CAPC

DronesPark

Изготвено от:

Име: Божидар Навчев

Име: Борислав Христов

Въведение

Организация на текущия документ

Предназначение на документа

В този документ е представена архитектурата на системата за следене и управление на паркоместа **DronesPark**

Пояснене на използваните структури

- **Декомпозицията на модулите:**

    - Показва с високо ниво на абстракция приожението и неговите отделни компоненти под формата на модули и подмодули. Основните модули са: **User manager, Drone manager, Database, Database collection, Drone, Payment API, Penalty, Technical team, Web app** и **Mobile app.** Всеки модул притежава **Communication API,** който служи за камуникация между отделните модули. **Drone** е комбинацията от хардуер и софтуер на дрона. **Web app** и **Mobile app** са потребителски приложения. Модула **Technical team** отговаря за техническия екип. Имаме модул **Penalty,** който служи за създаването на електронни фишове и комуникация с групите по контрол (т.нар ,,паяци'').
- **Структура на внедряването:**

    - Диаграмата показва как отделните модули ще се рапределят върху различните софтуерни/хардуерни системи. Структурата на внедряването също показва взетите архитектурни решения - използването на два или повече сървъра за отделните компоненти и комуникацията между тях.
- **Структура на процесите:**

    - Този стил структура показва процеса на използване на приложението, от гледна точка на потребителите. Трябва да се обърне внимание, че структурата е представена с по-високо ниво на абстракция с цел по-лесно разбиране. Насочена е главно към потребители и други заинтересовани лица без технологични познанния.

Структура на документа

- **Въведение (секция 1)**
- **Декомпозиция на модулите (секция 2)**
    - Общ вид на декомпозицията на модулите
    - Подробно описание

    - Описание на възможните вариации
- **Структура на внедряването (секция 3)**
    - Първично представяне
    - Описание на елементите и връзките
    - Описание на обкръжението
    - Описание на възможните вариации
- **Структура на процесите (секция 4)**
    - Първично представяне
    - Описание на елементите и връзките
    - Описание на обкръжението
    - Описание на възможните вариации
- **Архитектурна обосновка (секция 5)**

1.ОБЩИ СВЕДЕНИЯ ЗА СИСТЕМАТА

Последните години показват, че намирането на паркоместа в големите градове се превръща в сериозно предизвикателство. Системата DronesPark предлага различен подход към този проблем. Тя предоставя възможност за наблюдение на всички паркоместа в реално време, с помощта на автономни дронове. Благодарение на лесния достъп до системата, потребителите ще могат да проверят за наличие на свободни места за паркиране около желаната им дестинация още преди да са пристигнали там. Също така регистрираните потребители ще могат да се абонират за свободно място, премахвайки проблемите, свързани с намирането му.

Разширен терминологичен речник

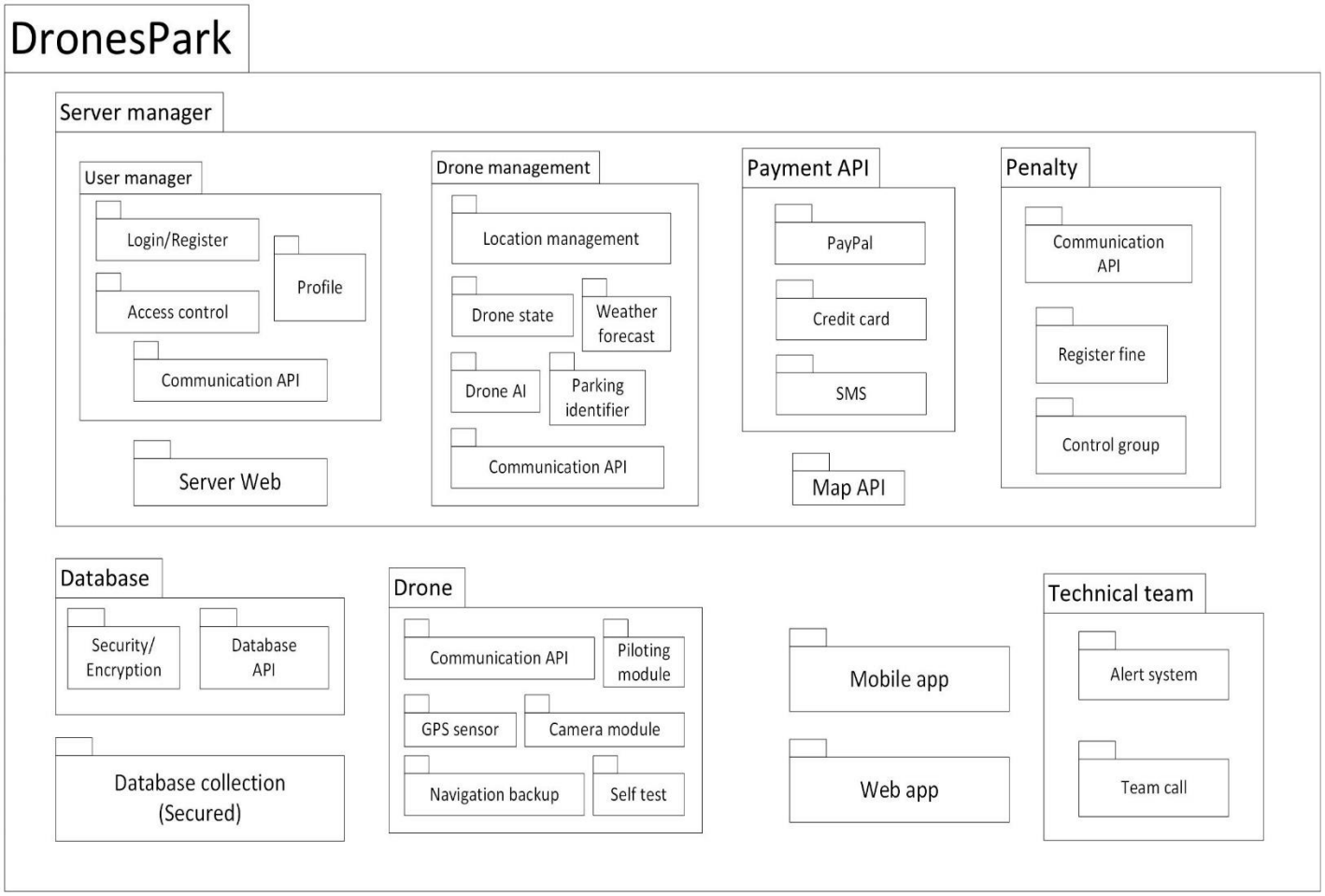
Списък на софтуерните елементи

- Server manager
- User manager
    - Login/Register

    - Access control
    - Profile
    - Communication API
- Server Web
- Drone manager
    - Location management
    - Drone state
    - Weather forecast
    - Drone AI.
    - Parking identifier
    - Communication API
- Database
    - Security/Encryption
    - Database API
- Database collection
- Mobile app
- Web app
- Map API
- Drone
    - Communication API
    - Self-test
    - Piloting module
    - GPS sensor
    - Camera module
- Payment API
    - PayPal
    - Credit card
    - SMS
- Penalty
    - Communication API
    - Register fine
    - Control group
- Technical team
    - Alert system
    - Team call

Декомпозиция на модулите

Общ вид на декомпозицията на модули за системата



User manager се грижи за достъпа на всичките потребители. **Access control** е подмодул, който определя достъпа до определени функционалности, на всеки потребител. Информацията се предава към **Communication API.**

Комуникацията между отделните части на системата се извършва чрез съответните подмодули **Communication API.** Те имат дефинирани входно-изходни интерфейси.

Database е директно свързан с **Database collection**. **Database collection** е изграден на локална мрежа и няма външен достъп освен през **Database API**. Останалите модули правят заявки към **Database API**, когато се нуждаят от информация от базата данни.

Payment API е свързан с **външни системи за плащане** и с **Mobile app**. Позволява използването на един и същ интерфейс за плащане независимо от коя външна система се използва, така добавянето на нов метод за плащане се случва лесно и бързо. Различните подмодули отговарят за конкретна услуга на плащане.

Map API също е специфичен модул. Той предоставя общ интерфейс за работа с различни карти. Характеризира се с комуникация с модулите **Drone** и **Drone management.**

Penalty комуникира с **Web app** и **Mobile app.** Благодарение на **Control group** подмодул, може да се осъществи връзка с т.нар ,,паяци''. Чрез **Communication API,** подмодулът **Register fine** изпраща данните за изготвени фишове към системата.

Technical team комуникира с **Drone management** и **Web app.** При установена повреда, подмодулът **Alert system** получава информацията, определя естеството на проблема и при необходимост съобщава за техническа неизправност на **Team call.** Подмодулът **Team call** осъществява комуникация със специализирани технически екипи, които се грижат за дроновете. Съответно ако има проблем в системата, информирани ще бъдат администраторите.

Подмодулите на **Drone management** - **Location management, Drone state** и **Weather forecast, Parking identifier,** са копоненти, съдържащи необходимата информация за сформиране на алгоритмите на **Drone AI**. Резултатите от тези алгоритми се изпращат към модулите **Drone** и **Technical team**. Подмодулът **Parking identifier** получава информация от подмодула **Drone** под формата на снимки, които използва за определянето на дадено

паркинг място като свободно или заето. Тази информация се изпраща към модула **User manager**.

Drone комуникира с **Map API** и с **Drone management. Drone** се характеризира с интензивна комуникация с подмодулите си. **Piloting module** събира данните от останалите подмодули, необходими за управление на дрона. Чрез **Communication API**, с периодични съобщения до **Drone management** се следи и управлява всеки един дрон. Ако подмодула **Self test** установи неизправност, той веднага изпраща данните заедно с координатите от **GPS sensor** до **Drone management** (Отново с помощта на **Communication API**). При неизправност на **GPS sensor** сме имплементирари резервен сензор за координатите на дрона, който се активира при дадената повреда. С тази отговорност се заема подмодула **Navigation backup**. **Camera module** се грижи за заснемането на изображенията, тяхното форматиране и изпращане към **Communication API .**

Подмодулът **Server web**, осъществява комуникация между **Server Manager** и останалите вътрешни и външни модули на системата.

2.Подробно описание на всеки модул:

Server manager

Предназначение:

- Осигуряване на единна среда за разположение на основните системни модули.
- Улеснява комуникацията между модулите.
- Осигурява арбитраж на информационния поток.

Отговорности:

- Защита на сървърната среда.

- Контролиране и наблюдаване на комуникацията с останалите модули.

User manager

Предназначение:

- Контролиране на достъпа на всички типове потребители.

Отговорности:

- Предпазване на системата от неоторозириан достъп.
- Защита на личните данни от злонамерено ползване.
- Обслужва клиентите, както и техните изисквания свързани с профилите им.

Интерфейси:

- **User manager**
  - Register(User)
    - Входни данни: Лични данни на нов потребител
    - Резултат: Регистриране на нов потребител
    - Грешки и изключения:
      - При невалидна заявка се връща подходящо съобщение
    - Зависимости от други елементи:
      - Заявката се изпраща към **Database** след верифициране
  - Login(User)
    - Входни данни: Име и парола на потребител
    - Изходни данни: Уникален токен за потребителя, с който се счита за успешно влязал в системата
    - Грешки и изключения:
      - При невалидна заявка се връща подходящо съобщение
    - Зависимости от други елементи:
      - Заявката се изпраща към **Database** след верифициране
  - ViewProfile(user)
    - Входни данни: Идентификация на потребител
    - Изходни данни: Информацията за дадения потребител в системата

        - Потребителя задължително се верифицира преди изпълняване на заявката
        - Грешки и изключения
            - При невалидна заявка се връща подходящо съобщение
        - Зависимости от други елементи:
            - Заявката се изпраща към **Database** след верифициране
    - UpdateProfile(User, New_Information)
        - Входни данни: Идентификация на потребител и обновена информация
        - Резултат: Личната информация на потребителя в системата е обновена
        - Потребителя задължително се верифицира преди изпълняване на заявката
        - Грешки и изключения:
            - При невалидна заявка се връща подходящо съобщение
        - Зависимости от други елементи:
            - Заявката се изпраща към **Database** след верифициране
    - ReportComplaint(User, Car_lot, Complaint_Information)
        - Входни данни: Идентификация на потребител, абонираното от него паркинг място и допълнителна налична информация
        - Резултат: Изпращане на съобщения за инцидента към **Control group**
        - Зависимости от други елементи:
            - Заявката се изпраща към **Penalty**

Database

Предназначение:

- Установява комуникация с базата данни посредством външен интерфейс.
- Криптира потока на информация с цел по-висока защита.

Отговорности:

- Изолиране на базата данни.
- Упражнява контрол до достъпа на информация.
- Сортира заявките към различните инстанции на базата данни.

Интерфейси

- **Database**

- searchCarlots(Corrdinates)
  - Входни данни: Координати
  - Изходни данни: Най-близките паркиг места по коодринати
  - Грешки и изключения:
    - При невалидна заявка се връща подходящо съобщение
  - Зависимости от други елементи:
    - Заявката се изпраща към **Database Collection** след верифициране
- Register(User)
  - Входни данни: Лични данни на нов потребител
  - Резултат: Регистриране на потребител в системата
  - Грешки и изключения:
    - При невалидна заявка се връща подходящо съобщение
  - Зависимости от други елементи:
    - Заявката се изпраща към **Database Collection** след верифициране
- Login(User)
  - Входни данни: Име и парола на потребител
  - Изходни данни: Уникален токен за потребителя, с който се счита за успешно влязал в системата
  - Грешки и изключения:
    - При невалидна заявка се връща подходящо съобщение
  - Зависимости от други елементи:
    - Заявката се изпраща към **Database Collection** след верифициране
- ViewProfile(User)
  - Входни данни: Токен на потребителя
  - Изходни данни: Информацията за дадения потребител в системата
  - Грешки и изключения:
    - При невалидна заявка се връща подходящо съобщение
  - Зависимости от други елементи:
    - Заявката се изпраща към **Database Collection** след верифициране
- UpdateProfile(User, New_Information)
  - Входни данни: Токен на потребителя и обновена информация
  - Резултат: Личната информация на потребителя в системата е обновена
  - Грешки и изключения:
    - При невалидна заявка се връща подходящо съобщение
  - Зависимости от други елементи:
    - Заявката се изпраща към **Database Collection** след верифициране

- ReportComplaint(User, Car_lot, Complaint_Information)
    - Входни данни: Идентификация на потребител, абонираното от него паркинг място и допълнителна налична информация
    - Резултат: Изпращане на съобщения за инцидента към **Penalty**
    - Зависимости от други елементи:
        - Заявката се изпраща към **Penalty**
- GetCarlotInfo(Car_lot)
    - Входни данни: Идентификационен номер на паркинг място
    - Изходни данни: Информация за състоянието на паркинг мястото (дали е свободно или заето)
    - Грешки и изключения:
        - При невалидна заявка се връща подходящо съобщение
    - Зависимости от други елементи:
        - Заявката се изпраща към **Database Collection** след верифициране
- UpdateCarlotInfo(Car_lot, New_Information)
    - Входни данни: Идентификационен номер на паркинг място и допълнитена информация за него
    - Резултат: Променяне на информацията за даденото паркинг място
    - Грешки и изключения:
        - При невалидна заявка се връща подходящо съобщение
    - Зависимости от други елементи:
        - Заявката се изпраща към **Database Collection** след верифициране

Database collection

Предназначение:

- Съхранява цялата необходима информацията на системата.

Отговорности:

- Изпълняване на заявки за търсене на информация в базата данни.
- Изпълняване на зявки за изтриване на информация в базата данни.

Интерфейси

- **Database**
    - executeQuery(SQL string)
        - Входни данни: SQL заявка
        - Изходни данни: отговорът на заявката под формата на JSON
        - Грешки и изключения:
            - При невалидна заявка се връща подходящо съобщение

Payment API

Предназначение:

- Използва външни системи за заплащане на услуги.
- Предоставя единен интерфейс за работа с външните системи.

Отговорности:

- Осигурява защита на личните данни на потребителя.
- Лесно интегриране на други методи на плащане.

Интерфейси:

- **Payment API**
    - Pay(User, Service)
        - Входни данни: Идентификация на потребител и услуга, която ще използва
        - Резултат: Заплащане на услугата по предпочитан от потребителя начин
        - Потребителя задължително се верифицира преди изпълняване на заявката
        - Грешки и изключения:
            - При невалидна заявка се връща подходящо съобщение
        - Зависимости от други елементи:
            - Заявката се изпраща към **Data Collection** след верифициране
            - Заявката се изпраща към правилната външна система за заплащане

Web app

Предназначение:

- Предоставя на крайния потребител справка на издадените му електронни фишове.
- Наблюдаване на системата.
- Администране на системата от администратори.

Отговорности:

- Осигурява защита на личните данни на потребителя.
- Осигурява интерфейс, който помага на потребителя при изготвяне на неговата справка, както и нейното представяне.

Интерфейси

- **Web GUI**
    - Графичен потребителски интерфейс, който позволява на администраторите използват всички функционалности на системата.
    - Графичен потребителски интерфейс, който позволява на потребителите да правят справка за техните електронни фишове.

Mobile app

Предназначение:

- Управление на потребителския профил.
- Преглед/Търсене на налични свободни паркоместа и абонаменти
- Създаване на абонамент
- Заплащане на услуги

Отговорности:

- Правилно свързване и комуникация между отделните модули, за да се позволи използването им чрез приложението.

Интерфейси

- Mobile GUI
    - Графичен потребителски интерфейс, който позволява на обикновените потребители да търсят свободни парко места, заплащат за такива и да управляват потребителския си профил.

Drone management

Предназначение:

- Обработване на състоянието на всички дронове.
- При нередност изпращане на съобщение до техническия екип.
- Използване на цялата налична информация(снимки, прогноза за време, налични трафик часове), за да се състави качествен и надежден алгоритъм, който drone AI ще ползва.
- Осигурен мониторинг и от оператора.

Отговорности:

- Мониторинг на показателите на всички налични дронове в реално време.
- Ръководител на полетите.
- Свързване на всички подмодули.
- Осигуряване на лесна комуникация между подмодулите.
- Осигуряване на контролирана комуникация с базата данни.

Интерфейси

- **DroneManagement-API**
    - searchCarlots(Corrdinates)
        - Входни данни: Координати
        - Изходни данни: Най-близките паркиг места по коодринати
        - Грешки и изключения:
            - При невалидна заявка се връща подходящо съобщение
        - Зависимости от други елементи:
            - Заявката се изпраща към **Database Collection** след верифициране
    - GetCarlotInfo(Administrator, Car_lot)
        - Входни данни: Идентификационен номер на паркинг място и Администратор
        - Изходни данни: Информация за състоянието на паркинг място
        - Администратора задължително се верифицира преди изпълняване на заявката
        - Грешки и изключения:
            - При невалидна заявка се връща подходящо съобщение

        - Зависимости от други елементи:
            - Заявката се изпраща към **Database Collection** след верифициране
    - UpdateCarlotInfo(Car_lot, New_Information)
        - Входни данни: Идентификационен номер на Администратор, паркинг място и допълнитена информация за него
        - Резултат: Променяне на информацията за даденото паркинг място
        - Администратора задължително се верифицира преди изпълняване на заявката
        - Грешки и изключения:
            - При невалидна заявка се връща подходящо съобщение
        - Зависимости от други елементи:
            - Заявката се изпраща към **Database Collection** след верифициране
    - DroneFailure(Drone, Location)
        - Входни данни: Идентификация на дрона и локацията му към този момент
        - Резултат: Изпращане на съобщения за неизправност в дрона към **Technical Team**
        - Зависимости от други елементи:
            - Заявката се изпраща към **Technical Team**

Drone

Предназначение:

- Обработва информацията подадена от сензорите.
- Пилотаж.
- Самодиагностика.
- Изпращане на периодични съобщения до **Drone management.**

Отогворности:

- Изпращане на съобщение за авария(при наличие на такава) до **Drone management.**
- Изпращане на данни от самодиагностиката и локацията до **Drone management.**
- Заснемане и изпращане на изображенията до **Drone management.**

Интерфейс:

- **Drone**
    - TakePhoto()
        - Изходни данни: Заснето изображение на паркомясто под дрона.
        - Грешки и изключения:
            - При проблем с камерата се изпраща съобщение за неизправност.
        - Зависимости от други елементи:
            - Изображението се изпраща на **Communication API**
    - GetSensorInfo()
        - Изходни данни: Актуална информация от всички сензори на дрона
        - Грешки и изключения:
            - При невалидна заявка се връща подходящо съобщение
        - Зависимости от други елементи:
            - Отговорът на заявката се изпраща към **Drone management-API**
    - SeltTest()
        - Резултат: Дронът проверява за нередности със сензорите или вградената система и другата част на хардуера
        - Изходни данни: При нередност се изпраща съобщение към Drone management-API
        - Зависимости от други елементи:
            - Отговорът на заявката се изпраща към **Drone management-API**

Map API

Предназначение:

- Използване на външни системи за карти.
- Предоставяне на единен интерфейс за работа с всички външни системи.

Отговорности:

- Бърз отговор на всички заявки.
- Лесно интегриране на други услуги за карти.

Интерфейси:

- GetLocation(Drone)
    - Входни данни: Идентификатор на дрон
    - Изходни данни: Локация на велосипеда в момента
    - Грешки и изключения:
        - При невалидна заявка се връща подходящо съобщение

Penalty

Предназначение:

- Осигурява възможност на **Control group** да се свързва с групите по контрол на паркирането(паяци).
- Получаване на електронни фишове на клиенти, които са паркирали неправилно.
- Дава възможност на крайният потребител да изпраща сигнал сигнал до **Control group** на паркирането за неправомерно заето от друг място, за което са абонирани.
- Подмодулът **Register Fine** е отделен, за да може да покрие горните точки и да подсигури изправността на целия **Penalty** модул.

Отговорност:

- Защита на личните данни при получаване и обработка на електронните фишове.
- Защитена комуникация със системата и нейната бази данни.

Интерфейси:

- **Control group**
    - ReportViolation(User, Location, Car_lot)
        - Входни данни: Идентификатор на потребител, местоположение, паркомясто.
        - Резултат: Екипите по контрол на паркирането биват уведомени за неправомерно паркиран автомобил, както и данни за местополежнието му.
        - Грешки и изключения:
            - При невалидна заявка се връща подходящо съобщение.
    - FineIssue(User, Location, Car_lot, photo)
        - Входни данни: Идентификатор на потребител, местоположение, паркомясто, изображение на нарушението
        - Резултат: Снимката и издадения фиш се запазват в системата.
        - Грешки и изключения:

        - При невалидна заявка се връща подходящо съобщение.
    - Зависимости от други елементи:
        - Външната система на групите по контрол на паркирането.
        - Заявката се изпраща към **Database** след верифициране

Technical Team

Предназначение:

- Администриране и наблюдение на системата от адмнситраторите и техническия екип.
- Обработване на съобщенията за нередности в системата.
- Свързване с аварийни екипи при неизправност с даден дрон.

Отговорности:

- Осигуряване на постоянна връзка с аварийните екипи.
- Защита на данните при отстраняване на неизправности в системата.

Интерфейси:

- **Tech**
    - ViewProfile(Administrator, User)
        - Входни данни: Идентификатор на Администратор и Потребител
        - Изходни данни: Информацията за дадения потребител в системата
        - Грешки и изключения:
            - При невалидна заявка се връща подходящо съобщение
        - Зависимости от други елементи:
            - Заявката се изпраща към **Database** след верифициране
    - UpdateProfile(Administrator, User, New_Information)
        - Входни данни: Идентификатор на Администратор и Потребител и обновена информация
        - Резултат: Личната информация на потребителя в системата е обновена
        - Администратора задължително се верифицира преди изпълняване на заявката
        - Грешки и изключения:
            - При невалидна заявка се връща подходящо съобщение

        - Зависимости от други елементи:
            - Заявката се изпраща към **Database** след верифициране
    - GetDroneInfo(Administrator, Drone)
        - Входни данни: Идентификационен номер на Дрон и Администратор
        - Изходни данни: Информация за състоянието и движението на дрона
        - Администратора задължително се верифицира преди изпълняване на заявката
        - Грешки и изключения:
            - При невалидна заявка се връща подходящо съобщение
        - Зависимости от други елементи:
            - Заявките се изпращат към **Drone management**, за да се получат актуални данни.
            - Заявката се изпраща към **Database** след верифициране(технически параметри)
    - UpdateDroneInfo(Administrator, Drone, New_Information)
        - Входни данни: Идентификационен номер на Администратор, Дрон и допълнитена информация за него
        - Резултат: Променяне на информацията за дадения Дрон
        - Администратора задължително се верифицира преди изпълняване на заявката
        - Грешки и изключения:
            - При невалидна заявка се връща подходящо съобщение
        - Зависимости от други елементи:
            - Заявката се изпраща към **Database** след верифициране
- **Team call**
    - ReportDroneMalfunction(Drone, Location, Actual_Data)
        - Входни данни: Идентификационен номер на Дрон, местоположение и наличната допълнителна информация(сензори, самодиагностика и т.н)
        - Резултат: Техническите екипи биват уведомени за настъпилата авария с летателния апарат и получават допълнителната информация
        - Зависимости от други елементи:
            - Заявката се изпраща към външните екипи

Описание на възможните вариации

- **Map API** може да бъде заменен с COTS, защото е стандартен модул за използване на карти.
- Обмена на информация между подмодулите се извършва по протокола **REST (** ***стил софтуерна архитектура за реализация на уеб услуги включваща взаимодействията между сървър и клиент, осъществени по време на трансфера на данни*** **)**, защото в системата ще се прехвърля голямо количество информация и е необходим протокол, който има възможно най-малко допълнителна информация.

Внедряване

Deployment е ключова структура за **DronesPark**. В декомпозицията на модули всеки отделен модул е на отделен **хардуерен** сегмент. Някои от модулите могат да бъдат дублирани на няколко **машини(сървъри)**.Подмодулите, които са от особено значение, са отделени на различни сървъри, заради високото натоварване. В частност и заради обема на данни при комуникация между отделните елементи, както и изискването подмодулите да не прекъсват работа.

Едно от изискванията към системата е да работи на 100% без отказ, в периода на светлата част на работния ден. Поради тази причина

Database Collection, User Manager, Drone Management, Technical team и **Penalty,** както и **Server Manager**, трябва да са налични постоянно. Те са сървъри необходими за обслужването на потребителите. За да се постигне това трябва да се подходи с определени тактики, от които зависи дали тези сървъри трябва да имат по 2 или повече копия, които да обслужват заявки и да разпределят оптимално работата си и в случай на повреда на един от тях, останалите работещи ще си разпределят по равно работата. Или сървърите да имат резервно копие, което се стартира в случай ,че в основния настъпи повреда и не може да обработва заявки. Тези решения за системата няма как да бъдат описани в декомпозицията на модулите. Вследствие на това е необходима deployment структура. Благодарение на deployment диаграма може да се опише как трябва да работят сървърите. Също така може да се покаже дали даден

сървър работи с 2 копия, дали има backup, дали е необходима синхронизация и т.н. Това е своеобразно допълнение към декомпозицията на модулите, защото илюстрира допълнителните връзки между различни модули и подмодули, когато бъдат разделени на отделни сървъри.

Изискване към системата е тя да може динамично да определя броя на летящите в момента дронове, както и маршрута на всеки един от тях. Това се постига чрез предвиждане, на базата на натрупаните данни(динамика на паркиране, честота на заемане/освобождаване на места, часови диапазон и т.н) Така може да се постигне пик в указаното натоварване върху системата вследствие на голям (необходим) брой дронове във въздуха. От друга страна системата трябва да е на разположение на голям брой и различни на тип потребители, както и да обслужва техните специфични заявки.(Групи потребители като: Администратори/Оператори/Аварийни групи/Групи по контрол на паркирането(т.нар. ,,паяци'')/ Регистрирани и обикновени потребители. Изискванията в този параграф отново предполагат горепосочените решения, защото ако всеки сървър има само едно копие, то при срив дори на един от тях, цялата система ще бъде неизползваема. Поради тази причина отново трябва да бъдат взети решения, касаещи критичните компоненти в системата и тяхната работоспособност, дори при спиране на една от машините, на която е инсталиран даден модул.

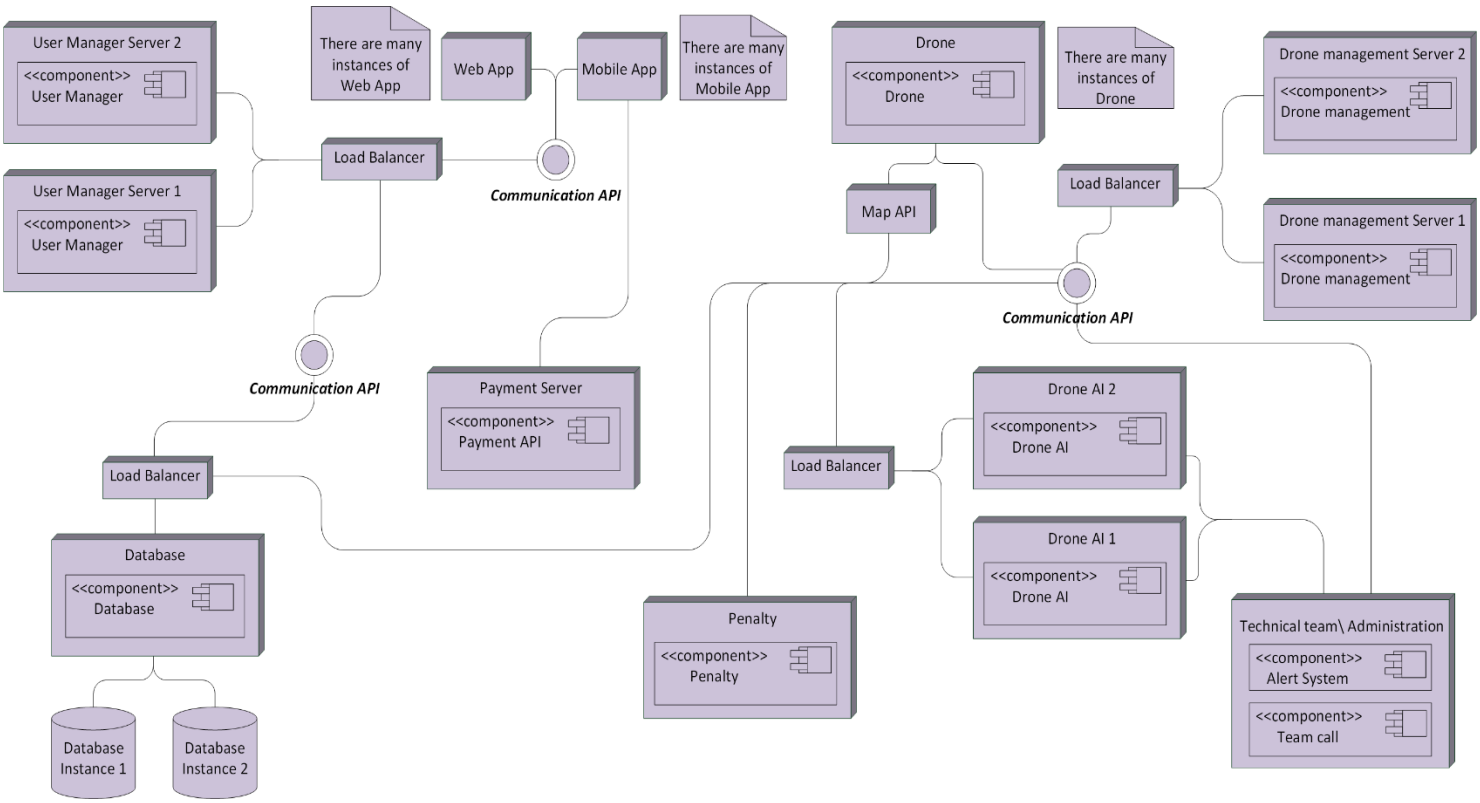
Последното критично изискване, за което е необходимо deployment диаграма, защото декомпозицията на модули не демонстрира достатъчно ясно как ще бъде реализирано е комуникацията с аварийните групи.

Тази връзка се осъществява, когато някой дрон излезе от строя. Отговорниците за техническото състояние на дроновете трябва да получат информация за предполагаемия район, в който се намира летателния апарат, и да остранят повредата. Това изискване предполага, необходимостта от обработка в реално време на всички данни получени от индивидуалните сензори на всеки дрон, както и на външните услуги за ,,карти'' и метеорологични условия. Цялата тази информация се анализира от **Drone management**. От друга страна, на базата на тези данни, както и на други получени от заснетите изображения на паркоместа, и ,,предвиждането''(споменато в предишния параграф) се определя поведението на **Drone AI**. Той отговаря за логичексата свързаност и се превръща в своеобразен ръководител на полет за отделните дронове. Въпреки че е подмодул на **Drone management**, двата модула не могат да бъдат на една и съща машина и не могат да имат единствено

копие. Причина за тези решения е голямото натоварване, защото количеството информацията от дроновете е огромно. Необходимостта от неотложна обработка на информацията също допринася към тази тежест. Тя е породена от факта, че сигурността на потребителите и на дроновете зависи от тези данни.

Заради тези изисквания и архитектурата, която те предполагат, е необходимо да се направи deployment структура за системата.(Допълнително изяснение на архитектурата)

Първично представяне



Всеки елемент на диаграмата представлява отделно хардуерно устройство.

User Manager Server X, Payment Server, Drone Management Server X, Drone AI X, Load Balancer, Technichal Team/Administration, Penalty са сървъри към системата.

Mobile App- Мобилно приложение, чрез което клиентите използват системата.

Web App- Уеб приложение, чрез което администраторите и екипа по поддръжката използват системата, а също така потребители мога да правят справка относно фишове.

Описание на връзките и елементите

- **Payment Server** предоставя интерфейс за плащане, който е еднакъв за всички различни методи на плащане към **Mobile App**. Той е отделен на отделна машина с единствен канал за комуникация ,за да се постигне по-високо ниво на сигурност.
- **Map API** предоставя единен интерфейс за работа с всички карти, които са свързани със системата.
- **Database**- както й съответните подмодули на модула **Database** са на отделен сървър. Свързани са с един канал за комуникация към **Load Balancer**
- **User Manager** сървърът обработва профилите на всички потребители. Има няколко инстанции на този сървър, за да може да издържи на натоварването при пик в броя на потребителие и пик в броя на заявките. Тези заявки се изпращат към **Communication API -**подмодул, който физически се намира на **User manager server** и през, който минават всички заявки от и към сървъра.
- **Mobile app** са всички отделни смартфони, които имат достъп до приложението Дронсепакр. Хардуерът на устройството може да произволен и не е от значение за системата. Този модул осъществява връзка между **User Manager** сървърите в системата. По този начин предоствая на потребителите цялата функционалност.
- **Drone** е вградена система на всеки дрон. Тя отговаря за неговата автономност, мониторинг на сензорите и комуникация с останалите модули.
- **Drone AI 2** сървърите са съставени от подмодула **Drone AI** на **Drone management**. Поради голямото натоварване, което се оказва от трафика и размера на данните, които трябва да се обработят. Модулът е отделен на няколко сървъра, пред които има **Load Balancer**.
- **Load Balancer** е сървър, който отговаря за разпределение на трафика. Той филтрира и изпраща единствено необходимата информация към отделните сървъри.
- **Technical Team/Administration** сървърът упражнява мониторинг на системата като към него се изпращат заявки с приоритет. Например съобщения за авария. Този сървър може да е само един, защото не е необходимо да обработва огромно количество информация.

- **Web app -** Сървър, който осигурява административен достъп до системата. От друга страна, потребители могат да правят справка за издадени електронни фишове.
- **Penalty** - Сървър изграден от компонентите на **Penalty** модул. Отговаря за установените нарушения, обработка на фишове и комуникация с екипите по контрол на паркирането. Един канал на комуникация за повишена сигурност. Едно копие, защото не се очаква да обработва огромно количество информация.
- **Drone Management** сървърите имат целия **Drone Management** модул, с изключение на **Drone AI** подмодул. Тъй като са отговорни за обработката на информацията от дроновете и на други данни, се очаква голямо натоварване. Поради тази причина има няколко иснтанции, които имат собствени **Communication API**- подмодул, който се намира на всеки един сървър и е отговорен за вход/изход на информация.
- Важно е да се отбележи, че сървърите **Web app** и **Mobile app** са свързани чрез един комуникационен канал към **Load balancer.** Единствено след успешното им верифициране от **User manager** сървърите, потребителите получават достъп към системата. Това предпазва допълнително системата от външна и неправомрна намеса.

Описание на обкръжението

- **Payment server**-е свързван с множество външни системи и комуникира с тях по специфичен начин.
- **MAP API** сървърът е wrapper за различните видове карти, които се използват от системата. Свързан е с външните системи, които представялват услугите на тези карти.

Описание на възможните вариации

- Модулът **Server Manager** може да бъде на отделен сървър, към който ще се насочва потока от информация. В този случай ще е желателно да има повече от едно копие, зад се разпределя трафика. Или един резервен, който осигурява пространство за работа в случай, че главния сървър претърпи срив.
- Може гореспоменатите сървъри вместо да бъдат машини да са в cloud среда.

Структура на процесите

Следващата приложена диаграма представя процеса на вход в приложението, последвано от заявка за абонамент или за наличие на свободни паркоместа, съответно обратната връзка от системата. Всичко това е изобразено от гледната точка на потребителя. Поради тази причина е необходимо едно опростено представяне на архитектурата, в което са посочени различните процеси, както и разположението им във времето. Споменато е, че има обработване на голямо количество информация в реално време. Чрез структура на процесите може да се покаже как се случва това от гледна точка на крайния потребител. Поради тази причина диаграмата на процесите е подходяща за всички групи заинтересовани лица. Например: регистрирани и обикновени потребители, администратори, аварийни групи, групи по контрол на паркирането(т.нар ,,паяци''), оператори и д.р.

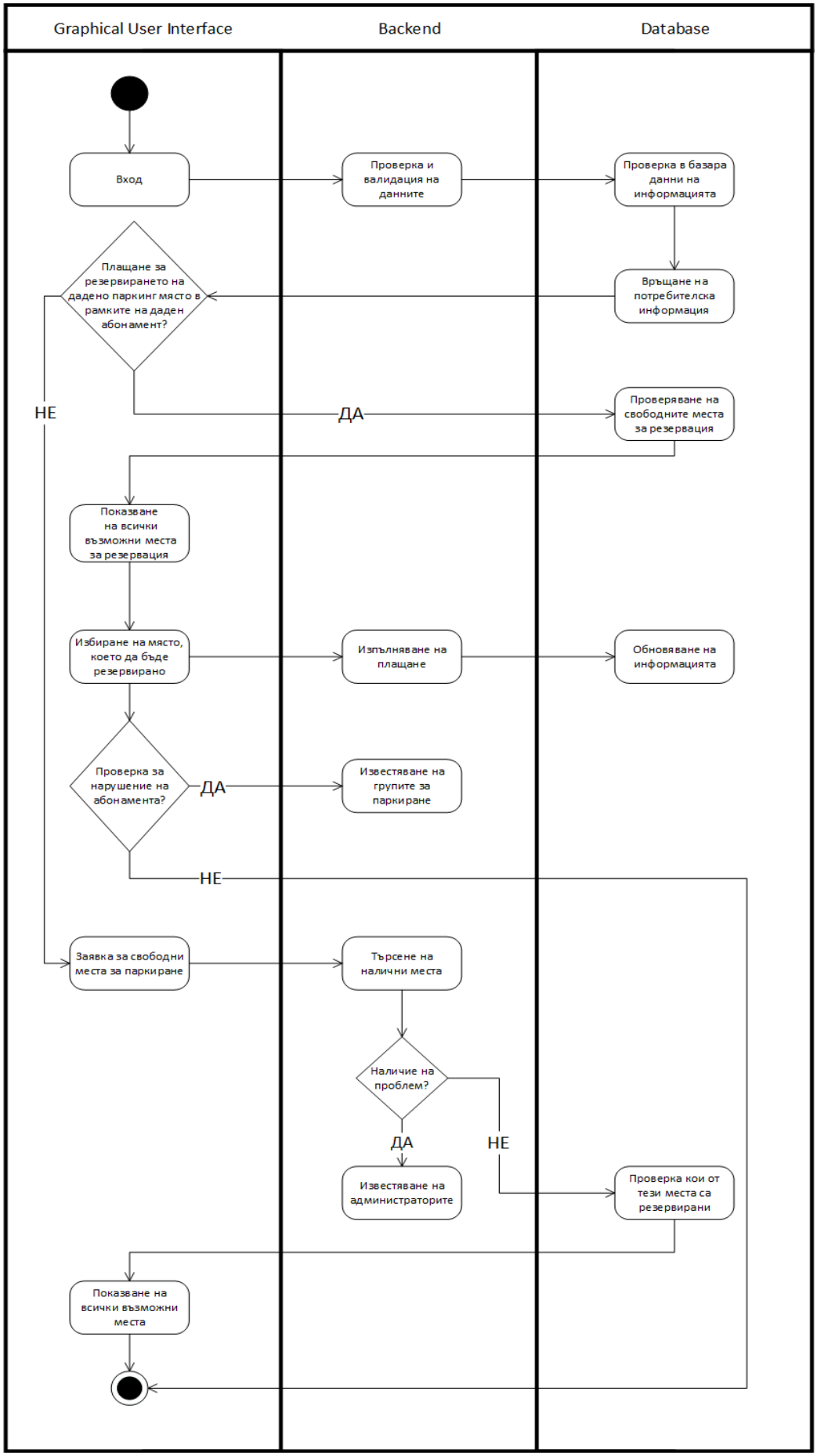
На диаграмата е показано как са изпълнени някои от системните изисквания към **DronesPark:**

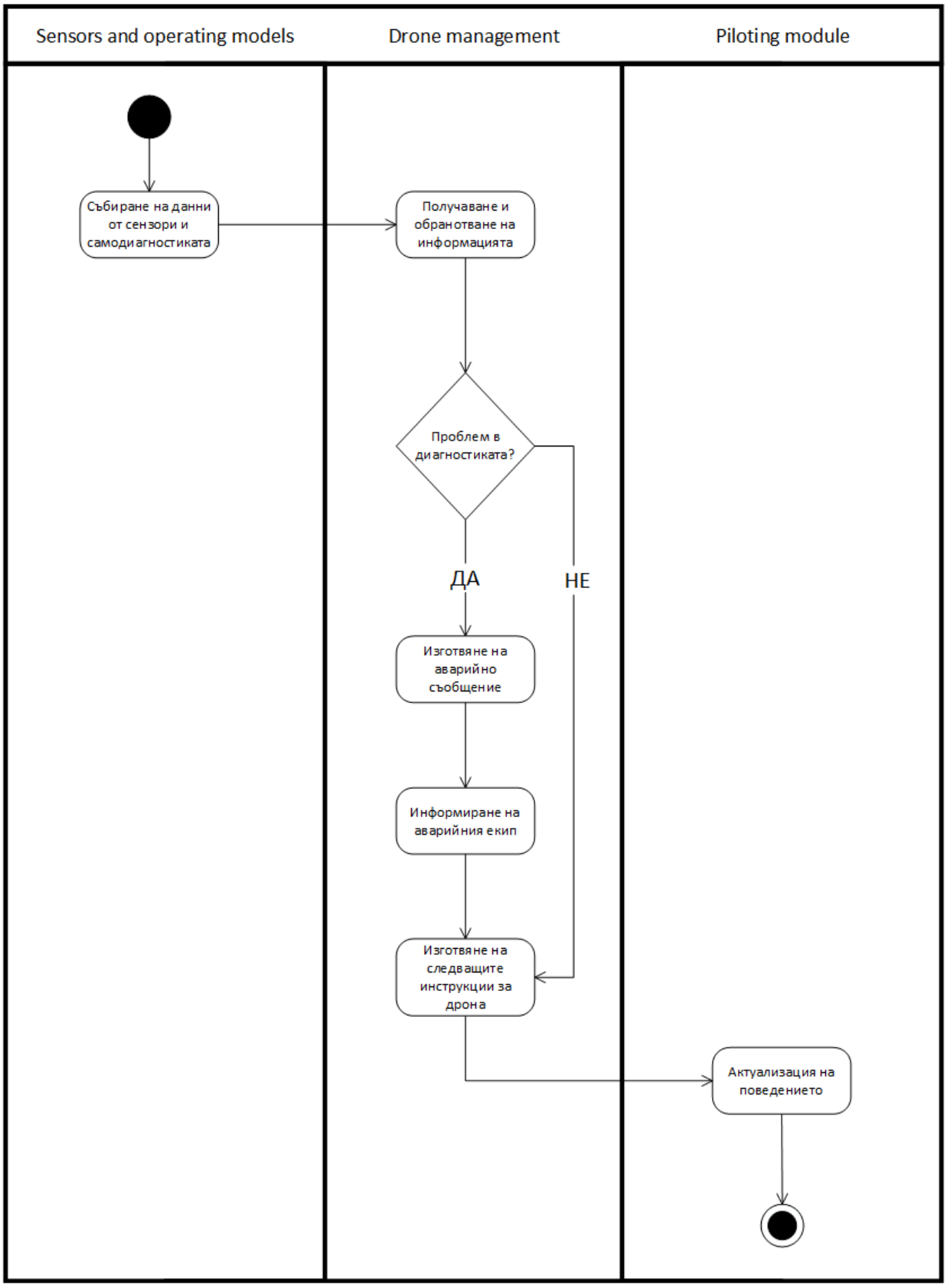
- *Потребителите трябва да имат 100% защитен от външна намеса достъп до системата.*
- *Системата поддържа следните групи потребители: администратор, оператор, аварийни групи, групи по контрол на паркирането (т.нар. ,,паяци''), регистрирани и обикновени потребили.*- Горе споменатите изисквания са изпълнени чрез разделянето на процеса на вход и регистация, както и на комуникация между потребителите и системата, на три слоя- Graphical user interface, Backend и Database. Връзките между тях са криптирани и използват специален протокол за защита.
- *Потребителите могат да заплатят абонаменти чрез дебитна/кредитна карта, PayPal или СМС, като в бъдеще може да се добавят и други начини на разплащане*- при създаване на абонамент, backend се свързва със системата за плащане и извършва съответното плащане, спрямо метода избран от потребителя.
- *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.*- това е осигурено от непрестанната връзка между всеки дрон, приложението и Backend-а. При необходимост

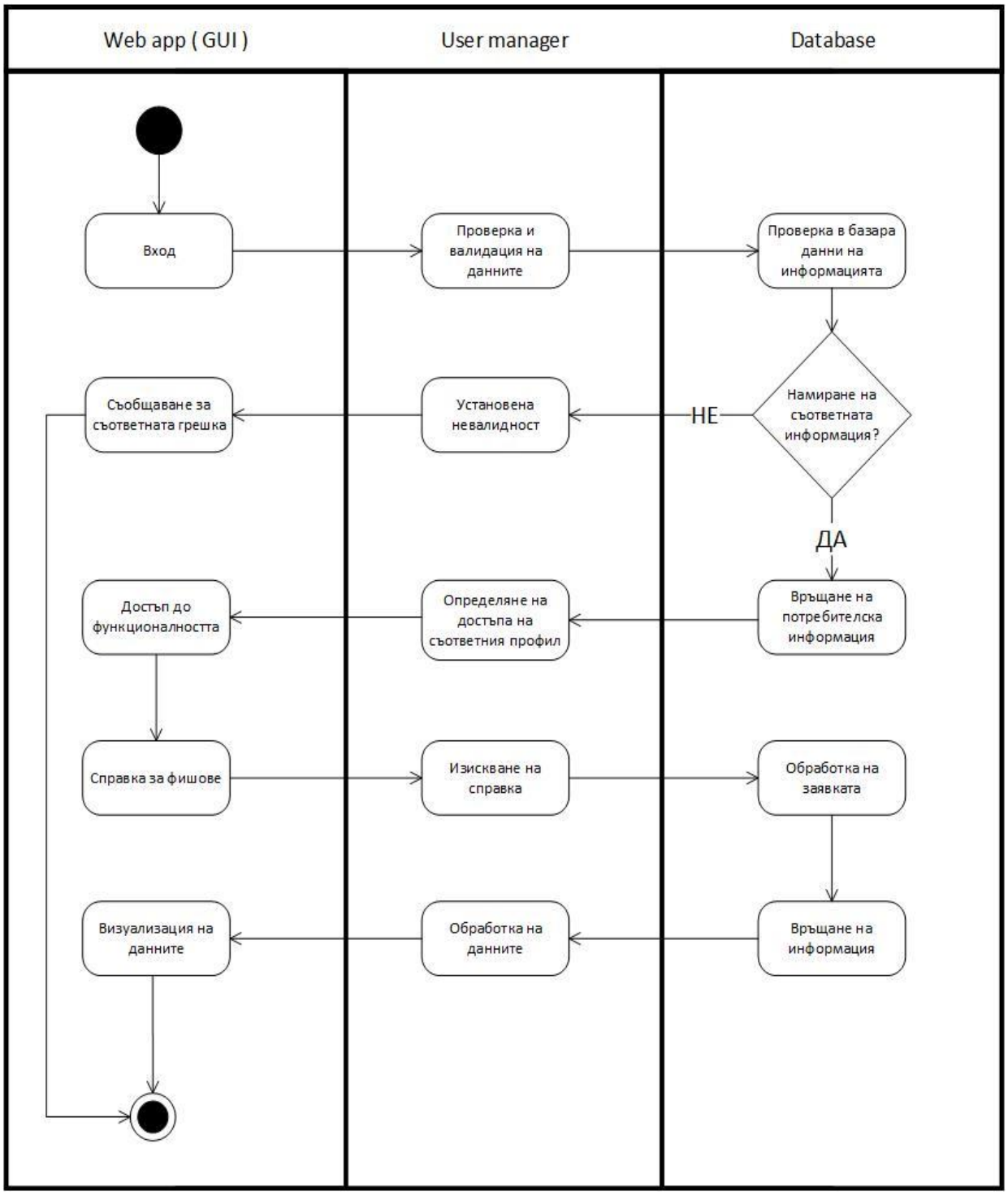
системата реагира бързо, предоставяйки всички необходими данни на аварийните екипи.

- *Обикновените потребители, регистрираните потребители, аварийните и групите по контрол на паркирането използват системата през мобилно приложение, като може да заемат само свободните места, за които няма абонамент.*
- *Регистрираните потребители могат да заплащат абонамент за определено парко-място, 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.*- Изброените, в последните 2 точки, изисквания към системата биват реализирани благодарение на интензивната комуникация между backend-а и database-а. По този начин, заедно с помощта на външните модули за карти, се постига бързодействие, чийто резултат достига потребителите благодарение на Graphical user interface. Чрез него те могат да взаимодействат със системата.

Първично представяне


Graphical User Interface
Backend
Database
Вход
Проверка и валидация на данните
Проверка в базара данни на информацията
Плащане за резервирането на дадено паркинг място в рамките на даден абонамент?
Връщане на потребителска информация
НЕ
ДА
Проверяване на свободните места за резервация
Показване на всички възможни места за резервация
Избиране на място, което да бъде резервирано
Изпълняване на плащане
Обновяване на информацията
Проверка за нарушение на абонамента?
ДА
Известяване на групите за паркиране
НЕ
Заявка за свободни места за паркиране
Търсене на налични места
Наличие на проблем?
ДА
НЕ
Известяване на администраторите
Проверка кои от тези места са резервирани
Показване на всички възможни места


Sensors and operating models
Drone management
Piloting module
Събиране на данни от сензори и самодиагностиката
Получаване и обработване на информацията
Проблем в диагностиката?
ДА
НЕ
Изготвяне на аварийно съобщение
Информиране на аварийния екип
Изготвяне на следващите инструкции за дрона
Актуализация на поведението


Web app (GUI)
User manager
Database
Вход
Проверка и валидация на данните
Проверка в базара данни на информацията
Намиране на съответната информация?
НЕ
ДА
Установена невалидност
Съобщаване за съответната грешка
Връщане на потребителска информация
Определяне на достъпа на съответния профил
Достъп до функционалността
Справка за фишове
Изискване на справка
Обработка на заявката
Връщане на информация
Обработка на данните
Визуализация на данните

Описание към първата диаграма

Описание на елементите и връзките

- Извършва се проверка на потребителксите данни при вход. След това базата данни проверява дали тези потребителски данни съществуват. След тази проверка, приложението изпраща заявка за извличане на последните данни за свободни паркоместа. Отговорът, както и останалите потребителски функции, се връщат на потребителя. Той може да заплати абонамент за дадено паркомясто. Ако то е налично, след извършване и потвърждение на плащането, мястото бива маркирано.Тези промени веднага биват отразени в системата и в базата данни. През цялото това време дроновете комуникират със системата.

Описание на обкръжението

- При създаване на абонамент, системата се свързва с външни услуги за плащане.
- При търсене на свободни паркоместа и/или създаване на абонамент, се осъществява връзка с базата данни на системата, както и данни от **Parking identifier**. Също така се осъществява връзка с API, който отговаря за карти. Извлечените данни се нанасят върху картите и биват предоставени на потребителите.

Описание на възможните вариации

- При настъпване на проблем в системата или в някой от летателните апарати, то администраторите биват уведомени/или аварийните групи съответно.

Архитектурна обосновка

1. **Свободните паркоместа се идентифицират от системата от дронове, които обикалят града и заснемат зоните за паркиране(отгоре).**

   Това изискване предпога, че в архитектурата ще трябва да се предвиди начин за заснемане на зоните. От тука ще се изисква модул за камера за всеки дрон. Постига се от подмодула на **Drone – Camera module.** Също така ще се изисква модул за индентификация дали дадено място е свободно или не. Това се постига с подмодула на **Drone management - Parking identifier.**

2. **Броят на летящите в момента 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.**

   От тука ясно виждаме, че ни трябва **AI** система, която от базата на натрупани знания да определя адекватно движението на дроновете. Също ще трябва да използваме външна услуга за времето, която **AI** системата да използва за управлението на дронове. Модулът, който се грижи за **AI** системата е **Drone AI,** а за външната услуга е **Weather forecast.** Също в диаграмата на внедряването е показано, че поради голямото натоварване, което се оказва от трафика и размера на данните, които трябва да се обработят, модулът **Drone AI** е отделен на няколко сървъра. Тези сървари ще си поделят работата по равно като така увеличаваме производителността и намаляваме времето, което системата трябва да чака. Ако един от тези сървари се повреди то следва, че останалите работещи ще си поделят работата по равно, докато не се отстрани повредата. Трябва и да се вземе на предвид, че ще има много дронове на различни локации. Това само по себе си изисква да се направи модул за геолокация. Това се изпълнява от модула **Map API.** Информацията за локацията на всеки един от дроновете се приема от модула **Location management,** който ги оформя в подходящи за **Drone AI** данни.

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

   Ще трябва да използваме модул, който да се грижи за външната услига. Този модул е **Weather forecast.**

4. **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.**

   За улеснение на аварийните групи трябва да се сложи **GPS** сензор на всеки дрон, който ще определя координатите му. Тук използваме модула **GPS sensor.** Трябва периодично всеки дрон да се проверява дали правилно работи. За това служи модула **Self test.** Изпраща се информацията на главната система. Ако даден дрон е в неизгодно положение ще трябва тогава **AI** системата да го вземе в предвид и да се адаптира подходящо към ситуяцията. За това имаме модула **Drone state,** който приема информацията за състоянието на всички дронове и я предава в подящи за **Drone AI** данни**.** После се дава сигнал на аварийните групи за повреден дрон и неговата локация. Същевременно, ако след даден период от време даден дрон не изпраща информация за състоянието си, системата приема, че дронът е повреден и се взимат същите мерки, както когато дронът праща информация, че има дефект.

5. **Регистрираните потребители могат да заплащат абонамент за определено парко-място, 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.**

От това изискване следва, че архитектурата ще трябва да осигури възможността за отбелязване на паркоместата като: заети, свободни, с или без абонамент към тях. Това е постигнато чрез подмодула на **Drone management- Parking identifier.** Важно е тези промени да се съхранят в системата, за да бъдат достъпни на заинтересованите при употреба на функционалностите свързани със създаването на такъв абонамент. Неговото създаване се осъществява чрез заплащане(по различни начини: PayPal, смс, кредитна карта и д.р). Отговорен за този процес е модулът **Payment**. Той предоставя на потребителя единен интерфейс за работа с изброените методи на плащане. Също така дава възможност за добавяне на нови, което е част от изискванията към системата. Важно е да се отбележи, че този модул допринася за защитата на личните данни при плащания.

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

Без никакво съмнение е, че профилите на потребителите трябва да са защитени от външна намеса. В тях има лична информация, която трябва да бъде предпазена. Това е и основна характеристика на всяко едно приложение. Модулът, който се грижи за това е **Data collection.** Също така на диаграмата на внедряването е показано, че достъпа до данни е регламентиран през няколко сървъра. В допълнение, данните се съхраняват съгласно различни методики и протоколи за повишаване на защитата.

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

Очаква се системата да бъде натоварена в определен часови диапазон, през който се изпозлва в пъти повече спрямо останалата част на денонощието. Необходимо е да се осигури отказоустойчива разбота на системата в такъв период. Също така е важно архитектурата да е съставена така, че да позволява системата а scale-ва

лесно. **Това е постигнато чрез разделянето на приложението върху множество сървъри, свързани чрез сървър за комуникация. При възникване на проблем в един от сървърите, останалите помат натоварването. Описано е в секцията Внедряване.**

8. Системата да поддържа архив на данните за динамиката на паркирането и всички издадени фишове за глоби за 25 години назад във времето, както и архив на заснетите изображения за 3 години назад.

Важно е да се съхраняват горепосочените данни, за да се осигури възможност за справки. **Модулът Data collection отговаря за съхранението на тези архиви.**

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

Това изискване предполага системата да бъде изградена по начин, по който новите методи на разплщане, да могат да се имплементират лесно. Поради тази причина модула **Payment** се състои от различни подмодули (**PayPal, Credit Card, SMS**), които отговарят за всеки един начин за плащане. Този подход осигурява, както допълнителна защита, така и позволява на модула да се разширявя без да нарушава неговата структура.

10. Системата поддържа следните групи потребители:

- **Администратор**
- **Оператор**
- **Аварийни групи**
- **Групи по контрол на паркирането(т.нар. ,,паяци’’)**

- **Регистрирани потребители**
- **Обикновени потребители**

Това е постигнато чрез мобилното приложение и модула за техническия екип (Декомпозиция на модули)