ExpenseBuddy

Курсова работа по Софтуерни архитектури и разработка на софтуер

Стефан Велев, № 62537

Даниел Халачев, № 62547

Съдържание

1. Въведение

a. Обща информация за текущия документ

i. Предназначение на документа

Този документ представя архитектурните решения за проекта ExpenseBuddy. Основната му цел е структурирано да представи цялата система, заедно с всички нейни характеристики и специфики.

ii. Описание на използваните структури на архитектурата

- **Модулна структура**
  - **Декомпозиция на модулите** - представя компонентите (софтуерните елементи на системата), тяхното разделение и връзки между тях
- **Структура на процесите** - представя най-важните процеси в системата - добавяне на разход към бюджет (ръчно и чрез външна система за плащания), добавяне на потребител към споделен бюджет
- **Структура на разположението** - представя разположението на софтуера на системата върху хардуера

За проекта освен модулната декомпозиция, са от значителна важност също **Структурата на процесите** и **Структурата на разположението**.

Структурата на процесите избрахме, защото поради множеството модули в системата настъпват множество взаимодействия между тях и операции, които за потребителя изглеждат атомарни (например добавяне на разход към бюджет), но всъщност са сложни и многоетапни.

Предпочетохме **Структурата на разположението** пред Разпределение на работата поради избрания архитектурен стил – "Клиент-Сървър", който предполага множество различни физически локации на хардуера и софтуера на системата. Без описанието на **Структурата на разположението**, би могло да настъпи неразбиране или двусмислие по отношение на това коя част от софтуера на какъв хардуер се намира.

iii. Структура на документа

Секция	Предназначение
1.	*Общи сведения за документа и същността на проекта*
2.	*Декомпозицията на модулите*
2.a.	*Общ вид на декомпозицията*
2.b.	*Контекстна диаграма*
2.c.	*Описание на Client*
	Описание на ClientManager
	Описание на ServerManager
	Описание на MainServer
	Описание на PaymentManager
	Описание на BackupServer
2.d.	*Описание на възможните вариации*
3.	*Допълнителни структури*
3.1.	*Структура на процесите*

b. Общи сведения за системата

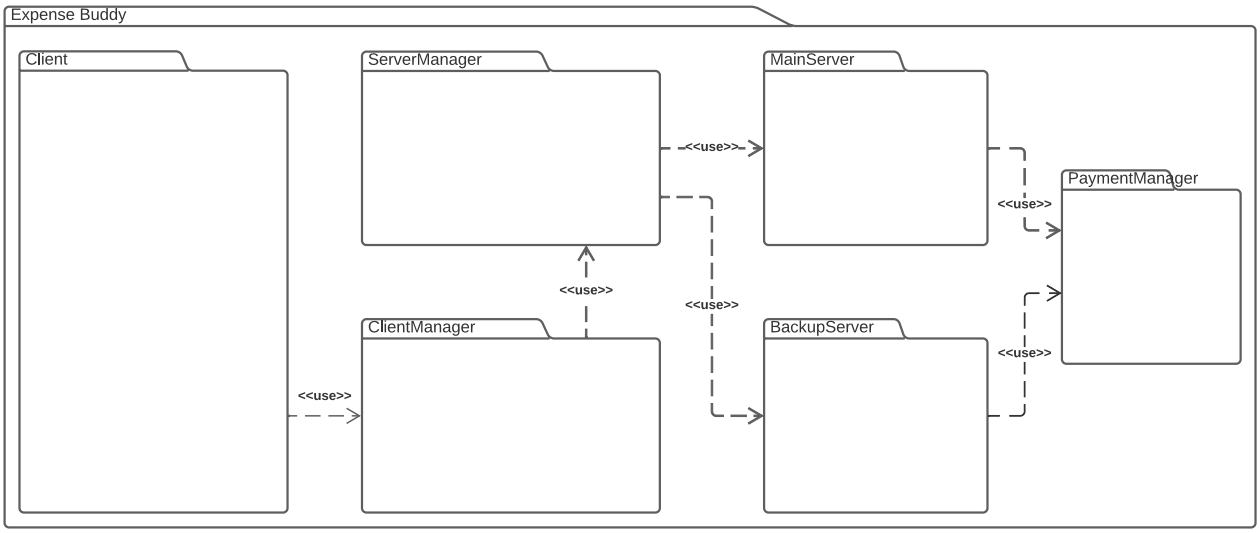
ExpenseBuddy е софтуерна система, предназначена за управление на разходи. Тя позволява на потребителите да създават индивидуални или споделени бюджети за от 2-ма до 5-ма души. Основната цел на системата е потребителите да имат възможност да съхраняват информация за направени разходи за определен период от време и да получават анализи на база направените разходи.

c. Терминологичен речник

Дума	Значение	Секция
Active Redundancy	*Тактика за дублиране на функционалност с цел повишаване на надеждността*	***2.a.***
Front-end	*Частта от едно уеб приложение, с която взаимодейства потребителят*	***2.c./ClientManager***
Back-end	*Частта от едно уеб приложение, която имплементира логиката, която front-end-а представя*	***2.c./ClientManager***
Bug	*Грешка в кода, която не е видима при компилация*	***2.d.***
HTTP	*Hypertext Transfer Protocol - един от основните протоколи за интернет комуникация*	***2.c./ClientManager***
REST	*Representational State Transfer - стил софтуерна архитектура, базиран на следните шест характеристики:* *1. Клиент-Сървър архитектура* *2. Липса на състояние - клиентът отговаря за следене на сесиите* *3. Кеширане* *4. Многослойна система* *5. Код при поискване* *6. Единен интерфейс*	***2./Обосновка***
JSON	*JavaScript Object Notation - формат на данни, който е четим и от човек, и от машина*	***2.c./ClientManager***
DBMS	*Database Management System - Система за управление на базата данни (СУБД)*	***2.d./DBMS***

2. Декомпозиция на модулите

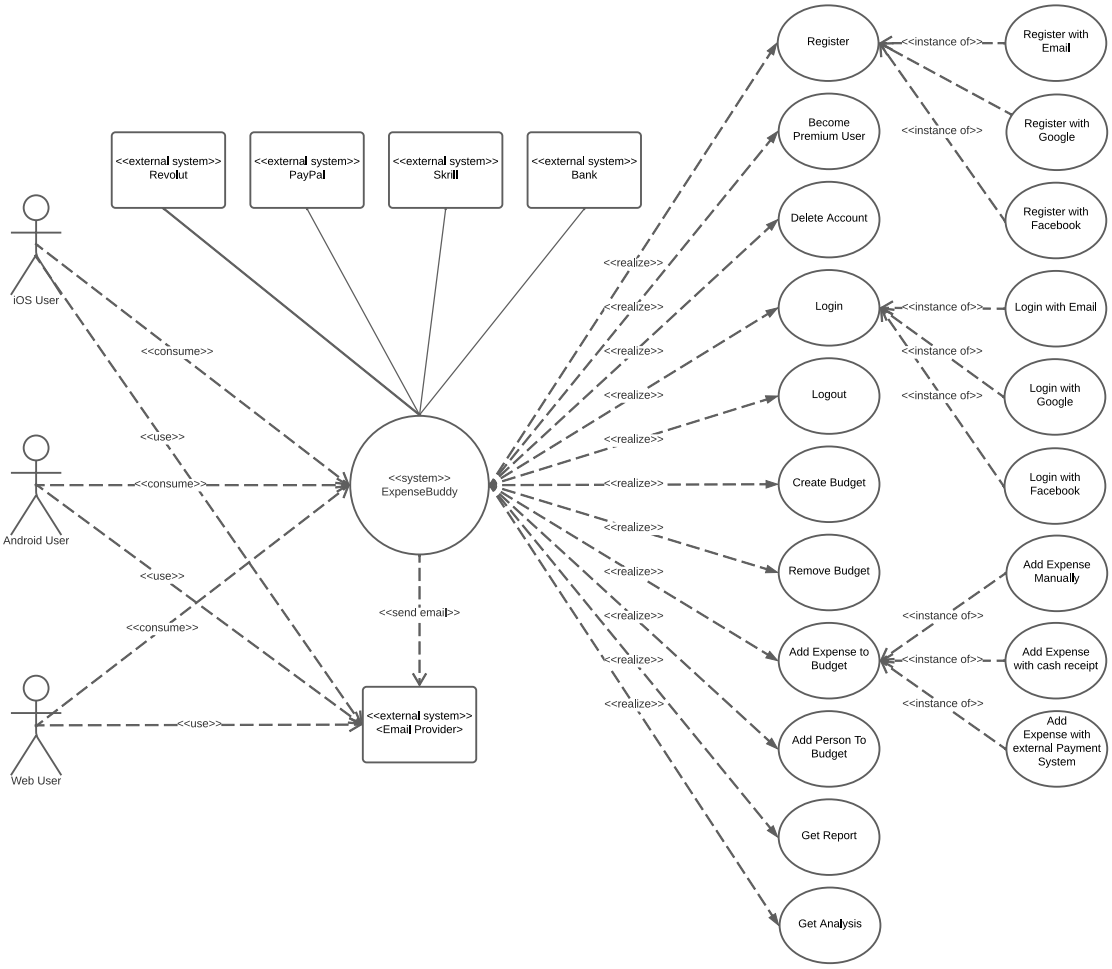
a. Общ вид на декомпозицията на модули за системата



Диаграма 1: Общ вид на декомпозицията на модулите

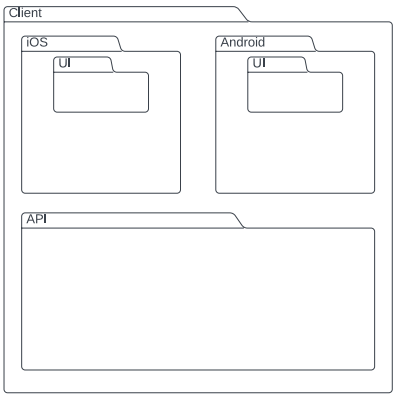
Поради изискването системата да бъде достъпвана от множество различни по вид устройства чрез интернет връзка, тя ще бъде изградена в архитектурния стил “Клиент-Сървър”. Затова системата ще бъде разделена на две основни части - клиентска част, която имплементира различните възможности за thin client (т.е, само визуално представяне на данните) и сървърна част, която имплементира генерирането, употребата и обработката на данните. С цел висока надеждност сървърът ще бъде дублиран чрез Active Redundancy, а модулът ServerManager ще следи кой сървър е активен - основният (MainServer) или резервният (BackupServer), поместени в съответните за тях модули. Тъй като клиентът не би могъл да бъде наясно кой от двата сървъра е активен, клиентът първо ще се свързва с NameServer, който ще пренасочва към активния сървър, получавайки информация кой е той от ServerManager модула.

b. Контекстна диаграма



c. Подробно описание на всеки модул

1. Client



Диаграма 2: Модулът за клиента и декомпозицията му на подмодули

- Предназначение: имплементира функционалността в клиентската част на системата и позволява лесната промяна на кода за всеки вид клиент
- Основни отговорности:
    - Осигурява интуитивен потребителски интерфейс

- Позволява достъп до системата от iOS и Android
- Позволява на потребителя да извършва всички налични за ролята му функционалности, сред които: регистрация, вписване, управление на разходи и бюджети, генериране и преглед на отчети и анализи
- Представя данните на потребителя, които сървърът е обработил

- Интерфейси: Client

1.1. IOS

1.2. Android

1.3. API

1.*.1. UI

2. ClientManager

Диаграма 3: Модулът за "посрещане" на завките от клиента ClientManager и неговите три подмодула

- Предназначение: имплементира Web клиента, “посреща” заявките на клиентите, пренасочва трафика към активния в момента на заявката сървър
- Основни отговорности:
  - Предоставя front-end и back-end за Web версията на системата
  - Интерпретира HTTP заявки и изпраща такива в отговор на клиентите
  - Получава информация от ServerManager кой е работещият сървър и пренасочва трафика към него
  - Създава допълнителна преграда пред достъпа до същинската функционалност на системата (тактика Фасада)

2.1. Web

- Предназначение: имплементира Web клиента за достъп до системата чрез уеб браузър
- Основни отговорности:
  - Предоставя графичен интерфейс за уебсайт, чрез който клиентите да могат да използват функционалностите на системата, които ролите им позволяват
  - Предоставя възможност на потребителите да извършват дейности в системата чрез логика, управляваща графичния интерфейс
  - Извършва желаните от потребителя действия като изпраща HTTP Request заявки към сървъра, подобно на модула Client при iOS и Android устройства

2.2. RequestHandler

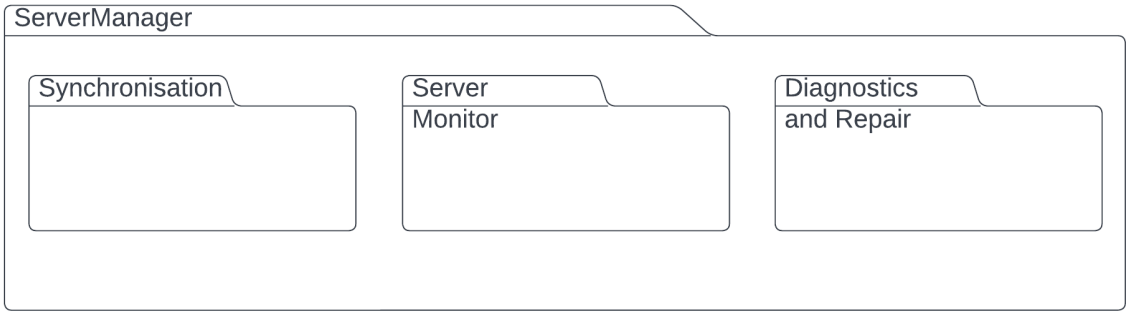
- Предназначение: приема получените от клиентите HTTP заявки и ги свежда до разбираеми за сървърната част инстанции на класове и методи
- Основни отговорности:
  - Интерпретира HTTP заявките
  - Извлича инстанции на класове от JSON нотацията в тялото на HTTP заявките

- Извиква съответните методи на AccountManager, BudgetManager, ReceiptInterpreter, ReportManager, AnalysisManager и PaymentManager, които съответстват на желаните от потребителя действия в HTTP заявките, изпратени от Client
- Интерфейси: RequestHandler

2.3. NameServer

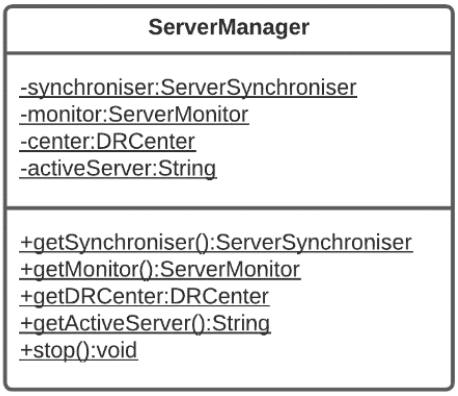
- Предназначение: информира за работещата сървърна част (активния сървър) и пренасочва към него трафика
- Основни отговорности:
  - Получава информация от ServerManager кой е работещият сървър и пренасочва трафика към него
- Интерфейси: NameServer

3. ServerManager



Диаграма 4: Модулът за управление на сървърната част ServerManager и неговите три подмодула Synchronisation, ServerMonitor и Diagnostics and Repair, предназначени съответно за синхронизацията на двата сървъра, следенето на състоянието и диагностиката

- Предназначение: администрира основния и резервния сървър и съобщава за NameServer кой от двата е активен
- Основни отговорности:
  - Следи за изправността на основния и резервния сървър
  - Синхронизира състоянието на системата между двата сървъра
  - Прави диагностика на двата сървъра и при неизправност на някой от тях опитва автоматично да я отстрани
  - Уведомява NameServer кой от двата сървъра е активен
- Интерфейси: ServerManager



Диаграма 5: Класът ServerManager, отговарящ за администрирането на сървърната част

```
// При спиране на работа на MainServer
this.ServerManager.getMonitor().notify();
this.ServerManager.getDRCenter.swtichToBackupServer();
if (this.ServerManager.getDRCenter.repairMainServer() == true) {
    this.ServerManager.getSynchronizer.sync();
    this.ServerManager.getDRCenter.switchToMainServer();
}
```

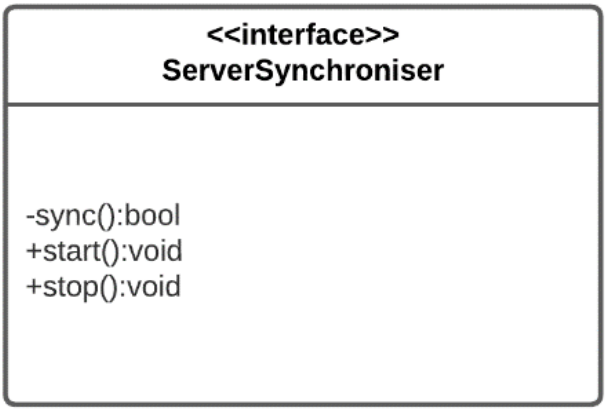
```
// В NameServer:
String serverLocation = this.ServerManager.getActiveServer();
```

```
public class ServerManager {
    private static final ServerMonitor monitor = new DefaultServerMonitor();
    private static final DRCenter center = new DefaultDRCenter();
    private static final ServerSynchronizer synchronizer = new
DefaultServerSynchronizer();
    private static URL activeServer = null;

    public ServerManager() {
        monitor.start();
        center.start();
        synchronizer.start();
    }
}
```

3.1. Synchronization

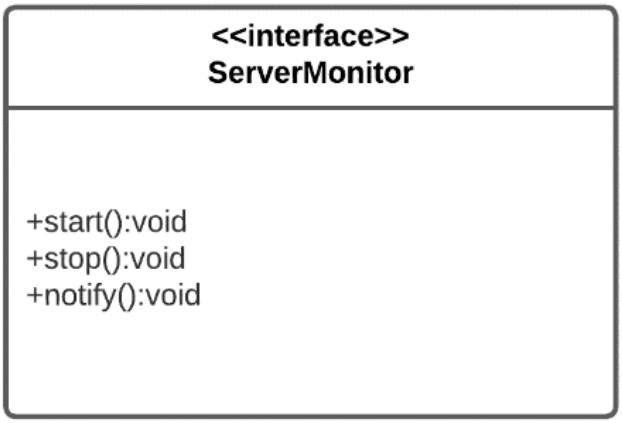
- Предназначение: синхронизира състоянието на системата между двата сървъра
- Основни отговорности:
  - Периодично проверява дали състоянието на данните е еднакво и на двата сървъра
  - При несъвпадение синхронизира данните
  - При настъпване на значимо събитие синхронизира данните
- Интерфейси: ServerSynchronizer



Диаграма 6: Интерфейсът ServerSynchroniser, отговарящ за синхронизирането на състоянието на двата сървъра

3.2. ServerMonitor

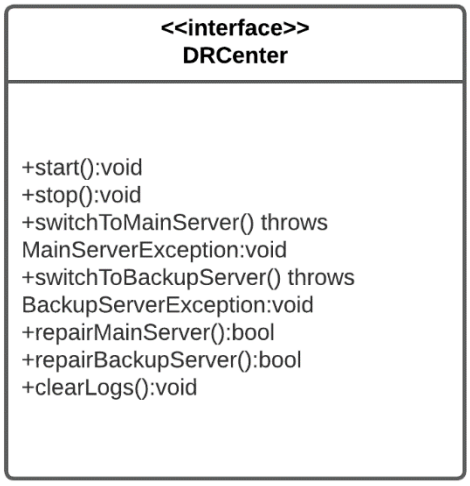
- Предназначение: следи за изправността на двата сървъра
- Основни отговорности:
  - Извършва периодични проверки за изправността на сървърите
  - Уведомява модула DiagnosticsAndRepair при откриване на неизправност
- Интерфейси: ServerMonitor



Диаграма 7: Интерфейсът ServerMonitor, отговарящ за следенето на състоянието на сървърите

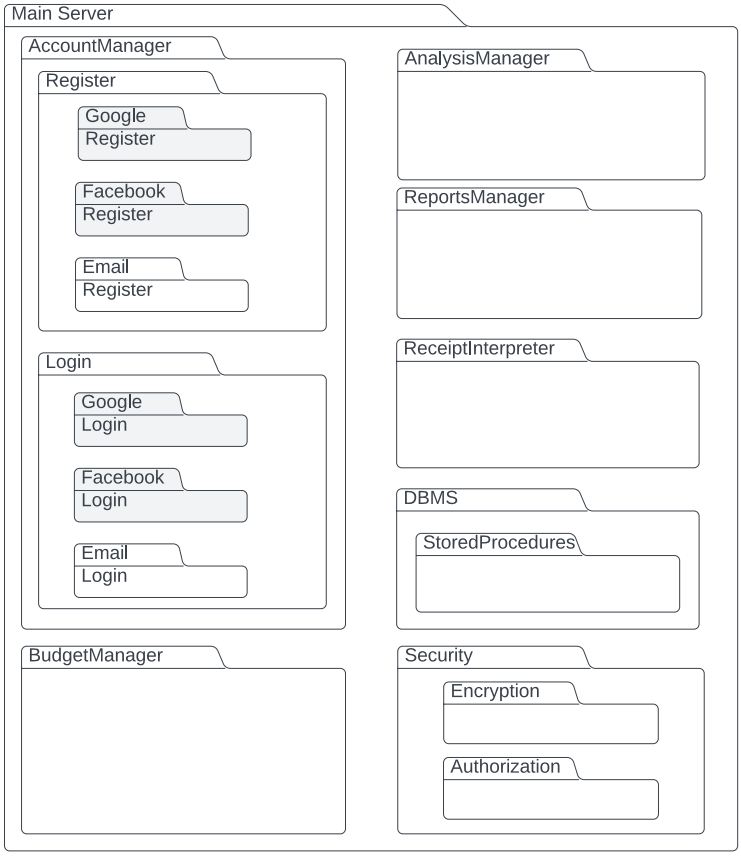
3.3. DiagnosticsAndRepair

- Предназначение: Осигурява надеждността на системата чрез замяна на активния сървър със заместник при нужда
- Основни отговорности:
  - Превключва към резервния сървър в случай на неизправност, открита от ServerMonitor
  - Прави опит да поправи открити неизправности чрез готови, описани от администратора, процедури
  - Създава лог файловете на системата
  - Предоставя събраните данни на администратора при профилактика
- Интерфейси: DRCenter



Диаграма 8: Интерфейсът DRCenter, отговарящ за диагностиката и самопоправката

4. MainServer



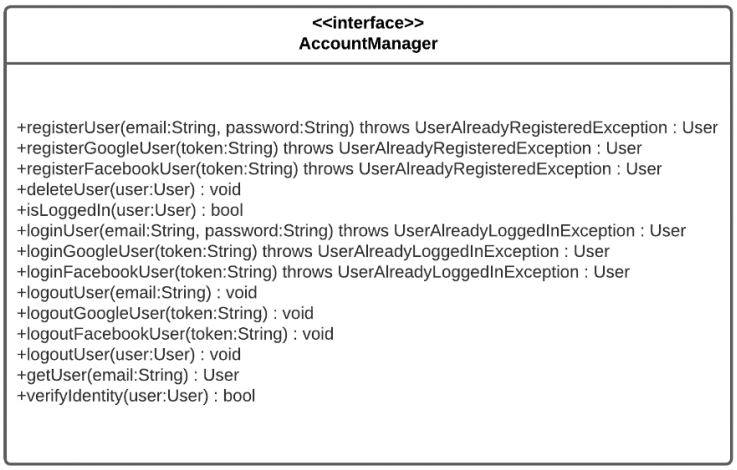
Диаграма 9: Модулът MainServer, имплементиращ основата бизнес логика на системата, и неговите подмодули

- Предназначение: имплементира Бизнес логиката на системата
- Основни отговорности:
    - Администрира акаунтите
    - Извършва отчитането и промяната на разходите и бюджетите на потребителите

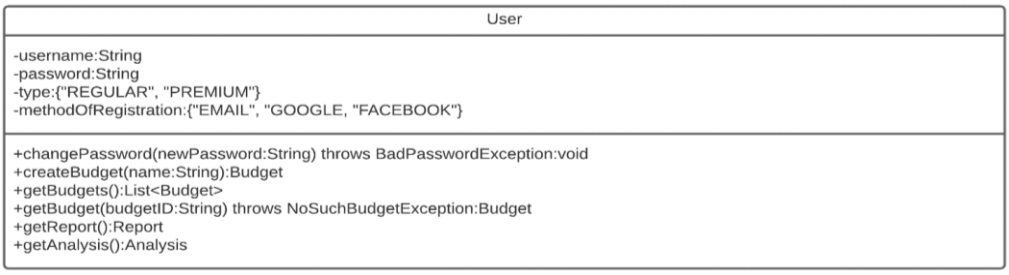
4.1. AccountManager

- Предназначение: имплементира Системата за администриране на акаунтите в собствен модул
- Основни отговорности:
    - Регистрира нови потребители
    - Вписва вече регистрирани потребители в системата
    - Блокира неправомерен достъп в съвместна работа с модула Authentication
    - Позволява на администратора да променя или изтрива акаунти
    - Осигурява разделението на Обикновени и Премиум потребители
    - Не позволява на Обикновени потребители достъп до Премиум функционалности

- Интерфейси: AccountManager



Диаграма 10: Интерфейсът AccountManager, отговорен за управлението на акаунтите



Диаграма 11: Възможна имплементация на класа User, представящ един акаунт в системата

```
// взимане на конкретен потребител и извършване на действие с него
ACCOUNT_MANAGER.getUser(providedEmail).createBudget("MyBudget");
```

```
// за регистриране на потребител
try{
    ACCOUNT_MANAGER.registerUser(providedEmail, providedPassword);
}
catch (UserAlreadyRegisteredException ex){
    ErrorForm.Load(ex.Message+": "+providedEmail);
}
```

```
// за логване
try{
    ACCOUNT_MANAGER.loginUser(providedEmail, providedPassword);
}
catch (UserAlreadyLoggedInException ex){

     // suspicion for unauthorized access -> Log the user out
    ACCOUNT_MANAGER.logoutUser(providedEmail);
}
```

```
// за изтриване на потребител
ACCOUNT_MANAGER.deleteUser(toDelete);
```

4.1.1. Register

- Предназначение: имплементира регистрирането на нови потребители в системата
- Основни отговорности:
    - Позволява регистрация на нов потребител чрез имейл, Google или Facebook акаунт
    - Не позволява повторна регистрация с вече използван имейл или Google/Facebook акаунт

4.1.1.1. Google Register	Извършва регистрирането с Google акаунт	Създава акаунт в системата, основан на предоставения Google акаунт
4.1.1.2. Facebook Register	Извършва регистрирането с Facebook акаунт	Създава акаунт в системата, основан на предоставения Facebook акаунт
4.1.1.3. Email Register	Извършва регистрирането с имейл адрес	1. Създава акаунт в системата чрез предоставени имейл и парола. 2. Проверява валидността на имейла чрез изпращане на потвърждаващо писмо 3. Чрез потвърждаващото писмо предотвратява регистрирането на bot-ове

4.1.2. Login

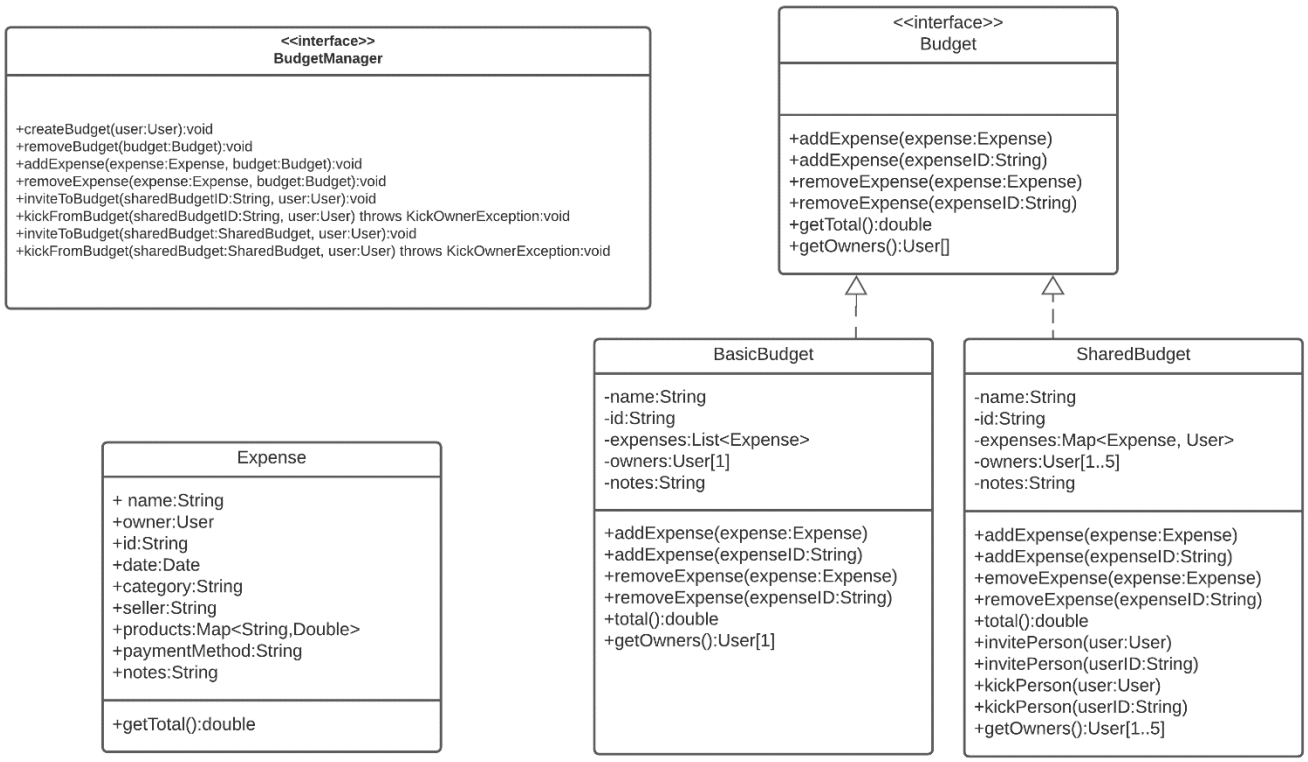
- Предназначение: Позволява вписването на вече съществуващ потребител чрез имейл, Google или Facebook акаунт
- Основни отговорности:
    - Вписва вече съществуващ потребител в системата при предоставени верни входни данни
    - Не позволява вписване при предоставени грешни входни данни
    - Не позволява вписване на несъществуващ потребител

4.1.2.1. Google Login	Извършва вписването с Google акаунт	Влиза в системния акаунт чрез предоставения Google акаунт
4.1.2.2. Facebook Login	Извършва вписването с Facebook акаунт	Влиза в системния акаунт чрез предоставения Facebook акаунт
4.1.2.3. Email Login	Извършва вписването с имейл адрес	Влиза в системния акаунт чрез предоставени имейл и парола

4.2. BudgetManager

- Предназначение: Управлява бюджетите на потребителите
- Основни отговорности:
    - Позволява добавянето и премахването на разходи в бюджет
    - Позволява добавянето и премахването на потребители в бюджета (при Премиум)

- Интерфейси: BudgetManager



Диаграма 12: Примерна имплементация на интерфейса BudgetManager и свързаните с него класове, представящи бюджети и разходи

```
private static final AccountManager ACCOUNT_MANAGER = new
DefaultAccountManager();
private static final BudgetManager BUDGET_MANAGER = new
DefaultAccountManager();
```

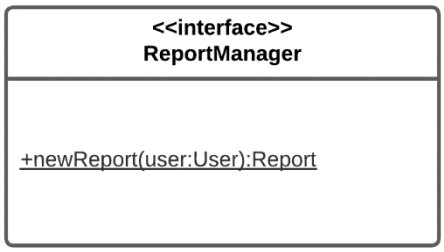
```
// добавяне на разход към бюджет MyBudget чрез касов бон
Expense ex = RECEIPT_INTERPRETER.interpret(providedImage);
ACCOUNT_MANAGER.getUser(providedEmail).getBudget("MyBudget").addExpense(ex);
// или
BUDGET_MANAGER.addExpense(ex,
ACCOUNT_MANAGER.getUser(providedEmail).getBudget("MyBudget"));
```

```
// премахване на потребител от споделен бюджет
try{
    BUDGET_MANAGER.kickFromBudget(myBudget, user)
}
catch (KickOwnerException ex){
    ErrorForm.Load(ex.Message);
}
```

4.3. ReportManager

- Предназначение: Имплементира функционалностите, свързани с Отчетите
- Основни отговорности:
    - Изчислява необходимите за отчетите статистически данни
    - Обединява данните за един отчет в инстанция на клас Report

- Интерфейси: ReportManager

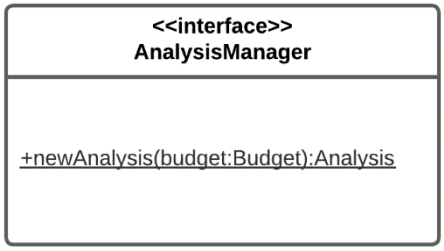


Диаграма 13: Интерфейсът ReportManager, отговарящ за генерирането на отчети

```
// Генериране на Report
Report MyReport = REPORT_MANAGER.newReport(user);
```

4.4. AnalysisManager

- Предназначение: Имплементира функционалностите, свързани с Анализи
- Основни отговорности:
  - Изчислява необходимите за анализите статистически данни
  - Обединява данните за един анализ в инстанция на клас Analysis
- Интерфейси: AnalysisManager



Диаграма 14: Интерфейсът AnalysisManager, отговарящ за генерирането на анализи

4.5. Security

- Предназначение: Осигурява сигурността на системата чрез защита от неправомерен достъп и криптиране на данните
- Основни отговорности:
  - Криптира данните, които се изпращат от сървъра
  - Поддържа списък от влезлите в системата потребители
  - Потвърждава влизането в системата в съвместна работа с модула Account Manager
  - Изисква допълнителни данни за потвърждение на самоличността при съмнение за неправомерен достъп

4.5.1. Encryption

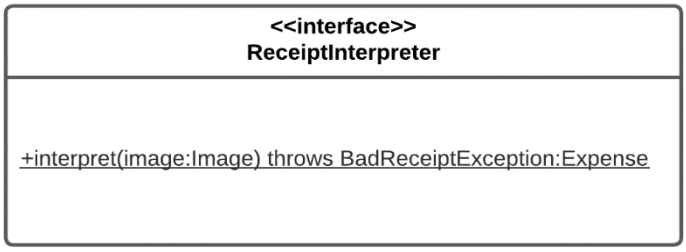
- Предназначение: Криптира данните, изпращани до локации извън сървъра
- Основни отговорности:
  - Криптира данните чрез разнообразие от алгоритми като AES, DES и т.н.
  - Периодично променя методът на криптиране и уведомява модулите, които изпращат данните за криптиране, за промяната.

4.5.2. Authorization

- Предназначение: Предотвратява неправомерен достъп до системата и потвърждава самоличността на влизащите потребители
- Основни отговорности:
    - Поддържа списък от влезлите в системата потребители
    - Потвърждава влизането на потребители в системата в съвместна работа с модула Account Manager
    - Изисква допълнителни данни за потвърждение на самоличността на потребителя при съмнение за неправомерен достъп

4.6. ReceiptInterpreter

- Предназначение: Дигитализира данните за даден разход при добавяне на снимка на касова бележка
- Основни отговорности:
    - Интерпретира изображение на касов бон чрез алгоритми (например на основата на AI)
    - Превръща интерпретираните данни в инстанция на Expense
- Интерфейси: ReceiptInterpreter



Диаграма 15: Интерфейсът ReceiptInterpreter, отговарящ за интерпретирането на касови бонове

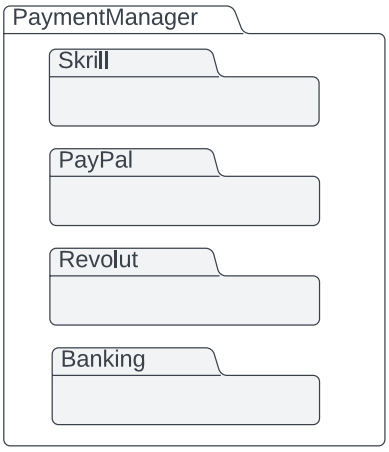
4.7. DBMS

- Предназначение: Осигурява връзката между бизнес логиката на софтуера и базата данни
- Основни отговорности:
    - Превръща изисквани от системата данни в SQL заявки
    - Изпраща SQL заявките до Базата данни
    - Интерпретира върнатите от Базата данни резултати и ги превръща в обекти
    - Оптимизира използването на базата данни чрез употребата на StoredProcedures и използване на данни от вече направени заявки
- Интерфейси: DBMS

4.7.1. StoredProcedures

- Предназначение: Превръща изискваната информация в SQL заявка/и
- Основни отговорности:
    - Превръщане на обект/и в кортежи от Базата данни
    - Изпращане на заявките до Базата и изчакване на отговор
- Превръщане на отговора в обект

5. PaymentManager



Диаграма 16: Модулът PaymentManager и неговите подмодули, имплементиращи операциите с външни системи за плащане

- Предназначение: Имплементира работа с външни системи за плащане
- Основни отговорности:
  - Извършва надеждни и сигурни (криптирани) плащания
  - Превръща данни за плащане чрез някоя от тези системи в Expense обект, който може да бъде обработен (например добавен към бюджет)
- Интерфейси: PaymentManager

5.1. Skrill	Имплементира плащането с външната система Skrill	1. Извършва плащанията в Skrill 2. Изпраща имейл на потребителя при плащане 3. Превръща данни за плащане в Skrill в инстания на Expense
5.2. PayPal	Имплементира плащането с външната система PayPal	1. Извършва плащанията в PayPal 2. Изпраща имейл на потребителя при плащане 3. Превръща данни за плащане в PayPal в инстания на Expense
5.3. Revolut	Имплементира плащането с външната система Revolut	1. Извършва плащанията в Revolut 2. Изпраща имейл на потребителя при плащане 3. Превръща данни за плащане в Revolut в инстания на Expense
5.4. Banking	Имплементира плащането по банков път	1. Извършва плащанията по банков път 2. Изпраща имейл на потребителя при плащане[1]

[1] Тази отговорност се извършва само при активирана от потребителя опция.

6. BackupServer

- Предназначение: Осигурява резервен сървър в случай на отказ на активния основен сървър (Active Redundancy)
- Основни отговорности:
  - Поема извършването на функционалността на системата в случай на отказа на основния сървър
  - Замества главния сървър при планирана профилактика (например при прилагане на тактиката Извеждане от употреба)
- Интерфейси: BackupServer

Модул	Архитектурна обосновка
1. Client	Архитектурата на системата е "Клиент-Сървър". Клиентът изпълнява ролята да представя данните, а сървърът - да ги генерира, обработва и предоставя. Освен разликата във функционалността, има разлика и в разположението на физическите устройства, върху които ще се изпълнява кода на клиента. В допълнение, клиентът има няколко разновидности със съществени различия в имплементацията. Затова е необходимо кодът за клиента да е отделен в собствен модул.
1.1. iOS	Имплементацията за iOS се различава от тази за Android, която се различава от тази за Web(2.1). Необходим е модул за всеки от тези случаи, защото макар и сходна, функционалността не е идентична. Освен това разделението улеснява тестването, редактирането и повторната употреба на кода.
1.2. Android	
1.{1,2}.1. UI	Клиентската част от приложението има две задачи: да получи данните от сървъра и да ги представи на потребителя. Втората задача е по-специфична, докато логиката е неразделна част от клиента. Освен това, разделяйки потребителския интерфейс от останалия код за конкретния вид клиент, се увеличава преизползваемостта на кода.
1.3. API	Въпреки своите различия, трите варианта за клиент имат обща характеристика: всички те ще се обръщат към сървъра с еднотипни заявки. Ако сървърът бъде имплементиран да приема и отговаря с HTTP заявки (например чрез XML или REST API + JSON), и трите модула могат да използват заявките, които модулът API ще генерира, изпраща, получава и интерпретира.
2. ClientManager	MainServer и BackupServer имплементират бизнес логиката, а ServerManager се грижи за поддръжката на сървърите, но е необходим и модул, който да посредничи между клиентите и същинската логика на сървъра, т.е, "да обслужва клиентите". На това е посветен ClientManager
2.1. Web	За разлика от iOS и Android случаите, Web клиентът няма къде да разположи своя код в машината на потребителя. Налага се да се обособи модул, който да предоставя front-end и back-end за уеб достъп до системата.

2.2. RequestHandler	Клиентът и сървърът комуникират помежду си не чрез обекти, а чрез HTTP заявки. Client "превежда" за клиента отговорите на сървъра. Огледално, налага се да има модул, който да "превежда" HTTP заявките на клиента до познатите за сървъра обекти. Още повече, че заявка на клиента може да съдържа привидно атомарно за клиента действие, което за сървъра обаче е поредица от действия. RequestHandler ще "инструктира" сървъра коя заявка в кой ред да се изпълни и ще обединява резултата
2.3. NameServer	С цел повишаване на надеждността, системата ще използва тактиката Redundancy. За целта, сървърната част ще разполага с два сървъра - основен и резервен, който ще дублира първия. Когато клиент иска да подаде заявка към сървърната част, той не може да знае кой от двата сървъра е поел изпълнението. Целта на модула NameServer е да пренасочва трафика към съответния активен сървър, получавайки информация кой е той от модула ServerManager. Тоест, NameServer (и изобщо ClientManager) е вариация на Фасада, която добавя още една преграда между сървъра и външната част. Обособен е модул NameServer вместо запитванията да се извършват директно към ServerManager, за да се намали натоварването към модула ServerManager.
3. ServerManager	Наличието на два сървъра също поражда проблеми - необходимо е данните да бъдат синхронизирани между работещата и резервната част, а също да се следи за грешки и сривове и в тези случаи отговорностите да бъдат прехвърлени на резервната част и обратно. Това са сложни задачи, които регулират работата на цялата сървърна част, а не само на един сървър (в смисъл на устройство) и затова е необходимо да бъдат в собствен модул, който отговаря за тези функционалности. Трите подзадачи са разпределени в собствени модули, защото са различни една от друга по своята същност и по кода си, но всички са подмодули на този, защото са силно зависими една от друга (имат сравнително висока свързаност помежду си, но ниска с останалите модули).
3.1. Synchronization	Този модул отговаря изцяло за синхронизацията на данните и състоянията между Основния и Резервния сървър. Неговата функционалност съществено се различава от тази на останалите модули. Затова е отделена от останалия код.
3.2. ServerMonitor	Необходимо е да се правят постоянни периодични проверки за състоянието на основния сървър, за да може в случай на отказ той да бъде заменен възможно най-скоро и ServerManager и DRCenter да "разберат" най-бързо за това събитие. Единственият начин това да стане бързо и независимо от останалите операции е поместването в собствен модул. Проверките за състояние биха

	могли да се осъществяват чрез тактиката Откриване на отказ (Ping или Echo).
3.3. DiagnosticsAndRepair	Програмистите и администраторите са хора, тоест невинаги са налице. Грешките и сривовете могат да станат по всяко време на денонощието при употреба на системата от различни части на света. Често грешките в един сървър са сходни по източник и резултат, съответно и начините за отстраняване са сходни. Този модул отговаря за три задачи: да се опита да отстрани настъпилите грешки и откази в Основния или Резервния сървър, да извършва периодично "Извеждане от употреба" с цел профилактика и да документира всички настъпващи събития в системата, за да може в случай на фатални грешки администраторът да успее да разбере източника им от лог файловете.
4. MainServer	Този модул е посветен на основните функционалности, които потребителят ще може да извършва в системата. При това - собственият код, а не код от външни системи (като например PaymentManager). Тези функционалности нямат общо с чисто програмистките действия по поддръжка на софтуера и хардуера (например ServerManager) и затова са в собствен модул.
4.1. AccountManager	Администрирането на акаунтите е сред най-важните дейности в системата. Сложна е и сама по себе си може да е или за Регистриране, или за вписване. С цел лесно тестване, промяна и преизползване на кода, той е поместен в този собствен модул.
4.1.1. Register	Регистрацията се различава от вписването по различни проверки - за съществуване на имейла/акаунта; извършва се потвърждение на регистрацията. Затова е в собствен модул.
4.1.1.1. GoogleRegister	Всеки от тези модули ще се различава значително от останалите. Google и Facebook ще съдържат код от външни библиотеки и адаптери, които да го свеждат до важните за тази система данни, докато Email ще съдържа собствен код. Редно е да не бъде смесван собствен код с код от външни системи. Освен това, това разделение ще позволи лесна замяна на кода (а Google Identity API и Facebook Login често търпят промени с цел максимална сигурност. Възможно е тези промени да се отразят и на тази система)
4.1.1.2. FacebookRegister	
4.1.1.3. EmailRegister	
4.1.2. Login	Вписването се различава от регистрацията - трябва да се правят проверки за идентичността на потребителите и дали вече са вписани в системата. Затова вписването е в собствен модул.
4.1.2.1. GoogleLogin	Аналогично за огледалните модули в Login, и тук има съществени разлики в имплементацията на трите случая.
4.1.2.2. FacebookLogin	
4.1.2.3. EmailLogin	
4.2. BudgetManager	Системата е предназначена за финансово счетоводство и самоотчет. Това е сърцевината на нейната функционалност. Тук ще се извършват операции, които по същество значително се различават от останалите в сървъра - затова кодът за тях е поместен в собствен модул. Този модул

	не е декомпозиран до по-детайлно, защото функционалността тук (опростено казано) представлява създаване/изтриване на бюджет, добавяне на разход към бюджет и добавяне на потребител към споделен бюджет. Тоест, модулът имплементира функционалностите, които се отнасят за *един* бюджет, чрез методи на интерфейса BudgetManager.
4.3. ReportManager	В секцията за Отчети могат да бъдат показани всички бюджети и наличните разходи за тях. Наличието на такава обобщена справка и още повече самото генериране на тези отчети е функционалност, която изисква обособяване в самостоятелен модул. Това позволява по-голяма свобода при имплементацията и ускоряване на генерирането. В допълнение, така се улеснява тестването и преизползваемостта както на BudgetManager, така и на ReportManager (аналогично за AnalysisManager). Генерирането на отчети може да се счете за функционалност, присъща и за BudgetManager (и това е възможна вариация в решението), но BudgetManager извършва действие за един бюджет, а ReportManager използва *множество* бюджети.
4.4. AnalysisManager	В секцията за Анализи могат да бъдат представени различни статистики за разходите по даден бюджет за даден период. Модулът отговаря за генерирането на самите анализи (извършване на изчисления), както и изпращането на известия за необичайна активност в разходите за даден бюджет. Тези задачи сами по себе си следва да бъдат отделени от останалите функционалности на системата по същия начин, както и за ReportManager.
4.5. Security	Сигурността е от ключова важност за системата поради работата с ценни лични данни на потребителите. Но осигуряването на сигурност на данните не бива да бъде смесвано с обработката на данните - т.е., това е функционалност, която трябва да бъде независима от останалите. Начинът да бъде постигната ниска свързаност е чрез самостоятелен модул.
4.5.1. Encryption	Кодирането на данните позволява те да останат защитени дори и при успешен неоторизиран достъп. По същество кодирането се различава от проверката на самоличността на потребителя и използва сложни и различни алгоритми. Затова е в собствен модул.
4.5.2. Authorization	Понякога въпреки употребата на пароли и ограничаването на времетраенето на потребителските сесии, е възможен неоторизиран достъп, но при определени условия той може да бъде различен от оторизирания. Този модул отговоря за изискването на по-подробна информация от потребителя при съмнение за неправомерен достъп - функционалност, която се различава съществено дори от криптирането. Затова е обособена в самостоятелен модул.

4.6. ReceiptInterpreter	Системата трябва да може да въвежда разход по предоставена касова бележка. Разчитането на изображения е сложна операция с характерни алгоритми, които съществено се различават от останалия код на системата. Функционалността е уникална сама по себе си и съществено се различава от останалия код.
4.7. DBMS	Комуникирането с базата данни е дейност, различаваща се от бизнес логиката, която обработва тези данни. Затова е в собствен модул.
4.7.1. StoredProcedures	Заявките, които ще се изпращат към базата данни се различават от Обектно-ориентирания модел на останалия код, но са еднотипни. Този модул отговаря за използването на Stored Procedures за взимане на информация от Базата данни. Но не всички операции с нея са Stored Procedures. Затова този модул е подмодул на DBMS, който поема тази по-конкретна функционалност. Останалите действия с Базата данни ще са имплементирани в DBMS.
5. PaymentManager	Плащането ще се извършва с външни системи, а не в нашата система. За разлика от билбиотеките за управление на акаунти на Google и Facebook, тези библиотеки не са толкова тясно свързани с тази система, а действията по плащане ще се извършват изцяло във външни за системата сървъри, а не частично в сървърите на ExpenseBuddy. Затова връзките с тях са отделени в собствен модул, който е извън MainServer.
5.1. Skrill	Всяка платежна система е имплементирана по различен начин и изисква различни данни. Затова кодът за всяка платежна система е отделен от този на останалите в собствен модул, въпреки че изпълнява една и съща по вид функционалност – “Плащане”.
5.2. PayPal	
5.3. Revolut	
5.4. Banking	
6. BackupServer	За да е ефективна тактиката Active Redundancy трябва дублирането да не е идентично с MainServer. Поради различната имплементация (и дори възможна различна модулна структура), кодът за резервния сървър е обособен в собствен модул, въпреки че изпълнява сходни на MainServer функции.

d. Описание на възможните вариации (ако е необходимо)

- Модулът BackupServer е приложение на тактиката Redundancy - той дублира модулите в MainServer. Важно е да се отбележи, че дублирането не трябва да е дословно, защото това предразполага bug или изключение, настъпили в MainServer да се случат и в BackupServer. Разликата между MainServer и BackupServer може да бъде под следните форми:
    - Имплементиране само на жизненоважните модули - AccountManager, ReportManager, Security и DBMS. В този случай по време на неизправността на MainServer ще бъдат налични само функционалностите по вписване, регистриране и преглед на информация. Няма да бъдат налични промени по бюджети като

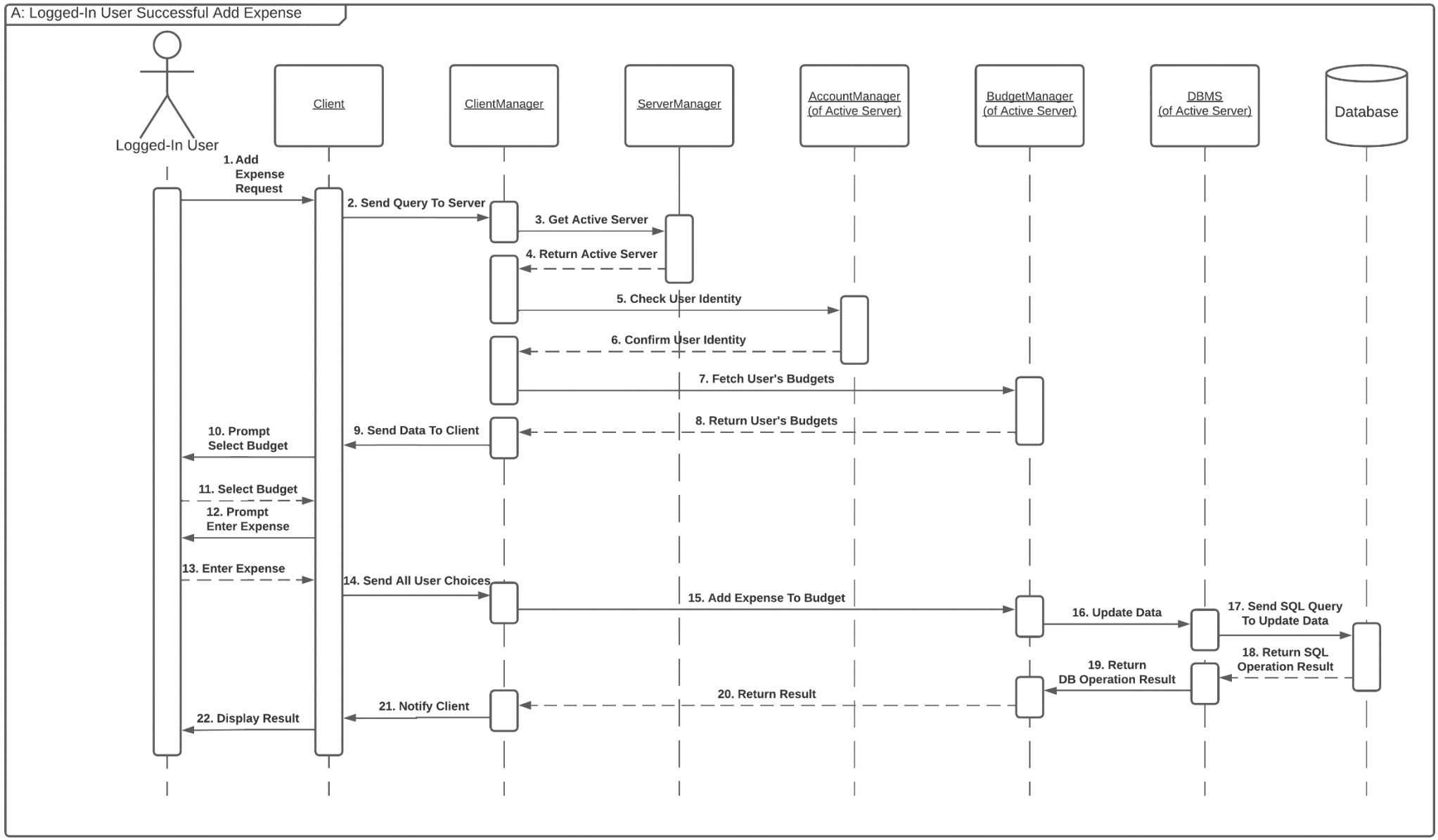
добавяне/премахване на разходи и/или бюджети и генериране на Анализи. Това решение би намалило цялостната наличност на системата, но спестява значителни разходи по разработката и позволява използването на по-евтин хардуер за BackupServer (който ще е разположен на друго физическо устройство от MainServer).

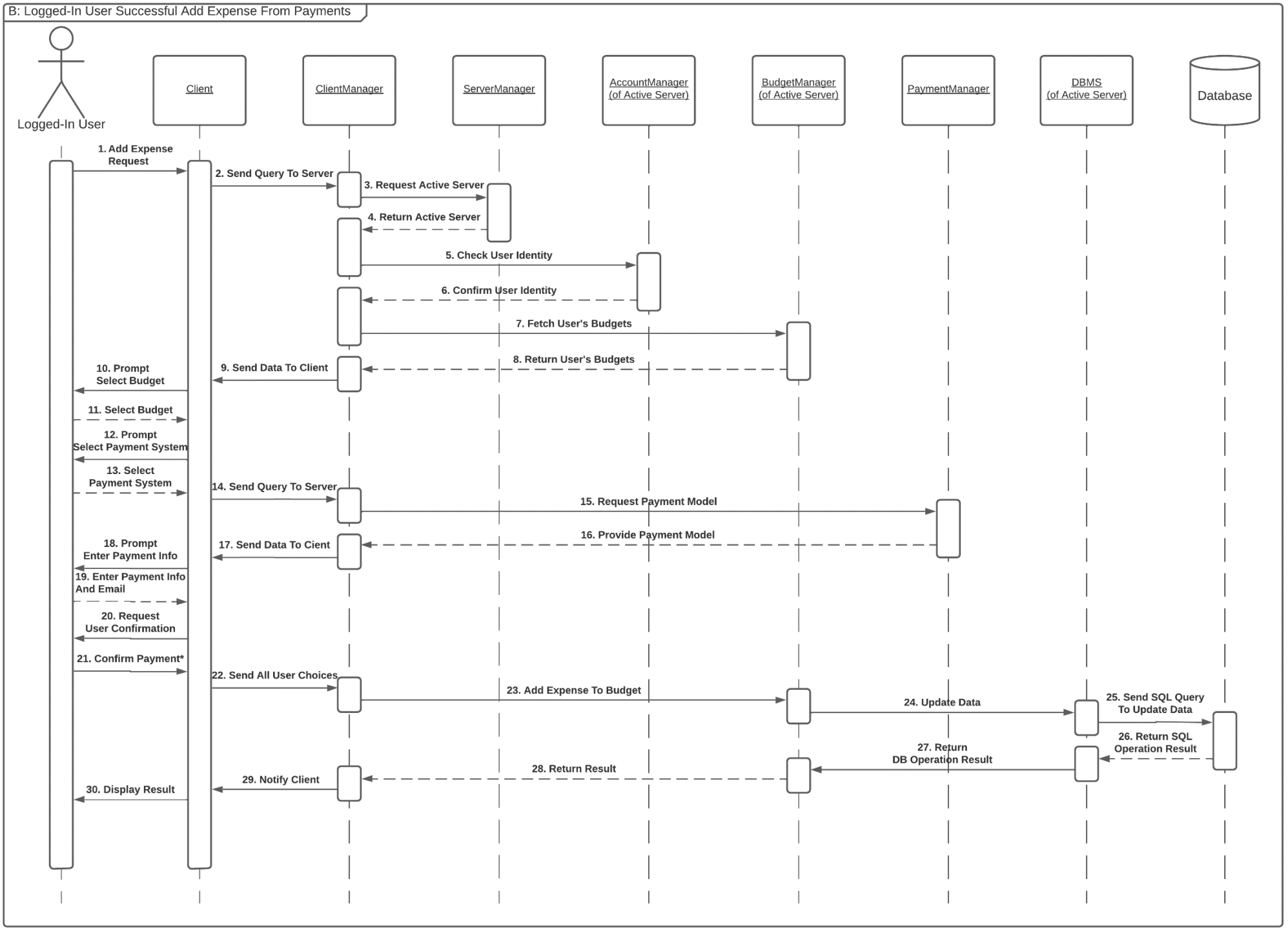
- Разпределяне на работата по MainServer и BackupServer на различни екипи, в които няма програмисти, участващи и в двата екипа едновременно. Това ще намали вероятността да се допуснат едни и същи логически грешки (оттам и bug-ове) в двата сървъра.

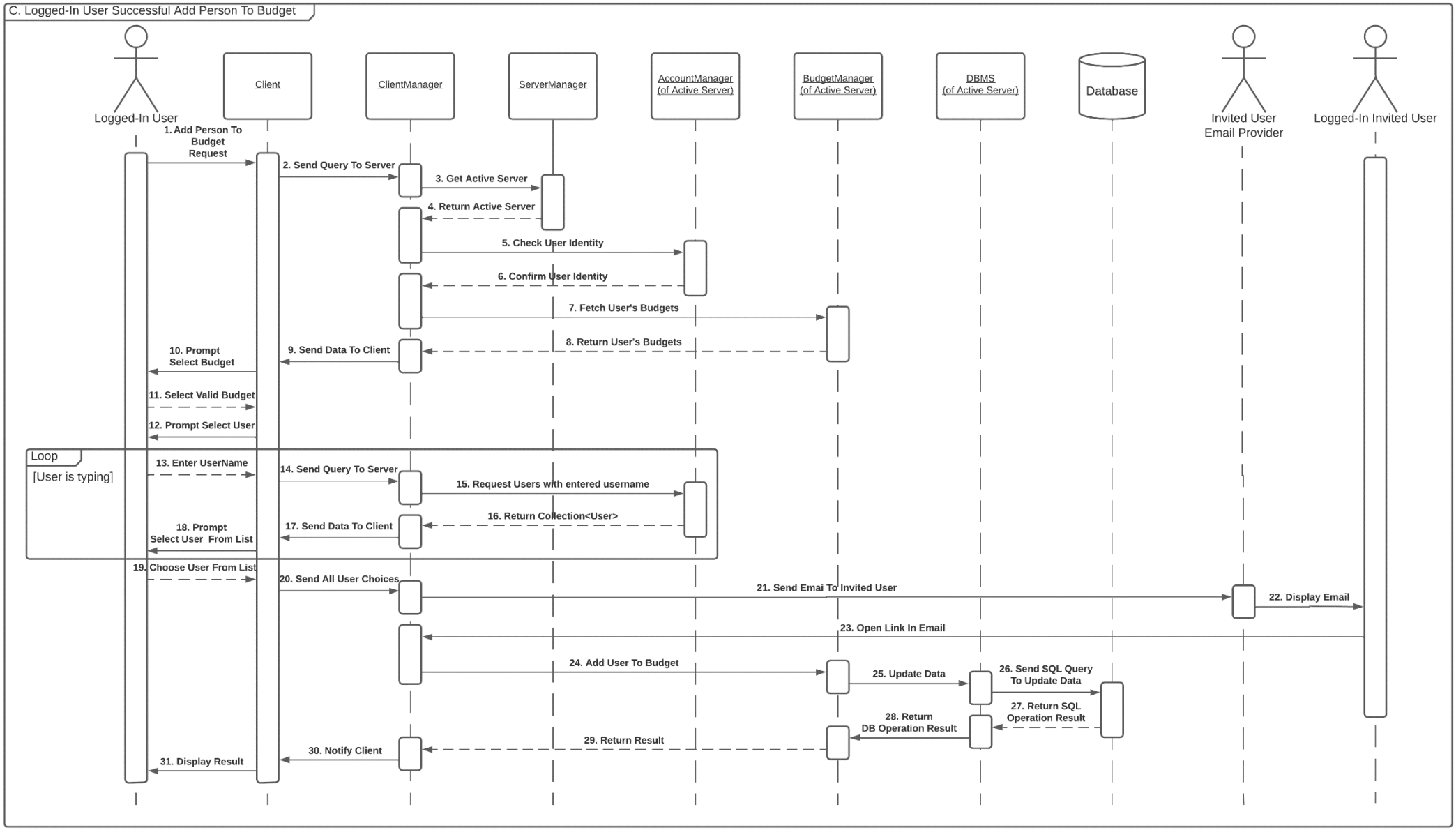
3. Описание на допълнителните структури

3.1. Структура на процесите

a. Първично представяне


A: Logged-In User Successful Add Expense
Logged-In User
Client
ClientManager
ServerManager
AccountManager (of Active Server)
BudgetManager (of Active Server)
DBMS (of Active Server)
Database
1. Add Expense Request
2. Send Query To Server
3. Get Active Server
4. Return Active Server
5. Check User Identity
6. Confirm User Identity
7. Fetch User's Budgets
8. Return User's Budgets
9. Send Data To Client
10. Prompt Select Budget
11. Select Budget
12. Prompt Enter Expense
13. Enter Expense
14. Send All User Choices
15. Add Expense To Budget
16. Update Data
17. Send SQL Query To Update Data
18. Return SQL Operation Result
19. Return DB Operation Result
20. Return Result
21. Notify Client
22. Display Result


B: Logged-In User Successful Add Expense From Payments
Logged-In User
Client
ClientManager
ServerManager
AccountManager (of Active Server)
BudgetManager (of Active Server)
PaymentManager
DBMS (of Active Server)
Database
1. Add Expense Request
2. Send Query To Server
3. Request Active Server
4. Return Active Server
5. Check User Identity
6. Confirm User Identity
7. Fetch User's Budgets
8. Return User's Budgets
9. Send Data To Client
10. Prompt Select Budget
11. Select Budget
12. Prompt Select Payment System
13. Select Payment System
14. Send Query To Server
15. Request Payment Model
16. Provide Payment Model
17. Send Data To Cient
18. Prompt Enter Payment Info
19. Enter Payment Info And Email
20. Request User Confirmation
21. Confirm Payment*
22. Send All User Choices
23. Add Expense To Budget
24. Update Data
25. Send SQL Query To Update Data
26. Return SQL Operation Result
27. Return DB Operation Result
28. Return Result
29. Notify Client
30. Display Result


C. Logged-In User Successful Add Person To Budget
Logged-In User
Client
ClientManager
ServerManager
AccountManager (of Active Server)
BudgetManager (of Active Server)
DBMS (of Active Server)
Database
Invited User Email Provider
Logged-In Invited User
1. Add Person To Budget Request
2. Send Query To Server
3. Get Active Server
4. Return Active Server
5. Check User Identity
6. Confirm User Identity
7. Fetch User's Budgets
8. Return User's Budgets
9. Send Data To Client
10. Prompt Select Budget
11. Select Valid Budget
12. Prompt Select User
Loop
[User is typing]
13. Enter UserName
14. Send Query To Server
15. Request Users with entered username
16. Return Collection<User>
17. Send Data To Client
18. Prompt Select User From List
19. Choose User From List
20. Send All User Choices
21. Send Emai To Invited User
22. Display Email
23. Open Link In Email
24. Add User To Budget
25. Update Data
26. Send SQL Query To Update Data
27. Return SQL Operation Result
28. Return DB Operation Result
29. Return Result
30. Notify Client
31. Display Result

b. Описание на елементите и връзките

№	Съобщение	Събитие	Обосновка
A1	Add Expense Request	Потребителят натиска елемент на графичния интерфейс, който започва операцията по добавяне на разход към бюджет. Модулът Client започва да изисква от сървъра необходимата информация	Модулът Client съдържа в имплементацията си инструкциите за добавяне на разход към бюджет (кои елементи от графичния интерфейс да покаже; директно изискване на бюджети). Така се избягва питането на сървъра за поредицата от действия, които трябва да се извършат, и се ускорява действието. Допитването кой е активният сървър става само веднъж, защото заявката се извършва за твърде кратко време, за да бъде ново допитване ефективно. Потвърждаването на самоличността става само веднъж, защото разговорът между Client и Server се осъществява като диалог чрез HTTP Request и HTTP Response и има начини за следене на потребителските сесии. След конкретизиране и на бюджета, и на разхода от страна на клиента, цялата информация се изпраща накуп, за да не се налага ClientManager да я съхранява до този момент. В противен случай ClientManager би бил претоварен модул, изпълняващ твърде много отговорности, но понеже той "посреща" трафика, това би довело до забавено
A2	Send Query To Server	Изпраща се заявка до сървъра чрез HTTP Request за предоставяне на необходимата информация. Модулът ClientManager свежда HTTP заявката до ООП класовете в системата.	
A3	Get Active Server	ClientManager изисква от ServerManager информация кой е активният сървър	
A4	Return Active Server	ServerManager връща информация кой е активният сървър	
A5	Check User Identity	ClientManager проверява идентичността на потребителя преди да изисква данните	
A6	Confirm User Identity	AccountManager потвърждава (или отхвърля) идентичността на потребителя, изпратил заявката	
A7	Fetch User's Budgets	Сега ClientManager може да извика необходимите методи на BudgetManager, които връщат всички бюджети на този потребител	
A8	Return User's Budgets	BudgetManager връща структура, съдържаща всички бюджети на този потребител	
A9	Send Data to Client	ClientManager превръща тези инстанции на класове в JSON формат и връща отговор на клиента чрез HTTPResponse	
A10	Prompt Select Budget	Модулът Client представя информацията за бюджетите на потребителя и го подканва да избере един от тях	
A11	Select Budget	Потребителят избира един бюджет чрез графичния интерфейс	
A12	Prompt Enter Expense	Модулът Client предоставя графичен интерфейс за ръчно въвеждане на разход	
A13	Enter Expense	Потребителят въвежда информацията за един разход в предоставения графичен интерфейс	
A14	Send All User Choices	Модулът Client изпраща накуп цялата информация, необходима за операцията - конкретен бюджет и конкретен разход	
A15	Add Expense To Budget	ClientManager превръща информацията в обекти и извиква необходимите методи на BudgetManager с тези обекти като параметри	
A16	Update Data	BudgetManager добавя бюджета, но това трябва да бъде отразено в	

		Базата данни. Изпраща се подкана да се обнови базата данни	изпълнение на заявките. Рискът за сигурността на данните е минимален, защото ще се прилага криптиране, а освен това съобщението ще съдържа цялостна информация само за разхода. Бюджетът ще бъде представен само чрез своето ID, което не носи информация за съдържанието.
A17	Send SQL Query To Update Data	Модулът DBMS приема новата информация и изпраща SQL заявка до базата данни	
A18	Return SQL Operation Result	Базата данни връща отговор дали заявката се е изпълнила успешно	
A19	Return DB Operation Result	Модулът DBMS връща информация дали операцията е минала успешно	
A20	Return Result	BudgetManager връща информация на ClientManager дали добавянето на бюджета е минало успешно	
A21	Notify Client	ClientManager превръща резултата в HTTP Response и го изпраща на клиента	
A22	Display Result	Модулът Client форматира пристигналия HTTP Response в четим за потребителя формат и го показва, за да го информира	

№	Съобщение	Събитие	Обосновка
B1	Add Expense Request	Потребителят натиска елемент на графичния интерфейс, който започва операцията по добавяне на разход към бюджет. Модулът Client започва да изисква от сървъра необходимата информация	Модулът Client съдържа в имплементацията си инструкциите за добавяне на разход към бюджет (кои елементи от графичния интерфейс да покаже; директно изискване на бюджети). Така се избягва питането на сървъра за поредицата от действия, които трябва да се извършат, и се ускорява действието. Допитването кой е активният сървър става само веднъж, защото заявката се извършва за твърде кратко време, за да бъде ново допитване ефективно. Потвърждаването на самоличността става само веднъж, защото разговорът
B2	Send Query To Server	Изпраща се заявка до сървъра чрез HTTP Request за предоставяне на необходимата информация. Модулът ClientManager свежда HTTP заявката до ООП класовете в системата.	
B3	Get Active Server	ClientManager изисква от ServerManager информация кой е активният сървър	
B4	Return Active Server	ServerManager връща информация кой е активният сървър	
B5	Check User Identity	ClientManager проверява идентичността на потребителя преди да изисква данните	
B6	Confirm User Identity	AccountManager потвърждава (или отхвърля) идентичността на потребителя, изпратил заявката	
B7	Fetch User's Budgets	Сега ClientManager може да извика необходимите методи на BudgetManager, които връщат всички бюджети на този потребител	
B8	Return User's Budgets	BudgetManager връща структура, съдържаща всички бюджети на този потребител	
B9	Send Data to Client	ClientManager превръща тези инстанции на класове в JSON формат и	

		връща отговор на клиента чрез HTTPResponse	между Client и Server се осъществява като диалог чрез HTTP Request и HTTP Response и има начини за следене на потребителските сесии. Всяка система за плащане използва различен формат за документиране на плащането. Затова е необходимо допитване до модула PaymentManager какъв точно е този формат. Като резултат може да се върне списък с необходими данни или дори хипервръзка, която потребителят да отвори и да го препрати към уебсайт на системата за плащане. След конкретизиране и на бюджета, и на разхода от страна на клиента, цялата информация се изпраща накуп, за да не се налага ClientManager да я съхранява до този момент. В противен случай ClientManager би бил претоварен модул, изпълняващ твърде много отговорности, но понеже той "посреща" трафика, това би довело до забавено изпълнение на заявките. Рискът за сигурността на данните е минимален, защото ще се прилага
B10	Prompt Select Budget	Модулът Client представя информацията за бюджетите на потребителя и го подканва да избере един от тях	
B11	Select Budget	Потребителят избира един бюджет чрез графичния интерфейс	
B12	Prompt Select Payment System	Модулът Client подканва потребителя да избере система за плащане от предварително програмиран списък	
B13	Select Payment System	Потребителят избира система за плащане от списъка	
B14	Send Query To Server	Модулът Client изпраща заявка до сървъра за модел на плащането с избраната система за плащане (т.е какви данни се използват в тази система за плащането)	
B15	Request Payment Model	ClientManager превръща HTTP заявката в обекти и извиква методите на модула PaymentManager с тях	
B16	Provide Payment Model	Модулът PaymentManager връща информацията за модела на плащане	
B17	Send Data To Client	ClientManager превръща информацията в JSON формат и изпраща HTTP Response	
B18	Prompt Enter Payment Info	ClientManager подканва потребителя да въведе информацията за плащане по модела, който PaymentManager е предоставил	
B19	Enter Payment Info And Email	Потребителят въвежда информацията. Валидацията става чрез графичния интерфейс в Client.	
B20	Request User Confirmation	Client изпраща молба за потвърждение до потребителя (чрез имейл или изскачащ прозорец)	
B21	Confirm Payment	Потребителят потвърждава плащането: 1. Ако е чрез имейл, чрез кликване върху линк в имейла, който препраща към графичния интерфейс на Client 2. Ако е чрез изскачащ прозорец – чрез натискане на елемент от графичния интерфейс	
B22	Send All User Choices	След потвърждаване на плащането, информацията за него се изпраща до сървъра.	
B23	Add Expense To Budget	ClientManager превръща информацията в обекти и извиква необходимите методи на BudgetManager с тези обекти като параметри	
B24	Update Data	BudgetManager добавя бюджета, но това трябва да бъде отразено в Базата данни. Изпраща се подкана да се обнови базата данни	

B25	Send SQL Query To Update Data	Модулът DBMS приема новата информация и изпраща SQL заявка до базата данни	криптиране, а освен това съобщението ще съдържа цялостна информация само за разхода. Бюджетът ще бъде представен само чрез своето ID, което не носи информация за съдържанието.
B26	Return SQL Operation Result	Базата данни връща отговор дали заявката се е изпълнила успешно	
B27	Return DB Operation Result	Модулът DBMS връща информация дали операцията е минала успешно	
B28	Return Result	BudgetManager връща информация на ClientManager дали добавянето на бюджета е минало успешно	
B29	Notify Client	ClientManager превръща резултата в HTTP Response и го изпраща на клиента	
B30	Display Result	Модулът Client форматира пристигналия HTTP Response в четим за потребителя формат и го показва, за да го информира	

№	Съобщение	Събитие	Обосновка
C1	Add Expense Request	Потребителят натиска елемент на графичния интерфейс, който започва операцията по добавяне на разход към бюджет. Модулът Client започва да изисква от сървъра необходимата информация	Модулът Client съдържа в имплементацията си инструкциите за добавяне на потребител към бюджет (кои елементи от графичния интерфейс да покаже; директно изискване на бюджетите на потребителя, допитване за потребители на базата на критерии за търсене). Така се избягва питането на сървъра за поредицата от действия, които трябва да се извършат, и се ускорява действието. Допитването кой е активният сървър става само веднъж, защото заявката се извършва за твърде кратко време, за да бъде ново допитване ефективно. Потвърждаването
C2	Send Query To Server	Изпраща се заявка до сървъра чрез HTTP Request за предоставяне на необходимата информация. Модулът ClientManager свежда HTTP заявката до ООП класовете в системата.	
C3	Get Active Server	ClientManager изисква от ServerManager информация кой е активният сървър	
C4	Return Active Server	ServerManager връща информация кой е активният сървър	
C5	Check User Identity	ClientManager проверява идентичността на потребителя преди да изисква данните	
C6	Confirm User Identity	AccountManager потвърждава (или отхвърля) идентичността на потребителя, изпратил заявката	
C7	Fetch User's Budgets	Сега ClientManager може да извика необходимите методи на BudgetManager, които връщат всички бюджети на този потребител	
C8	Return User's Budgets	BudgetManager връща структура, съдържаща всички бюджети на този потребител	
C9	Send Data to Client	ClientManager превръща тези инстанции на класове в JSON формат и връща отговор на клиента чрез HTTPResponse	
C10	Prompt Select Budget	Модулът Client представя информацията за бюджетите на потребителя и го подканва да избере един от тях (като предварително е	

		филтрирал тези, които са достигнали лимита за споделяне от 5ма души)	на самоличността става само веднъж, защото разговорът между Client и Server се осъществява като диалог чрез HTTP Request и HTTP Response и има начини за следене на потребителските сесии. След конкретизиране и на бюджета, и на потребителя от страна на клиента, цялата информация се изпраща накуп, за да не се налага ClientManager да я съхранява до този момент. В противен случай ClientManager би бил претоварен модул, изпълняващ твърде много отговорности, но понеже той "посреща" трафика, това би довело до забавено изпълнение на заявките. Рискът за сигурността на данните е минимален, защото ще се прилага криптиране, а освен това съобщението ще съдържа цялостна информация само за разхода. Бюджетът ще бъде представен само чрез своето ID, което не носи информация за съдържанието.
C11	Select Valid Budget	Потребителят избира един бюджет чрез графичния интерфейс	
C12	Prompt Select User	Client подканва потребителя да въведе потребител	
C13	Enter UserName	Потребителят започва да въвежда информация за човека, с когото иска да сподели бюджета	
C14	Send Query To Server	Client изпраща заявка до сървъра за списък от всички потребители, които отговарят на въведените от потребителя данни	
C15	Request Users with Entered Usernames	ClientManager преработва заявката до инстанции на обекти и извиква необходимите методи на AccountManager, които връщат списък от потребители, отговарящи на критериите	
C16	Return Collection<User>	AccountManager връща списък на потребителите, които отговарят на критериите за търсене	
C17	Send Data To Client	ClientManager форматира до JSON формат данните и ги изпраща на клиента	
C18	Prompt Select User From List	Client разчита данните и ги представя на потребителя за избор. Потребителят може да избере човек от предоставения списък или да продължи да пише информация в полето/полетата за търсене докато не получи удовлетворителен резултат. Ако потребителят продължи да пише, действията C13-C18 се повтарят циклично.	
C19	Choose User From List	Потребителят избира човек от предоставения списък.	
C20	Send All User Choices	Модулът Client изпраща накуп цялата информация, необходима за операцията - конкретен бюджет и конкретен потребител за добавяне	
C21	Send Email To Invited User	ClientManager изпраща имейл до поканения потребител (до неговия Email Provider)	
C22	Display Email	Email Provider (част от обкръжението) представя полученото съобщение (имейл), използвайки собствена имплементация и интерфейс (външни за системата), на поканения потребител	
C23	Open Link In Email	Поканеният потребител отваря хипервръзката, която пренасочва към сървъра и извършва заявката, добавяща го към бюджета	
C24	Add User To Budget	ClientManager извършва заявката, като превръща информацията в обекти и извиква необходимите методи на	

		BudgetManager с тези обекти като параметри	
C25	Update Data	BudgetManager добавя потребителя към бюджета, но това трябва да бъде отразено в Базата данни. Изпраща се подкана да се обнови базата данни	
C26	Send SQL Query To Update Data	Модулът DBMS приема новата информация и изпраща SQL заявка до базата данни	
C27	Return SQL Operation Result	Базата данни връща отговор дали заявката се е изпълнила успешно	
C28	Return DB Operation Result	Модулът DBMS връща информация дали операцията е минала успешно	
C29	Return Result	BudgetManager връща информация на ClientManager дали добавянето на потребител към бюджета е минало успешно	
C30	Notify Client	ClientManager превръща резултата в HTTP Response и го изпраща на клиента	
C31	Display Result	Модулът Client форматира пристигналия HTTP Response в четим за потребителя формат и го показва, за да го информира	

c. Описание на обкръжението

Добавянето на потребител към бюджет изисква потвърждение от втория потребител, което се осъществява чрез имейл. Следователно потвърждението зависи от външна система - доставчик на електронна поща.

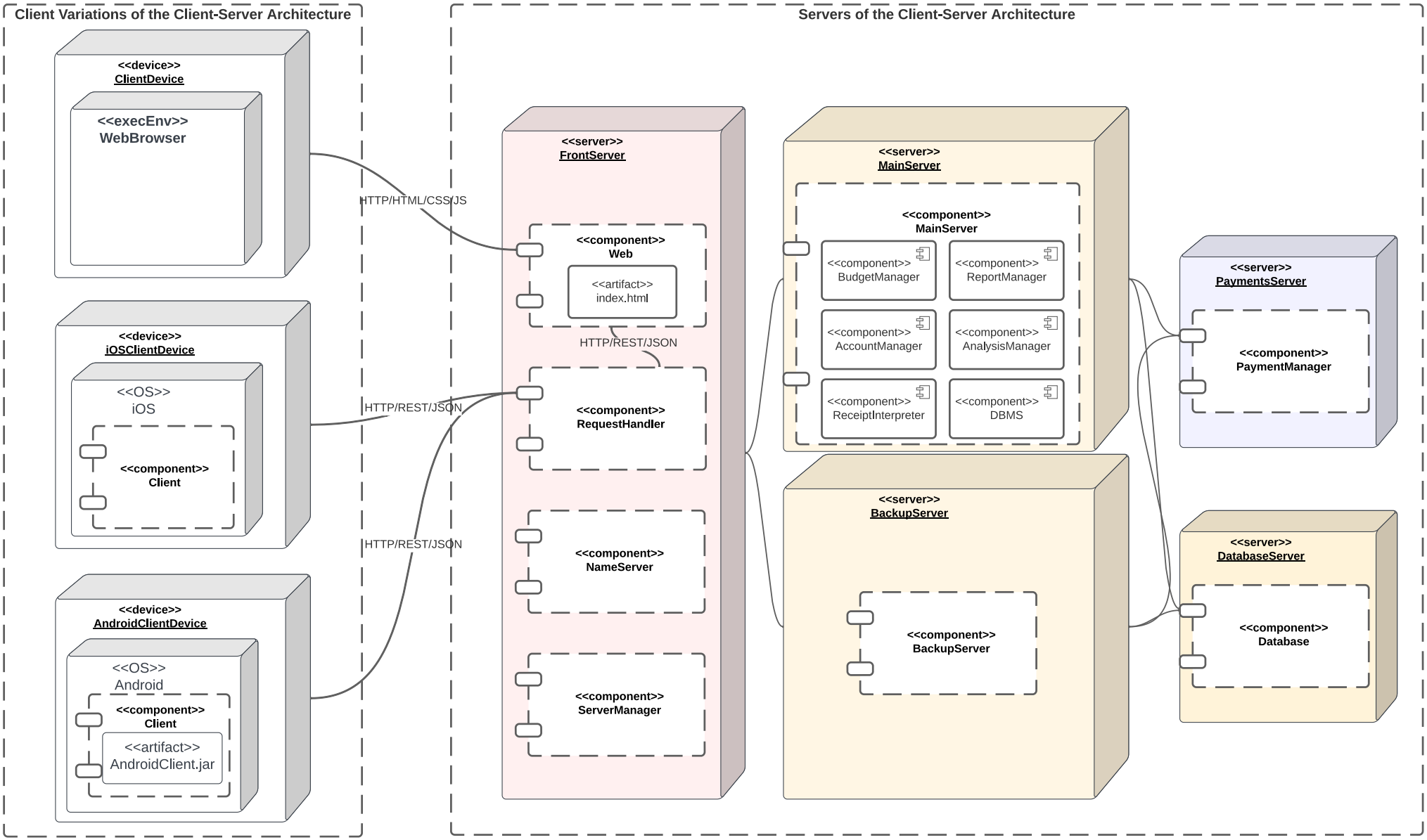
Добавянето на разход към бюджет чрез външни системи зависи от тези външни системи. При неизправност вътре в тях, настъпва неизправност и в тази система. За щастие неизправността тук ще е само временната липса на тази функционалност, а има и други две възможности за добавяне на разход към бюджет.

d. Описание на възможните вариации

Разположението на различните процеси върху различен хардуер се влияе непосредствено от разположението на модулите, в които се извършват тези процеси, върху хардуера. Невъзможно е процес, предизвикан от даден модул, да се намира на хардуер, различен от този на модула. Разпределението на модулите върху хардуера и възможните вариации са описани в **3.2. Структура на разположението.**

3.2. Структура на разположението

a. Първично представяне



Диаграма 17: UML диаграма на внедряването на цялата система

b. Описание на елементите и връзките

Системата ще бъде изградена основно в стила Клиент-Сървър. Следователно тя ще бъде разположена на две основни физически места:

- Машината на потребителя (клиента)
    - Вариант iOS

    IOS клиентът ще представлява приложение, което е написано на Swift и ще се изпълнява на операционната система iOS. Това мобилно приложение ще съдържа инстанция на модула Client в себе си.
    - Вариант Android

    Android клиентът, подобно на iOS клиентът, ще съдържа приложение, написано да се изпълнява на операционната система Android и ще съдържа екземпляр на модула Client в себе си. Машината ще съдържа като артефакт програмни файлове на Java/Kotlin
    - Вариант Web Browser

    Машината на клиента няма как да съдържа код за клиентската част. За целта, достъпът ще се осъществява чрез уеб браузър, кодът за който ще се помещава на сървъра в модула Web, който е аналогичен на модула Client в първите два варианта.

И трите модула ще изпращат HTTP заявки до сървъра (в REST стил и данни в JSON формат), по-точно до модула ClientManager.RequestHandler в сървърната част, който ще ги обработва.

- Сървърът

Сървърната част ще представлява съвкупност от няколко сървъра (физически устройства), чиято взаимна зависимост е сведена до минимум:

- FrontServer - посреща потребителския трафик и управлява сървърите за бизнес логиката

  Съдържа в себе си модулите:
    - ClientManager – осигурява Web клиентът на системата; посреща потребителските заявки и ги свежда до ООП модела на MainServer и BackupServer
    - ServerManager - контролира състоянието на сървърите за бизнес логика и осигурява надеждността на системата
- MainServer - изпълнява бизнес логиката и съдържа модулът MainServer
- BackupServer - изпълнява бизнес логиката, когато BackupServer е извън строя; съдържа модулът BackupServer
- PaymentsServer - Помещава единствено модулът PaymentManager, за да се осигури максимална надеждност на работата с външни системи за плащания и за да може да се ползва и от MainServer, и от BackupServer и да не се налага дублирането му
- DatabaseServer - помещава базата данни

c. Описание на обкръжението

Архитектурата “клиент-сървър” е надеждна и ефективна, но и тя има недостатък - за функционирането ѝ е необходима надеждна интернет връзка. Този недостатък не може да бъде преодолян поради същността на системата - да обслужва множество клиенти от разстояние. Единственото, което системата може да гарантира, е предвиждане и обработка на грешките във всички случаи, когато се прекъсне връзката между клиент и сървър, за да се осъществи бързо възстановяване по-късно. Аналогична е ситуацията между сървъра и системите за външни плащания. Тяхната работоспособност не зависи от тази система и в случай на отказ на някоя тях, системата ExpenseBuddy ще е принудена да изчака до отстраняването на неизправността. За щастие, в този случай всички останали функционалности освен плащането чрез външна система ще бъдат налични.

d. Описание на възможните вариации

- Модулът BackupServer дублира функционалността на MainServer, но не и имплементацията му. Следователно физическото устройство BackupServer може да предоставя по-малко изчислителни мощности от MainServer.
- С цел максимално разделение на функционалността е възможно модулът Web да се помещава на свое собствено физическо устройство (сървър), но това решение е оправдано само ако:
    - потребителите използват предимно Web клиенти за сметка на мобилни приложения
    - Backend-ът на Web клиента изисква значителни изчислителни мощности
    - Има финансови възможности за осигуряването на ново устройство (сървърите са скъпа техника)
- Възможно е модулът PaymentManager да бъде дублиран в MainServer и BackupServer и физическото устройство PaymentsServer да бъде премахнато (например поради финансови причини).
- Възможно е модулът ServerManager да бъде поместен в собствено физическо устройство, ако се окаже, че диагностиката на сървърите отнема значителни ресурси, от които ClientManager се нуждае повече.
- Дублиране може да се направи и на базата данни, но това поражда проблеми по синхронизацията на информацията в двете бази. В допълнение оскъпява системата. Може да се приложи при голямо увеличение на количеството данни, което няма да настъпи изведнъж. Затова една база данни е гарантирано достатъчна за системата в първите месеци на експлоатация.
- Вместо собствена база данни може да се използва външен доставчик (External Database Provider).

4. Архитектурна обосновка

Изискване[1]	***Драйвер***	***Обосновка***
1. *Достъпът до системата трябва да може да се осъществява или през браузър, или чрез мобилен клиент за iOS и Android*	*Да*	*Изискването се удовлетворява от модулът Client, който ще бъде разполаган на всяко iOS/Android физическо устройство, и модулът Web за уеб клиента, който е разположен в сървъра, в модула ClientManager. Модулът Client съдържа подмодул UI, който позволява разделянето на чисто визуалния елемент на системата от логиката за задвижване на графичния интерфейс. Модулът API увеличава преизползваемостта на кода, предоставяйки унифициран за всички видове клиенти начин на комуникация със сървъра.*
2. *Системата поддържа 2 типа потребители:* – *Обикновени потребители, които могат да използват ограничен набор от функционалностите на системата* – *Привилегировани потребители, които имат достъп до всички функционалности на системата*	*Да*	*Двете потребителски роли се обслужват от клас User, който съдържа поле за типа на потребителя, от модула AccountManager, който позволява регистрирането и вписването на тези два вида потребители, и имплементацията на модула BudgetManager, която връща резултати, съответстващи на ролята на потребителя.*
3. *Потребител може да се регистрира в системата чрез имейл, потребителско име и парола, или чрез връзка с външна система. Потребител може да се впише в системата чрез имейл или потребителско име и парола, или чрез външна система.* *a. Възможни външни системи са например Google и Facebook.* *b. Системата трябва да предоставя възможност за добавяне на допълнителни външни системи.*	*Да*	*За регистрация и вписване, системата разполага с модула AccountManager, който отделя тази функционалност от останалия код на системата. Трите начина на вписване (имейл, Google, Facebook) са твърде различни един от друг, а два от тях се имплементират чрез външни системи. Осигуряването на подмодул за всеки вид регистрация и вписване осигурява необходимата независимост на елементите в кода и увеличава неговата преизползваемост. Значително се улеснява и отстраняването на грешки.*
4. *Бюджетът представлява съвкупност от разходи. Всеки разход съдържа следната информация:* ***a.*** *списък на закупените продукти/услуги и цена за всеки от тях* ***b.*** *търговец* ***c.*** *дата на разхода* ***d.*** *категория* ***e.*** *начин на плащане*	*Не*	*Данните за бюджет се съхраняват в базата данни (поместена в сървърната част на системата), където с помощта на DBMS модула се превръщат в Разход и Бюджет обекти, които BudgetManager обработва.*

f. *опционално: при споделен бюджет се вписва автоматично името на потребителя, въвел разхода* ***g.*** *опционално: бележки от потребителя*		
5. *Системата трябва да поддържа два типа бюджет:* *a. Индивидуален - право на достъп има само потребителят, който е създал бюджета.* *b. Споделен - създава се от един потребител и има възможност да добавя към него до 4 нови потребителя.* *i. Добавяне на потребител към споделен бюджет се случва чрез изпращане на имейл покана.*	*Не*	*Разликата във видовете бюджет би могла да се имплементира чрез два класа, които наследяват общ интерфейс и чрез полиморфизъм позволяват унифицирано използване от BudgetManager и другите компоненти в системата* *Имейл поканата е осъществена в имплементацията на клиента.*
6. *Всеки потребител може да създава бюджети или да се присъедини към създаден от друг потребител споделен бюджет*	*Не*	*Това изискване е изпълнено чрез интерфейсите BudgetManager и AccountManager в едноименните модули.*
7. *Нов разход може да бъде добавен по следните начини:* *a. чрез снимка на касов бон: Потребител може да направи или качи снимка на касов бон и системата автоматично ще попълни формата за разход с данни от касовия бон, а на база на търговеца приел плащането, системата определя към коя категория той принадлежи. След потвърждение от страна на потребителя, разходът бива добавен към избрания от него бюджет* *b. чрез връзка с онлайн система за плащания (пример PayPal, Revolut, Skrill, банкови системи) - Потребител може да свърже бюджета си с онлайн системи за плащания. В този случай при извършване на плащане, потребителят ще получава известие или имейл, чрез който може да потвърди плащането и да го добави нов разход към бюджета на база*	*Да*	*Добавянето на разход чрез снимка на касов бон се осъществява с помощта на модула ReceiptInterpreter, който разчита касовите бонове и ги превръща в обект от тип Разход(Expense). Системата определя категорията в имплементацията на BudgetManager. Плащането с външни системи се осъществява чрез модула PaymentManager, където за всяка система за плащане има подмодул, който го имплементира. Така се гарантира независимост в методите на плащане на различните системи, лесно тестване и отстраняване на грешки и преизползваемост на кода. Ръчното добавяне на разход се имплементира чрез модула Client, който осигурява компоненти на графичния интерфейс, които разчитат и предават за интерпретация от модула въведените от потребителя данни. Впоследствие те се изпращат чрез HTTP комуникация до сървъра, където обратно се преобразуват в обекти, които се обработват в системата и се записват в базата данни.*

плащането. Категорията на разхода се определя от системата на база търговеца получил плащането. *с. чрез ръчно въвеждане на разход - Потребителят може да попълни форма за информация за разхода и да го добави към избрания от него бюджет.*		
***8.** Обикновените потребители ще имат достъп до секцията Отчети, в която ще могат да виждат всичките си бюджети и наличните разходи за тях. Обикновен потребител може да премине към Премиум при заплащане на месечен абонамент.*	*Не*	*Информацията за вида потребител се пази в поле на класа User. Данните за отчетите се създават чрез модула и едноименния компонент ReportManager и се предават като обект от тип Report.*
***9.** Премиум потребителите ще имат достъп до допълнителна секция Анализи. В тази секция са налични анализи за разходите по даден бюджет за даден период:* *a. Премиум потребители могат да избират период за анализ.* *b. Премиум потребителите могат да филтрират по категория разход и/или начин на плащане. Наличните категории могат да са: храна, услуги, развлечение, разни и др.* *c. Премиум потребителите ще получават известия, ако има необичайна активност спрямо разходите за даден бюджет.* *Пример: разходите в категория храна са увеличени с 30% спрямо предходния месец.*	*Да*	*Генерирането на анализи се осъществява в модула AnalysisManager и едноименния интерфейс. Изборът на период за анализ и филтриране се осигуряват от графичния интерфейс и логиката за него в модулите Client и Web. Информацията за категория се съдържа като поле във всеки обект разход.* *Чрез Stored Procedures (и едноименния модул, посветен на тях в DBMS) е възможно периодичното генериране на анализи и известяването на потребители за необичайна активност.*
***10.** Възможни са опции за плащане с банкова карта, Revolut, Paypal и Skrill.*	*Не*	*Плащането с външни системи е поместено в модула PaymentManager, където за всяка опция - банкова карта, Revolut, Paypal, Skrill, има модул - Banking, Revolut, Paypal и Skrill.*
***11.** Данните в системата трябва да бъдат защитени от нерегламентиран достъп. Особено важно е комуникацията между системата и външните системи за разплащане да*	*Да*	*Сигурността на системата е имплементирана основно по 3 начина:* *контрол на потребителските сесии и достъпа до системата в модула AccountManager* *Допълнителни проверки при съмнения за идентичността се извършват от модула Security* *Модулът Encryption криптира данните преди напускането им на сървъра.*

предоставя защита на пренасяните данни		
***12.** Допуска се профилактика веднъж месечно в рамките на 6 часа. През останалото време, системата трябва да е 99,95% налична.*	*Да*	*Високата надеждност на системата се гарантира чрез тактиките Redundancy - модул BackupServer, копиращ функционалността (но не и имплементацията) на модула MainServer (който е предназначен да изпълнява бизнес логиката при нормални условия), и осигуряване на допълнителни изчислителни мощности чрез осигуряване на различно физическо устройство за всеки от тези модули, и допълнително - за модула PaymentManager. С цел улеснение и максимално бързо извършване на профилактиката, модулът ServerManager разполага с подмодули, които да помагат в логването на грешки и автоматичното им отстраняване (вградени модули за мониторинг).*
***13.** При извършване на плащане чрез външна система, известието за направеното плащане трябва да достига до потребителя в рамките на 30 сек.*	*Не*	*Осигуряването на собствен модул и физическо устройство за плащанията позволява операцията да става възможно най-бърза и надеждна. Спазването на изискването плащането да става за 30 секунди също зависи от имплементацията на модула PaymentManager.*
***14.** С цел актуалност на информацията, генерирането на агрегираните анализи трябва да става до 3 секунди.*	*Не*	*Скоростта на генериране на анализите се подобрява от наличието на собствен модул за това в MainServer, но зависи и от имплементацията на модула. Наличието на резервен модул BackupServer и изчислителни мощности за него позволява при невъзможност на MainServer да изпълни генерирането на анализа в срок, това да стане в BackupServer (Active Redundancy).*
***15.** Всеки потребител трябва да има възможност да изтрива данни от системата, асоциирани с него по всяко време спрямо GDPR.*	*Не*	*Възможността потребителя да изтрива данните за себе си е гарантира от графичния интерфейс в модула Client и класовете и интерфейсите в модула AccountManager*
***16.** Архитектурата трябва да позволява лесно добавяне на нови системи за плащания.*	*Да*	*Всяка система за плащане разполага със собствен подмодул в модула PaymentManager и всяка нова система за плащане също ще има такъв. Благодарение на факта, че PaymentManager е извън MainServer и BackupServer и е на собствено физическо устройство на практика премахва повечето физически ограничения върху броя и сложността на работа със системите за плащане.*