Drones Park

Мария Иванова 9MI0600045

Николай Костандиев 6MI0600046

Март 2023 – Май 2023

Съдържание

I. Въведение

1. Обща информация за текущия документ
   1.1. Предназначение на документа
   1.2. Описание на използваните структури и архитектурата
   1.3. Структура на документа
2. Общи сведения за системата
3. Терминологичен речник

II. Декомпозиция на модулите

1. Account Manager
   1.1. Basic Functionalities
      1.1.1. Register
      1.1.2. Login
      1.1.3. Modify
   1.2. Payment Manager
   1.3. Subscription Manager
   1.4. Functionalities
      1.4.1. View free parking spots
      1.4.2. Send signal
      1.4.3. View slip
      1.4.4. Register intruder
      1.4.5. Notify emergency
2. Map API
3. User Authenticator
4. Database 1
5. UI and Adapter
6. Server
   6.1. Garbage Collector
   6.2. Process killer
7. Server backup
8. Recognition Manager
   8.1. Count of Drones and Route of Drones
   8.2. Image Processing
   8.3. Caching data
9. Database 2
10. Whether API

III. Допълнителни структури

1. Структура на процесите
   1.1. Процес на разглеждане на свободните паркоместа
      1.1.1. Първично представяне
      1.1.2. Описание на елементите и връзките
      1.1.3. Описание на обкръжението
      1.1.4. Описание на възможните вариации
   1.2. Процес на абониране за паркомясто
      1.2.1. Първично представяне
      1.2.2. Описание на елементите и връзките
      1.2.3. Описание на обкръжението

1.2.4. Описание на възможните вариации
1.3. Процес на изчисляване на броят и маршрутът на дроновете
1.3.1. Първично представяне
1.3.2. Описание на елементите и връзките
1.3.3. Описание на обкръжението
1.3.4. Описание на възможните вариации
2. Структура на внедряването
2.1. Първично представяне
2.2. Описание на елементите и връзките
2.3. Описание на обкръжението
2.4. Описание на възможните вариации
IV. Архитектурна обосновка

I. Въведение

1. Обща информация за текущия документ
   1.1. Предназначение на документа

Този документ има за цел да представи системата DronesPark - софтуерна система за следене и управление на свободните паркоместа.

1.2. Описание на използваните структури и архитектурата

Декомпозиция на модулите

Декомпозицията на модулите е съществена структура при проектирането на една софтуерна архитектура. Тя показва логическото разделение на приложението на модули и подмодули, както и връзките между тях. В нашата система основните модули са Account manager, Mobile UI, Adapter, Server, User Authenticator, Recognition manager.

Структура на процесите

Представени са три диаграми на последователността на основните процеси в системата. Те са: разглеждане на свободните паркоместа, абониране за паркомясто, изчисляване на броят и маршрутът на дроновете. Чрез тези диаграми е показано едно по-голямо ниво на абстракция спрямо архитектурата. Така се дава възможност за по-добро разбиране на това как се извършват основните процеси и начинът, по който отделните части от системата са свързани помежду си. Използваният език е опростен, за да може да е достъпен до заинтересованите лица.

Структура на внедряването

Deployment е една от най-ключовите структури за DronesPark. Чрез нея се дава по-ясна представа за това как някои от съществените изисквания ще бъдат изпълнени. “Системата да работи 100% без отказ в рамките на светлата част на работния ден”. Чрез Декомпозицията на модулите и Структурата на процесите, по никакъв начин не става ясно как ще се удовлетворят изисквания от това естество. Deployment структурата ни дава информация за разполагането на модулите върху хардуерните устройства, за наличността на бекъпи за най-важните части от системата, което е ключово за постигането на отказоустойчивост и висока сигурност на системата.

1.3. Структура на документа

Секция 1 – Въведение

Описва се структурата на документацията, представят се основните функционалности, предоставя се мотивация за избора на по-долу описаните структури, архитектурните драйвери и терминологичен речник за улеснение на потребителя.

Секция 2 – Декомпозиция на модулите

Представя се общ вид на декомпозиция на модулите, както и детайлна информация за всички модули и подмодули на системата.

Секция 3 – Описание на допълнителни структури

Представя се детайлна информация за структурата на някои от най-съществените процеси за системата (3.1).

Представя се информация за разполагането на модулите върху хардуерните устройства(3.2).

Секция 4 – Архитектурна обосновка

Защитават се архитектурните решения, които са взети за системата, с оглед отделните изисквания и използваните тактики за постигането им.

2. Общи сведения за системата

DronesPark е софтуерна система за следене и управление на свободните паркоместа. Системата има за цел да улесни ежедневието ни и по-точно паркирането в града. Предназначена за всеки, който иска бързо да намери място за колата си без досадно обикаляне. Свободните паркоместа се идентифицират от система от дронове, които обикалят града и заснемат зоните за паркиране. Аварийните групи се грижат при проблеми с дроновете. За динамиката на обновяване на данните за свободни места се грижи оператора на системата. Потребителите на системата се разделят на обикновени, които могат да плащат местата само динамично, и регистрирани, които могат също да се абонирам за място. При случай на неправомерно паркиране групите по контрол на паркирането преместват превозното средство, заснемат събитието и издават фиш.

Архитектурни драйвери

- Идентифициране на свободни паркоместа

Системата трябва да може да отговаря на заявки за наличност на свободни паркоместа в дадена зона от потребителите. За тази цел софтуерната архитектура трябва да е такава, че: в зависимост от дадени условия във всеки момент, да се определя броя и маршрута на дроновете; чрез заснетите изображения от дроновете, посредством специфичен алгоритъм и външна услуга да се определят свободните паркоместа; да е съвместима с тази външна услуга; да се обновява информацията на определен интервал. Част от условията за определяне на броя и маршрута на дроновете са натрупаните данни за динамиката на паркиране в съответния час и ден от седмицата. Поради тази причина архитектурата също трябва да поддържа база данни, която трябва да е добре скалируема и да съдържа цялата необходима информация.

- Достъпът до системата трябва да се осъществява чрез мобилно приложение

Системата трябва да бъде налична за Аварийни групи, Групи по контрол на паркирането, Регистрирани потребители, Обикновени потребители през мобилно приложение, което се поддържа от всички типове операционни системи за мобилни устройства. Функционалностите за всяка една от групите трябва да бъдат отделени от останалите групи, като за тази цел при разработването трябва да бъдат имплементирани правила за сигурност. Освен до собствените си функционалности, регистрираните потребители трябва да имат достъп и до функционалностите на обикновените потребители. Системата трябва да поддържа и отделна база данни за потребителите на системата, съдържаща тяхната лична информация, разплащателни данни, информация за фишове и глоби, и други.

- Заплащане на абонамент

Първоначално системата трябва да поддържа следните начини на плащане: кредитна/дебитна карта, СМС или PayPal. При проектирането, трябва да се съобразим с изискването за добавяне на допълнителни възможности за плащане, което би подобрило използваемостта.

- Надеждност на системата

Системата трябва да бъде структурирана по такъв начин, че да може да устоява на опити за неразрешена употреба, без това да пречи на легитимните потребители. Също така трябва да бъде проектирана по такъв начин, че да устоява на голямо количество заявки и да предоставя услугите си безпроблемно, съгласно спецификацията, в рамките на светлата част на работния ден.

3. Терминологичен речник

Adapter – Трансформира информацията по такъв начин, че тя да е съвместима между двете страни

Android, iOS – видове мобилни ОС

API – (Application Programming Interface) интерфейсът на изходния код, който операционната система или нейните библиотеки от ниско ниво предлагат за поддръжката на заявките от приложния софтуер или компютърните програми

Authenticate – Валидиране на данните

Backup – Резерва

bool – Логически тип данни

Cache – Място, където се съхраняват данни с цел по бърза обработка

Database – (база данни) колекция от информация, която е така организирана, че да може лесно да се достъпва, управлява и актуализира

Docker – Инструмент за създаване и работа с контейнери

enum – Тип данни, с предварително определени стойности.

HTTP – протокол, чрез който достъпваме и създаваме ресурси

id – Уникален идентификатор

intruder – нарушител

Json – формат на представяне на данни, така че да е лесен за хора и машини

Kubernetes – Инструмент за менажиране на множество контейнери

map - карта

Server – Локална или отдалечена компютърна машина, която извършва множество различни операции

slip – нарушения/глоби

SQL – Заявка към базата данни

String – Структура от данни, представляваща текст.

TCP – Транспортен протокол

UI – (User Interface) това, което потребителят вижда

unsigned – (unsigned int) цяло неотрицателно число в интервал [0; 2^32 - 1)

user – потребител

Web server – сървър, който приема и връща отговор на заявки, направени от клиент.

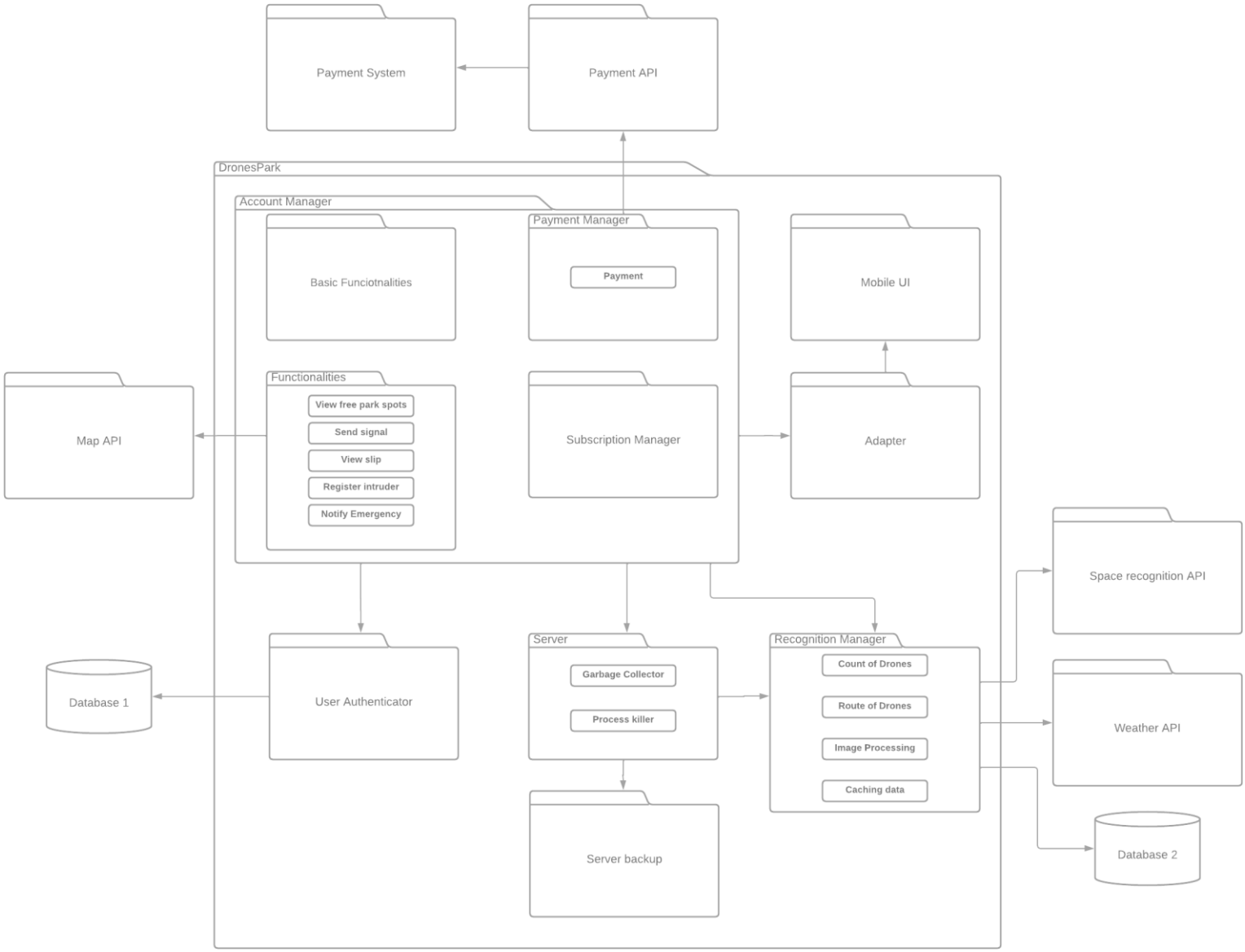
wrapper – обвивка

Архитектурна обосновка - Мотивира избраната архитектурна конфигурация и се описва как и защо се удовлетворяват основните изисквания към системата

Външна система – отдалечена софтуерна система, която е източник на данни или функционалности, които се използват от настоящата система

модул – Част от системата, обособена за конкретна задача

II. Декомпозиция на модулите



1. Account Manager

Този модул предоставя лесна употреба на мобилно приложение от неговите потребители. Съставен е от 4 подмодула.

1.1. Basic Functionalities

Позволява регистрация, вход и редактиране на акаунт.

1.1.1. Register

bool register(const std::string&,const std::string&,const std::string&,userType);

Вход: Потребителско име, имейл, парола, тип акаунт

Изход: true – успех ; false – неуспех

Резултат: При успешно регистриране информацията се запазва в базата данни

1.1.2. Login

userToken login(const std::string&,const std::string&);

Вход: Потребителско име и парола

Изход: Уникален токен за потребителя, с който се счита за успешно влязъл в системата – успех ; подходящо съобщение за грешка – неуспех

1.1.3. Modify

bool ChangeName(const std::string&);

Вход: ново потребителско име

Изход: true при успех ; false при неуспех

Резултат: При успех информацията в базата данни се обновява

bool ChangePassword(const std::string&, const std::string&);

Вход: стара парола, нова парола
Изход: true при успех; false при провал
Резултат: При успех информацията в базата данни се обновява

bool ChangeEmail(const std::string&);
Вход: нов имейл
Изход: true при успех; false при провал
Резултат: При успех информацията в базата данни се обновява

bool DeleteProfile(const std::string &);

Вход: парола

Изход: true при успех; false при провал

Резултат: При успех информацията в базата данни се обновява

1.2. Payment Manager

Този модул осигурява наличността на различните начини за плащане. Структуриран е така, че добавянето на нов метод за плащане да се интегрира лесно. Payment Manager

е пряко свързан с Payment System, през Payment API, като те авторизират и подсигуряват плащането. Payment System е външна система за плащане, а Payment API преобразува информацията, таке че да е съвместима със системата. Payment API връща на системата съобщение за успешността на транзакцията.

bool payment(Type,userCredentials,systemCredentials);

Вход: Тип на плащане (enum), данни за потребителя, данни за системата
Изход: true – успех, false – неуспех
Резултат: Заплащане на услугата по предпочитания от потребителя начин
Задължително се осъществява верифициране на потребителя през authenticator.

1.3. Subscription Manager

Следи за абонаментите за паркоместа на Регистрираните потребители. При неплатена месечна такса съответното паркомясто спира да се маркира като заето и да се пази за абонирания потребител.

bool addSubscription(userCredentials,subscriptionInfo);

Вход: Данни за потребителя, информация за абонамента

Изход: true – успех, false – неуспех

bool removeSubscription(userCredentials,subscriptionInfo)
Вход: Данни за потребителя, информация за абонамента
Изход: true – успех, false – неуспех

Задължително се осъществява верифициране на потребителя през authenticator.

1.4. Functionalities

Това е модул, които представлява основните функционалности на потребителите в мобилното приложение. Методите са представени, така че достъп до тях да имат само групите потребители, за които се отнасят. Тази проверка се извършва през Authenticator.

1.4.1. View free parking spots

Позволява на Регистрираните потребители и на Обикновените потребители, да преглеждат картата с маркирани заети и свободни паркоместа. За тази цел модула е свързан с Map API, който осигурява наличността на картата.

parkSpotInfo viewParkingSpot(region);

Вход: Информация за желания регион

Изход: Информация за паркоместата в този район – успех ; Подходящо съобщение - неуспех

1.4.2. Send signal

Позволява на Регистрираните потребители да изпращат сигнали за неправомерно заето от друг място, за което са абонирани.

bool sendSignal(userCredentials, parkSpotInfo);

Вход: Данни на потребителя подаващ сигнала, Информация за паркомястото

Изход: true – успешно подаден сигнал ; false - неуспех

1.4.3. View slip

Този метод препраща потребителя към външен публичен сайт, в който може да открие подробна информация за всички свои нарушения.

1.4.4. Register intruder

Този метод е предназначен за групите по контрол, които чрез него регистрират нарушител - заснемат настъпилото събитие и издават фиш за него.

bool registerIntruder(userCredentials, intruderCredentials, std::string)

Вход: Данни за потребителя, извършващ регистрацията ; данни за нарушителя ; описание на нарушението

Изход: Изход: true – успешна регистрация на сигнал; false - неуспех

1.4.5. Notify emergency

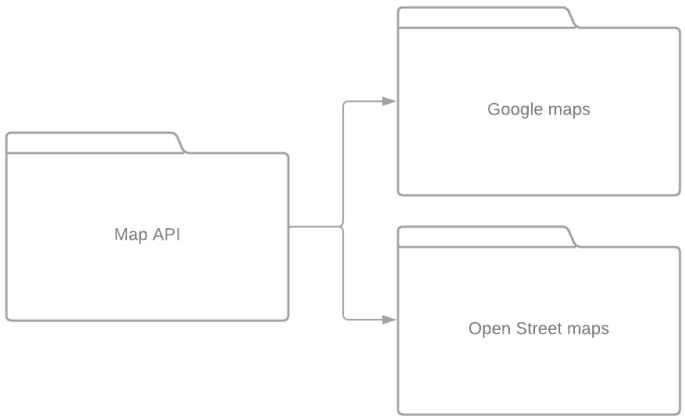
Уведомява аварийните групи при излизане на някой дрон от строя.

bool notifyEmergency(id)

Вход: Идентификационен номер на повредения дрон

Изход: true – успешно изпратен сигнал ; false – неуспех

2. Map API



Модулът има за цел да адаптира информация от няколко сайта за навигация и карти. Негова отговорност е да взима данни от сайтовете и да ги преработва в подходящи и смислени за системата.

3. User Authenticator

Осигурява автентикация на данните на потебителите, като сверява дали подадените от потребителя данни съвпадат с тези от базата данни.

const userReport& (userCredentials)

Вход: Данни на потребителя

Изход: Заявка за базата данни

bool userValidator (const userReport&)

Вход: Заявка за базата данни

Изход: true – успех ; false – неуспех

4. Database 1

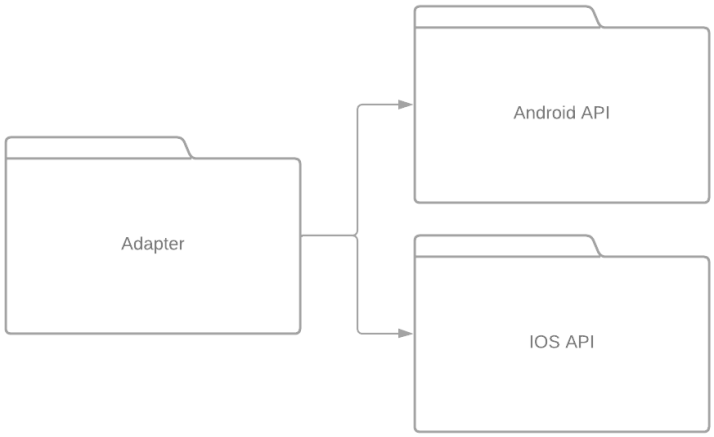
Това е основната база данни, която съдържа информацията за акаунтите на потребителите, техните абонаменти и фишове.

executeQuery(SQL string)

Вход: SQL заявка

Изход: отговор на заявката в JSON формат – успех ; подходящо съобщение – неуспех

5. UI and Adapter



UI включва всички видими части за потребителя. Предоставя на потребителя лесен начин за използване на функционалностите на системата.

Adapter има за цел да направи платформата съвместима с основните операционни системи. Модулът е свързан с Android API и IOS API. Те спомагат данните да се представят във формат на съответната среда.

6. Server

Модулът Server представлява ядрото на софтуерната архитектура. Неговата задача е да осигурява функционирането на цялата система. Всеки един от модулите в системата е свързан със сървъра. Преди да се премине към тях, сървърът трябва да обработва правилно потребителските заявки и да ги пренасочва към подходящите модули за тяхното изпълнение. Самият Server съдържа два подмодула, които са от съществено значение за неговата пълноценна и безпроблемна работа.

6.1. Garbage Collector

Този модул има важната задача да се грижи за пълноценното използване на паметта на системата. В случай, че има заделена памет, която вече не е нужна и не се използва, garbage collector-а се грижи за нейното освобождаване. Това включва и премахване на данните, след като периодът, в който трябва да се съхраняват изтече.

unsigned * findAllNotUsedMemory();

Вход: Няма

Изход: Масив от ненужно заетата памет

bool releaseAllNotUsedMemory(unsigned *);

Вход: Масив от ненужно заетата памет

Изход: true – успех ; false - неуспех

6.2. Process killer

Модулът има за цел да прекрати работата на неотговарящи (зациклили) или неактивни за дълъг период от време процеси в системата. По този начин намаляваме натоварването върху сървъра и увеличаваме неговата производителност.

unsigned* findInactiveProcesses(unsigned);

Вход: Максимален период на неактивност

Изход: Масив от process id на всички неактивни за този период процеси

bool killInactiveProcesses(unsigned *);

Вход: Масив от process id на всички неактивни за този период процеси

Изход: true – успех ; false - неуспех

7. Server backup

Модулът представлява резерва на главния сървър. В случай на авария, всички потребителски заявки трябва веднага да бъдат пренасочени към резервния сървър, за да може работата да продължи безпроблемно до локализирането и отстраняването на проблема. Освен това, резервният сървър може да се използва и в случай на прекалено голямо натоварване, за да поеме част от заявките. За тази цел, той трябва по всяко време да бъде в съответствие с главния.

8. Recognition Manager

Модулът има за цел да осъществи функционалностите свързани с разпознаването на свободни паркоместа и дроновете. Той е съставен от 4 подмодула.

8.1. Count of Drones and Route of Drones

Модулите се грижат за определянето на броя и маршрута на летящите дронове във всеки един момент. Това става на базата на информацията за честотата на заемане и освобождаване на местата в съответните зони, както и на информацията за активни абонаменти на потребители. Освен това броя и маршрута на дроновете зависи и от метеорологичните условия, информация за които получаваме от Weather API. При трайно намалена видимост, която води до невъзможност да се заснемат паркоместата, броят и маршрута на дроновете е празна стойност и се известява оператора на системата. Модулите използват пряко данните от Caching data.

unsigned countOfDrones(parkSpotInfo,weatherInfo);

Вход: Данни за паркоместата, данни за метеорологичните условия

Изход: Брой дронове

route getDroneRoute(parkSpotInfo,weatherInfo);

Вход: Данни за паркоместата, данни за метеорологичните услови

Изход: Маршрут на дронове

8.2. Image Processing

Модулът има за цел да обработи заснетите от дроновете снимки. Той пряко използва Space recognition API, което представлява външна услуга, която на база заснетите снимки и данни за активните абонаменти на потребители, определя свободните места. Като данните се обновяват на определен период от време (минимум 1 минута), предварително зададен от оператора на системата. Отговорност на API-то е да обработва подадените данни, и да връща подходяща информация, която да бъде смислена за системата.

8.3. Caching data

Модулът работи пряко с база данни и има за цел да кешира информацията получена от нея за определен период (в който тя се счита за актуална). По този начин при заявки, данните ще се взимат от кеша и това значително би увеличило бързодействието.

bool addToCache(parkSpotInfo)

Вход: Данни за паркоместата

Изход: true – успех ; false - неуспех

parkSpotInfo getFromCache()

Вход: няма

Изход: Данни за паркоместата

9. Database 2

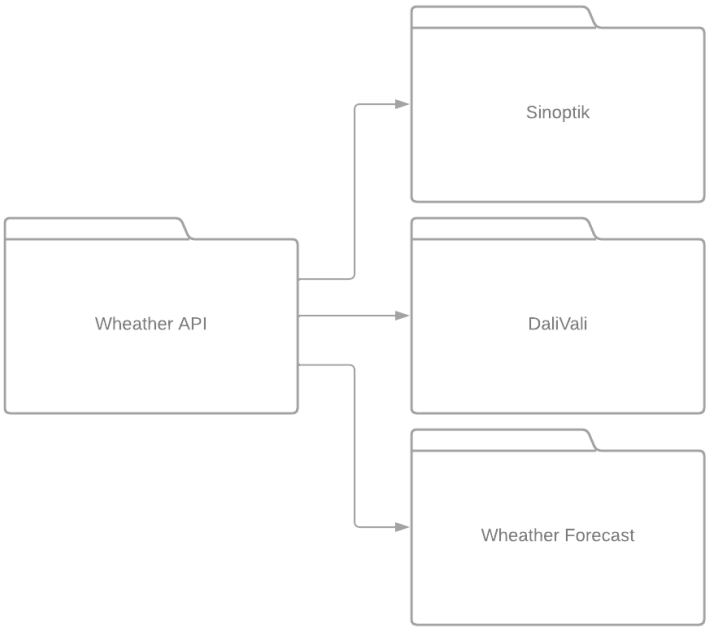
База данни, в която се съхранява информация за честотата на заемане и освобождаване на паркоместата и заснетите изображения, като се взимат в предвид и активните абонаменти за потребителите. Причината ,поради която е отделена от основната база данни е ,че информацията тук е различна от тази за потребителите и така има по-добро структуриране и по-лесно достъпване.

executeQuery(SQL string)

Вход: SQL заявка

Изход: отговор на заявката в JSON формат – успех ; подходящо съобщение – неуспех

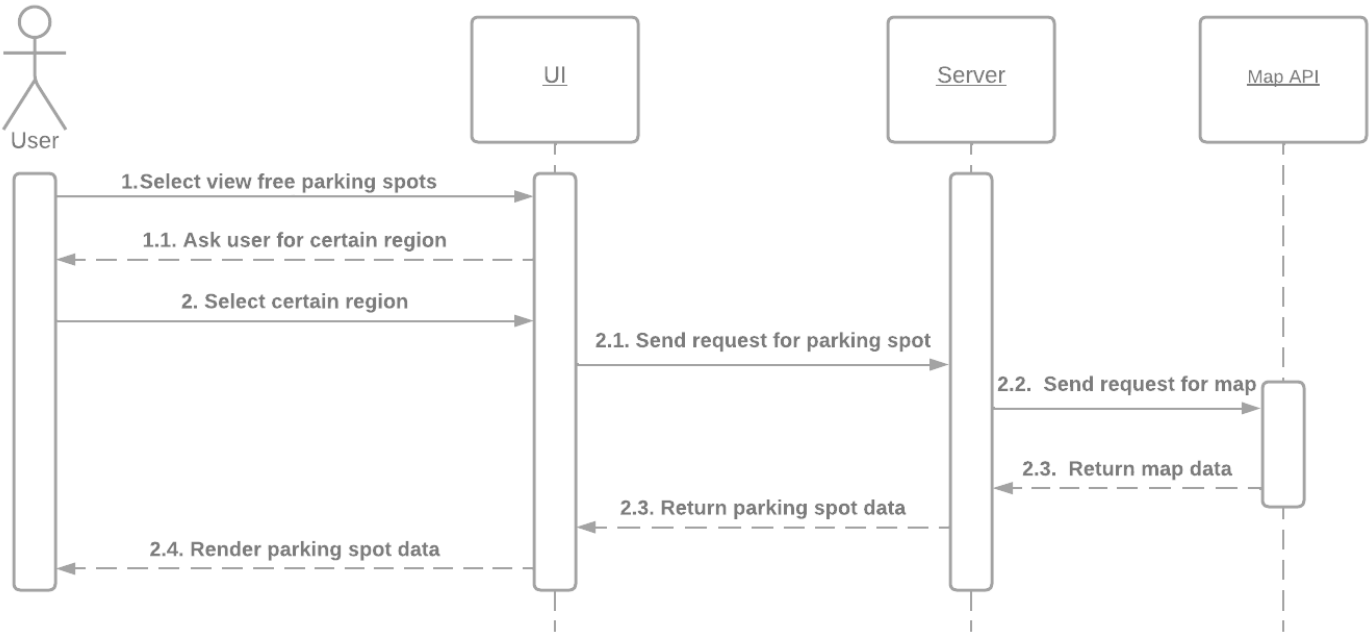
10. Whether API



Модулът има за цел да адаптира информация от няколко сайта за прогнози за времето. Негова отговорност е да взима данни от сайтовете и да ги преработва в подходящи и смислени за системата.

III. Описание на допълнителните структури

1. Структура на процесите
   1.1. Процес на разглеждане на свободните паркоместа
      1.1.1. Първично представяне



      1.1.2. Описание на елементите и връзките

User е потребителят, който иска да разглежда свободните места през мобилното приложение.

UI е интерфейсът на мобилното приложение, през който потребителят достъпва функционалностите на приложението.

Server съдържа модулите Acount Manager и Recognition Manager.

Схемата започва с избор от потребителя на функционалността за разглеждане на свободните паркоместа. Тогава интерфейсът пита потребителя за конкретен регион, за който да се покажат свободните паркоместа. Когато потребителят направи избора си, интерфейсът изпраща заявка към сървъра за тях. Сървърът извлича информацията от модула Recognition Manager, както и изпраща запитване към Map API за актуална карта. След като сървърът получи нужната информацията, той връща на интерфейсът карта за свободните паркоместа в региона, а след това интерфейсът я изобразява на потребителя.

1.1.3. Описание на обкръжението

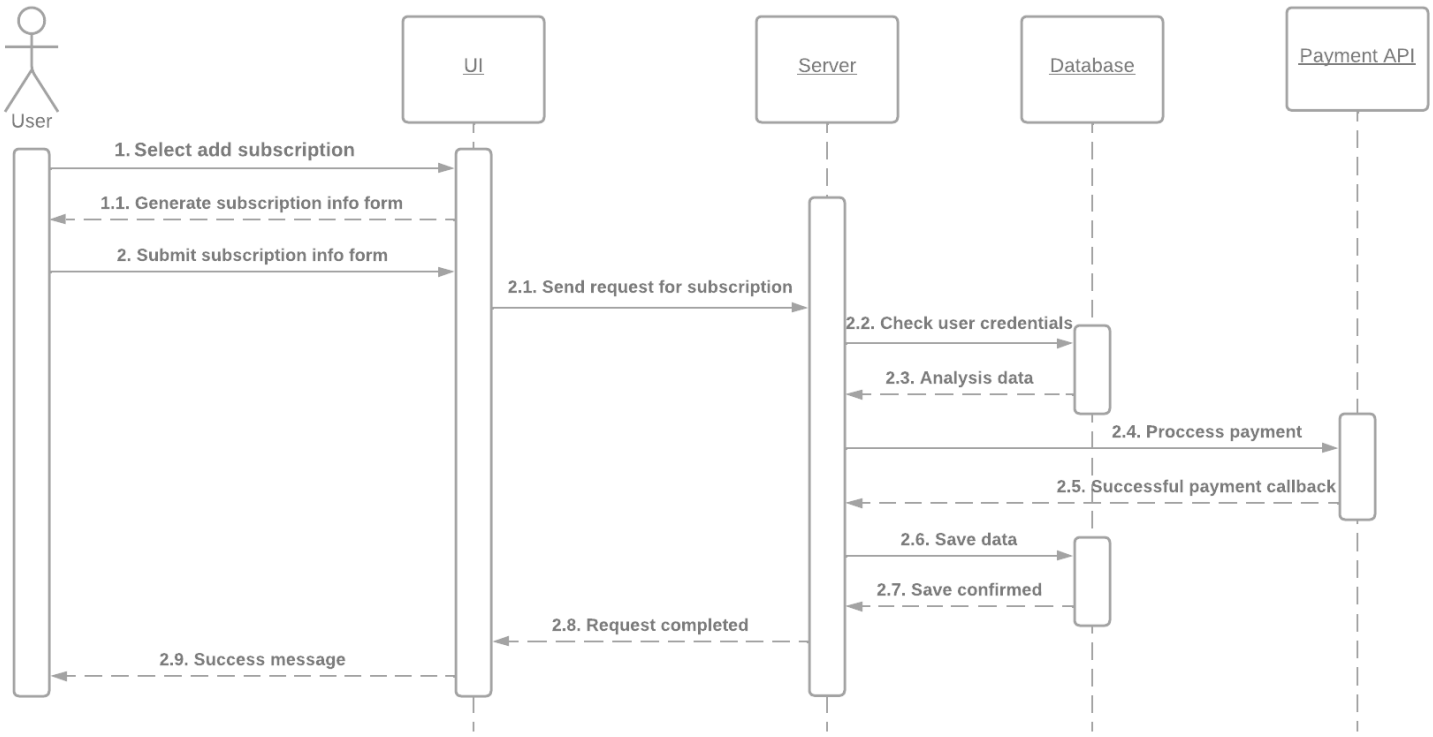
Map API е wrapper, който предоставя използването на различни видове карти.

1.1.4. Описание на възможните вариации

Показаният **Server** може да бъде разделен на няколко инстанции: General Server, Acount Manager и Recognition Manager.

1.2. Процес на абониране за паркомясто

1.2.1. Първично представяне



1.2.2. Описание на елементите и връзките

User е потребителят, който иска да закупи абонамент за паркомясто през мобилното приложение.

UI е интерфейсът на мобилното приложение, през който потребителят достъпва функционалностите на приложението.

Database е представител на модула Database 1, в който се съдържа информацията за потребителите.

Ако потребителят избере да използва функционалността за абониране за паркомясто, интерфейсът ще му изпрати форма за попълване на информация за желаното паркомясто, лични данни и начин на плащане. След като потребителя изпрати формата, интерфейсът праща към сървъра питане за потвърждаване на абонамента. Тогава сървърът извършва проверка с информацията от базата данни на наличността на желаното паркомясто, както и на данните на потребителя. Ако проверките за успешни, то сървърът изпраща плащането към Payment API. При потвърдена транзакция, сървърът запазва данните за абонамента в Database. Когато сървърът получи потвърждение за записа, изпраща към интерфейса, че абонамента е успешен, а интерфейса също изпраща съобщение към потребителя за успешността.

1.2.3. Описание на обкръжението

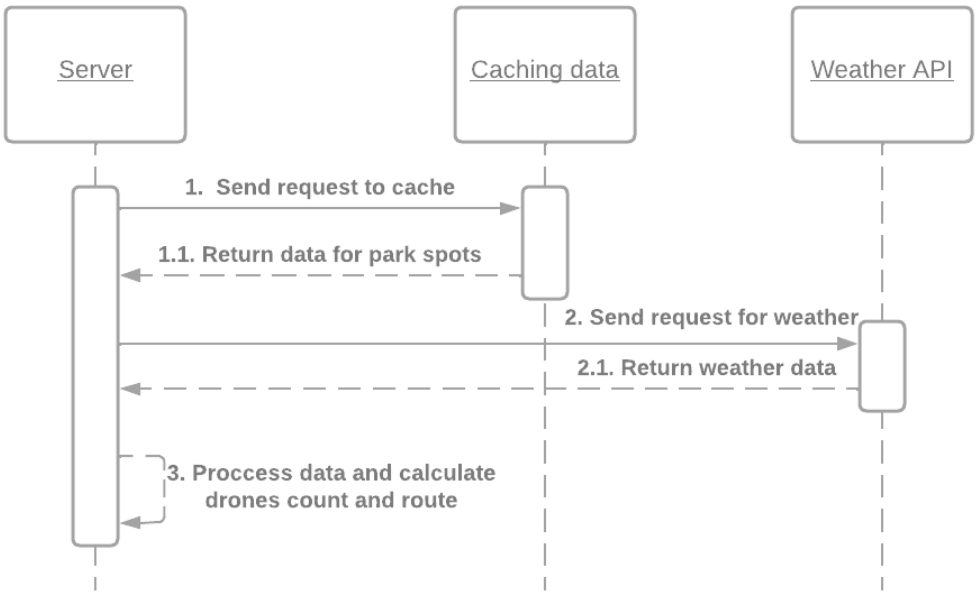
Payment API е wrapper, който авторизира и подсигурява плащането.

1.2.4. Описание на възможните вариации

Показаният **Server** може да бъде разделен на няколко инстанции: General Server, Acount Manager и Recognition Manager.

1.3. Процес на изчисляване на броят и маршрутът на дроновете

1.3.1. Първично представяне



1.3.2. Описание на елементите и връзките

Caching data е модулът, в който се съхранява кешираната информация за паркоместата.

Броят и маршрутът на дроновете се определя периодично от сървъра. Той изпраща заявка за извличане на данни за паркоместата към кеша. При успешно обработване на заявката, информацията се връща към сървъра. След това, изпраща заявка за извличане

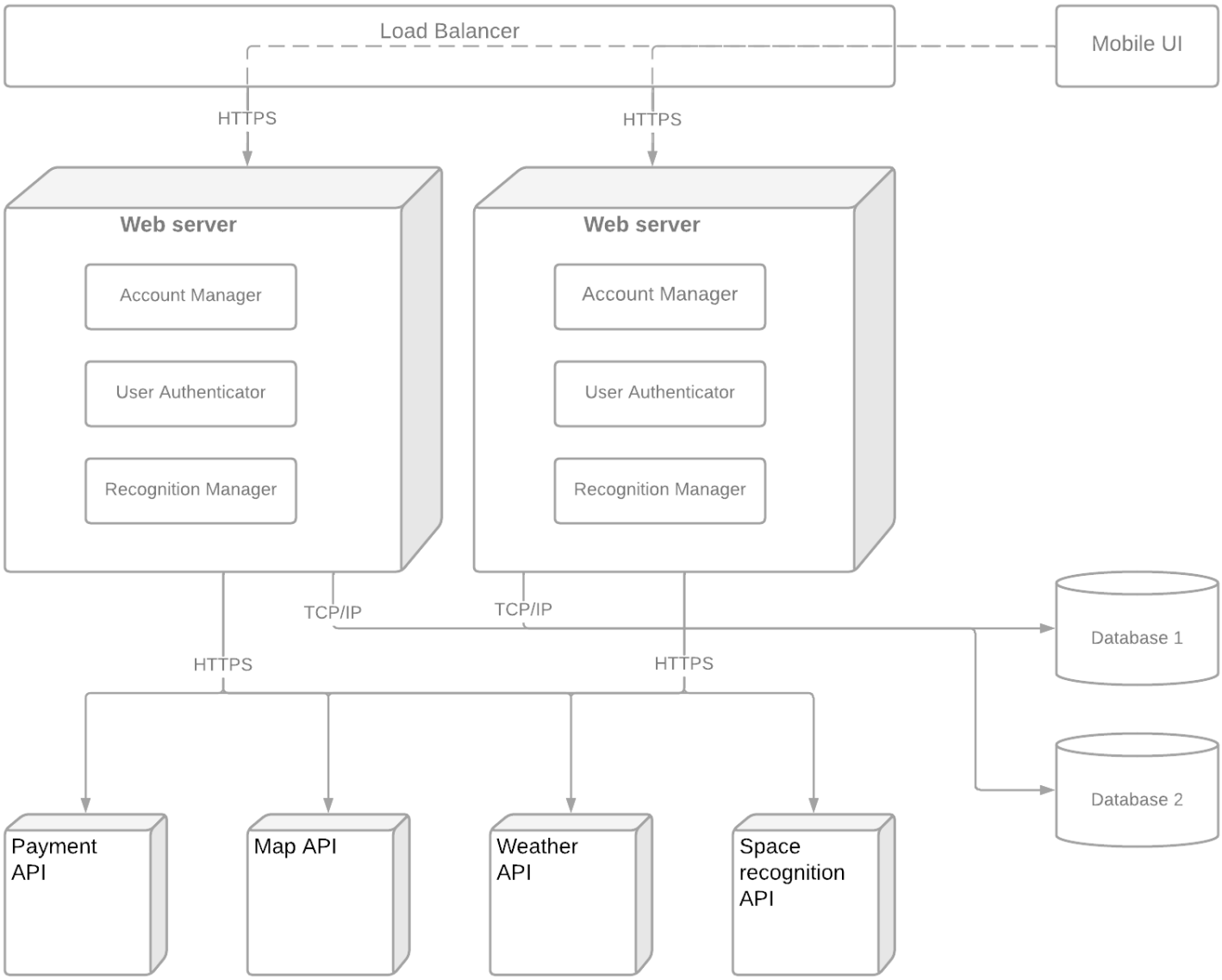
на данни за метеорологичните условия. При успех данните се връщат към сървъра. На база събраната информация, сървърът изчислява необходимия брой дронове, както и маршрута на всеки един от тях.

1.3.3. Описание на обкръжението

Weather API е wrapper, който предоставя използването на различни сайтове за прогноза за времето.

2. Структура на внедряването

2.1. Първично представяне



2.2. Описание на елементите и връзките

Системата ще бъде налична за мобилни устройства. Посредством UI-а потребителите ще могат лесно да се възползват от всички налични функционалности. Комуникацията между отделните подсистеми ще се осъществи посредством API calls. Ще се използва HTTPS протокола за достъпването и модифицирането на ресурси. За транспортен протокол ще използваме TCP, като по този начин ще си гарантираме по голяма сигурност при предаването на sensitive данни.

Първоначално ще имаме две копия на системата на различни сървъри, като в зависимост от моментното натоварване, load balancer-а ще определя към кое от тях да насочват заявките, така че да се увеличи ефективността и да се намали забавянето.

Външните системи съответно ще се намират на различни сървъри, като когато е необходимо посредством API-тата ние ще извличаме необходимата информация. Базите данни също ще са на отделни сървъри, като това увеличи сигурността на данните в случай, че системата бъде атакувана от злонамерни лица. Тъй като е възможно да се получи bottleneck, за да го избегнем ще използваме horizontal scaling.

2.3. Описание на обкръжението

Payment API е wrapper, който авторизира и подсигурява плащането.

Map API е wrapper, който предоставя използването на различни видове карти.

Weather API е wrapper, който предоставя използването на различни сайтове за прогноза за времето.

Space recognition API е wrapper към външна услуга, която използва специфичен алгоритъм за определяне на свободните места.

2.4. Описание на възможните вариации

Възможно е при прекалено голямо натоварване да се добавят нови scale-ове, които да облекчат натоварването, като поемат част от трафика.

Начин на реализация и технологии:

Отделните компоненти на приложението ще бъдат реализирани като микросървиси. Всеки един от тях ще бъде обособен за конкретна задача, като отделните микросървиси ще са независими един от друг. Това ни дава редица ползи:

- Отделните микросървиси могат да бъдат реализирани от различни екипи, използвайки различни програмни езици и технологии, без това да оказва влияние върху останалите.
- При проблем в някой от компонентите, това по никакъв начин няма да окаже влияние върху останалата част от системата, което би удовлетворило изискването за наличност.
- При промяна в някой от тях, това ще доведе до билдване и внедряване само на конкретния микросървис, а не на цялата система, което значително би увеличило бързодействието.
- При необходимост от скалиране, то може да се осъществи само върху конкретен компонент, вместо върху цялата система, което би увеличило гъвкавостта и спестило много ресурси.

Микросървисите от своя страна ще бъдат реализирани като контейнери. Ще използваме технология за създаване и работа с контейнери (като Docker). Те ще ни послужат, за да пакетираме отделните компоненти заедно с всички библиотеки и дипендънсита, които са им необходими за да работят. Така ще осигурим изолираност и ще избегнем евентуални конфликти свързани с използването на различни версии на едни и същи библиотеки от различни компоненти. Освен това ще улесним много процеса на преместване от една система на друга, при необходимост.

Отделните контейнери ще бъдат менажирани посредством технология за организиране и управляване на контейнери (като Kubernetes). Ще разпределим отделните контейнери върху отделни подове. Обикновено подовете съдържат един контейнер (представляват обвивка на контейнера).При нужда може да съдържа множество контейнери, които трябва да работят заедно. Подовете сами се грижат за контейнерите в тях, тоест ако някой от тях спре работа, той автоматично ще го рестартира. Kubernetes използва т.нар. health check, който периодично проверява състоянието на подовете и в случай, че някой от тях е unhealthy, то се създава нов под, който да го замени. По този начин ние значително ще увеличим надеждността и отказоустойчивостта на системата. Освен това, се поддържа и load balancer, който в зависимост от натоварването на даден под, може да създаде нов, който да послужи като отдушник и да поеме част от заявките, което би увеличило ефективността.

IV. Архитектурна обосновка

- Броят на летящите в даден момент дронове и маршрутът на всеки от тях се определя динамично, на базата на предвиждане, за честотата на заемане/освобождаване на места в съответните зони. Това предвиждане зависи от натрупаните данни за динамиката на паркиране в съответния ден и час от седмицата и метеорологичните условия.

За изпълнението на това изискване са интегрирани функционалности Count of drones и Route of drones в модула Recognition Manager. Също така модулът е свързан Weather API и Database 2, които предоставят информация за динамиката на паркиране и метрологичните условия. (Декомпозиция на модулите – 8.1.)

- За определяне на метеорологичните условия да се ползва външна услуга за прогноза за времето.

Външната услуга е Weather API, която е свързана с модула Recondition Manager. От своя страна външната услуга използва сайтовете за метеорологично време Sinoptik, DaliVali и Whether Forecast. (Декомпозиция на модулите – 10.)

- Системата използва специфичен алгоритъм и външна услуга за разпознаване на свободните места, на база на заснетите изображения.

В нашата архитектура специфичният алгоритъм се имплементира във функционалността Image processing на модула Recognition Manager. Този модул е свързан с външната услуга Space recognition API, която разпознава свободните места. (Декомпозиция на модулите – 8.2.)

- Ако някой дрон излезе от строя, незабавно трябва да се уведомят аварийните групи, които да получат информация за предполагаемия район, в който се намира дрона и да отстранят повредата.

В случай, че дрон излезе от строя, аварийните групи се уведомяват чрез функционалността Notify emergency на подмодула Functionalities на Account Manager. (Декомпозиция на модулите – 1.4.5.)

- Информацията за свободните места се обновява на определен интервал от време, който се задава от оператора на системата и може да е най-малко 1 минута.

Подмодулът Image Processing на Recognition Manager се грижи за обработването на снимките. Това се случва на интервал от време, зададен от оператора на системата. (Декомпозиция на модулите – 8.2.)

- При трайно намалена видимост (напр. мъгла, пушек и др.), която води до невъзможност да се заснемат паркоместата, да се вземат мерки за известяване на оператора на системата.

При трайно намалена видимост оператора се уведомява през функционалностите Count of drones и Route of drones в модула Recognition Manager. (Декомпозиция на модулите – 8.1.)

- Регистрираните потребители могат да заплащат абонамент за определено паркомясто, което се маркира като заето в рамките на периода на абонамента, независимо дали заснетите от дроновете изображения, показват наличието на автомобил на него или не. За целта трябва да се поддържа карта на наличните места.

Потребителя може да избере място, за което да се абонира през подмодула Subscription Manager на Account Manager. След това потребителя се изпраща към подмодула Payment Manager на Account Manager, където може да извърши плащането на абонамента си. (Декомпозиция на модулите – 1.2., 1.3., 1.4.1., 2.)

- Ако няма абонамент, свободните места за парикране може да са безплатни или да се таксуват динамично, като цената се определя според предвиждане за честотата на заемане/освобождаване в съответния ден/час, както и от прогнозата за времето.

Регистрираните и обикновените потребители имат достъп до функционалността View free park spots от подмодула Functionalities на Account Manager. Тя им дава възможност да разглеждат паркоместата. Модулът е свързан с Map API, което предоставя карта на районите и е свързано със сайтовете за сателитни карти Google maps и Open Street maps. Оттам потребителя може да избере място, за което иска да плати. При този случай потребителя се изпраща към подмодула Payment Manager на Account Manager, където може да извърши плащането. (Декомпозиция на модулите – 1.2., 1.4.1.)

- Плащането може да се извършва чрез дебитна/кредитна карта, PayPal или СМС, като в бъдеще може да се добавят и други начини на разплащане.

Плащането на абонаменти и на паркоместа в нашата система се извършва чрез метода Payment на Payment Manager, който от своя страна е подмодул на Account Manager. Този подмодул е свързан с Payment System, през Payment API. Чрез тях се авторизират и подсигуряват плащането. Функционалността Payment е такава, че приема различни начини на плащане, т.е. въвеждането на нов начин на плащането ще бъде лесно и няма да изисква добавянето на нова функционалност. (Декомпозиция на модулите – 1.2.)

- Обикновените потребители, регистрираните потребители, аварийните и групите по контрол на паркирането използват системата през мобилно приложение.

Архитектурата ни предоставя приложение. Модулът Account Manager се грижи за функционалностите на приложението от различните потребители. То включва

базови функционалности в подмодула Basic Functionalities, основните за системата функционалности в подмодула Functionalities, както и тези за абониране и плащане в Subscription Manager и Payment Manager. Функционалностите са имплементирани по начин, който позволява да се използват само от потребителите, за който са предназначени. Системата поддържа UI – потребителския интерфейс, който представлява всички видими части за потребителя. Account Manager се свързва с UI през Adapter, който направи платформата съвместима с основните операционни системи. (Декомпозиция на модулите – 1., 5.)

- Обикновените потребители може да заемат само свободните места, за които няма абонамент.

Във функционалността View free parking spots се изобразяват свободните паркоместа. При нея се прави проверка: при абониране е позволено да се избере всяко паркомясто, а при динамично таксуване – само тези, който са свободни. (Декомпозиция на модулите – 1.4.1.)

- Останалите потребители трябва да имат 100% защитен от външна намеса достъп до системата.

Всички функционалности и плащания на системата минават през модула User Authenticator, който осигурява автентикация на данните на потебителите. (Декомпозиция на модулите – 3.)

- Групите по контрол на паркирането следят дали няма нарушители (неплатили или заели място за което нямат абонамент). При засичане на нарушител, освен принудителното преместване на автомобила, заснемат настъпилото събитие, като снимката се съхранява директно в системата и след това се издава електронен фиш за глоба. Снимката и фишът трябва да са достъпни и през публичен сайт, който се зарежда чрез уеб-браузър, но само за виновното лице.

Групите по контрол могат да заснемат настъпилото събитие и да издадат електронен фиш чрез функционалността Register intruder на подмодула Functionalities на Account Manager. От страна на виновното лице, той може да достъпва своите фишове през метода View slip от същия подмодул. Този метод го препраща към външен публичен сайт. (Декомпозиция на модулите – 1.4.3., 1.4.4.)

- Регистрираните потребители може да подават сигнал до групите по контрол на паркирането за неправомерно заето от друг място, за което са абонирани.

Чрез метода Send signal на подмодула Functionalities на Account Manager, регистрираните потребители могат са изпращат сигнали за неправомерно заето от друг място, за което са абонирани. (Декомпозиция на модулите – 1.4.2.)

- Системата да работи 100% без отказ в рамките на светлата част на работния ден (9:00 до 17:00 зимно време и 8:00 – 19:00 лятно време).

В системата са предвидени няколко метода и модули, който да увеличат отказоустойчивостта. Подмодулът Process killer на модула Server прекратява работата на зациклили или неактивни за дълъг период от време процеси, като така намаляваме натоварването върху сървъра и увеличаваме неговата производителност. Модулът Server backup е резерва на главния сървър и се употребява в случай на авария или на претоварване на главния сървър. В модула

Recognition Manager има подмодул Caching data, който е свързан с базата данни и кешира информацията получена от нея за определен период, като така се увеличава бързодействието при заявки. Освен това чрез структурата на внедряването се добива още по-ясна представа за начинът на постигане на висока отказоустойчивост (Декомпозиция на модулите – 6.2., 7., 8.3.)

- Системата да поддържа архив на данните за динамиката на паркирането и всички издадени фишове за глоби за 50 години назад във времето, както и архив на заснетите изображения за 5 години назад.

Архитектурата ни предоставя две бази данни. Database 1 съдържа информацията за акаунтите на потребителите, включително и информация за техните фишове и изображенията на събитията. Database 2 съхранява данните за динамиката на паркирането, както и заснетите от дроновете изображения. Чрез две отделни бази данни има по-добро структуриране и по-лесно достъпване. За освобождаването на ненужните и остарели данни се грижи подмодула Garbage Collector на модула Server. (Декомпозиция на модулите – 4., 6.1., 9.)