СОФИЙСКИ УНИВЕРСИТЕТ
„СВ. КЛИМЕНТ ОХРИДСКИ"
ФАКУЛТЕТ ПО
МАТЕМАТИКА И ИНФОРМАТИКА

SOFIA UNIVERSITY
St. KLIMENT OHRIDSKI
FACULTY OF
MATHEMATICS AND INFORMATICS

# Проект

по

## Софтуерни архитектури и разработка на софтуер

на тема

# Velocity– система за наемане на велосипеди

**Изготвен от:**
Андрея Дяксова, 62455
Айше Джинджи, 62470

# Съдържание

# 1. Въведение

## 1.1 Обща информация за текущия документ

### 1.1.1 Предназначение на документа

В този документ е представена архитектурата на Velocity - система за отдаване на велосипеди под наем.

### 1.1.2 Описание на използваните структури на архитектурата

#### 1.1.2.1 Декомпозиция на модулите

Показва разделянето на системата на модули (сървиси) и подмодули, както и съответните връзки между тях. Основните модули на системата са Mobile UI, Web UI, Mobile App, Web App, Payment Manager, User Manager, Bike Manager, Fault Manager и Geographic Maps API. Показани са връзките между всеки един от модулите, както и връзката с базата данни. Тъй като модулът Bike Manager си комуникира с голям брой модули, за него сме обособили подмодул Communication API, който да се грижи комуникацията да се извършва безпроблемно. Базата данни е разделена на две отделни части - Bikes DB и Users DB, за да гарантира по-голяма сигурност на данните. Тази структура е от изключителна необходимост за разработчиците, QA, тестерите, както и системния администратор.

#### 1.1.2.2 Структура на процесите

В система като Velocity от съществено значение са високата надеждност и бързодействието при наличие на инцидент или технически проблем с велосипеда. Чрез структура на процесите, която показва основните процеси в системата и операциите, извършващи се между тях по време на каране на велосипеда, най-добре може да се покаже как са удовлетворени изискванията, свързани с реакцията на системата при проблем. Тази структура е от полза за разработчиците, тестерите и поръчителите, но високото ниво на абстракция, което предоставя тя, я прави подходяща и за потребителите, клиентите, системния администратор и хората от техническата поддръжка.

#### 1.1.2.3 Структура на внедряването

Системата Velocity е изградена от множество различни компоненти, които е необходимо да бъдат ясно показани посредтвом някоя структура на разположението, и по-конкретно - структура на внедряването. Структурата на внедряването предоставя информация за това кои модули върху какъв хардуер се разполагат. Тъй като голяма част от изискванията предполагат взимане на архитектурни решения, които водят до това, че един модул от декомпозицията трябва да бъде внедрен върху няколко машини или да съществуват

няколко инстанции на даден модул, е необходимо декомпозицията на модулите да бъде подкрепена от структура на внедряването. Структурата на внедряването (deployment) ще бъде от полза на следните заинтересовани лица: project мениджъри, тестери, маркетинг отдела, support-а, поръчителя, системния администратор и др. Възможно е да бъде полезна и на самите разработчици или на клиента.

### 1.1.3 Структура на документа

Документът се състои от 4 секции:
**Секция 1**: Въведение.
**Секция 2**: Представяне на структурата “Декомпозиция на модулите”, както и описание на всеки от модулите
**Секция 3**: Описание на допълнителните структури - “Структура на процесите” и “Структура на внедряването”.
**Секция 4**: Обосновка на архитектурата, в която се дават причините за изборите относно архитектурния стил и архитектурните драйвери

## 1.2 Общи сведения за системата

VeloCity е софтуерна система, предоставяща възможност за наемане на електрически велосипед за придвижване в рамките на града. През мобилно приложение потребителите могат да намерят най-близкия до тях свободен велосипед, който е снабден с електрическа система за задвижване, но може да се задвижва и чрез педали. След регистрация/вход в системата и плащане чрез избрания от тях метод (кредитна карта, СМС или чрез предварително закупени талони) потребителите могат да наемат велосипеда, като при наличие на проблем с велосипеда или някакъв инцидент се предприемат нужните мерки, за да се осигури максимална безопасност на наемателите.

## 1.3 Терминологичен речник

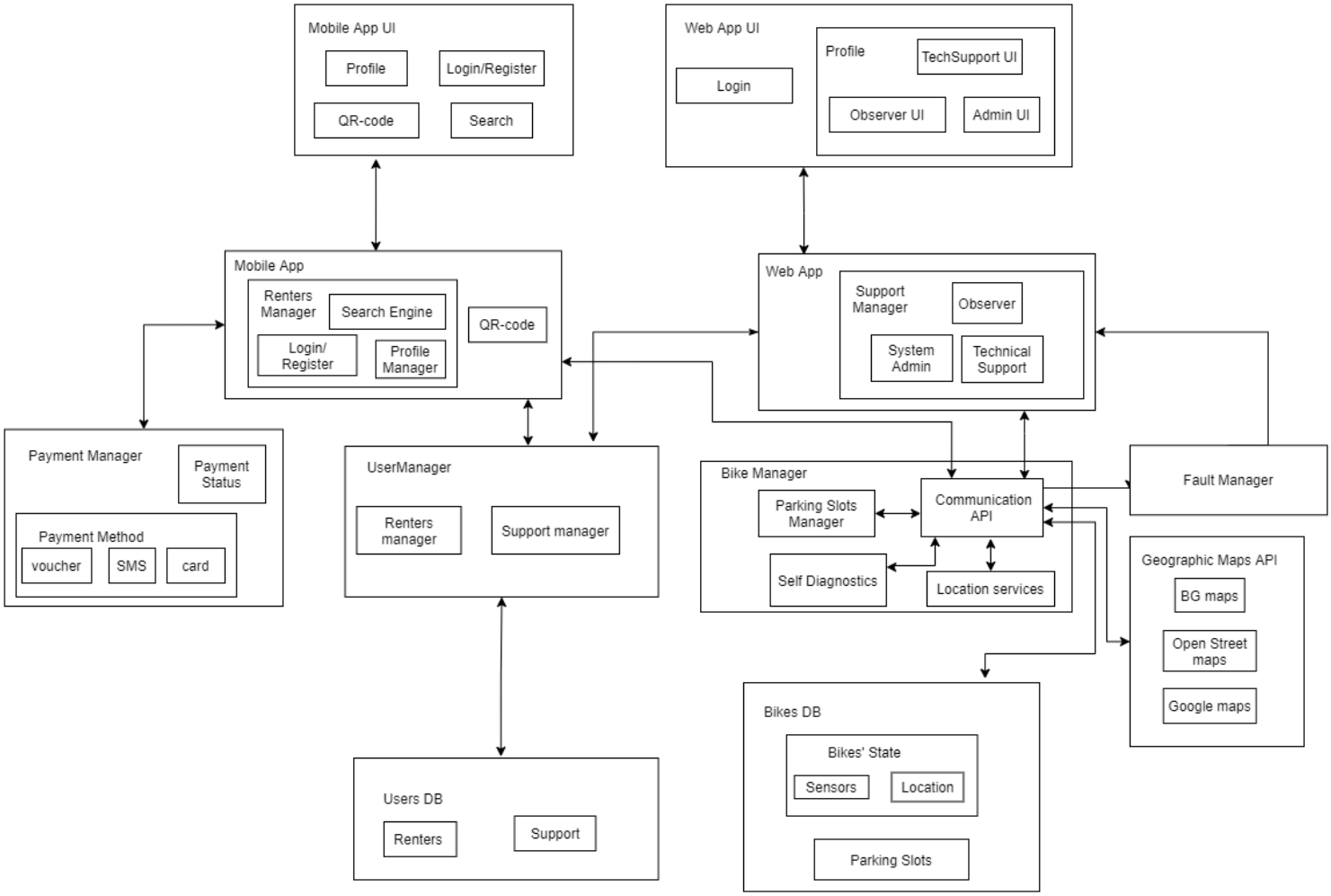
- Интерфейс (interface) – споделена граница, между която два отделни компонента на компютърна система си обменят информация;
- Модул – логически обособена софтуерна единица;
- Процес – съвкупност от стъпки, която изгражда логическо действие и стига определена цел;
- Декомпозиция – софтуерна структура, показваща как системата се разделя на отделни модули. Типовете елементи изграждащи тази структура са модули, а връзките между тях са от типа “X е подмодул на У”;
- Структура на процесите – софтуерна структура, показваща даден процес през какви условия и действия преминава;
- Структура на внедряването – софтуерна структура, показваща как софтуерът се разполага върху хардуера и комуникационното оборудване. Елементите са процеси, хардуерни устройства и комуникационни канали;
- Контекстна диаграма – диаграма, която дефинира границата между системата или част от системата и нейната среда, показваща обектите, които взаимодействат с нея;

- API - Application Programming Interface;
- Load Balancer - отнася се до процеса на разпределение на набор от задачи върху набор от ресурси, с цел да се направи цялостната им обработка по-ефективна;
- База от данни – колекция от информация, която е така организирана, че да може лесно да се достъпва, управлява и актуализира;
- Front-end - частта от Уеб приложение или страница, която е видима за крайния потребител;
- Back-end - логиката на приложението и управлението на данните;
- Сървър - стартирана инстанция на софтуерна система, която може да приема заявки от клиент и да връща подходящи отговори;
- Passive redundancy - архитектурна тактика, която се изразява в прилагането на допълнителен (излишен) елемент само ако главният се провали;
- PIN (Personal Identifying Number) - ЕГН

# 2. Декомпозиция на модулите

## 2.1 Общ вид на декомпозиция на модули за системата

Структурата представя Microservice архитектурата на Velocity. Системата е разделена на модули (сървиси), като всеки модул представлява отделно логическо ядро на системата. Показана е връзката между различните модули, т.е. как всеки от тях си комуникира с останалите. Всеки модул се състои от подмодули, които са тясно свързани един с друг, но отговарят за изпълнението на различни задачи в системата. Модулите в системата са разпределени на отделни нива, всяко от които представлява отделна смислова част. На най-високо ниво стоят двата вида интерфейс на системата. Под тях се разполагат модулите, които осъществяват връзката между интерфейса и Backend-а на системата. На по-ниско ниво са разположени модулите, които осъществяват работата на системата. На най-ниско ниво са обособени две отделни бази данни, които да пазят информация съответно за потребителите и за велосипедите.




## 2.2 Контекстна диаграма

Диаграмата показва кои модули от декомпозицията на модулите си комуникират с различни външни системи, т. е как се осъществява връзката на системата с външния свят.

**Mobile App UI** и **Web App UI** представляват интерфейса на системата. Те осъществяват комуникацията на системата с нейните потребители. **Mobile App UI** предоставя интерфейса на мобилното приложение, което цели да бъде използвано от наемателите на системата. **Web App UI** предоставя интерфейса на Уеб приложението, което да се използва от системния администратор, наблюдателя на системата и екипа по техническа поддръжка.
Връзката между интерфейса и системата се осъществява от модулите **Mobile App** и **Web App**, които комуникират както със съответния интерфейс, така и с модулите от по-ниско ниво.
**Payment Manager** осъществява връзката с външни системи за плащане, като предоставя единен интерфейс за плащане, независимо от вида на външната система. Това позволява

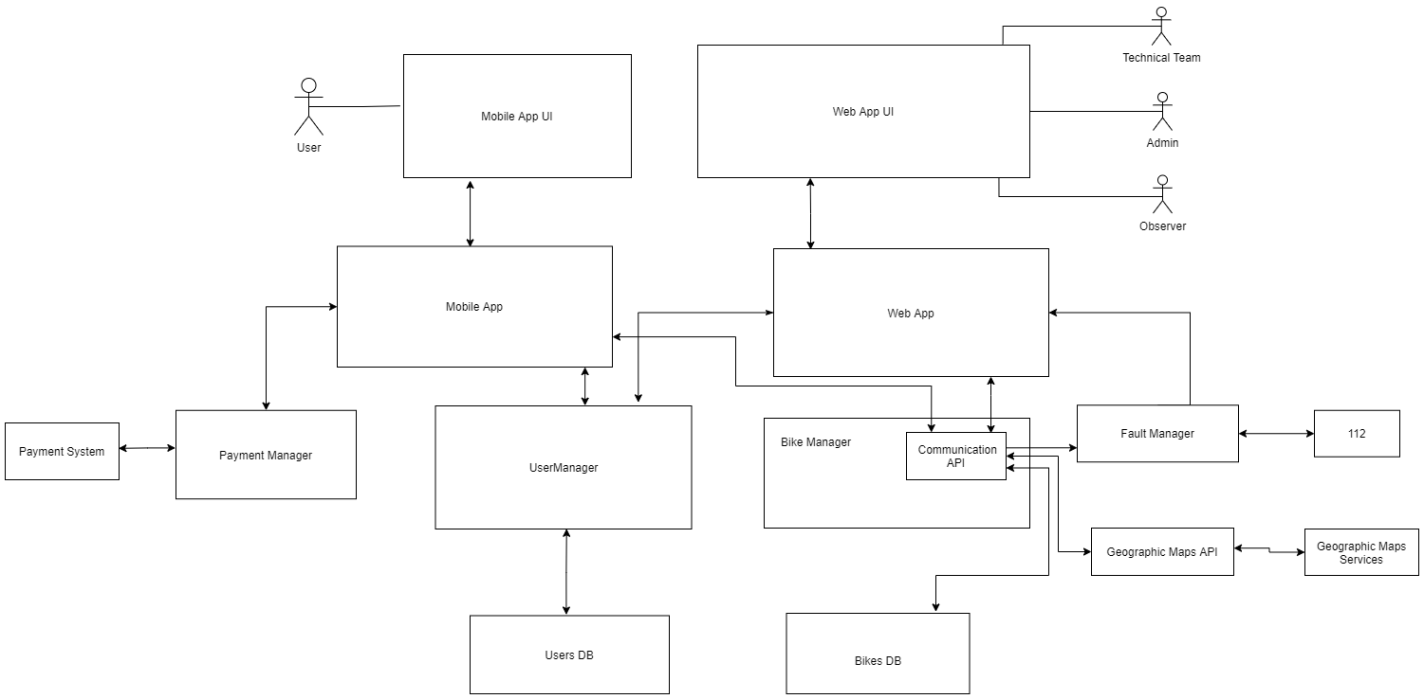
добавянето на нов метод за плащане или премахването на вече съществуващ такъв да се осъществява възможно най-лесно, без да се налага промяна в останалите модули от системата.

**User Manager** осъществява връзката на системата с потребителската база данни, като по този начин гарантира максимално скриване на личните данни на потребителите и осигурява по-голяма сигурност в базата. Като единствен модул, който си комуникира с **Users DB**, ако се наложи промяна в базата данни, останалите модули няма да бъдат засегнати.

**Bike Manager** главен модул в системата, който трябва да си комуникира с почти всички модули. Посредством подмодула **Communication API**, **Bike Manager** изпраща събраната в модула информация до **Fault Manager** и **Web App**, като същевременно обменя данни с модулите **Geographic Maps API**, **Mobile App** и **Web App** и осъществява връзката с **Bikes DB** - базата данни, която пази информация за велосипедите.

При настъпване на инцидент в системата, **Fault Manager** получава информация от **Bike Manager**, известява наблюдателя на системата (**Observer**), и изпраща сигнал до бърза помощ в рамките на определеното в изискванията време, като по този начин осъществява връзката между системата и спешните служби.

**Geographic Maps API** си комуникира както с **Bike Manager**, така и с външни системи за онлайн географски карти и, подобно на **Payment Manager**, ни осигурява единен интерфейс за работа с различни географски системи, независимо от вида им.



## 2.3 Mobile App UI

### 2.3.1 Предназначение на модула

Mobile App UI представлява мобилната версия на VeloCity. Mobile App UI съдържа подмодули, предоставящи интерфейс за вход/регистрация в системата (Login/Register), профила на потребителите (Profile), търсачка на свободни велосипеди, както и четец на QR-кодове.

### 2.3.2 Основни отговорности на модула

Предоставя възможност на наемателите да се свързват с VeloCity чрез мобилната версия на приложението.

## 2.4 Web App UI

### 2.4.1 Предназначение на модула

Web App UI представлява Уеб версията на VeloCity. Този модул съдържа подмодул за вход в системата (Login) и подмодул, предоставящ различен интерфейс на профила (Profile) на потребителите в зависимост от тяхната роля в системата - член на екип от техническата поддръжка (TechSupport UI), наблюдател (Oberserver UI) или системен администратор (Admin UI).

### 2.4.2 Основни отговорности на модула

Предоставя възможност на хората от група "поддръжка" да се свързват с VeloCity чрез Уеб версията на приложението.

## 2.5 Web App

### 2.5.1 Предназначение на модула

В Web App се намира подмодулът Support Manager, който отговаря за потребителите, определени като част от модул “поддръжка”. Модулът има за цел да разграничи различните потребители, които ще използват Уеб приложението:  наблюдател (**Observer**), системен администратор (**System Admin**) и член по техническата поддръжка (**Technical Support**)), както и да предостави съответните права за всяка роля - наблюдаване на системата и извличане на справки за данните (**Observer**), администриране на системата **(System Admin**) и постоянен мониторинг на велосипедите за нужда от ремонт или поддръжка (**Technical Support**).

### 2.5.2 Основни отговорности на модула

Модулът Web App е отговорен за ограничаването на комуникацията между Frontend частта и същинската имплементация на системата. Той е част от т.нар. Backend for Frontend модел и осъществява връзката между Web App UI и останалите модули на системата.

### 2.5.3 Описание на интерфейсите на модула

Support

- **login(string username,string password)**
    - **вход**: потребителско име и парола
    - **резултат**: вход в системата
    - **грешки и изключения**: при неправилни потребителско име или парола се извежда съобщение за грешка и се дава възможност за ново въвеждане (до 5 пъти)
    - **зависимости от други елементи**: потребителското име и паролата се изпращат до User Manager, който проверява валидността им в базата данни (Users DB)
- **requestBikeData(int bikeID)**
    - **вход**: идентификатор на велосипед, за който да бъде получена информация
    - **резултат**: изпраща заявка към модулите, управляващи данните на велосипедите; в заяквата са включени ID на велосипеда и ID на администратора, изпратил заявката
    - **ограничения при употреба**: може да се използва само от профили с тип System Admin
    - **грешки и изключения**: при невалиден идентификатор връща съобщение за грешка
    - **зависимост от други елементи**: заяквата се изпраща към модула BikeManager

## 2.6 Mobile App

### 2.6.1 Предназначение на модула

Модулът Mobile App е предназначен за работа с наемателите на системата. Аналогично на Web App, в него се намира подмодулът Renters Manager, който отговаря за обработването на всички заявки, свързани с наемателите. В него се намират подмодулите Login/Register, Profile Manager и Search Engine, които са предназначени съответно за управлението на потребителските профили и търсенето на велосипеди за наемане в системата. Също така, в Mobile App се намира подмодулът QR-code, който се грижи за разчитането и обработката на QR-кодовете, с които може да се осъществи плащането.

### 2.6.2 Основни отговорности на модула

Модулът Mobile App, също като Web App, е част от Backend for Frontend модела и е отговорен за ограничаването на комуникацията между Frontend частта и същинската имплементация на системата. Mobile App осъществява връзката както с външния свят (Mobile App UI), така и със сървисите от по-ниско ниво.

### 2.6.3 Описание на интерфейсите на модула

Profile

- **login(string username,string password)**
  - **вход**: потребителско име и парола
  - **резултат**: вход в системата
  - **грешки и изключения**: при неправилни потребителско име или парола се извежда съобщение за грешка и се дава възможност за ново въвеждане (до 5 пъти)
  - **зависимости от други елементи**: потребителското име и паролата се се изпращат до User Manager, който проверява валидността им в базата данни (Users DB)
- **register(string PIN,string username,string password,string additional_info)**
  - **вход**: ЕГН,потребителско име, парола (повторена 2 пъти), допълнителна информация, която може да бъде имена, номер на кредитна карта, телефонен номер и др.
  - **резултат**: създаване на профил на потребителя
  - **грешки и изключения**: при неудовлетворяващи изискванията парола/ потребителско име или невалидни ЕГН/ данни в допълнителната информация се изкарва съобщение за съответната грешка и се дава възможност за повторно въвеждане
  - **зависимости от други елементи**: данните се изпращат до User Manager, който от своя страна през DB API ги записва в базата данни (Users DB)
- **editProfile(int userID,vector<string> newInfo)**
  - **вход**: идентификатор на потребител и нова информация, която трябва да се запише при допълнителната информация за потребителя
  - **резултат**: записване на новите данни за потребителя
  - **грешки и изключения**: при невалидни идентификатор или нова информация (телефонен номер, номер на кредитна карта и др.) се извежда съобщение за съответната грешка
  - **зависимости от други елементи**: User Manager проверява валидността на подадените параметри и записва новите данни за дадения потребител в базата данни (Users DB)
- **findBike(Location location,int userID)**
  - **вход**: координати и идентификатор на потребителя
  - **резултат**: координати на 10 най-близки велосипеда
  - **грешки и изключения**: при невалидни координати или идентификатор се изкарва съобщение за грешка

    - **зависимости от други елементи**: В модула Bike Manager се проверява кои са най-близките до потребителя велосипеди
- **readQR(QRcode code,int userID)**
    - **вход**: QR-код, който да бъде обработен, идентификатор за потребител, който е заснел кода
    - **резултат**: автоматично въвеждане на уникален код за предварително закупени талони
    - **грешки и изключения**: при невалиден код  или идентификатор се връща съобщение за грешка
    - **зависимости от други елементи**: Mobile App получава QR-кода от Mobile App UI и го изпраща към Payment Manager

# 2.7 Payment Manager

## 2.7.1 Предназначение на модула

Payment Manager се грижи за правилното обработване на заявките за плащане и осъществява връзката с външни системи за плащане, като предоставя единен интерфейс за плащане в системата, който не е зависим от вида на Payment System. Подмодулът Payment Method предоставя възможност за различни методи на плащане (voucher, SMS, card, etc.), а подмодулът Payment Status сменя статуса на плащанията.

## 2.7.2 Основни отговорности на модула

Модулът Payment Manager е отговорен за сигурността при плащане в системата и защитата на личните данни на потребителите, както и за правилната работа на системата при интеграцията ѝ с външни системи за плащане.

## 2.7.3 Описание на интерфейсите на модула

Payment

- **pay(int userID,paymentMethod method,string info)**
    - **вход**: идентификатор на потребител, метод на плащане, нужна информация за извършване на плащането (info може да е телефонен номер, номер на кредитна карта или друго в зависимост от избрания method)
    - **резултат**:  заплащане на наем на велосипед
    - **ограничения при употреба**: заплащането може да се извършва само от регистрирани потребители
    - **грешки и изключения**: съобщение за грешка, ако плащането не е завършило успешно

## 2.8 User Manager

### 2.8.1 Предназначение на модула

User Manager управлява работата на различните групи потребители и държи информация за това кой потребител каква роля изпълнява. Той си комуникира единствено с модулите Web App и Mobile App и осъществява връзката с потребителската база данни (Users DB).

### 2.8.2 Основни отговорности на модула

Модулът User Manager е отговорен за правилното скриване на личните данни на потребителите и осъществяването на сигурна връзка с базата данни, както и за ограничаването на достъпа на останалите модули до базата данни (MobileApp и WebApp не комуникират директно с Users DB).

### 2.8.3 Описание на интерфейсите на модула

Users

- **verifyUser(string username,string password)**
    - **вход**: потребителско име и парола на потребител
    - **резултат**: връща резултат за валиден потребител
    - **грешки и изключения**: при невалидни данни връща индикатор за грешка
    - **зависимости от други елементи**: получава заявка за верификация на данните при вход на потребител в системата от модулите MobileApp и WebApp; проверява за наличието на данните в базата данни Users DB
- **createAccount(string PIN,string username,string password,vector<string> additional_info)**
    - **вход**: ЕГН, потребителско име, парола и допълнителни данни на профила, който да бъде създаден ( напр. имена, телефонен номер и др.)
    - **резултат**: връща идентификатор на създадения профил
    - **грешки и изключения**: при данни, които не отговарят на изискванията, както и при вече съществуващ профил се извежда съобщение за грешка
    - **зависимости от други елементи**: данните се получават от модула Mobile App и се проверяват в базата Users DB

## 2.9 Bike Manager

### 2.9.1 Предназначение на модула

Bike Manager следи състоянието на велосипедите (локация, батерия, време за използване и др.) посредством подмодулите Self Diagnostics и Location Services и известява съответните модули при наличието на проблем. Подмодулът Parking Slots Manager служи за управление на стоянките, на които се намират велосипедите. Communication API осъществява връзката между отделните подмодули в Bike Manager и останалите модули на системата, с които Bike Manager си комуникира.

### 2.9.2 Основни отговорности на модула

Модулът Bike Manager е отговорен за постоянния мониторинг на велосипедите и навременното известяване на съответните модули при засичане на проблем, както и за изпращането на данни, ако е необходимо. Също така, Bike Manager отговаря за контрола върху използването на велосипедите от потребителите (напр. при изтичане на времето за каране).

### 2.9.3 Описание на интерфейсите на модула

Bike Management

- **findBike(Location location,int userID)**
    - **вход**: координати (location), идентификатор на потребител, който търси свободен велосипед
    - **резултат**: координати на 10-те най-близки велосипеда
    - **грешки и изключения**: при невалидни координати се връща подходящо съобщение
    - **зависимост от други елементи**: чрез комуникация с Geographic Maps API и Bikes DB се откриват 10-те най-близки до потребителя велосипеда;

- **reportTechnicalProblem(Location location,int bikeID,int userID)**
    - **вход**: локация на велосипеда, идентификатор на велосипеда, в който е установен проблем, и идентификатор на потребител, който е наел велосипеда.
    - **резултат**: изпращане на съобщение до екипа по техническа поддръжка
    - **грешки и изключения**: при невалиден идентификатор на велосипед, невалиден идентификатор на потребител или невалидни координати се изкарва съобщение за грешка
    - **зависимост от други елементи**: Координатите (location) се изпращат от Geographic Maps API. Съобщението за технически проблем се изпраща към подмодула Technical Support на Web App.

- **reportLocationProblem(Location likelyLocation, int bikeID, int userID, string locationInfo)**
    - **вход**: най-вероятна позиция, на която се е намирал велосипеда в момента на изгубване на връзката, ID на велосипед, при който е засечен проблема, ID на потребител, наел велосипеда, данни за движението на велосипеда в последните 30 мин
    - **резултат**: изпращане на известие към наблюдателя на системата, както и данни за движението на велосипеда през последните 30 мин
    - **грешки и изключения**: при невалиден идентификатор на велосипед, невалиден идентификатор на потребител или невалидни координати се изкарва съобщение за грешка

    - **зависимост от други елементи**: Локацията (координатите) и данните за движението се получават от Geographic Maps API. Данните се изпращат към подмодула Observer на Web App

- **reportAccident(Location location, int bikeID, int userID)**
    - **вход**: локация на велосипеда, ID на велосипеда, за който е засечен инцидент, ID на потребителя, наел велосипеда
    - **изход**: изпращане на съобщение за инцидента до модула, който трябва да реагира
    - **грешки и изключения**: при невалиден идентификатор на велосипед, невалиден идентификатор на потребител или невалидни координати се изкарва съобщение за грешка
    - **зависимост от други елементи**:  съобщението за инцидента се изпраща до модула Fault Manager
- **getBikeData( int bikeID, int userID)**
    - **вход**: идентификатор на велосипед, чиято информация се търси, идентификатор на потребител, изпратил заявката
    - **резултат**: връща данните за велосипеда
    - **грешки и изключения**: при невалиден идентификатор на велосипед или идентификатор на потребител, който не е към администраторските,  се връща съобщение за грешка
    - **ограничения при употреба**: задейства се при получаване на заявка requestBikeData от Web App.
    - **зависимост от други елементи:** изпраща информацията до System Admin.

# 2.10 Fault Manager

## 2.10.1 Предназначение на модула

Fault Manager получава информация за състоянието на данните от Bike Manager, и при засичане на пътен (или друг) инцидент предприема необходимите мерки - уведомява наблюдателя на системата и автоматично изпраща сигнал до спешна помощ (112), ако е необходимо.

## 2.10.2 Основни отговорности на модула

Fault Manager трябва да гарантира бързодействие при засичането на инцидент. Той е отговорен за незабавната реакция на системата и навременното сигнализиране на необходимите лица, като по този начин осъществява връзка между системата и външния свят.

## 2.10.3 Описание на интерфейсите на модула

Fault Manager

- **reportTo112(int bikeID, Location location, int userID)**

    - **вход**: идентификатор на велосипед, локацията му (координати) и идентификатор на потребител, който е нает велосипеда
    - **резултат**: изпращане на сигнал до спешна помощ
    - **грешки и изключения**: при невалиден идентификатор на велосипед/наемател или невалидни координати се изкарва съобщение за грешка
    - **зависимост от други елементи**: идентификаторът на велосипеда (bikeID) и координатите (location) се получават от Bike Manager, като координатите в Bike Manager са изпратени от модула Geographic Maps API
- **reportToObserver(int bikeID, Location location, int userID)**
    - **вход**: идентификатор на велосипед, локацията му (координати) и идентификатор на потребител, който е нает велосипеда
    - **резултат**:  известяване на наблюдателя на системата
    - **грешки и изключения**: при невалиден идентификатор на велосипед/наемател или невалидни координати се изкарва съобщение за грешка
    - **зависимост от други елементи**: идентификаторът на велосипеда (bikeID) и координатите (location) се получават от Bike Manager, като координатите в Bike Manager са изпратени от модула Geographic Maps API

# 2.11 Geographic Maps API

## 2.11.1 Предназначение на модула

Geographic Maps API осъществява връзката между модула Bike Manager и онлайн услугите за географски карти и предоставя еднакъв интерфейс за работа с всички външни системи за географски карти.

## 2.11.2 Основни отговорности на модула

Geographic Maps API отговаря за интеграцията с онлайн услуги за географски карти и за предоставянето на бърз отговор на всички заявки, свързани с географски карти.

## 2.11.3 Описание на интерфейсите на модула

Location

- **get_location(int bike_ID)**
    - **вход**: идентификатор на велосипед
    - **резултат**: локацията на велосипеда (координати)
    - **грешки и изключения**: при невалиден идентификатор на велосипед се връща съобщение за грешка
    - **зависимост от други елементи**: информацията за координатите на велосипеда се изпращат до Bike Manager

## 2.12 Описание на възможните вариации

Модулът Geographics Map API може да бъде заменен от COTS, тъй като е стандартен модул за работа с географски карти.
Обменът на информация между отделните сървиси може да се извършва по протокола REST, тъй като в системата ще се прехвърля голямо количество информация и е необходим протокол, който да има възможно най-малко допълнителна информация.

## 2.13 Допълнителна информация

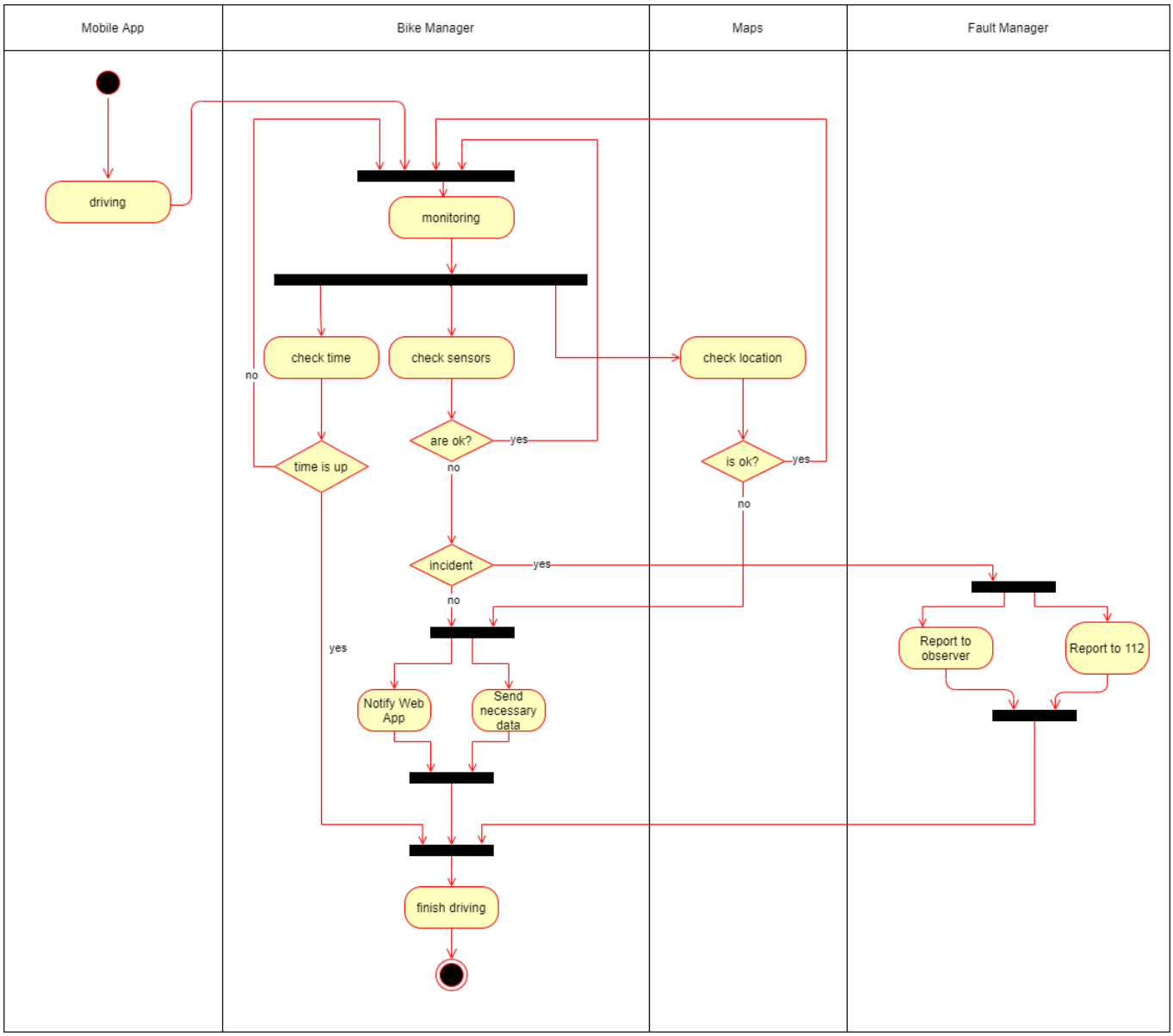
Location, QRcode и paymentMethod са сложни типове данни (специални структури), които да предоставят възможност за по-лесна обработка на данните, свързани с локацията (координатите) и с начина на плащане в системата.

# 3. Описание на допълнителните структури

## 3.1 Структура на процесите

### 3.1.1 Първично представяне

Activity диаграмата представя процеса на работа на системата при вече нает велосипед, който е в режим на каране. Разгледано е как системата реагира при различни ситуации, които могат да настъпят.




### 3.1.2 Описание на елементите и връзките

По време на самото каране модулите от декомпозицията на модулите, които са засегнати, са Mobile App, Bike Manager, Geographic Maps API и Fault Manager.

Процесът започва от състояние в **Mobile App**, при което велосипедът е вече нает и е в режим на каране (***driving***). Веднага след като велосипедът започне да се движи, се преминава към режим ***monitoring*** в модула **Bike Manager**, който следи за нередности по време на управление на велосипеда. В модулите Bike Manager и **Geographic Maps API** паралелно се следи текущото състояние на велосипеда, като за оставащото време за

използване на велосипеда (***check time***) и за смарт сензорите (***check sensors***) се грижи Bike manager, а проверката на локацията (***check location***) се случва в Geographic Maps API. Така, връщайки се към състояние *monitoring*, докато времето не е изтекло (time is up? -> no) и няма засечени проблеми със сензорите (are ok? -> yes) и локацията (is ok? -> yes), се постига постоянен контрол върху ситуацията.

В случаите, когато времето за използване е изтекло или се установи нередност в смарт сензорите (т.е. има технически проблем с велосипеда) (are ok? -> no) или с локацията (т.е. загуба на местоположение или излизане от рамките на града), цикълът се прекъсва и в зависимост от естеството на проблема, се предприемат различни мерки.

- Ако времето за използване на велосипеда е изтекло (time is up? -> yes), системата преминава към състояние 'finish driving' и процесът се прекратява.

- Ако засечен проблем със сензорите, който не се определя като инцидент (incident -> no), или е засечен проблем с локацията на велосипеда от модула Geographic Maps API (is ok? -> no), модулът Bike Manager предприема нужните мерки, като известява модула Web App (***Notify Web App***) и изпраща нужната информация, определена в изискванията (***Send necessary data***), отново се отива в състояние finish driving.

По този начин се удовлетворяват изискванията:
*10. При излизане на велосипед от рамките на града, трябва да се сигнализира наблюдателя в рамките на 1 мин, като се изпратят данни за движението на велосипеда в последните 30 мин.*

*11. При загуба на връзка с даден велосипед, трябва да се сигнализира наблюдателя в рамките на 10 сек, като се изпратят данни за движението на велосипеда в последните 30 мин, заедно с най-вероятната му позиция на която се е намирал в момента на изгубване на връзката.*

- Последният възможен случай включва ситуациите, в които е засечен пътен (или друг) инцидент (incident -> yes). Тогава, както е описано при определянето на модулите, Bike Manager изпраща информация до **Fault Manager**, който от своя страна изпраща сигнал до спешна помощ (***Report to 112***), известява наблюдателя на системата (***Report to observer***) и отново процесът се прекратява.

### 3.1.3 Описание на обкръжението

- За да се осигури постоянно наблюдение върху локацията на велосипедите, е необходимо VeloCity да се свързва с различните **онлайн услуги за географски карти** (Google maps, BG maps, Open Street maps и т.н.), като за тази комуникация отговаря модулът Geographic maps API.

- Другата външна система, с която VeloCity е възможно да си комуникира по време на каране на велосипеда, е **спешна помощ (112)**. С цел по-голяма сигурност и

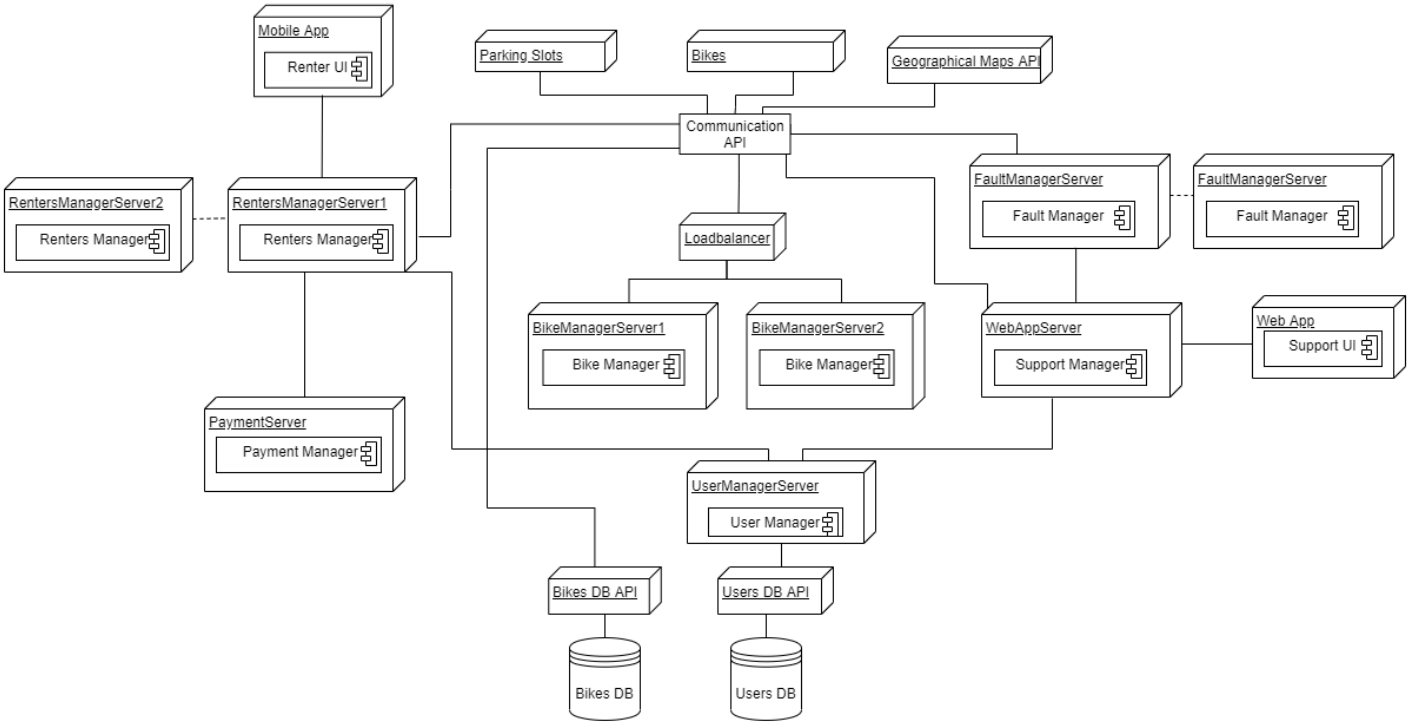
надеждност при инцидент с велосипеда или с велосипедиста, за връзката със 112 отговаря отделен модул - Fault Manager.

### 3.1.4 Описание на възможните вариации

Възможно е проверката на локацията да се осъществи в модула Bike Manager, като Geographic Maps API да се грижи само за изпращането на данните за локацията, но за да се намали натоварването върху Bike Manager, в случая тази задача е прехвърлена на Geographic Maps API.

## 3.2 Структура на внедряването

### 3.2.1 Първично представяне




Структурата на внедряването е основна структура в системата Velocity. Тя дава информация за това как различните модули се разполагат върху отделни машини. Най-важните модули в системата са внедрени на отделен сървър. Част от тях са дублирани с цел по-висока надеждност, а някои от сървисите са разделени на няколко отделни устройства, така че натоварването да е минимално дори в най-пиковите часове. По този начин се гарантира минимален шанс за срив на системата.

### 3.2.2 Описание на елементите и връзките

Тъй като при инцидент бързодействието и надеждността на системата са изключително важни, модулът Fault Manager е разположен на отделен сървър - (**FaultManagerServer** ).
Според изискванията се допуска ремонт и профилактика в интервала от 2:30 до 5:30 ч., а в останалата част на деня системата трябва да е 99,999% налична. Поради тази причина е създаден пасивен излишък на **RentersManagerServer** и на **FaultManagerServer,** т.е. за тях има по още един допълнителен сървър, който да се използва само ако главният сървър откаже.
Съществуват и изисквания към системата, според които тя трябва да бъде устойчива към пикови натоварвания в най-натоварените часове в денонощието, както и навременно да се осведомяват съответните лица при инцидент, технически проблем или излизане на велосипеда от рамките на града. Поради тази причина модулът Bike Manager е разделен на два сървъра (**BikeManagerServer1** и **BikeManagerServer2**), за да се намали натоварването и съответно да се увеличи надеждността на VeloCity, като за разпределянето на заявките между BikeManagerServer1 и BikeManagerServer2 се грижи **Load Balancer**.
През **Communication API** BikeManagerServer-ите се свързват с RentersManagerServer-ите и **WebAppServer**, както и с модулите, които осигуряват постоянен monitoring на състоянието на велосипедите, т.е. стоянките (**Parking Slots**), онлайн услугите за географски карти (**Geographical maps API**) и самите велосипеди (**Bikes**).

Също така Payment Manager, който предоставя единен интерфейс за плащане в системата, е внедрен на отделна машина - **PaymentServe**r за по-голяма сигурност при плащане.

Модулът UserManager, който държи информация за ролите на потребителите, се разполага върху отделен **UserManagerServer**, за да се осигурят ограничен достъп и пълна защита на личните данни на потребителите.

Обособени са две сигурни база данни, достъпът до които да се извършва само през **Users DB API** и **Bikes DB API**.

**MobileApp** представлява всички смарт телефони на наемателите, на които има изтеглено мобилното приложение, а **WebApp** - компютрите, чрез които хората от поддръжката (наблюдател, администратор, членове от техническата поддръжка) използват системата.

### 3.2.3 Описание на обкръжението

- **Payment Server** е свързан с множество външни системи и комуникира с тях, като дава възможност за интеграция независимо от типа на външната система
- **Map API** комуникира с външния системи за онлайн географски карти и дава възможност за интеграция независимо от типа на географската система.

- **Fault Manager** осъществява връзката между системата и спешна помощ, като при пътен (или друг) инцидент изпраща сигнал до 112 в рамките на 1 сек след засичане на инцидента.
- **Bikes DB API** и **Users DB API** осъществяват връзката с отделните бази, като гарантират максимална сигурност при достъпа до данните.

### 3.2.4 Описание на възможните вариации

Възможно е изискването за 99.999% наличност на системата да бъде изпълнено като вместо Passive redundancy за Fault Manager и Renters Manager се използва Active redundancy. Това би увеличило изправността на системата, тъй като времето за активиране на допълнителния сървър при срив ще стане много по-кратко, но разходите биха били драстично по-големи.
Разнородността в копията на Bike Manager може да бъде постигната по различни начини- разнородност в проектирането, по данни и по време.

# 4. Архитектурна обосновка

## 4.1 Архитектурен стил

Архитектурата на системата VeloCity е Microservice архитектура. Тъй като VeloCity е голяма система, състояща се от много на брой модули, е необходимо тя да бъде разделена на по-малки, лесноуправляеми микросървиси, които да са независими един от друг. По този начин се гарантира независимото скалиране на отделните модули, както и по-добрата fault изолация, т.е. наличието на проблем с някой от сървисите да не доведе до срив на цялата система.

## 4.2 Архитектурни драйвери

**1. Системата трябва да поддържа следните групи потребители:**

**a. Наемател на велосипед (обикновен потребител)**

**b. Член на група по техническа поддръжка на велосипедите**

**c. Системен администратор (техническа софтуерна поддръжка)**

**d. Наблюдател/отговорник по използването на велосипедите**

Това изискване е удовлетворено чрез разделяне на различните потребители на две групи – наематели (renters) и хора от поддръжката (support), като support-а от своя страна разграничава ролите член на група по техническата поддръжка, системен администратор и наблюдател по използването на велосипедите. Модулите, занимаващи се с различните роли в системата, са MobileApp, WebApp и

UserManager и те са подробно описани в декомпозицията на модулите ( **2.5,2.6,2.8**) Така се удовлетворяват и изискванията, че наемателите използват мобилната версия на приложението (Mobile App UI), докато системните администратори и наблюдателите използват системата през Уеб версията (Web App UI).

**2. Велосипедите се намират на стоянки, разположени на предварително определени позиции в рамките на града. Батериите на велосипедите се зареждат по време на престоя им на стоянката. Всеки велосипед е снабден с GPS устройство, както и със смарт-сензори за самодиагностика.**

Информацията, свързана с това изискване може да се намери в декомпозицията на модулите, и по-конкретно в описанието на модула Bike Manager (**2.9**). Този модул разполага с няколко подмодула, отговорни за постоянното следене на състоянието на велосипеда – Parking Slots Manager, Location services, Self Diagnostics.

Също така структурата на внедряването (**3.2.2**) показва физическото разположение на стоянките, велосипедите и двата сървъра (Bike Manager Server1 и Bike Manager Server2), отговорни за обработката на информация, свързана с велосипедите.

**3. Наемателите на велосипед се регистрират през мобилното приложение, като в профила им се включват следните лични данни: имена, ЕГН, както и данни за връзка. Личните данни на потребителите трябва да са абсолютно защитени от външна намеса. Достъпни са единствено до наблюдателя на правомерното използване на велосипедите.**

От структурата на внедряването (**3.2.2**) се вижда, че UserManagerServer-ът, отговарящ за различните роли в системата, е компонент, до който външният свят няма достъп. Комуникацията с него се осъществява единствено през вътрешна за системата мрежа, а именно RentersManagerServer и WebAppServer. Освен това данните на потребителите са физически отделени от информацията, свързана с велосипедите, като се съхраняват в отделна база данни - Users DB. За допълнително ниво на сигурност има и Users DB API, което да бъде единствената точка на достъп до Users DB.

**4. Потребителите може да заплащат услугата чрез кредитна карта, СМС или чрез предварително закупени талони, които съдържат уникален код. Кодът може да се въвежда ръчно или автоматично (QR-code).**

Това изискване се удовлетворява в модула Payment Manager от декомпозицията на модулите (**2.7**), където подмодулът Payment Method осигурява различни методи на плащане (кредитна карта, СМС, талони и др.), а подмодулът QR-code предоставя четец за QR-кодове, като естествено е предоставена и съответната визуализация в клиентската част (QR-code в MobileAppUI).

**6. При засичане на пътен или друг инцидент с велосипеда, се изпраща автоматично сигнал до спешна помощ (112), в рамките на 1 сек след засичане на инцидента. В рамките на 5 сек се известява и наблюдателя на системата.**

В структурата на внедряването (**3.2.2**) е показано, че Fault Manager е внедрен на отделна машина (Fault Manager Server), за да се увеличат надеждността и бързодействието при работата, тъй като това е модулът, чиято изправност в кризисни ситуации е от голямо значение за навременното уведомяване на съответните лица. Освен това за осигуряване на изправността на системата е използвана тактиката за пасивен излишък (passive redundancy), като е осигурен back-up Fault Manager Server2 в случай на срив в главния.

**7.Системата трябва да може да се интегрира с всички познати онлайн услуги за географски карти (Google maps, BG maps, Open Street maps и т.н.), като има възможност за бъдещо добавяне на нови карти.**

Това изискване е удовлетворено посредством микросървиса Geographic maps API от декомпозицията на модулите (**2.11**). Този модул съдържа отделни подмодули за различните картови системи и по този начин е осигурена възможността за интеграция и с други онлайн услуги за географски карти в бъдеще.

**8.Системата трябва да е устойчива към пикови натоварвания в най-натоварените в денонощието, часове за придвижване.**

За удовлетворяване на това изискване са приложени няколко тактики за изправност, представени чрез структурата на внедряването (**3.2.2**). Първо, съществуват два сървъра, които да обработват пристигащите заявки - BikeManagerServer1 и BikeManagerServer2, като тези заявки биват разпределяни между сървърите от Load Balancer. Също така комуникацията между Bike Manager Server-ите и останалите компоненти е сведена до минимум, използвайки Communication API. Освен това с цел по-висока изправност, ще бъде приложена тактиката за разнородност в копията на Bike Manager, за да се намали наличието на бъгове в системата.

Допълнително високата наличност на системата, посочена в изискванията (99,999% извън интервала за ремонт и профилактика), се постига чрез прилагането на тактиката за пасивен излишък (passive redundancy) за сървърите Renters Manager Server и Fault Manager Server.