности като автоматично категоризиране на разходите чрез машинно обучение, интеграция с банкови API-та за автоматично синхронизиране на транзакции, както и разширени анализи и визуализации за по-добро финансово планиране.

# **Използвани литературни източници и Уеб сайтове**

|  |  |
| --- | --- |
| [1] | V. Paradigm, „What is a sequence diagram?,“ [Онлайн]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/. |
| [2] | T. Target, „What is database normalization?,“ Tech Target, 2024. [Онлайн]. Available: https://www.techtarget.com/searchdatamanagement/definition/normalization. [Отваряно на 2024].` |
| [3] | S. Direct, „Transitive Dependency,“ Science Direct, 2016. [Онлайн]. Available: https://www.sciencedirect.com/topics/computer-science/transitive-dependency. |
| [4] | Microsoft, „MSDN,“ 9 2 2022. [Онлайн]. Available: https://learn.microsoft.com/en-us/ef/ef6/fundamentals/testing/mocking. [Отваряно на 10 4 2023]. |
| [5] | Microsoft, „CQRS Pattern,“ 21 02 2025. [Онлайн]. Available: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs. [Отваряно на 2025]. |
| [6] | B. Wagner, „Introduction to LINQ queries in C#,“ 24 03 2025. [Онлайн]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/linq/get-started/introduction-to-linq-queries. |
| [7] | C. Neidel и T. Tindall, „Review: We Tested the YNAB App,“ nerdwallet, 24 11 2024. [Онлайн]. Available: https://www.nerdwallet.com/article/finance/ynab-app-review. |
| [8] | B. Holzhauer и D. Whiteman, „PocketGuard Budgeting App Review 2025,“ Forbes, 03 04 2024. [Онлайн]. Available: https://www.forbes.com/advisor/banking/pocketguard-budgeting-app-review/.. |

# **Приложения**

[Фигура № 1: Страница, на която потребителят може да види в обобщен вид информация за своите финанси. 2](#_Toc194943954)

[Фигура № 2: Страница, на която потребителят може да види транзакциите си. 2](#_Toc194943955)

[Фигура № 3:Страница, на която потребителят може да проследява спестовните си цели. 2](#_Toc194943956)

[Фигура № 4: Страница, на която потребителят може да проследява състоянието на зададените си бюджети. 2](#_Toc194943957)

[Фигура № 5: Страница, на която потребителят може да следи своето финансово си състояние. 2](#_Toc194943958)

[Фигура № 6: Страница, на която потребителят може да види обобщен годишен анализ на своите финанси. 2](#_Toc194943959)

[Фигура № 7: Страница за настройките на потребителя. 2](#_Toc194943960)

[Фигура № 8: Страница с напомнянията за сметки 2](#_Toc194943961)

[Фигура № 9: Диаграма на случаите за употреба (Use Case) 2](#_Toc194943962)

[Фигура № 10: Диаграма на дейността №1(Activity diagram) 2](#_Toc194943963)

[Фигура № 11: Диаграма на дейността №2(Activity diagram) 2](#_Toc194943964)

[Фигура № 12: Диаграма на дейността №3(Activity diagram) 2](#_Toc194943965)

[Фигура № 13: Диаграма на дейността №4(Activity diagram) 2](#_Toc194943966)

[Фигура № 14: Диаграма на дейността №5(Activity diagram) 2](#_Toc194943967)

[Фигура № 15: Диаграма на дейността №6(Activity diagram) 2](#_Toc194943968)

[Фигура № 16: Диаграма на дейността №7(Activity diagram) 2](#_Toc194943969)

[Фигура № 17: Диаграма на дейността №8(Activity diagram) 2](#_Toc194943970)

[Фигура № 18: Диаграма на дейността №9(Activity diagram) 2](#_Toc194943971)

[Фигура № 19: Диаграма на дейността №10(Activity diagram) 2](#_Toc194943972)

[Фигура № 20: Диаграма на дейността №11(Activity diagram) 2](#_Toc194943973)

[Фигура № 21: Диаграма на дейността №12(Activity diagram) 2](#_Toc194943974)

[Фигура № 22: Диаграма на последователността (Sequence diagram) 2](#_Toc194943975)

[Фигура № 23: Диаграма на последователността (Sequence diagram) 2](#_Toc194943976)

[Фигура № 24: Диаграма на последователността (Sequence diagram) 2](#_Toc194943977)

[Фигура № 25: Диаграма на отношенията между моделите (Entity Relation) 2](#_Toc194943978)

[Фигура № 26: Диаграма на не нормализирана таблица 2](#_Toc194943979)

[Фигура № 27: Диаграма на таблицата в първа нормална форма 2](#_Toc194943980)

[Фигура № 28: Диаграма на таблица във втора нормална форма (2NF) 2](#_Toc194943981)

[Фигура № 29: Диаграма на отношенията между главните проекти 2](#_Toc194943982)

[Фигура № 30: Диаграма на отношенията между проектите и техните зависимост 2](#_Toc194943983)

[Фигура № 31: Диаграма на слоя от данни 2](#_Toc194943984)

[Фигура № 32: Диаграма на бизнес слоя 2](#_Toc194943985)

[Фигура № 33: Диаграма на презентационния слой 2](#_Toc194943986)

[Фигура № 34: Метод от класа за тестване на транзакции 2](#_Toc194943987)

[Фигура № 35: Примерни тестове за производителност 2](#_Toc194943988)

[Фигура № 36: Част от тестовете за сигурност на защита на потребителските данни 2](#_Toc194943989)

[Фигура № 37: Примерен интеграционен тест за бюджетите 2](#_Toc194943990)