|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

**Факультет «Информатика и системы управления»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

Дисциплина «Разработка интернет-приложений»

Техническое задание

Тема: «Парковки МГТУ»

Студент: Поляков Д.Д.

Группа ИУ5-53Б

Преподаватель: Канев А.И.

2023г.

**СОДЕРЖАНИЕ**

[1. Цель 3](#_Toc150775711)

[2. Назначение 3](#_Toc150775712)

[3. Задачи 3](#_Toc150775713)

[4. Методы веб-сервиса 4](#_Toc150775714)

[5. Функциональные требования 13](#_Toc150775715)

[6. Требования к программному обеспечению: 21](#_Toc150775716)

[7. Требования к аппаратному обеспечению: 22](#_Toc150775717)

## Цель

Реализовать систему для автоматизации документирования выдачи абонементов на парковки МГТУ, которая будет состоять из веб-приложения, веб-сервиса, отдельного асинхронного веб-сервиса сервиса для расчёта времени въезда на парковку и кроссплатформенного мобильного приложения.

## Назначение

Основное назначения разработанной системы заключается в автоматизации и оптимизации процесса работы по выдаче и заказу абонементов на парковки МГТУ. Система позволит быстро находить информацию об парковках и автомобилях, которые на них стоят. Также она позволит сотрудникам – быстро и легко создавать заявки на добавления новых абонементов, а администраторам – принимать / отклонять их, тем самым повысив эффективность документирования исторических открытий.

## Задачи

* 1. Разработать дизайн системы приложения и сайта
  2. Создать базу данных PostgreSQL
  3. Реализовать веб-сервис на технологии Django REST Framework
  4. Создать интерфейс гостя на технологии React
  5. Развернуть веб-приложение React на GitHub Pages
  6. Написать документацию методов API
  7. Добавить аутентификацию и авторизацию в веб сервис, основанную на JWT и Redis
  8. Создать интерфейс пользователя на React, с использованием Redux Toolkit
  9. Создать кроссплатформенное мобильное приложение на React Native
  10. Создать асинхронный веб-сервис для расчета времени въезда на парковку на Go
  11. Создать интерфейс администратора на React
  12. Подготовить набор документации, включающий ТЗ, РПЗ и набор диаграмм

## Методы веб-сервиса

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Метод** | **Описание** | **URL** | **Входные данные** | **Выходные данные** |

|  |
| --- |
| 4.1 Методы парковок |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4.1.1 | GET | Возвращает список парковок, удовлетворяющих переданным критериям  Будут возвращены парковки, отфильтрованные по статусу «действует», а также id чернового абонемента, пользователя если он есть  Доступно всем пользователям | api/parkings/search | {  “query”: char(100),  } | {  “draft\_ticket\_id”: int,  “patkings”: [  {  “id”: int,  “name”: char(100),  “adress”: text,  “status”: int,  “image”: char(100),  “places\_count”: int,  },  …  ]  } |
| 4.1.2 | GET | Возвращает парковку  Доступно всем пользователям | api/parkings/< parking\_id>/ | {  “patking\_id”: int  } | {  “id”: int,  “name”: char(100),  “adress”: text,  “status”: int,  “image”: char(100),  “places\_count”: int,  } |
| 4.1.3 | PUT | Обновляет информацию о парковке  Доступно только администратору | api/parkings /<parking\_id>/update/ | {  “patking\_id”: int,  “access\_token”: int  } | {  “id”: int,  “name”: char(100),  “adress”: text,  “status”: int,  “image”: char(100),  “places\_count”: int,  } |
| 4.1.4 | DELETE | Меняет статус парковки на  «удален»  Доступно только администратору  Возвращает список из всех парковок, отфильтрованных по статусу «действует», | api/parkings/<patking\_id> /delete/ | {  “patking\_id”: int,  “access\_token”: int  } | [  {  “id”: int,  “name”: char(100),  “adress”: text,  “status”: int,  “image”: char(100),  “places\_count”: int,  },  …  ] |
| 4.1.5 | POST | Создает новую парковку  Доступно только администратору | api/parkings/create/ | {  “access\_token”: int  } | [  {  “id”: int,  “name”: char(100),  “adress”: text,  “status”: int,  “image”: char(100),  “places\_count”: int,  },  …  ] |
| 4.1.6 | POST | Добавляет парковку в абонемент  Возвращает список всех парковок в абонементе  Доступно только авторизованным пользователям | api/parkings/<parking\_id> /add\_to\_ticket/ | {  “patking\_id”: int,  “access\_token”: int  } | [  {  “id”: int,  “name”: char(100),  “adress”: text,  “status”: int,  “image”: char(100),  “places\_count”: int,  },  …  ] |
| 4.1.7 | GET | Возвращает картинку парковки в бинарном виде  Доступно всем пользователям | api/parkings/<parking\_id> /image/ | {  “patking\_id”: int  } | {  “image”: bytes  } |
| 4.1.8 | PUT | Обновляет картинку парковки  Возвращает обновленную картинку парковки  Доступно только администратору | api/pargkings/<parking\_id> /update\_image/ | {  “patking\_id”: int,  “image”: bytes  } | {  “image”: bytes  } |

|  |
| --- |
| 4.2 Методы абонементов |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4.2.1 | GET | Возвращает список абонементов, удовлетворяющих переданным критериям  Критерии:  Статус, пользователь, диапазон даты формирования  Доступно только авторизированным пользователям | api/tickets/search/ | {  “access\_token”: char(100),  “status”: int,  “user”: int,  “date\_start”: long,  “date\_end”: long  } | [  {  "id": int,  "date": int,  "status": int,  "date\_created": datetime,  “date\_formation": datetime,  "date\_complete": " datetime  "owner": {  “id”: int,  “email”: bool,  “is\_moderator”: bool  },  “moderator”: {  “id”: int,  “email”: bool,  “is\_moderator”: bool  },  "patkings": [  {  “id”: int,  “name”: char(100),  “adress”: text,  “status”: int,  “image”: char(100),  “places\_count”: int,  },  …  ]  },  …  ] |
| 4.2.2 | GET | Возвращает абонемент  Доступно только авторизированным пользователям | api/tickets/<ticket\_id>/ | {  “ticket\_id”: int,  “access\_token”: char(100)  } | {  "id": int,  "date": int,  "status": int,  "date\_created": datetime,  “date\_formation": datetime,  "date\_complete": " datetime  "owner": {  “id”: int,  “email”: bool,  “is\_moderator”: bool  },  “moderator”: {  “id”: int,  “email”: bool,  “is\_moderator”: bool  },  "patkings": [  {  “id”: int,  “name”: char(100),  “adress”: text,  “status”: int,  “image”: char(100),  “places\_count”: int,  },  …  ]  } |
| 4.2.3 | PUT | Обновляет дату въезда в абнементе  Доступно с ключом доступа | api/tickets/<ticket\_id>/update\_time\_entry/ | {  “ticket\_id”: char(100),  “year”: int,  “secret\_key”: char(100)  } | Статус 200 / 403 |
| 4.2.4 | PUT | Обновляет информацию об абонементе  Возвращает обновленный  абонемент  Доступно только авторизированным пользователям | api/tickets/<ticket\_id>/update/ | {  “ticket\_id”: int,  “access\_token”: char(100)  } | {  "id": int,  "date": int,  "status": int,  "date\_created": datetime,  “date\_formation": datetime,  "date\_complete": " datetime  "owner": {  “id”: int,  “email”: bool,  “is\_moderator”: bool  },  “moderator”: {  “id”: int,  “email”: bool,  “is\_moderator”: bool  },  "patkings": [  {  “id”: int,  “name”: char(100),  “adress”: text,  “status”: int,  “image”: char(100),  “places\_count”: int,  },  …  ]  } |
| 4.2.5 | PUT | Обновляет статус абонемента с «черновик»  на «в работе»  В случае успеха меняет дату формирования абонемента на текущую  Можно обновить только при наличии статуса «в работе»*,* иначе возвращает ошибку 405  Возвращает обновленный абонемент  Доступно только авторизированным пользователям | api/tickets/<ticket\_id>/update\_status\_user/ | {  “ticket\_id”: int,  “access\_token”: char(100)  } | {  "id": int,  "date": int,  "status": int,  "date\_created": datetime,  “date\_formation": datetime,  "date\_complete": " datetime  "owner": {  “id”: int,  “email”: bool,  “is\_moderator”: bool  },  “moderator”: {  “id”: int,  “email”: bool,  “is\_moderator”: bool  },  "patkings": [  {  “id”: int,  “name”: char(100),  “adress”: text,  “status”: int,  “image”: char(100),  “places\_count”: int,  },  …  ]  } |
| 4.2.6 | PUT | Меняет статус абонемента с «в работе» на «завершён» или «отклонён»  В случае успеха меняет дату завершения абонемента на текущую  Можно обновить только при наличии статуса «в работе»*,* иначе возвращает ошибку 405  Возвращает обновленный абонемент  Доступно только авторизированным пользователям | api/tickets/<ticket\_id>/update\_status\_admin/ | {  “ticket\_id”: int,  “access\_token”: char(100)  } | {  "id": int,  "date": int,  "status": int,  "date\_created": datetime,  “date\_formation": datetime,  "date\_complete": " datetime  "owner": {  “id”: int,  “email”: bool,  “is\_moderator”: bool  },  “moderator”: {  “id”: int,  “email”: bool,  “is\_moderator”: bool  },  "patkings": [  {  “id”: int,  “name”: char(100),  “adress”: text,  “status”: int,  “image”: char(100),  “places\_count”: int,  },  …  ]  } |
| 4.2.7 | DELETE | Меняет статус абонемента с «черновик» на «удалён»  Можно выполнить только при наличии у абонемента статуса «черновик»*,* иначе возвращает ошибку 405  В случае успеха возвращает статус 200  Доступно только авторизированным пользователям | api/tickets/<ticket\_id>/delete/ | {  “ticket\_id”: int,  “access\_token”: char(100)  } | Статус 200 / 404 / 405 |
| 4.2.8 | DELETE | Удаляет парковку из абонемента  Возвращает обновлённый список парковок в абонементе  Доступно только авторизированным пользователям | api/tickets/<ticket\_id>/delete\_oatking/<parking\_id>/ | {  “ticket\_id”: int,  “patking\_id”: int,  “access\_token”: char(100)  } | {  "patkings": [  {  “id”: int,  “name”: char(100),  “adress”: text,  “status”: int,  “image”: char(100),  “places\_count”: int,  },  …  ]  } |

|  |
| --- |
| 4.4 Методы авторизации и аутентификации |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4.4.1 | POST | Регистрирует нового пользователя в базе данных  Если в системе уже существует пользователем с такой почтой, то возвращает ошибку 409  В случае успеха возвращает статус 201, id и access\_token созданного пользователя  Доступен всем пользователям | api/register | {  “email”: char(100),  “password”: char(100)  } | {  “user\_id”: int,  “access\_token”: char(100)  } |
| 4.4.2 | POST | Выполняет аутентификацию пользователя по логину и паролю  В случае отправки невалидных данных возвращает ошибку 401  В случае успеха возвращает статус 200, а также данные о пользователе | api/login | {  “email”: char(100),  “password”: char(100)  } | {  “user\_id”: int,  “email”: char(100),  “is\_moderator”: bool,  “access\_token”: char(100)  } |
| 4.4.3 | POST | Выполняет аутентификацию пользователя по токену  В случае отправки невалидного токена возвращает ошибку 401  В случае успеха возвращает статус 200, а также данные о пользователе | api/check | {  “access\_token”: char(100)  } | {  “user\_id”: int,  “email”: char(100),  “is\_moderator”: bool,  “access\_token”: char(100)  } |
| 4.4.4 | POST | Добавляет его токен в черный список  Доступно только авторизированным пользователям | api/logout | {  “access\_token”: char(100)  } | Статус 200 / 401 |

|  |
| --- |
| 4.5 Методы удаленного сервиса |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4.5.1 | POST | Определяет время въезда в абонементе  Действие запускается через 3 секунды. Вероятность успеха 70% | api/calculate\_time\_entry | {  “ticket\_id”: int  } | Статус 200 / 403 |

## Функциональные требования

* 1. Главное меню
     1. Парковки – переход на страницу со списком / таблицей парковок
        1. Доступно всем пользователям
        2. Для обычных пользователей выполняется переход на страницу со списком парковок (5.4)
        3. Для администраторов выполняется переход на страницу с таблицей парковок (5.5)
     2. Абонементы – переход на страницу с историей открытий (5.8)
        1. Доступно только авторизированным пользователям.
     3. Вход – переход на страницу с формой авторизации (5.2)
        1. Доступно только неавторизированным пользователям.
     4. Выход
        1. Доступно только авторизированным пользователям.
        2. При нажатии отправляется запрос на API (4.4.4), который добавляет токен пользователя в черный список. Также токен удаляется из cookies, обнуляется состояние пользователя в Redux и происходит переадресация на страницу со списком первооткрывателей (5.4)
  2. Страница с формой авторизации
     1. Доступна только неавторизованным пользователям
     2. Содержит
        1. Тестовое поле для ввода почты
        2. Текстовое поле для ввода пароля
        3. Кнопку Войти
        4. Кнопку Ужезарегистрированы**?**
     3. Действия
        1. При нажатии на кнопкуВойти отправляется запрос на API (4.4.2), в котором передаются введенные данные формы. Если введенные данные верны, то в ответе возвращается access токен, который сохраняется в cookies, а также происходит переадресация на страницу со списком парковок (5.4)
        2. При нажатиина кнопку Ужезарегистрированы**?** происходит перенаправление пользователя на страницу регистрации (5.3)
  3. Страница с формой регистрации
     1. Доступна только неавторизованным пользователям
     2. Содержит
        1. Тестовое поле для ввода почты
        2. Текстовое поле для ввода пароля
        3. Кнопку Зарегистрироваться
        4. КнопкуУжеестьаккаунт**?**
     3. Действия
        1. При нажатии на кнопкуЗарегистрироватьсяотправляется запрос на API (4.4.1), в котором передаются введенные данные формы. Если введеные данные верны, то в ответе возвращается access токен, который сохраняется в куки, а также происходит переадресация на страницу со списком парковок (5.4)
        2. При нажатиина кнопку Уже есть аккаунт? перенаправляет пользователя на страницу авторизации (5.2)
  4. Страница со списком парковок
     1. Доступна всем пользователям
     2. Содержит
        1. Кнопку **«**Новыйабонемент**»**
           1. Доступна только авторизированным пользователям (неадминистраторам)
        2. Текстовое поле для поиска парковки по названию
        3. Парковки в виде карточек. Каждая карточка состоит из:
           1. Фото парковки
           2. Названия парковки
           3. Кнопки **«**Открыть**»**
           4. Кнопка **«**Добавить**»**

Доступна только авторизированным пользователям (неадминистраторам)

* + 1. Действия
       1. При нажатии на кнопку **«**Открыть**»** (5.4.2.2.3) происходит перенаправление пользователя на страницу с подробным описанием парковки (5.6)
       2. При нажатии на кнопку **«**Добавить**»** (5.4.2.2.4) отправляется запрос на API (4.1.6), а также меняется состояние кнопки **«**Новыйабонемент**»** (5.4.2.1)
       3. При нажатии на кнопку **«**Новыйабонемент» (5.4.2.1) происходит перенаправление пользователяна страницу с черновым абонементом (5.7), если оно существует у пользователя
       4. При каждом наборе символов в поле поиска (5.4.2.2) отправляется запрос на API (4.1.1), который возвращает список парковок, отфильтрованный по названию
  1. Страница с таблицей парковок
     1. Доступна только администраторам
     2. Содержит
        1. Текстовое поле для поиска парковки по названию
        2. Таблицу с парковкой, содержащую столбцы:
           1. № парковки
           2. Название парковки
           3. Адрес парковки
           4. Количество мест на парковке
     3. Действия
        1. При нажатии на любую строку таблицы (5.2.2.2) происходит перенаправление пользователя на страницу редактирования парковки (5.9)
        2. При наборе символов в поле поиска (5.5.2.1) отправляется запрос на API (4.1.1), который фильтрует таблицу парковок по названию
  2. Страница с подробным описанием парковки
     1. Доступна всем пользователям
     2. Содержит:
        1. Название парковки
        2. Фото парковки
        3. Адрес парковки
        4. Количество мест на парковке
  3. Страница чернового абонемента
     1. Доступна только авторизованным пользователям (неадминистраторам)
     2. Содержит
        1. Карточек парковок в абонементе. Каждая карточка содержит:
           1. Название парковки
           2. Кнопку **«**Удалить**»**
        2. Кнопку **«**Удалить**»**
        3. Кнопку **«**Отправить**»**
     3. Действия
        1. При нажатии на кнопку **«**Удалить**»** (5.7.2.1.2) отправляется запрос на API (4.2.8), который удаляет парковки из абонемента
        2. При нажатии на кнопку **«**Удалить**»** (5.7.2.5) отправляется запрос на API (4.2.7), который удаляет черновой абонемент и перенаправляет пользователя на страницу со списком парковок (5.4)
        3. При нажатии на кнопку **«**Отправить**»** отправляетсязапрос на API (4.2.5), который формирует новый абонемент и после чего перенаправляет пользователя на страницу с историей его абонементов (5.8)
  4. Страница с историей абонементов
     1. Доступна только авторизованным пользователям
     2. Содержит
        1. Таблицу с историей абонементов, содержащую столбцы
           1. № Абонемента
           2. Статус
           3. Названия парковок
           4. Дату формирования абонемента
           5. Время въезда
           6. Кнопка подтвердить

Доступен только администраторам

* + - * 1. Кнопка отклонить

Доступен только администраторам

* + - 1. Фильтр по пользователю
         1. Доступен только администраторам
         2. Выполняется на стороне клиента
      2. Фильтр по дате формирования открытия в виде диапазона
      3. Фильтр по статусу абонемента
         1. Пользователю доступны статусы

Черновик

В работе

Завершён

Отклонён

Удалён

* + - * 1. Администратору доступны статусы

В работе

Завершён

Отклонён

* + 1. Действия
       1. При изменении состояния фильтров (5.8.2.3 и 5.8.2.4) отправляется запрос на API (4.2.1), который обновляет таблицу с абонементами
       2. При нажатии на любую строчку таблицы происходит перенаправление пользователя на страницу абонемента (5.8.2)
       3. При нажатии на кнопку Подтвердить(5.8.2.1.6) отправляется запрос на API (4.2.6), который меняет статус абонемента с «в работе»на «завершён»
       4. При нажатии на кнопку Отклонить (5.8.2.1.7) отправляется запрос на API (4.2.6), который меняет статус абонемента с «в работе» на «завершён»
  1. Страница редактирования / добавления парковки
     1. Доступна только администраторам
     2. Содержит
        1. Файловое поле, для загрузки / изменения фото парковки
        2. Текстовое поле для ввода / изменения названия парковки
        3. Текстовое поле для ввода / изменения адреса парковки
        4. Текстовое поле для ввода количества парковочных мест на па парковке
        5. Кнопку **«**Удалить**»**
        6. Кнопку **«**Сохранить**»**
        7. Кнопку **«**Добавить**»**
     3. Действия
        1. При нажатии на кнопку **«**Удалить**»** (5.9.2.6)выполняется запрос на API (4.1.4), а также происходит переадресация на страницу со списком парковок (5.4)
        2. При нажатии на кнопку **«**Сохранить**»** (5.9.2.7)выполняется запрос на API (4.1.4), а также происходит переадресация на страницу со списком парковок (5.4)
        3. При нажатии на кнопку **«**Добавить**»** (5.9.2.8)выполняются два запроса на API: сначала на добавление новой парковки (4.1.5), а затем на обновление парковки (4.1.3), также происходит переадресация на страницу со списком парковок (5.4)

## Требования к программному обеспечению:

* 1. Серверная часть
     1. Операционная система (любая из)
        1. Linux **(**6.4.12 и выше)
        2. Windows (22H2 и выше)
     2. Minio (RELEASE.2023-12-14T18-51-57Z и выше)
     3. Redis (7.2 и выше)
     4. Python (3.10 и выше) с фреймворком Django (4.1 и выше)
     5. СУБД PostgreSQL (16.1 и выше)
     6. Доступ к интернету
  2. Клиентская часть
     1. Операционная система (любая из)
        1. Windows (22H2 и выше)
        2. MacOS (11.7.10 и выше)
        3. Linux **(**6.4.12 и выше)
     2. Веб-браузер (любой из)
        1. Safari (16.5.2 и выше)
        2. Chrome (119.0.6045 и выше)
        3. Opera (105.0.4970.16 и выше)
        4. Mozilla Firefox (121.0 и выше)
     3. Доступ к интернету

## Требования к аппаратному обеспечению:

* 1. Серверная часть
     1. Процессор минимум 2-ядерный с частотой от 2 ГГц
     2. Оперативная память от 4 Гб
     3. Место на жестком диске от 2 Гб
  2. Клиентская часть
     1. Процессор с частотой от 1ГГц
     2. Оперативная память от 512 Мб