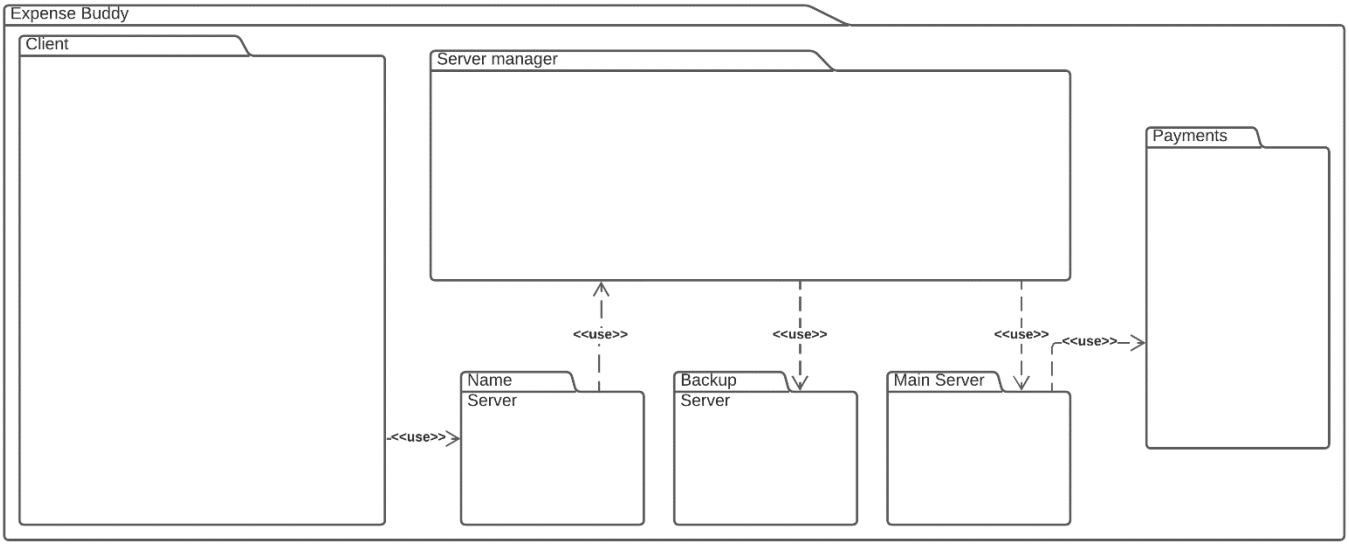
Софтуерни архитектури и разработка на софтуер
Домашна работа № 2
Вариант 1 – ExpenseBuddy

Стефан Велев, № 62537

Даниел Халачев, № 62547

Декомпозиция на модулите

а. Общ вид на декомпозицията на модули за системата

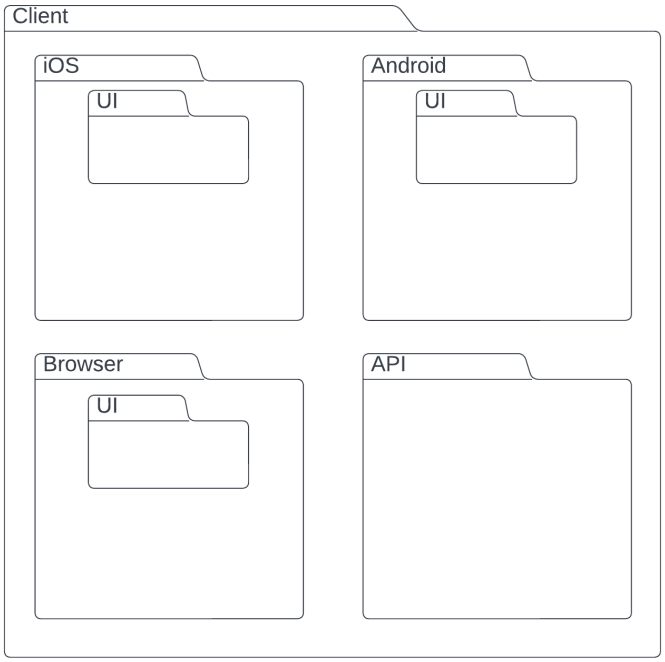


Поради изискването системата да бъде достъпвана от множество различни по вид устройства чрез интернет връзка, тя ще бъде изградена в архитектурния стил “Клиент-Сървър”. Затова системата ще бъде разделена на две основни части - клиентска част, която имплементира различните възможности за thin client (т.е, 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 Server Manager 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. Подробно описание на всеки модул

1. Client



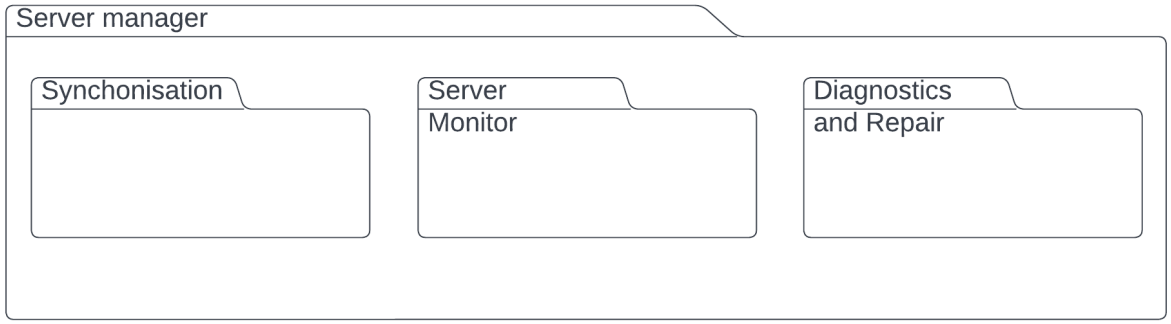
- Предназначение: имплементира функционалността в клиентската част на системата и позволява лесната промяна на кода за всеки вид клиент
- Основни отговорности:
    - Осигурява интуитивен потребителски интерфейс
    - Позволява достъп до системата от iOS, Android и Web browser
    - Позволява на потребителя да извършва всички налични за ролята му функционалности, сред които: регистрация, вписване, управление на разходи и бюджети, генериране и преглед на отчети и анализи
    - Единствено представя данните на потребителя, които сървърът е обработил

1.1. IOS
1.2. Android
1.3. Web
1.4. API
1.5. UI

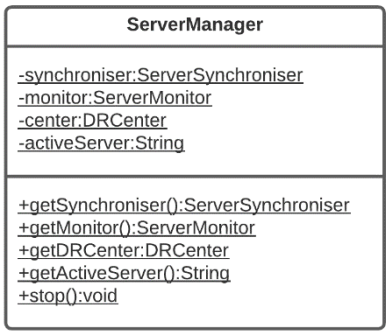
2. NameServer

- Предназначение: “посреща” клиентите, пренасочва трафика към активния в момента на заявката сървър
- Основни отговорности:
    - Получава информация от ServerManager кой е работещият сървър и пренасочва трафика към него
    - Създава допълнителна преграда пред достъпа до същинската функционалност на системата

3. ServerManager



- 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



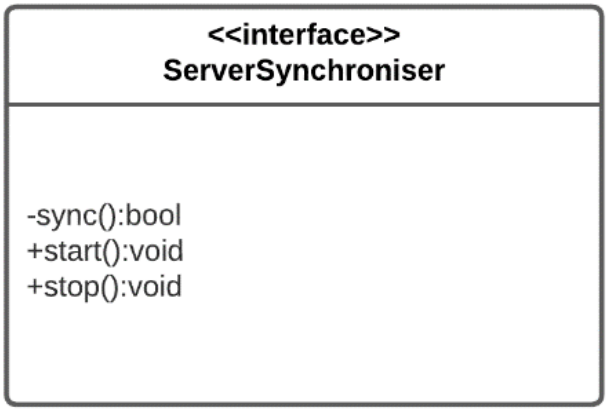

```
public class ServerManager {
    private static final ServerMonitor monitor = new DefaultServerMonitor();
    private static final DRCenter center = new DefaultDRCenter();
    private static final ServerSynchronizer synchronizer = new
DefaultServerSynchronizer();
    private static URL activeServer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

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

```
// 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();
```

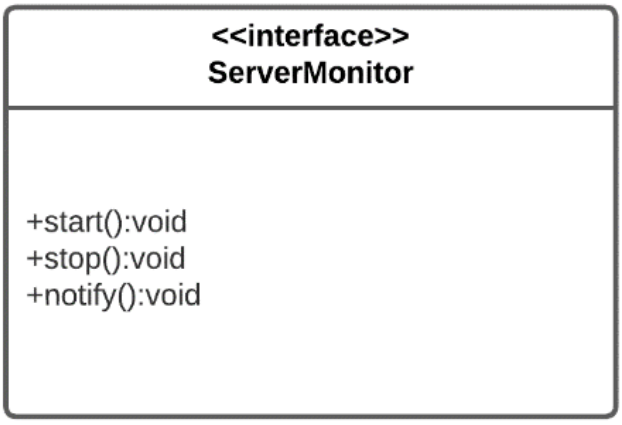
3.1. Synchronization

- 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



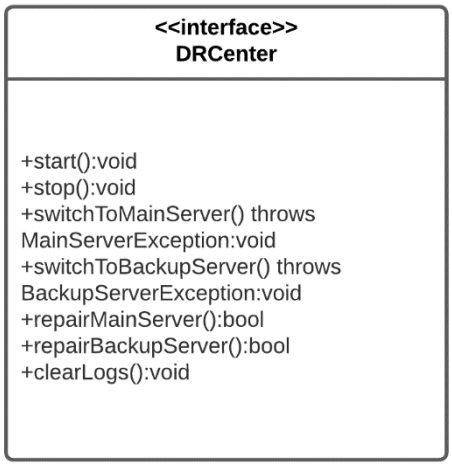
3.2. ServerMonitor

- 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

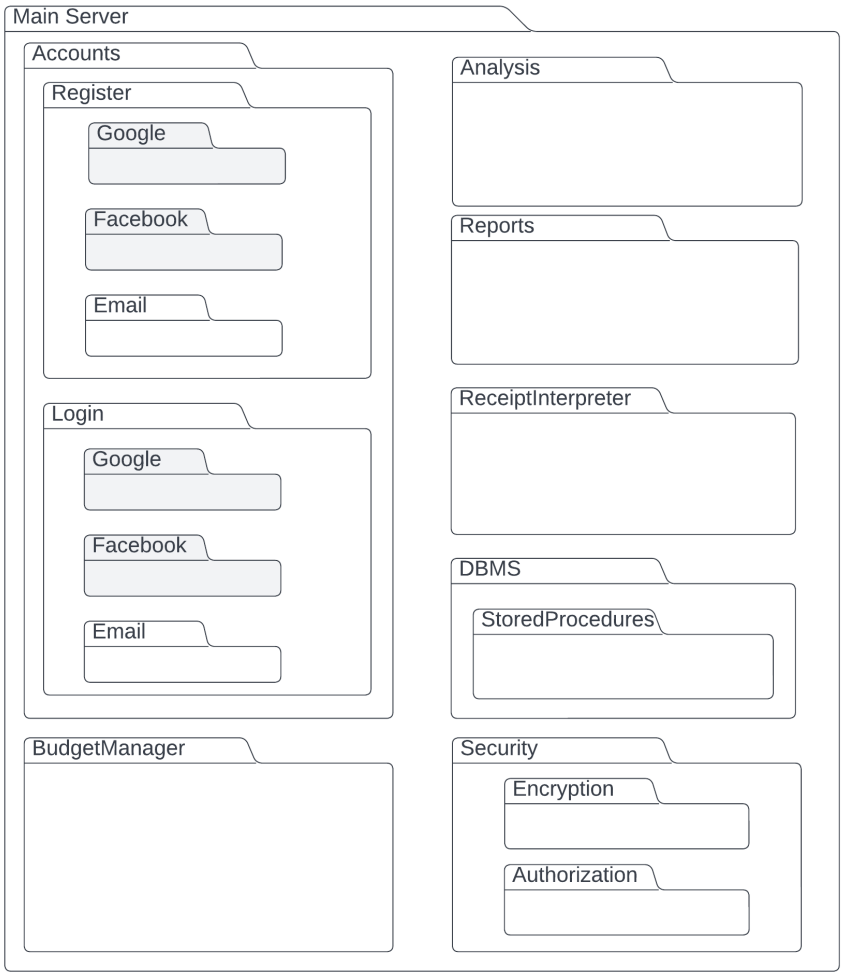


3.3. DiagnosticsAndRepair

- 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



4. MainServer



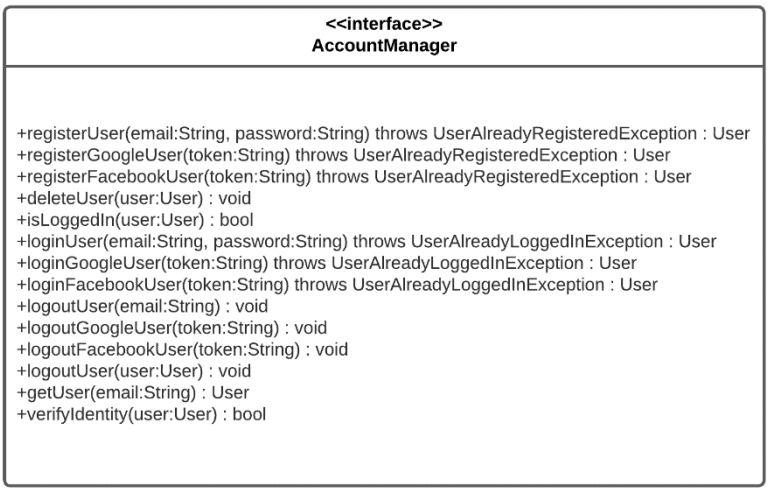
- Предназначение: имплементира Бизнес логиката на системата
- Основни отговорности:
    - Администрира акаунтите
    - Извършва отчитането и промяната на разходите и бюджетите на потребителите

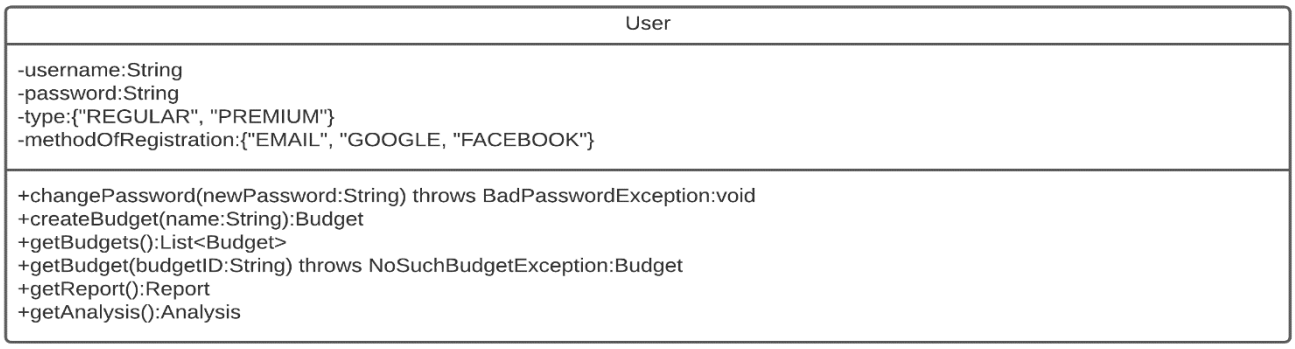
4.1. Accounts

- 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






```
// 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	Извършва регистрирането с Google акаунт	Създава акаунт в системата, основан на предоставения Google акаунт
4.1.1.2. Facebook	Извършва регистрирането с Facebook акаунт	Създава акаунт в системата, основан на предоставения Facebook акаунт
4.1.1.3. Email	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

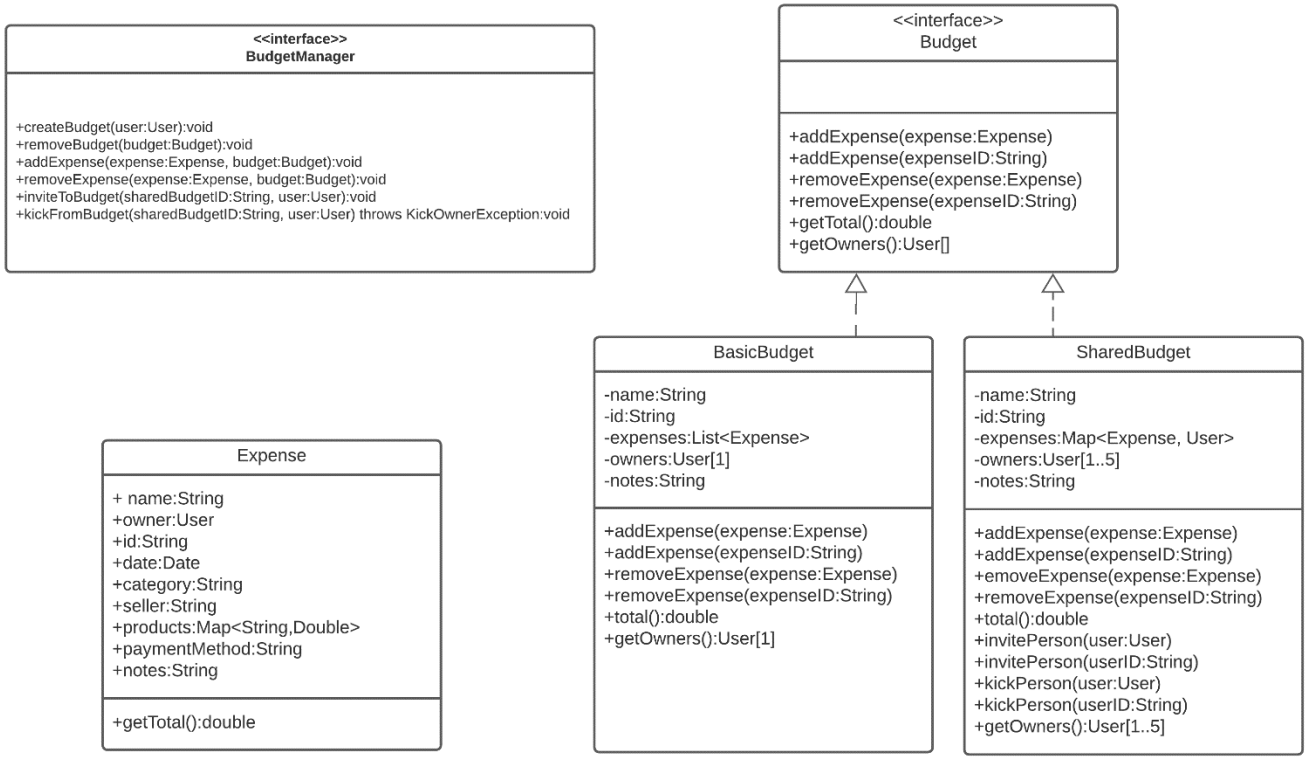
- Предназначение: Позволява вписването на вече съществуващ потребител чрез имейл, Google или Facebook акаунт
- Основни отговорности:
  - Вписва вече съществуващ потребител в системата при предоставени верни входни данни
  - Не позволява вписване при предоставени грешни входни данни
  - Не позволява вписване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

4.1.2.1. Google	Извършва вписването с Google акаунт	Влиза в системния акаунт чрез предоставения Google акаунт
4.1.2.2. Facebook	Извършва вписването с Facebook акаунт	Влиза в системния акаунт чрез предоставения Facebook акаунт
4.1.2.3. Email	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




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

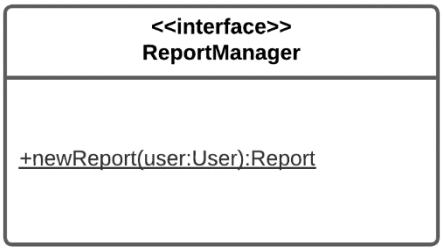
```
// добавяне на разход към бюджет MyBudget чрез касов бон
Expense ex = RECEIPT_INTERPRETER.interpret(providedImage);
ACCOUNT_MANAGER.getUser(providedEmail).getBudget("MyBudget").addExpense(ex);
// или
BUDGET_MANAGER.addExpense(ex,
ACCOUNT_MANAGER.getUser(providedEmail).getBudget("MyBudget"));
```

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

4.3. Reports

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

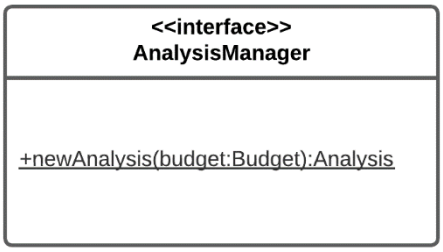
- Интерфейси: ReportManager




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

4.4. Analysis

- 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



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 Login
  - 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.

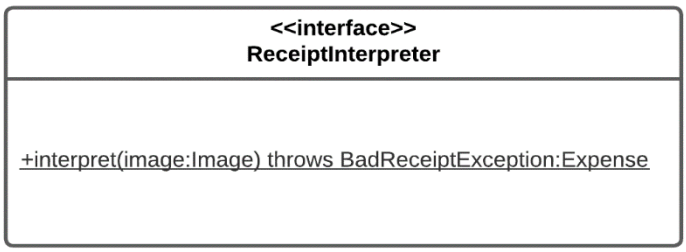
4.5.2. Authorization

- 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 Login
- 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



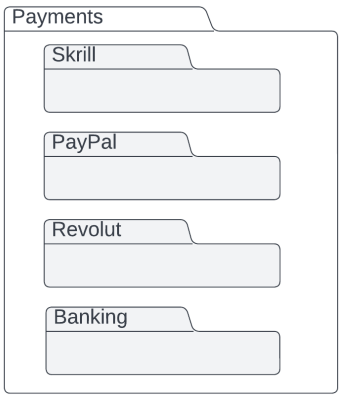
4.7. DBMS

- 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

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. Payments



- 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)

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]

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)

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

1.1. iOS	Имплементацията за iOS се различава от тази за Android, която се различава от тази за Web. 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	
1.3. Web	
1.{1,2,3}.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.
1.4. 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. NameServer	С цел повишаване на надеждността, системата ще използва тактиката Active 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. Т.е, NameServer е вариация на Фасада, която добавя още една преграда между сървъра и външната част
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.
3.3. DiagnosticsAndRepair	Програмистите и администраторите са хора, а не роботи. Но светът не спи за разлика от тях! :-D 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 Payments). Тези функционалности нямат общо с чисто програмистките действия по поддръжка на софтуера и хардуера (например ServerManager) и затова са в собствен модул.
4.1. Accounts	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. Google	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. Facebook	
4.1.1.3. Email	
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. Google	Аналогично за огледалните модули в Login, и тук има съществени разлики в имплементацията на трите случая.
4.1.2.2. Facebook	
4.1.2.3. Email	
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-детайлни, защото функционалността (опростено казано) представлява

	добавяне на елементи в списък и заявяване на отчети и анализи, които ще се изчисляват в собствени модули (т.е, BudgetManager ще използва наготово отчети и анализи от модулите Reports и Analysis).
4.3. Reports	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. В допълнение, отделянето на тази функционалност от останалите позволява резултатът да бъде използван наготово (от BudgetManager) без да се нарушава енкапсулацията.
4.4. Analysis	В секцията за Анализи могат да бъдат представени различни статистики за разходите по даден бюджет за даден период. Модулът отговаря за редица от функционалности, сред които 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.
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. Payments	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.

c. Описание на възможните вариации (ако е необходимо)

Модулът BackupServer е приложение на тактиката Active 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 - Accounts, Reports, 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.