# Курсов проект по Софтуерни архитектури и разработка на софтуер на тема: “ExpenseBuddy”

Изготвен от Илиян Мариянов Йорданов ФН: 62546, втора група
и Юсмен Забанов ФН: 62556, втора група

# 1. Въведение

## a) Обща информация за текущия документ

### i. Предназначение на документа

В документа е представена архитектурата на софтуерната система „ExpenseBuddy“, която е предназначена за управнение на разходи. Главната идея е да запознае заинтересованите лица, към проекта, с това как е разработена архитектурата на системата, как са реализирани нейните функционалности и логиката зад самата нея.

### ii. Описание на използваните структури на архитектурата.

- **Декомпозиция на модулите**

  Декомпозицията на модулите представя на читателя поглед над логическото разпределение на модулите и подмодулите. В допълнение на гореспоменатото, в декомпозицията на модулите са показани и връзките между отделните модули и подмодули, това как те си комуникират като без една такава структура читателят няма как да бъде запознат с това как дадената система работи и как самата тя е проектирана. Основните модули в системата са четири на брой като те са UI, Server, Monitoring и Database controller. UI е модула, който отговаря за визуалната част на потребителя, Server извършва цялата логика и функционалност на системата, Database controller служи за междинна връзка между сървъра и базата данни и Monitoring за следене работоспособността на системата.

- **Структура на разположението**

  В тази структура се показва как ще се разпределят модулите на върху различните хардуери или софтуерни системи и нужните компоненти за работоспособността на системата. Тази структура показва и един различен поглед над как е структрурирана системата, как е решено тя да бъде изградена, като се вижда и в по-детайлен вид някои допълнителни неща (решения) и тактики за справяне с проблеми. Мотивацията за избора на тази структура е описана в точката която е самата тя.

- **Структура на процесите**

  Като не на последно място ще покажем и още един различен поглед над това как системата е функционира, а именно с помощта на структури на процесите. Те погазват в по-детайлен вид как и какво се случва като се стартира даден процес в системата, като е показано и по-подробно как някои модули си комуникират.
  Мотивацията за избора на тази структура е описана в точката която е самата тя.

## iii. Структура на документа

# Съдържание



## b) Общи сведения за системата

Софтуерната система „ExpenseBuddy“ е система предназначена за управление на разходи. Потребителя ще може да си създава бюджети, като самите разходи са на база тези бюджети. Потребителя ще може да избира между два типа бюджети – индивидуален и споделен, в който ще могат да участват от 2 до 5 души, като се добавят с покана по имейл. Той ще може да достъпва тези отчети през секция отчети. Системата ще поддържа два типа потребители – обикновени и привилегировани, като единствената разлика ще бъде, че привилегированите ще могат да имат достъп до секция анализи. Тази секция ще предоставя възможността потребителя да следи разходите си за период, по категория и ще бъде известяван при разлики в месечните разлики. За да стане един потребител привилегирован, той ще трябва да заплати месечна такса, след което автоматично ще стане такъв. Системата ще може да поддържа външни системи за плащане, като лесно и безпроблемно ще могат да бъдат добавяни и още такива. Начините на добавяне на разходи ще бъдат по три начина:

- Чрез снимка на касов бон като системата автоматично ще попълва данните от бележката в системата.
- Чрез връзка с онлайн система за плащане като потребителя ще трябва да свърже бюджета си със съответната система
- Чрез ръчно добавяне на разход.

Системата ще може да бъде достъпвана чрез браузър, приложение на телефон като поддържани операционни системи ще са iOS и Android. Потребителя ще може да се вписва по стандартен начин с въвеждане на никнейм и парола, и чрез външни системи като Google, Facebook и други. Особен фокус ще се даде над сигурността на системата, като тя ще бъде защитена от достъп от външни нерегламентирани лица.

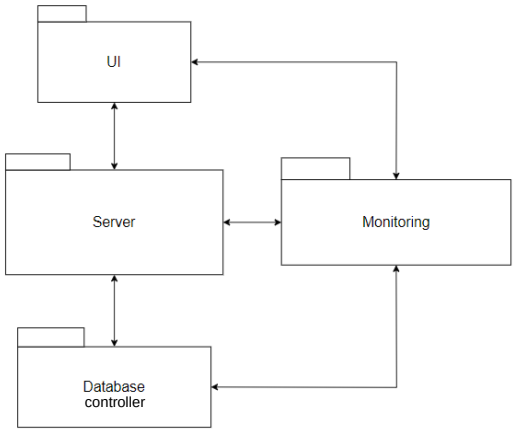
## c) Терминологичен речник

1. **UI** – Графичен изглед на потребителя, през който той използва системата.
2. **API** – (Приложен програмен интерфейс) Интерфейс за програмиране на приложения.
3. **Системен администратор** – Човек, отговарящ за работоспособността на системата, който оправя проблемите на софтуерно и хардуерно ниво.
4. **Authorzation** – Упълномощаване на потребителя, определяне на неговите права в системата.
5. **Authentication** – Удостоверяване на самоличност в системата на потребителя.
6. **Interface** – Споделена граница между два разделени компютърни компонента, обменящи информация.
7. **Monitoring** – Следене на параметри в системата.
8. **Servlet** – Java програма, в която ще се разполага изпълнимата част на сървъра.
9. **HTTPS** – Протокол за защитена комуникация в компютърната мрежа в интернет.
10. **TCP** – Мрежов протокол за управление на обмена на информация в интернет.
11. **TCP/IP** – Семейство от интернет протоколи за комуникация между компютрите.
12. **Component** – Устройство, на което ще се извършват изчисления и обработка на данни.
13. **Data accessor** – Устройство, което ще получава данни и ще извършва нещо с тях.
14. **Communicator** – Устройство, което ще комуникира с дадено друго устройство.
15. **MSSQL Database** – База данни, в която ще се използва технология на microsoft сървър.
16. **Web/Mobile interface** – Визуалната част на системата, която ще бъде уеб базирана и на мобилно устройство.
17. **Encryption** – Криптиране на данните с цел защитата им от външна намеса.
18. **Decryption** – Декриптиране на данните, за да могат да бъдат удобно ползвани.
19. **Execution environment** – Изпълнима среда, на която ще е Servlet.
20. **OS** – Операционна система (iOS, Android).
21. **Connection** – Връзка, по която ще си комуникират устрийствата
22. **Communication** – При установена вече връзка да се извършва комуникация.
23. **PayPal/Revolut/Skrill** - Външни системи за разплащане и за извличане на разходи.
24. **Permissions** – Метаданни за какви позволения има потребителя в системата.
25. **Data controller** – Устройство, което ще служи за разпределяне и извличане на данни от базата данни
26. **Synchroniser** – Устройство, което служи за синхронизация на сървъри.
27. **GDPR** – Регламент (правила за обработка на личните данни).

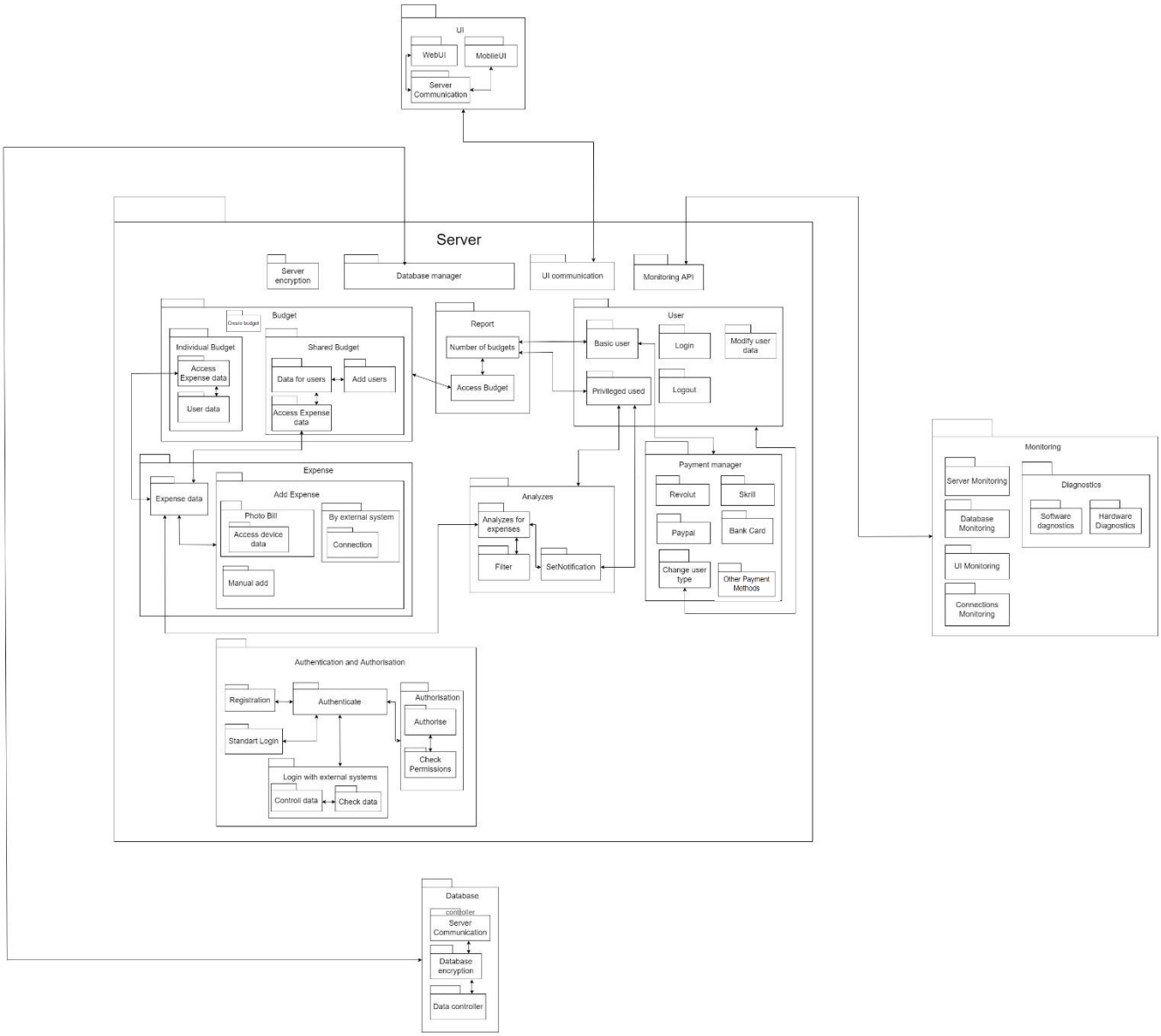
## 2. Декомпозиция на модулите

### а) Общ вид на декомпозицията на модули за системата

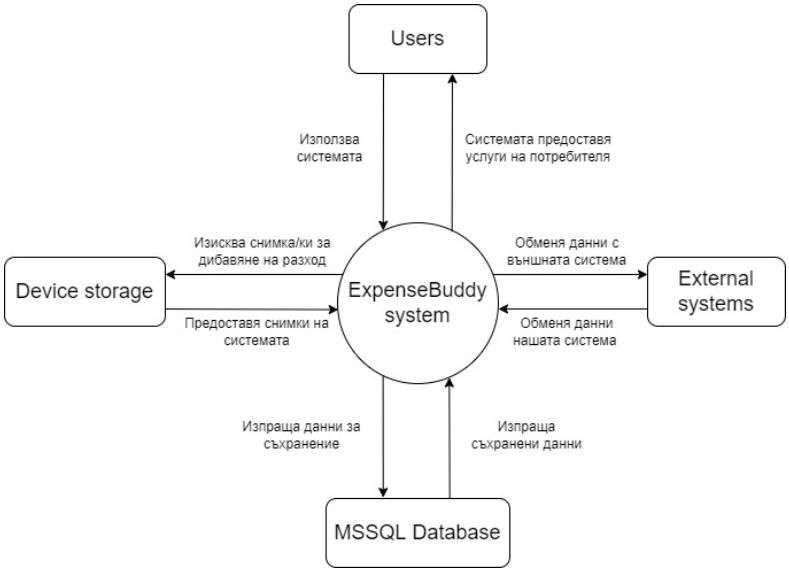
#### Абстрактен поглед над системата




#### Цялостен поглед над системата

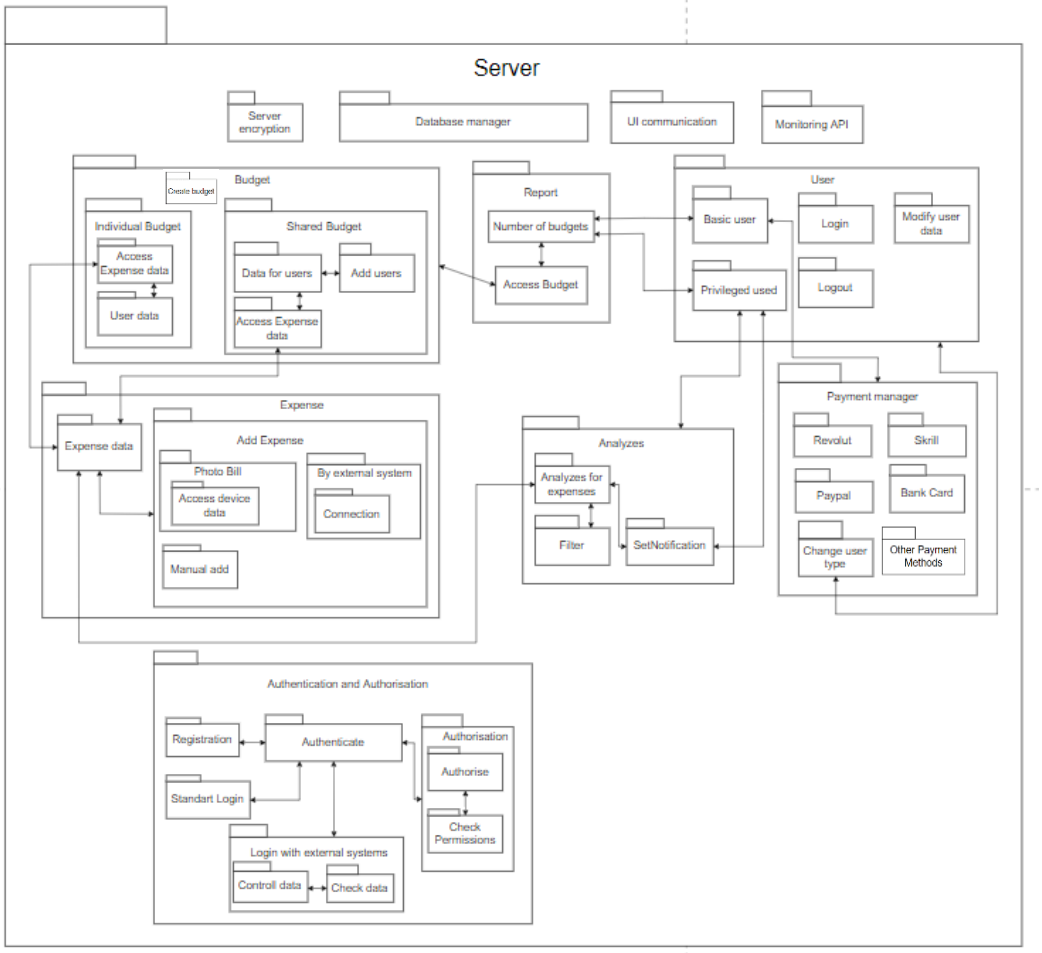


## b) Контекстна диаграма




## c) Подробно описание на всеки модул

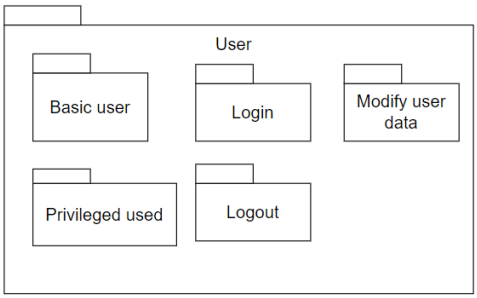
### 1. Server



1.1. Предназначение и основни отговорности

В този модул ще се извършва цялата логика и функционалност на системата ни. Тук се извършват аутентикацията и ауторизацията и по този начин се определят правата на потребителя. Основната отговорност е да обработва правилно заявките на потрябителя/лите, като това трябва да се случва безпроблемно не само вътре в него, а и в момента когато комуникира (приемане и предаване на информация) с другите модули.

1.2. Описание на подмодулите:




1.2.1. **User (Account)** – модул в който се дефинират различните типове потребители (Basic and Privileged user), с което се стига до различните функционалности според всеки тип. В този модул ще се съдържат различните модули съответно за :

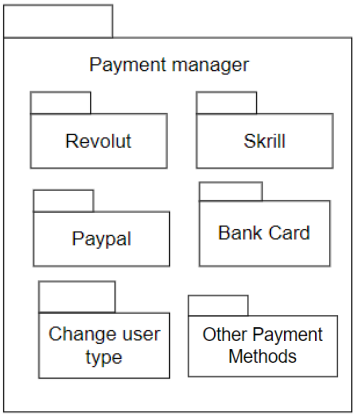
1.2.1.1. **Basic user** – Базовият потребител който има ограничен достъп до дадени секции от системата. Той е свързан с модулът Payment manager, поради факта че той ако пожелае да стане привилегирован потребител, трябва да му се подсигури начин за плащане, за да стане такъв, а и със модулът Report, където той ще може да достъпи секцията, където се визуализират неговите брой бюджети с метода getNumberOfBudgets().

1.2.1.2. **Privileged user** – Следващият тип потребител, за който всичките функционалности на системата са налични. Той е свързан както в гореописаната точка в модул Report и има същия метод за извикване на reports, също е свързан и с модул Analyzes където той ще може да получава анализи за свойте разходи. Това ще се извършва с метод под името getAnalyzesForExpenses().

1.2.1.3. **Login** – Модул, които взима данните които потребителя подава в login полето и веднага ги подава на модула Authentication and Authorization, за да се проверят дали са верни данните и да върне през Standart Login, дали данните са верни и да бъде вписан в системата. Като подмодули ще има и за вписване с външна система, система която не е за разплащане или за добавяне на разходи, която ще осъществява връзката с външните системи и ще има подмодул, който пази метаданни за тях.

1.2.1.4. **Logout** – Модул, които отговаря за отписването на даден потребител от системата, като прекратява всички процеси в системата за него.

1.2.1.5. **Modify user data** – Модул които при извикване от потребителя на метода modifyUserData() потребителя ще може да избере кои данни може да премахне от системата, кои да не са видими за други потребители от изгледите, кои данни може да промени, като за пример нов имейл акаунт, да смени паролата си и различни.




1.2.2. **Payment Manager** е модул които се достъпва от обикновения потребител и в него съдържа модули на различни определени според спецификация начини на плащане с които той може да стане привилегирован потребител. След като опикновения потребител заплати зададената сума, за да стане потребител от другия тип, според наличните начини чрез : Revolut, Skrill, Paypal, Bank card и други. След като е верифицирана транзакцията от самите системи – на банката, на Revolut, на Skrill, на Paypal или на друга система, се връщат данните за успешна транзакция към модула Change user type, където той сменя типа на акаунт за дадения потребител като парите от платения месечен абонамент постъпват в сметка, открита от екипа зад системата ExpenseBuddy. Този модул съдържа следните модули:

1.2.2.1. **Revolut** – Които отговаря за връзката със системата на Revolut с даден подмодул за нея, задено с това съдържа данни за потребителската сметка в отделен (самата тя, но не и чувствителни данни като пароли и пин кодове на карти).

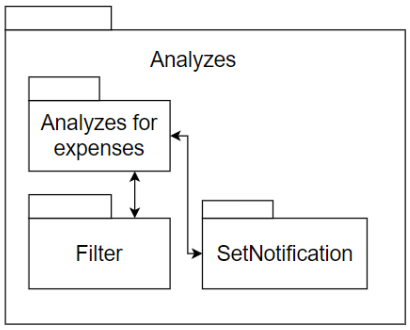
1.2.2.2. **Skrill** – Отговаря за връзката със системата на Skrill, и има модули като гореспоменатите със същите данни.

1.2.2.3. **Paypal** - Отговаря за връзката със системата на Paypal, и има модули като на точката за Revolut със същите данни.

1.2.2.4. **Bank card** - Отговаря за връзката със системата на съответната банка която потребителя е въвел че желае да бъде вписана в системата, и има модули като на точката за Revolut със същите данни.

1.2.2.5. **Change user type** – модул в които полученото от външните системи като данни, от гореспоменатите, ако транзакцията е успешна, той променя типа на потребителя, ако не, тогава подава съобщение за неуспешна транзакция (ако от дадената система също се появи грешка, тази грешка, като данни какво представлява и от какво се е получила също се изписва на потребителя).

1.2.2.6. **Other Payment Methods** – модул който позволява за се добавят други методи за плащане от други системи.

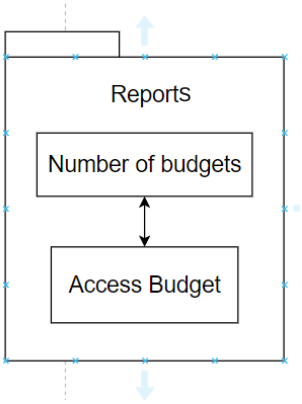



1.2.3. **Analyzes** е модул които може да бъде достъпван само от привилегирован потребител като той има ролята да изкара анализи за различни разходи по бюджет за даден период. В него ще има модули които ще отговарят за специфични дейности за това как могат да бъдат изпълнени дадени функционалности. Този модул съдържа следните в себе си:

1.2.3.1. **Analyzes for expenses** – Модула които отговаря да изкара анализите за разходите по даден бюджет за даден потребител. Той е свързан с модула Expense data в по-големият Expense, от където взима информация за бюджета (понеже са свързани) и за типа бюджет. После на база на броя (независимо от типа), за всеки един се генерира анализ за даден период (стандартно ще бъде зададено 1 месец), ако даденият потребител не зададе даден друг период (което се случва с помощтта на метода filterForPeriod(const std::string& startDate, const std::string& endDate)), и се генерират в въщата секция бутони от които потребителя ще може да натисне и да види анализа за дадения бюджет. Метода filterForPeriod - в зависимост от определения период за даден анализ от потребителя ще задава ограничения от кога до кога ще бъде генериран дадения анализ.

1.2.3.2. **Filter** – ще е свързан с Analyzes for expenses и ще може да показва по различни секции (тези секции от различни типове ще бъдат определяни според типовете разходи, които потребителя е въвел в системата) типовете разходи, където ще има анализи за различни типове разходи от датата на първия разход, въведен от дадения тип, до датата на последния.

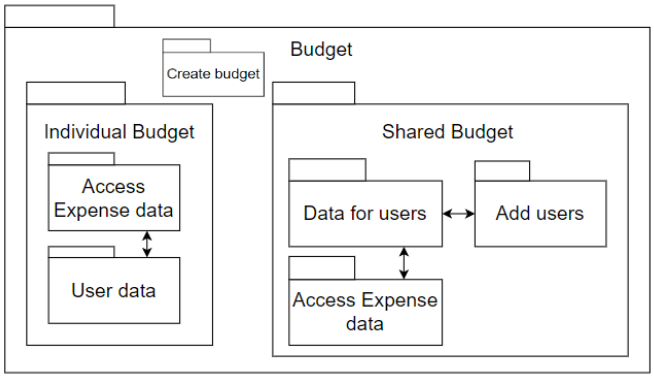
1.2.3.3. **SetNotification** – е модул които отговаря за изпращане на известие на потребител за необичайно повишение на разходите за дадения месец спрямо предходните и за дадена категория като например повишаване на разходите за развлечение с 50%. Всичкото това действие като измерване и изчисление на разходите ще се грижи този модул, защото е пряко свързан със Analyzes for expenses, а той от своя страна с Filter, и достъпва данните за разходи оттам.




1.2.4. **Report** е специален модул които съдържа данните за броя на бюджетите и може да ги достъпи. Този модул съдържа в себе си следните модули:

1.2.4.1. **Number of budgets** – Модул които отговаря за съхранението на броя на бюджетите, като за всеки бюджет ще си има отделен бутон, от които потребителя ще може да достъпи съответния си бюджет, защото този модул е свързан с Access Budget модула.

1.2.4.2. **Access Budget** – е модул от които осъществява връзката вече към модула Budget, от където вече потребителя има досъп до данните за дадения бюджет които е избрал.




1.2.5. **Budget** – това е един от основните модули в системата. Той предоставя възможността при регистрация на потребителя в системата да се създаде бюджет, който е съвкупност от разходи и този модул достъпва данните на модула Expense data**.** Също така този модул се достъпва и от модула Report (Access Budget), който дава информация на потребителя за бюджетите**.** Тук се дефинират двата типа бюджет, които са в два отделни модула (Individual и Shared budget):

1.2.5.1. **Create Budget** – Този модул служи за създаване на бюджет и има две опции (Individual и Shared budget).

1.2.5.2. **Individual Budget** - Този модул представлява индивидуалният бюджет, който може да се достъпва от потребителя, който е създал бюджета. Двата подмодула на Individual budget и Access Expense data са свързани помежду си, тъй като потребителят може да добавя разходи в бюджета. Съдържа два основни модула:

1.2.5.2.1. **Access Expense Data –** този модул служи за комуникация с модула Expense.

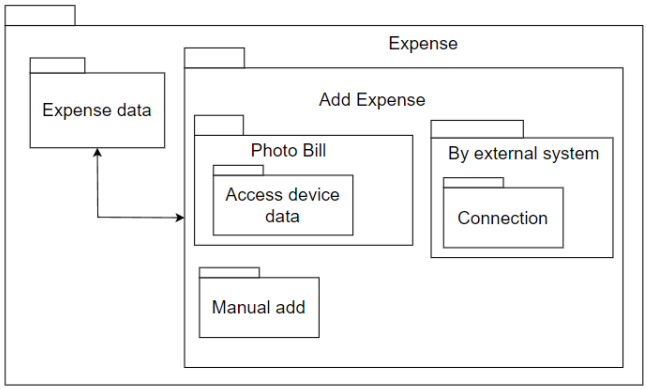
1.2.5.2.2. **User data -** съдържа данните за потребителя, който се е регистрирал и създал бюджета.

1.2.5.3. **Shared Budget** – Този модул представя споделения бюджет, който също достъпва данните за потребител и разход и има три основни подмодула. Трите подмодула са свързани помежду си, защото, за да се добавят нови потребители към споделения бюджет трябва да се достъпва информацията за тях и поради тази причина и модулът Expense също се достъпва. Трите модула са:

1.2.5.3.1. **Add Users –** този модул съдържа метод Add(const User& user), който служи за добавяне други потребители към споделения бюджет и съдържа проверка за ограничение до 4 на брой потребители. Методът SendEmail(const User& user) използвайки достъпа до модула User изпраща имейл покана за присъединяване към споделен бюджет.

1.2.5.3.2. **Access Expense data –** служи за извличане на данните от модула Expense, като разход може да бъде добавян от всички потребители в споделения бюджет.

1.2.5.3.3. **Data for users –** Съдържа данните за до четирите потребителя, които могат да бъдат в споделения бюджет




1.2.6. **Expense –** този модул съдържа всички функционалности и подмодули, които са свързани с разхода. Разходът съществува и при двата типа бюджет (Individual и Shared budget) т.е комуникира с модула Budget и може да се добавя към бюджет по различни начини. Формират се следните подмодули:

1.2.6.1. **Expense data** – това е основният модул, който образува разхода. Съдържа данните за обекта разход (списък на закупените продукти, търговец, дата на разхода, категория, начин плащане, бележки (опционално)), които се получават при регистриране на нов разход от потребителя чрез долупосочените начини.

1.2.6.2. **Add Expense –** това е модула за добавяне на нов разход, свързан е с модула Expenese data, който при добавяне на нов разход се попълва и тъй като има три начина за добавяне се образуват следните подмодули:

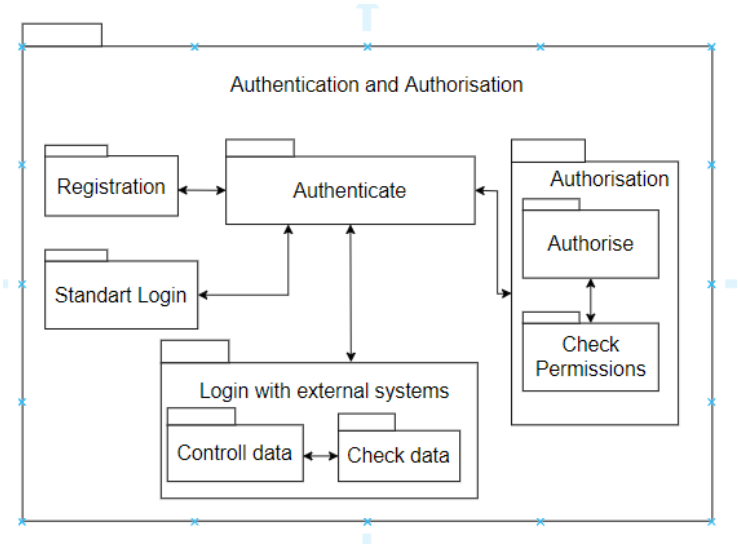
1.2.6.2.1. **By Photo Bill** – Това е модул за добавяне на разход чрез снимка на касов бон и след като за взети данните от устройството ги записва в модула за данните на разхода (Expense data).

1.2.6.2.2. **By External System** – модул, който попълва данните за разхода чрез външна система за разплащания (PayPal, Revolut, Skrill, банкови системи),

съдържа метод, който изпраща на потребителя имейл за потвърждаване на плащане и добавяне като разход:

1.2.6.2.2.1. **Connection –** Този модул свързва бюджета на системата с външните системи за плащане.

1.2.6.2.3. **Manual Add** – Този модул изгражда форма, чрез която потребителят ръчно да добави нов разход към избран от него бюджет и данните се записват в модула Expense data




1.2.7. **Authentication and Authorization** – В този модул се извършва аутентикация на потребителя и определяне на неговите права:

1.2.7.1. **Authenticate** – този модул верифицира потребителят, който се регистрира или влиза и първо се определя, дали може да достъпи системата.

1.2.7.2. **Authorization** – В този модул се определят правата на потребителя в системата след като вече е извършена аутентикация:

1.2.7.2.1. **Check permissions –** този модул определя правата на потребителя

1.2.7.2.2. **Authorize –** предоставя достъп до различните функционалности от системата в зависимост от притежаваните права.

1.2.7.3. **Registration –** подава на потребителя форма за регистрация и след като премине през модулите за ауторизация и аутентикация потребителят успешно се регистрира в системата.

1.2.7.4. **Standard Login –** взима данните от потребителската форма за вход и отново се извършва ауторизация и аутентикация.

1.2.7.5. **Login with external system** – този модул позволява вход в системата чрез външни системи като Facebook, Twitter, LinkedIn, Microsoft account и др. и тук също се извършва ауторизация и аутентикация:

1.2.7.5.1. **Check data** – този модул проверява дали данните от външната система са валидни.

1.2.7.5.2. **Control data** – този модул извлича само данните които са необходими на потребителя за системата.

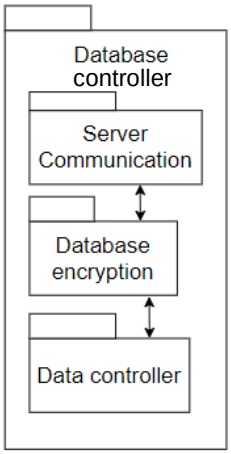
1.2.8. **Server Encryption** – криптира и декриптира всички данни които преминават през сървъра, за връзките с базата данни, UI и мониторинга. Също така криптира всички връзки които са към външните ситеми и сървъра като се използват специални криптиращи функции.

1.2.9. **Database Manager –** Този модул предоставя връзката с базата данни, в която могат да си добавят или да се изтриват вече съществуващи данни за регистрирания потребител.

1.2.10. **UI Communication** – служи за предаване към и приемане на данните от модула UI.

1.2.11. **Monitoring API** – е модул, който служи за извършване на връзка с модула Monitoring, който позволява да бъде извършено следене на модула Server за правилна работоспособност.

## 2. Database controller




2.1. Предназначение и основни отговорности

Модулът Database controller ще отговаря за най-важното нещо без което не може да работи системата, а това са необходимите данни за потребителите, номера на сметки, бюджети, разходи, анализи и дриги данни да бъдат защитени и разпределени правилно. Този модул ще бъде свързан към сървъра и ще бъде междинен модул между него и базата данни.

2.2. Основни модули:

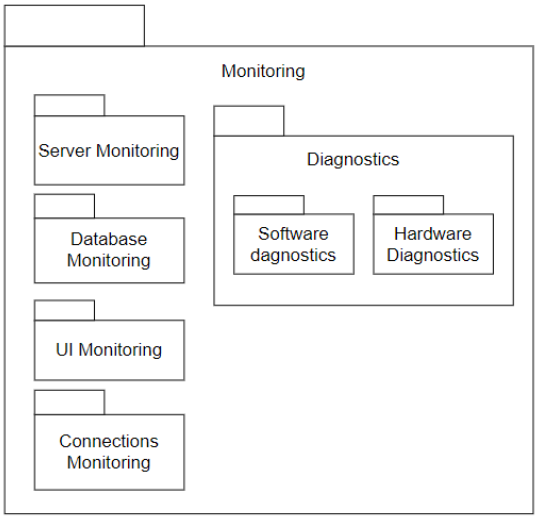
2.2.1. **Server Communication** – през този модул ще се осъществява връзката между сървъра и базата данни. Той ще бъде свързан с модула Database manager, от където ще се осъществява комунникацията между тези два големи модула, заедно с това когато вземе данните от сървъра които ще му бъдат подадени, той ще ги предаде на модула Database encryption.

2.2.2. **Database encryption** – след като получи данните от Server Communication, той ще може да ги криптира и декриптира, с цел сигурното съхранение на данните в нашата база данни. След като са получени данните от комуникационния модул, той ще криприта данните и ще ги предаде на модула Data controller . Когато трябва да бъдат

изкарани дадени данни за потребител или неща свързани с него данните необходими от базата данни ще бъдат извлечени, декриптирани и ще ги предаде на Server Communication, от където ще бъдат изпратени до сървъра.

2.2.3. **Data controller** - е модула които отговаря за предаване на данните (които са вече криптирани) на базата данни, като създаде запис с уникален номер, за да могат криптираните данни да бъдат разпознати после при извличането им. Когато ги извлече от базата данни, данни които са нужни за даден процес, след като е получил заявка че са нужни от database manager, той ги предава на Database encryption модула, които ги декриптира данните.

## 3. Monitoring




3.1. Предназначение и основни отговорности

Модулът Monitoring е също един от най-важните модули за нашата система, като неговото предназначение ще е за следене за работоспособността на системата и комуникацията между отделните модули.

3.2. Основни модули:

3.2.1. **Server Monitoring** – Този модул ще служи за следенето на модула Server и главно ще следи работоспособността му, комуникацията на модулите вътре и ако нещо се повреди го препраща съответната грешка (съобщение) към модула Diagnostics.

3.2.2. **Database Monitoring** – Този модул ще следи за работоспособността на database controller и базата данни, комуникацията на модулите вътре и ако нещо се повреди праща съобщение за грешка на модула Diagnostics.

3.2.3. **UI Monitoring** – Модулът ще служи за следене на работоспособността на модула UI и вътрешно в него помежду модулите му и ако нещо се срине като комуникация или се появят сривове или бъгове се изпраща съобщение за грешка към Diagnostics.

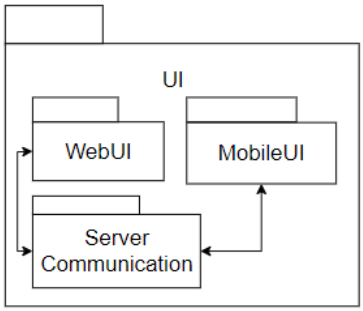
3.2.4. **Connections Monitoring** – Този модул ще следи за връзките между големите модули и ако се разпадне връзката веднага изпраща съобщение за грешка на модула Diagnostics.

3.2.5. **Diagnostics** – Модулът, който отговаря за известяването на проблемите, възникнали по време на работа на системата в зависимост дали са хардуерни или софтуерни неизправности.

3.2.5.1. **Software Diagnostics** – Този модул служи за получаване на оценка на системата за начина на работа и поведение на софтуерно ниво. Ако се появи съобщение за грешка от един от горепосочените модули, то то ще се появи тук. Всичките важни решения за това даден проблем на софтуерно ниво, как ще се справи, ще се взимат тук. При проблем настъпил в някой процес, решенията за това как ще може системата автоматично да се справи с него, ще се взимат тук. Също този модул ще може да известява със съобщение как може да бъде подобрена (оптимизирана) дадена част в софтуера при наличие на забавяния.

3.2.5.2. **Hardware Diagnostics** – Този модул служи за получаване на оценка на системата за начина на работа и поведение на хардъерно ниво. Ако се появи някъде неизправност в цялата система (независимо дали ще е в Server, Database, UI) на хардуерен компонент, съобщението за грешка ще се появи тук. Също този модул ще известява ако има някой хардуерен компонент, който достига лимита си или е започнал да се забавя (пример твърдия диск е на 80% от капацитета си или е започнал да се забавя).

## 4. UI




4.1. Предназначение и основни отговорности

Този модул представлява уеб приложение, позволяващо комуникацията на потребителя със системата и нейните функционалности и потребителят може да си създаде акаунт чрез регистрация, попълвайки форма. Този модул съдържа всички горе посочени данни и методи оформени в полета, форми за попълване, бутони, уеб страници и други.

4.1.1 **WebUI** – този модул позволява уеб приложението да работи на Windows, Linux и MacOS операционни системи, като използва технологии като JavaScript, HTML, CSS и др. и комуникира с модулите на системата.

4.1.2 **MobileUI** – позволява приложението да работи на всякакъв вид мобилни устройства под операционна система iOS и Android.

## 5. Интерфейси

Структурата, която ще използваме е класът Expense за съхранение на данните:

```
class Expense {

  private:
        std::vector<std::string> listOfProducts;
        std::string merchantName;
        std::string expenseDate;
        std::string category;
        std::string paymentMethod;
        std::string sharedBudgetUserPaymentName;
        std::vector<std::string> userNotes;
  public:
        Expense();
        Expense( Всички параметри с const& отпред);

        void addListOfProducts(const std::vector<std::string>& listOfProducts);
        std::vector<std::string> getListOfProducts() const;

        void setMerchantName (const std::string& merchantName);
        std::string getMerchantName() const;

        void setExpenseDate(const std::string& expenseDate);
        std::string getExpenseDate() const;

        void setCategory(const std::string& category);
        std::string getCategory() const;

        void setPaymentMethod(const std::string& paymentMethod);
        std::string getPaymentMethod() const;
        voidsetSharedBudgetUserPaymentName(const std::string&
      sharedBudgetUserPaymentName);
        std::string getSharedBudgetUserPaymentName() const;

        void addUserNotes(const std::vector<std::string>& userNotes);
        std::vector<std::string> getUserNotes() const;

};
```

За да е защитена самата структура ние правим данните private, за да не могат директно да бъдат достъпвани от потребителя и променяни. Единствения начин за промяната им и визуализацията им ще става през съответните setters и getters (за добавяне и визуализиране на данните), като така се спазват и принципите за енкапсулация според ООП. Ако от долупосочените интерфейси се вика, когато потребител е в shared budget, само тогава ще може да се задава със сетър void setSharedBudgetUserPaymentName името на потребителя направил плащането при добавяне на разхода.

### 5.1. Интерфейс на модула By Photo Bill:

std::vector<std::vector<int>> Photo [size] – масив в който ще се съхраняват матрици, в които ще се съхраняват снимките добавени от потребителя.

std::vector<Expense> expenses – масив, в който ще се съхраняват разходите добавени от потребителя.

**const Photo& AccessDeviceData() const** – това е метод който достъпва файловата система на устройството и взима необходимите снимки.

Входни данни: В случая няма да има защото устроиството, с което потребителя влиза в системата ще е регистрирано и така той ще може да влезе в галерията и да избере снимкакоято да качи, естествено като даде позволение приложеието да ползва галерия на телефон или утроиството от което ползва през браузър.

Изходни данни: Ще бъде получена снимката която потребителя е качил в системата. Ако потребителя откаже на системата да достъпи галерията на устройството, се прекратява операцията и нищо не излиза.

Грешки и изключения: Ако настъпи грешка при добавянето на снимката се хвърля грешка с подходящо съобщение.

**Expense& GetDataFromPhoto(const Photo& photo)** – този метод използвайки получените снимки попълва данните в класа Expense data и определя категорията на база търговеца.

Входни данни: Ще се взимат за входни данни снимката която е подадена от потребителя.

Изходни данни: Ще се получава инстанция на класа Expense с попълнени данни съответно по описаните точки.

Грешки и изключения: Ако настъпи грешка при извличането на данните от снимката се хвърля грешка с подходящо съобщение.

**void AddToBudget(const Expense& expense)** – този метод позволява новодобавените данни са разхода да бъдат добавени към избрания от потребителя бюджет.

Входни данни: Ще се взимат за входни данни разхода, който потребителя е решил да добави.
Изходни данни: Няма да има, вместо това ще се добавя разход в системата.
Грешки и изключения: Ако е настъпила грешка при добавяне на разхода се хвърля грешка с подходящо съобщение.

### 5.2. Интерфейс на модула Connections

std::vector<std::System> systems – масив, в който ще се съхраняват системите, които потребителя е въвел.
int Connection – ще връща число което е за идентификатор за връзка със системата;
std::vector<std::pair<Connection, System>> connections – масив от двойки връзки със съответната система.

**int ConnectToExternalSystem(const System& system)** – ще се свързва с външната система която е избрал потребителя.

Входни данни: Ще получава системата която потребителя иска да се върже към.

Изходни данни: След като е установена връзката с помощта на socket ще бъде върнато число, което ще е идентификатор на връзката.

Грешки и изключения: Ако е настъпила грешка при свързване се хвърля грешка с подходящо съобщение.

**Expense& GetData(std::pair<Connection, System> connection)** – взема данните, от плащането, което е извършил потребителя, заедно с категорията и ги записва в Expense data.

Входни данни: Ще се взимат за входни данни от системата, която е подадена от потребителя.

Изходни данни: Ще се получава инстанция на класа Expense с попълнени данни съответно по описаните точки.

Грешки и изключения: Ако настъпи грешка при извличането на данните от системата се хвърля грешка с подходящо съобщение.

**void AddToBudget(const Expense& expense)** – изпълнява същата функция, както при горепосочения начин за добавяне на разход.

Входни данни: Ще се взимат за входни данни разхода, който потребителя е решил да добави.
Изходни данни: Няма да има, вместо това ще се добавя разход в системата.
Грешки и изключения: Ако е настъпила грешка при добавяне на разхода се хвърля грешка с подходящо съобщение.

### 5.3. Интерфейс на модула Manual Add

**Expense& fillFormData()** – визуализира на потребителя формата която той трябва да полълни.

Входни данни: Няма да получава никакви входни данни, защото той ще попълва тази форма

Изходни данни: Ще се получава инстанция на класа Expense с попълнени данни съответно по описаните точки.

Грешки и изключения: Ако е настъпила грешка при попълване на данните се хвърля грешка с подходящо съобщение.

**void AddToBudget(const Expense& expense)** – взема данните от формата, която е попълнени от потребителя и изпълнява същата функция, както при горепосочените начини за добавяне на разход.

Входни данни: Ще се взимат за входни данни разхода, който потребителя е решил да добави.

Изходни данни: Няма да има, вместо това ще се добавя разход в системата.
Грешки и изключения: Ако е настъпила грешка при добавяне на разхода се хвърля грешка с подходящо съобщение.

## d) Описание на възможните вариации

Модулите за индивидуалния и споделеният бюджет могат да бъдат обединени в един модул, като ще трябва вътре като подмодул да бъде добавен още един, който ще е за съхранение на данните за това какъв е самият тип бюджет. Ако е индивидуален няма да може да се достъпва модула Add users и да бъдат добавяни потребители към него и User data ще се ползва само за един потребител, а ако е споделен ще може да се достъпи и да се добавят други потребители, които ще са до четири на брой според изискванията и техните данни ще бъдат съхранени в User data.

Модулът на Премиум потребителя може да се наследява от обикновения потребител, като за целта могат модулите Basic user и Privileged user да бъдат обединени в един модул и вътре да е като подмодул Privileged user, за да се покаже че така ще се случи наследяването.

## 3. Описание на допълнителните структури

### а) Структура на разположението

Deployment е изключително важна за нашата система, защото както в декомпозиция на модулите се вижда всеки модул за какво служи, сега ще трябва да се покаже как самата система седи на хардуерно ниво (как се разполага). Едно ключово изискване към самите нас като софтуерни архитекти, е да видим това как се случва, а именно с помощта на такава структура.

Като изпървоначално ни е подсказано още с първото изискване към системата да се насочим към това, че ни е нужна структура на разположението, а то е дадената система да бъде достъпвана чрез браузър или мобилен клиент (iOS/Android). Това не би успяло успешно да представи по какъв начин би се случило, без да е налична такава диаграма за това как ще се случи на хардуерно ниво.

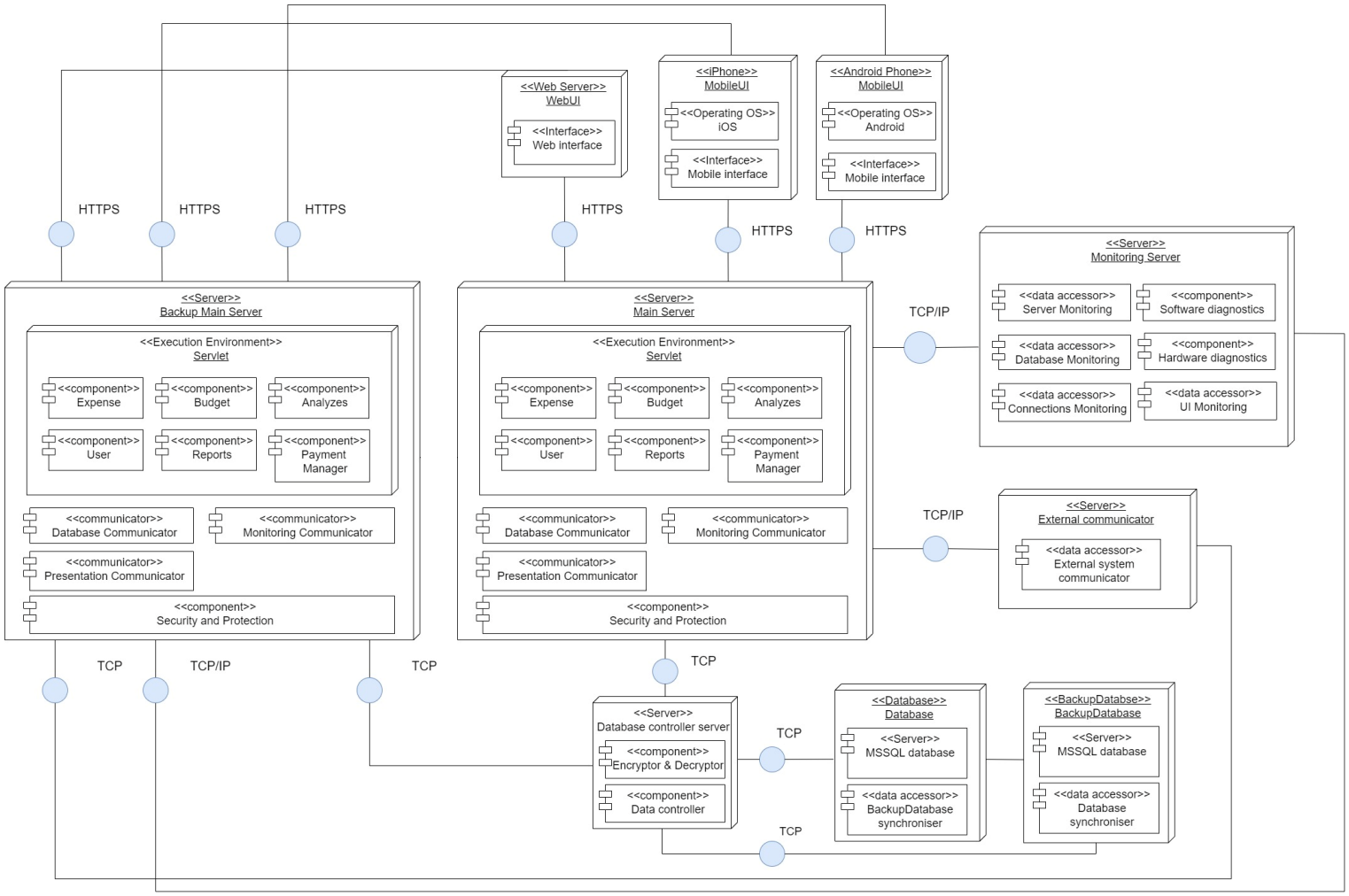
От изискванията (няколко на брой) се подсказва и че нашата система ще трябва да се свързва с външни системни, като нашата трябва да е напълно независима от тях (в случай че те се сринат, да не се срине и нашата). Затова ни трябва отново някакъв хардуер, който ще осъществи тази комуникация. Важно допълнително изискване е и да могат да се добавят лесно външни системи към нашата, като това отново ще се извършва с помощта на този хардуер, които споменахме. Всичко това отново не би се случило да се покаже без да е налична структура на разположението.

Изискването нашата система да е 99,95% налична, ние трябва да можем да преценим така нещата да могат да бъдат структурирани, че системата ни да е налична почти постоянно. Това се случва с помощта на по-долу описана тактика за справяне с проблема за неизправност с допълнителен сървър. Всичкото това нещо трябва да се случи както на софтуерно ниво да е добре разработена архитектурата, така и на хардуерно ниво, като едното не може без другото. Затова отново трябва да се допитаме към структурата на разположението, да се види как би се случило справянето с този проблем.

Профилактиката, която трябва да се случва максимум 6 часа на месец, отново ни подсказва, че трябва да имаме нещо което да следи за нормалното функциониране на системата и при случай на срив, да може бързо и успешно да се справим с него, като това да не засегне крайния потребител като данни и ползване на системата. Отново имаме отново по-долу описани тактики за справяне с проблемите в системата.

Не на последно място е нужно и да видим как бихме могли да се справим с проблема ако нашата база данни се срине да не изгубим данните, които имаме или пък данните постъпили/постъпващи по време на сриването на й, и какво бихме направили това нещо да бъде предотвратено, тъй като отново цялата ни система разчита на данните, които се съхраняват в нея.

## 1. Първично представяне:




## 2. Описание на елементите и връзките:

2.1 Web Server: е сървър, на който ще се разполага Web Interface за визуалнта част, която ще се визуализира на потребителя и от където ще може да ползва системата през браузър. Той комуникира със главния сървър чрез протокола HTTPS и по този начин се осигурява трансфера на данни. Независимо дали устроиството е компютър, телефон, конзола или друг вид устроиство имащо браузър, системата ще може да бъде достъпена през него.

2.2 iPhone и Android Phone са хардуерни устройства, на които, съответно от операционната система (iOS/Android според изискванията), която е на самото устроиство, ще бъде достъпена нашата система чрез мобилно приложение, в което ще бъде зареден Mobile Interface.

2.3 Main Server е главният сървър, на когото се разполага цялата функционалност на системата, като ще използваме за execution environment Servlet, като от своя страна той ще изпълнява главните компоненти на нашата система – Expense, Budget, Analyzes, User, Report, Payment Manager. Извън от изпълнимата среда на сървъра, имаме отделни компоненти, които са комуникатори със съответните други части на

системата. Тези комуникатори са за базата данни, за web и мобилният интерфейс и за връзката с мониторинг сървъра. На отделен компонент в рамките на сървъра под името Security and Protection се разполага софтуерната част, която е Server encryption и Authentication and Authorization.

2.4 Тъй като по изискване се изисква от нашата система тя да е налична 99.95% от времето, като се допуска месечна профилактика в рамките на 6 часа, ние трябва да подсигурим системата ни да работи почти безотказно. Затова ние сме избрали да имаме резервен сървър, който, ако падне главният, да бъде мигновенно заместен. Отделна функционалност която този сървър има е ако заявките от потребители започнат да претоварват главния сървър веднага да се включи и този, за да бъдат обработени всички заявки и да не се налага потребителите да чакат, за да могат да ползват системата.

2.5 Database controller server ще представлява голям сървър, с голям капацитет и мощ, която ще може да обработва по най-бързия начин заявките на потребителите и техните данни в самата система, като ги разпределя по базата данни. Върху този сървър ще се разполага софтруерната реализация на модула Database controller, като 1/3 от капацитета му ще бъде за криптиране на данните, още 1/3 ще е да се разпределят по базата данни с помощта на компонентите Encryptor & Decryptor и Data controller. Останалата 1/3 ще е в случай нещо ако се повреди, веднага да се взема от това свободно място и да се отговаря на нуждите на криптирането и декриптирането, и на контрола на данни.

2.6 Database: Това е нашата база данни, на която ние сме избрали да работи с MSSQL Server, в която се съхраняват всичките данни от нашата система. Отделно пак имаме компонент, който служи за синхронизиране на тази база данни и резервната.

2.7 BackupDatabase ще ни служи за в случай, че първата ни база данни се срине или се повредят по някакъв начин данните, независимо това дали е заради хардуерен проблем или софруерен проблем, като отново данните са синхронизирани и са в актуален вид.

2.8 Monitoring Server ще е сървър, който ще следи за изправността на другите части на нашата система и съдържа различни компоненти за следене на главния сървър, базата данни, на визуалната част и връзките помежду им. Имаме и компонентите Software и Hardware diagnostics, които ще са в случай на неизправност да извести администратора за типовете и грешките според типа. Отново тук идва и една важна отговорност на този сървър и тя е, както е описано в модулната декомпозиция на Monitoring, да следи за грешките, които навременно оправени (в реално време чрез ъпдейти), биха ни подсигурили системата да работи добре и профилактиката да се извършва веднъж месечно за максимум 6 часа. В допълнение тук се следи и в случай, че сървъра се пренатовари или падне, да се включи другия, както и ако базата данни се срине да се включи другата. За следенето на компонентите дали са в изправност ще изпраща сигнали до други компоненти и те ако не върнат резултат, значи може да има повреда и с помощта на exceptions се вижда и типа на грешка, която е настъпила. Според самата нея ще може да се види дали проблема е на хардуерно ниво или на софтуерно, като после се праща на компонента diagnostics и оттам следва частта, която е спомената малко-по нагоре в същата точка. Ако е

настъпил даден проблем от софтуерна гледна точка в даден процес, сървъра ще прецени дали ще може да се преинициализира или ще се създаде нов екземпляр.

2.9 External Communicator е малък сървър, който ще се грижи за връзката на нашата система с външните за нея системи и комуникацията между тях чрез компонента External system communicator. Този сървър ще ни осигури лесно добавяне на нови системи, който ще служи като фасада между нашата система и външните, като по този начин и ще осигури независимост на нашата система от останалите(ако не работи външна, да не пречи на работата на нашата).

Компонентите, описани в горните точки и представени в структрурата на разположението са асоциирани със съответните модули в Декомпозиция на модулите.

### 3. Описание на обкръжението

1. External communicator сървъра комуникира с множество външни разплащателни системи по специален начин с тях както е описано в точките, които се споменава за комуникация с външни системи.
2. Тъй като в базата данни ще се ползва външен сървър MSSQL, отново трябва да има добре направени комуникационни канали към него. За тях ще отговаря Data controller в Database controller server.

### 4. Описание на възможните вариации

1. Софтуерната реализация на модула Database controller може да бъде сложена да се изпълнява на сървъра, който е на базата данни, но при наличие на по-голям сървър за база данни. Отделно още една възможност е да се добави още един сървър Database controller server, който ще е за BackupDatabase и данните за тази база данни ще минават през него. Но това решение би било доста скъпо на компанията.
2. Ако потока на потребителски заявки е прекалено голям, резервният сървър се е включил, тогава можем и да включим резервната база данни и тя да действа като втора база данни, ако този поток е прекалено голям и първата не може да се справи.

## b) Структури на процесите

Структурите на процесите от своя страна биха ни дали един различен поглед над системата ни и това как тя функционира. Тази представа би могла да ни бъде дадена с помощта на такива структури, като ни представи как един процес се осъществява в системата. Тези структури ни дават представа и какви са споделените ресурси помежду модулите, как се развиват данните и какво се случва в последствие от дадения процес. За да си отговорим на всичките тези въпроси, тези неща как се случват в нашата система, ние трябва да имаме и такива структури. Избраните от нас процеси, които сме представили са описателни за това как нашата система работи и това как дадени модули си комуникират и логиката зад самите тях.

### Activity diagram на процесът Login в нашата система

1. **Първично представяне:**

Тъй като е прекаено голяма диаграмата, за да се побере на малкото пространство в този параграф, е добавен хиперлинк към самата нея в края на документа.
*Линк*

2. **Описание на елементите и връзките:**

Когато потребителят влезе в системата, той изпървоначално няма да има право да достъпи функционалностите на системата ако не се впише в нея. Приемаме, че потребителя има вече регистрация в системата. Когато той влезне в системата, независимо дали от мобилен телефон или от уеб браузър, той ще има бутон вписване в системата (Standard login). Когато той го натисне, сигналът от натискането през UI извиква форма за попълване на данни, която се визуализира на потребителя. Той от своя страна попълва данните си и натиска бутона log In. Данните от формата се предават на модула Login, където се изпращат на Standard Login в модула Authentication and Authorization, за да се провери дали потребителя е валиден потребител в системата. Standard Login в Authentication and Authorization взема данните и ги подава на Authentication, от където започва реалната част по проверка за валидността на данните от потребителя. Оттук данните се прехвърлят на модула Database manager, който както упоменахме по-рано в документацията, служи за връзка с базата данни. Той подава данните на Server communication, който пък служи от страна на базата данни за комуникация със сървъра. Данните се прехвърлят на Database encryption, защото желаем всички данни в базата данни да са криптирани, в случай на опит на външна намеса, криптират се, прехвърлят се на Data controller, които от своя страна проверява дали присъстват данните в базата данни. Ако данните не присъстват в системата, базата данни подава на Data controller съобщение, че не съществуват данните в нея, той от своя страна генерира съобщение за грешка и то се подава в обратен ред до потребителя, първо на Server communication, после на Database manager, Authentication, Login в User и накрая се визуализира съобщението за грешка на потребителя и показва и съобщение за повторен опит. Естествено при 5 неуспешни опита, Login в User няма да позволи повторни операции и ще генерира съобщение приканващо за смяна на парола. При сменена такава с различна, тогава потребителя ще има правото отново да се впише в системата. Сега следва и другата ситуация, която е ако данните присъстват в системата. Ако да, тогава базата данни дава на Data controller метаданни за потребителя: име, и какъв тип потребител е. Тъй като тези данни са криптирани, те се подават на Database encryption, където се декриптират и се подават на Check Permissions в Authorization по стандартния път, през Server communication и Database manager. Проверява се какъв е типа на потребителя и го подава на Authorize, където от своя страна дава позволение на потребителя да достъпи системата според типа на акаунта му и извежда на екрана през UI съобщение за успешна операция.

3. **Описание на обкръжението:**
   Когато се подават данните за проверка за валидността им, се осъществява връзката със сървъра MSSQL, който е на друга компания.
4. **Описание на възможните вариации:**

От Database manager, когато се връщат метаданните за позволенията на потребителя, могат да бъдат пуснати към Authentication, защото последната връзка минала през него е от Authentication и могат да бъдат пуснати там и от Authentication да бъдат прехвърлени на Check Permissions в Authorization.

## Activity diagram на процесът добавяне на разход чрез снимка

1. **Първично представяне:**
   Тъй като е прекаено голяма диаграмата, за да се побере на малкото пространство в този параграф, е добавен хиперлинк към самата нея в края на документа.

   *Линк*

2. **Описание на елементите и връзките:**

При натискане на потребителя бутона за отчети с помощта на подмодула Number of budgets в Reports, се показва списък с всички бюджети и потребителят може да избира към кой от бюджетите да добави разхода. След това се влиза в избрания бюджет и чрез модула Access Expense data се достъпват данните за разхода в Expense data. Оттам се започва процедурата по добавяне на новия разход, през модула Add expense. Потребителя избира да стане с добавяне на снимка, за което отговаря модула By Photo. Оттам Модулът Access device data достъпва файловата система на устройството и ако потребителя позволи да бъде достъпено вътрешното хранилище на устройството, през което влиза, и извлича снимката, чрез която ще се добави разхода. След това се попълват данните за разхода автоматично със специален алгоритъм в системата ни. Данните се подават на модула Database manager, който служи за връзка с базата данни. Той подава данните от разхода на модула Server communication, който служи за връзка на базата данни със сървъра. Данните са прехвърлят на Data Encryption, където се криптират и се предават на модула Data Controller. Тук се създава уникален запис в базата данни, за да могат криптираните данни да бъдат разпознати при извличането им. След като е създаден този запис в базата данни, тогава се връща в обратен ред до Expense data съобщение за успех и номер на записа в базата данни, записва се в даден контейнер, например масив в самият модул, номера и се визуализира на потребителя разхода, който е добавил като показването се случва във модула UI. Ако потребителя откаже досъп на системата до вътрешната памет на устройството, от което е влезнал в системата, тогава веднага се визуализира съобщение за отказан достъп до данните на устройството.

3. **Описание на обкръжението:**
   Когато се добава разход чрез снимка от телефон, връзката се осъществява с външно устроиство, от което потребителя я добавя.
4. **Описание на възможните вариации:**
   Вместо данните, които се попълват директно от снимката в модула Access device data, да бъдат пращани към модула Database manager, могат да бъдат върнати към Expense data и оттам да бъдат пратени на Database manager.

   Може да се добави между Expense data и Access device data още един разделител, който да се казва Add expense и в него да има процес, в който

потребителят да си избира някой от трите начина за добавяне на разход, но в случая ще е само By Photo, следователно може да е разделител Add expense и в него да има процес „Добавяне на разход чрез снимка" и вместо в Expense data да е „Добавяне на разход чрез снимка“ да е само „Добавяне на нов разход“ и последователността от процеси да има вида:

Добавяне на нов разход -> Добавяне на разход чрез снимка -> Запитване от системата дали може да достъпи вътрешната памет на устройството.

## 4. Архитектурна обосновка

- Достъпът до системата трябва да може да се осъществява или през браузър, или чрез мобилен клиент за iOS и Android.

Тъй като ще се разработва система, а тя няма как да съществува без да е определено по какъв начин ще е достъпна на потребителите. От самата точка се разбира на какъв хардуер ще работи самата система, като това ще е много ценно за разработката и по-доброто разбиране на това как ще се рабие на различни структури (модулни, на процесите и на разположението), като най-вече на разположението, защото се касае за две различни типове архитектури – Уеб и мобилна. Това е постигнато като цялата система работи на сървър, като по този начин се позволява функционалността да не е ограничена до даден тип устроиство. Що се касае до визуалната част на системата, модулите за Web и мобилно приложение, които изпълняват функционалността, са отделени от сървъра в техен модул, който е наречен UI. На хардуерно ниво, за уеб частта работи на Web Server под името WebUI, а за мобилното работи на смартфоните с iOS и Android под формата на приложение.

- 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.

От тази точка произлиза частта, в която се показва, че има разграничение между дава типа потребители, като за обикновения потребител, модулите за него няма да могат да комуникират с всички модули на системата, което същервеменно ще повлияе и на структурата на процесите за самия акаунт. Решението което сме взели е в модула User да има два подмодула Basic user и Privileged user, в което в последствие ще се покаже, че само привилегирования потребител ще има достъп до секция анализи. Останалата част от системата и двамата ще имат еднакъв достъп. Всичко това е показано и в точки 1.2.1.1 и 1.2.1.2.

- 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.

За да може да се впише в системата потребителя трябва да има регистрация, която позволява да има своя собствена енкапсулирана среда, в която той ще има достъп до своите данни и ще може само той да си ги достъпва без външна намеса. За реализацията на тази цел ние сме избрали в модула User да има подмодули Login и Logout, чрез които потребителя ще може да се впише в системата. Данните трябва да бъдат защитени по даден начин, но това ще бъде описано в следваща точка. Всичките данни на потребителите ще бъдат запазени в базата данни и ще бъдат добре защитени от външна намеса. Детайлният процес по вписване на потребителя в системата е показан в 3.б) – Диаграма на процеса Login в системата. Вписването с помощта на външните системи е вътре в самият Login модул, който модул е вързан към сървъра External communicator показано в структурата на разположението. В описанието на 3.а).2-2.9 е описано самият сървър как работи и за какво служи по-подробно, като накратко сега той служи за връзка с външните системи и позволява лесно добавяне на нови. Като се добавят нови, по този начин ще може потребителя да се впише в системата и чрез тях, стига те да са от тип мрежи като Facebook, Google, LinkedIn, Twitter и други, а не за разплащане, които ще бъдат описани по-надолу.

- 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
    f. опционално: при споделен бюджет се вписва автоматично името на потребителя, въвел разхода
    g. опционално: бележки от потребителя

Бюджетът е един от основните модули на системата, без която тя не може и съдържа данни, които са ключови за нея. Когато потребителя се регистрира в системата, за първи път той ще има опцията да си създаде бюджет. За реализацията на тази точка ние имаме два модула, които са Budget и Expense. В модула Budget има подмодул

Create budget, който ще отговаря за създаването на бюджета. По подробно е описано в точка 1.2.5. Разходите ще могат да бъдат достъпвани през бюджетите, което е по-подробно показано в точка 1.2.6, а самата реализация на това какво съдържа разхода и как се добавя е в интерфейса за добавяне на разход. Изпълнимата част на цялото изискване, на хардуерно ниво, ще си има специални компоненти Budget и Expense, които ще са в контейнер в главния ни сървър.

- 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.

Тази точка доразширява представата на по-долно ниво как ще бъдат съставени самите модули, като разширение на тези от горната точка. След като един път потребителя е вписан в системата, той ще има опция за създаване на споделен бюджет. За целта ще има специален модул, който ще съдържа данни за до 4 човека, и ще поддържа връзките от тези акаунти към модула за споделения бюджет на потребителя. Описаното е постигнато с целта разделянето в Budget на два подмодула Individual budget и Shared budget. По-подробното описание на архитектурната ни идея се намира в точка 1.2.5, като там е описано как се добавят хора към споделения бюджет и как самите модули и подмодули са изградени.

- Всеки потребител може да създава бюджети или да се присъедини към създаден от друг потребител споделен бюджет.

Добавянето на нови бюджети е есенциална функционалност на нашата система, която не ограничава потребителя до това той да има само един бюджет и разходите към него съответно. С помощта на модула Create Budget ще може да се създават нови бюджети от потребителя, като те ще бъдат отразени в секция Отчети, която ще бъде описана по- надолу какво прави. Там в нея ще вижда всичките си създадени от него бюджети.

- 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 плащането. Категорията на разхода се определя от системата на база търговеца получил плащането.
c) чрез ръчно въвеждане на разход - Потребителят може да попълни форма за информация за разхода и да го добави към избрания от него бюджет.

Това е една от най-важните функционалности на системата, а именно добавянето на разход. В модула Expense има подмодул Add Expense, който служи за добавяне на разходи по изискванията от дадената точка. Чрез снимка на касов бон от мобилно устройство ще отговаря модула By Photo Bill, които ще достъпват необходимите данни от телефона, а за уеб такъв, който достъпва файловата система на компютъра. Целия процес на добавяне на разход чрез снимка на касов бон е показано в точка 3.б)-добавяне на разход чрез снимка, където е по-детайлно показан процеса, а за това какво се съдържа в самия модул и как той е изграден може да се види по-подробно в секция 1 - Декомпозиция на модулите в точка 1.2.6.2.1. За добавяне на разход чрез външна система ще отговаря модула By external system, който ще е свързан със сървъра External communicator, който от своя страна ще служи за връзка с външни системи. По-подробно е показано какво съдържа самия модул в същата секция в точка 1.2.6.2.2. И не на последно място модулът за ръчно добавяне на разход е разположен вътре в модула Add Expense, където е описано в точка 1.2.6.2.3 как е изграден и какво прави.

- 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.

По визия на екипа ни, идеята от визуална гледна точка на потребителя е той като влезе в сайта/приложението да има един бутон за Отчети и като го натисне, ще му излезне под формата на списък всичките бюджети, които е създал и ще може да си ги достъпва с натискане на бутон. За тази цел ние имаме модул Reports, който има като подмодули Number of budgets, от където ще се генерира списъка и ще се пазят

метаданни за бюджетите и Access budget за достъпването на бюджет. Цялото това нещо е показано в точка 1.2.4. В структурата на разположението показана в секция 3 точка а), имаме и компонент, който изпълнява цялото това действие.

От друга страна в тази точка се показва и как даден обикновен потребител може да стане премиум потребител. Ако обикновеният потребител пожелае да стане премиум потребител (да има достъп до всички функционалности на системата) ние трябва да можем да му осигурим тази възможност, като съответно това ще се случва чрез даден вид плащане, което от бизнес гледна точка би ни подпомогнало. Реализацията на това действие е с помощта на модула Payment manager, който има като модули описаните в по-следващата точка начини на плащане и възможност за добавяне на други начини. Тези модули ще могат да се свързват към сървъра External communicator, показан в структурата на разположението, от където ще се достъпват системите за плащане. Описанието на това как се случва процеса по смяна на типа на потребителя и как е показана в точка 1.2.2 в секция 1.

- 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.)

Тъй като имаме вече обикновения потребител, който има ограничен достъп до някои функции на системата, сега ще имаме и такъв, които ще има достъп до всичките предоставени функционалности. В модулната декомпозиция на точка 1.2.1 е показано разделението на модулите на обикновения потребител и привилегирования как са. Вторият от своя страна ще може да достъпва секция анализи. Модулът анализи се достъпва само и единствено от Privileged user и съдържа като подмодули Analyzes for expenses, Filter и SetNotification, които ще отговарят за описаните от изискването поддточки, като в тях ще се реализира софтуерната функционалност. За по-подробно описание на това как и какво прави е описано в точка 1.2.3 в секция 1.

- Възможни са опции за плащане с банкова карта, Revolut, Paypal и Skrill.

Това е изискване е отразено в модула Payment manager, където четири от модулите отговарят за връзка с тези системи като всеки един от тях се свързва със сървъра External communicator и така се осъществява връзката между системата ни и външните разплащателни системи. За тях може да се види и в точка 1.2.2 на секция 1, където има по-подробно описано всеки един от четирите модула какво прави.

- Данните в системата трябва да бъдат защитени от нерегламентиран достъп. Особено важно е комуникацията между системата и външните системи за разплащане да предоставя защита на пренасяните данни.

Най-важното нещо, когато имаме една система е да е защитена, за да може потребителя да е уверен в това че я ползва, тъй като имаме работа с лични данни, като банкови сметки, карти и данни за самия потребител, за които трябва да бъде отделен огромен ресурс за защитата на данните. За целта ние имаме модул Server encryption, който отговаря за криптиране на всчиките данни които минават, във и през сървъра, като това са наистина сложни алгоритми за криптиране на данните от външна намеса и затова ние сме решили да бъде направено в нашата система, което е отразено в точка 1.2.8. От друга страна тези данни, които постъпват в базата данни също трябва да бъдат защитени от външна намеса, затова и там сме решили да сложим специален модул, който ще се грижи за криптирането и дектиприране на данните под името Database encryption. Описанието на това този модул как работи и какво се случва около него в следствие на криптирането на данните е в точка 2.2.2 в секция 1. За допълнителна сигурност на данните ние сме добавили и модул наречен Authentication and Authorization, който ще служи за аутентикация на данните, които потребителя е въвел при вписване в системата и после ауторизация за това какви права той има над самата система, защото той няма как да има достъп до данни, които няма право. Това е описано в точка 1.2.7 на секция 1.

- Допуска се профилактика веднъж месечно в рамките на 6 часа. През останалото време, системата трябва да е 99,95% налична.

Това е едно нефункционално изискване, на което е обърнато особено голям фокус в нашата система. Още в структурата на разположението може да бъде видяна една тактика, която ние сме предприели за изпълнението на това изискване и тя е да имаме резервни сървър и база данни в случай на срив, като активен излишък. Отделно и ако има прекалено много потребителски заявки да могат да се включат и резервните, за да може системата да се справи с потока им. По-подробно описание на справянето с дадения проблем е описано в секция 3, точка а)-2-2.4, 2.6, 2.7, 2.8. Но опирането само до резервен сървър и резервна база данни не решава целия въпрос как ще бъде постигната тази цел системата почти винаги да е налична. Затова ние имаме и още в модулната декомпозиция отделен голям модул наречен Monitoring, чиято работа е да следи изправността на системата, като по-точно главния сървър, базата данни,

визуалната част (UI) и връзките между всички тях. В случай на установена грешка през diagnostics модула системния администрятор ще бъде уведомен за дадения тип грешка – софтуерна или хардуерна и веднага ще бъдат предприети мерки. За този модул е по-подробно описано в модулната декомпозиция в секция 1 точка 3. Този модул в структурата на разположението е сложен на отделен сървър Monitoring server, на който ще се изпълнява софтуерната част описана нагоре. По-подробното описание на самия сървър какво представлява и какво прави е в секция 3 точка а)-2-2.8. За поддържането на компонентите сървъра Мониторинг ще ползва тактиката на откриване на откази като използва hearthbeat като праща сигнали до други компоненти и tsin е ако не върнат резултат, значи може да има повреда и с помощта на exceptions се вижда и типа на грешка, която е настъпила. Подробното описание на последвалата част също присъства в тази точка. В Database controller server сме предприели още една тактика за резерва, като сме оставили 1/3 от сървъра да е свободна, в случай че стане повреда на някой от компонентите служещи за целите на системата ни. Това е показано в точка а)-2-2.5 в секция 3. Споменатите до тук тактики за справяне с различни проблеми са едни от многото за справяне със всички възможни проблеми, които могат да настъпят в системата.

- При извършване на плащане чрез външна система, известието за направеното плащане трябва да достига до потребителя в рамките на 30 сек.

Бързодействието на системата ни е един от главните приоритети за нас, затова не само на архитектурно ниво, но и на имплементационно ниво ще желаем да няма забавяния. При извършено плащане в системата, модулите са директно свързани с External communicator сървъра, който приема данните и препраща към съответната система за плащания. В този сървър ще си има специална част, която ще отговаря за плащанията, от където под формата на вход и изход ще се приемат данните и директно ще се препращат към съответната система, което ще гарантира бързодействието на системата. След като бъдат обработени данните в техните системи и бъде пратено съобщение за успешна или неуспешна транзакция, в нашата система по същия начин под формата на приемане като вход и веднага праща като изход от сървъра към нашия и се визуализира на потребителя. Всичкото това нещо, благодарение на оптимизираната част на главния ни сървър и на сървъра за връзки с външни системи, ще се случи съобщението да пристигне до 30 секунди.

- С цел актуалност на информацията, генерирането на агрегираните анализи трябва да става до 3 секунди

Отново както в обосновката на горната точка ние се стремим нашата система да е изключително бърза и да не се случват забавяния. Справянето с проблема за бързината идва от това, че самият модул анализи е отделен от бюджета и разходите, което означава че потребителя като натисне да му се генерират анализите няма да се налага целия процес да минава както за добавяне на разход през модула за отчети, бюджета,

разходите и чак тогава анализите, което би забавило процеса. Затова ние сме го отделили в отделен модул, който си извършва своята функционалност. Естествено, анализите ще бъдат свързани с разходите за да се достъпи тази информация по някакъв начин, но това няма да представлява проблем при генерирането им. Отделно и на софтуерно ниво този процес е добре автоматизиран и се използват структури и такива алгоритми за бързо генериране на самите анализи, които потребителя ще пожелае да му бъдат визуализирани.

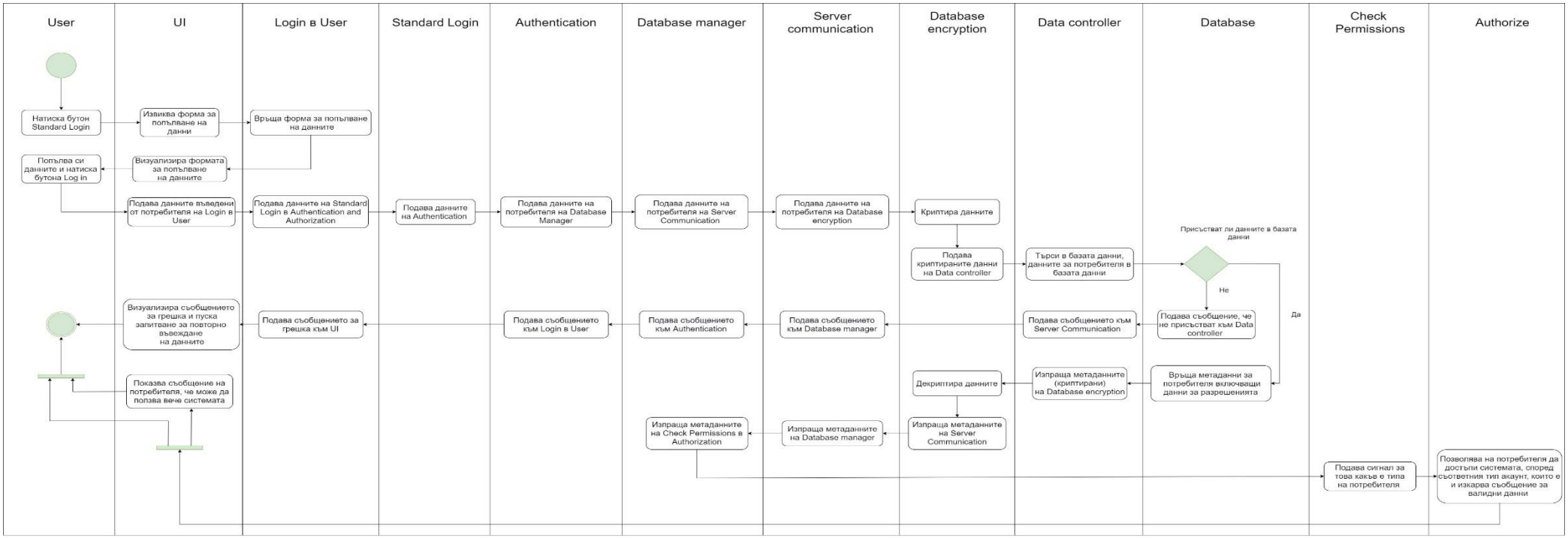
- Всеки потребител трябва да има възможност да изтрива данни от системата, асоциирани с него по всяко време спрямо GDPR.

Редакцията на личните данни е съществено важна за нас част, без която системата не би могла да осигури достатъчната комфортност на потребителя да може по всяко време да редактира своите данни. Тази функционалност сме я добавили в нашата система с помощта на модула Modify user data в модула User в декомпозицията на модулите. Показана е и как става в точка 1.2.1.5 на секция 1. Отделно, когато потребителя се регистрира в системата, ще му бъде визуализирано съобщение за съгласие дали позволява подадените от него данни да бъдат ползвани от системата.

- Архитектурата трябва да позволява лесно добавяне на нови системи за плащания.

Както и в по-горните точки беше описано, системата ще позволява лесно добавяне на нови системи за плащания с помощтта на модула Other payment methods в Payment manager, за който е описано по-подробно в 1.2.2.6 на секция 1. Отделно като имаме и сървър External communicator лесно през него ще могат да бъдат добавяни нови системи за плащане, като способността на този сървър да се добавят нови системи, които ще комуникира с тях е описано в точка а)-2.9 на секция 3.

## *Диаграма Activity diagram на процесът Login в нашата система*




## *Диаграма Activity diagram на процесът добавяне на разход чрез снимка*