МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
 ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
 **«Дальневосточный федеральный университет»**

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ  
 Департамент программной инженерии и искусственного интеллекта**

**РАЗРАБОТКА И РЕАЛИЗАЦИЯ АВТОМАТИЗИРОВАННЫЙ СИСТЕМЫ БРОНИРОВАНИЯ ПАРКОВОЧНЫХ МЕСТ**

КУРСОВАЯ РАБОТА

по дисциплине «Технологии коллективной промышленной разработки информационных систем» по образовательной программе подготовки бакалавров по направлению 02.03.03 «Технология программирования»

Выполнили студенты гр. Б9121-02.03.03тп  
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Гордеева А. О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Тархов В. Е.

Руководитель:  
Старший преподаватель ДПИиИИ ИМиКТ ДВФУ  
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Иваненко Ю. С.

г. Владивосток  
 2025

**Оглавление**

[Введение 4](#_Toc188291683)

[1. План проекта 6](#_Toc188291684)

[2. Разработка регламента проведения инспекции 7](#_Toc188291685)

[2.1. Верификация рабочих продуктов 7](#_Toc188291686)

[2.2. Критерии отнесения к формальной инспекции 7](#_Toc188291687)

[2.3. Перечень ролей участников инспекции и их обязанности 8](#_Toc188291688)

[2.4. Этапы инспекции 8](#_Toc188291689)

[2.5. Порядок организации 9](#_Toc188291690)

[2.6. Порядок подготовки и проведения 9](#_Toc188291691)

[2.7. Перечень статусов и степени важности замечаний 10](#_Toc188291692)

[2.8. Порядок верификации замечаний 11](#_Toc188291693)

[3. Модель состояний задач 12](#_Toc188291694)

[3.1. Перечень возможных состояний задачи и их интерпретация 12](#_Toc188291695)

[3.2. Правила создания новой задачи 12](#_Toc188291696)

[3.3. Правила перехода задачи из состояния в состояние 13](#_Toc188291697)

[4. Презентация проекта 15](#_Toc188291698)

[5. Требования к проекту 21](#_Toc188291699)

[5.1. Требования к подсистеме UI 21](#_Toc188291700)

[5.2. Требования к подсистеме S 21](#_Toc188291701)

[5.3. Требования к подсистеме BS 22](#_Toc188291702)

[6. Архитектура проекта 23](#_Toc188291703)

[7. Измерения проекта 26](#_Toc188291704)

[8. Перечень задач проекта 28](#_Toc188291705)

[9. Рекомендации по кодированию 31](#_Toc188291706)

[9.1. Рекомендации 31](#_Toc188291707)

[9.2. Требования 32](#_Toc188291708)

[9.3. Запреты 32](#_Toc188291709)

[Список литературы 34](#_Toc188291710)

# Введение

Промышленная разработка информационных систем включает в себя множество этапов, начиная от разработки плана проекта, заканчивая тестированием проекта для чего, очевидно необходимо множество специалистов различных профилей, а также унифицированные методы коммуникации между ними, с помощью которых можно разделить обязанности членов команды по их специализации.

Исходя из описанного выше необходимо использовать определенные технологии коллективной разработки для повышения эффективности работы в группе и соответствия конечного продукта заявленным требованиям.

В данной курсовой работе рассматривается задача коллективной разработки проекта «Автоматизированная система бронирования парковочных мест» и составление технической документации к нему.

Таким образом, целью курсовой работы является разработка разработки проекта «Автоматизированная система бронирования парковочных мест» с использованием подходов коллективной промышленной разработки.

Для достижения поставленной цели необходимо решить следующие задачи:

* разработать план проекта;
* разработать регламент проведения инспекции;
* разработать модель состояний задач;
* разработать презентацию проекта;
* разработать требования к проекту;
* разработать архитектуру проекта;
* разработать измерения проекта;
* разработать перечь задач проекта;
* разработать рекомендации по кодированию;
* разработать план тестирования проекта;
* протестировать проект.

# План проекта

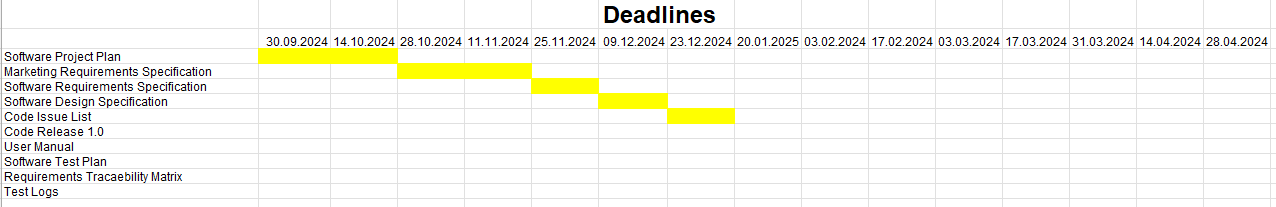


Рисунок 1 – План проекта с финальными датами

Т а б л и ц а 1 - Команда проекта

|  |  |
| --- | --- |
| **Команда** | |
| **Team Leader** | **Тархов В.** |
| **Coder 1** | **Тархов В.** |
| **Coder 2** | **Гордеева А.** |
| **Build Engineer** | **Тархов В.** |
| **Technical Writer** | **Гордеева А.** |

На рисунке 1 можно увидеть план проекта с названием этапов, их описания и финальных дат выполнения. В таблице 1 перечислен состав команды и роли участников.

# Разработка регламента проведения инспекции

## Верификация рабочих продуктов

Верификация рабочих продуктов является неотъемлемой частью процесса по обеспечению их качества. Современной технологией программирования выработаны специальные стандарты, подходы и механизмы проведения верификаций рабочих продуктов, в формате так называемых инспекций.

Инспекция – это мероприятие по обеспечению качества рабочих продуктов проектов по разработке ПО и иной деятельности, которая проводится разработчиками, возможно – с участием представителей заказчика.

Концептуально инспекция имеет следующие цели:

* обнаружить ошибки в функциях, логике, содержании или реализации рабочих продуктов на ранних этапах их разработки и предотвратить их наследование;
* рационально донести замысел или реализацию продукта до всех заинтересованных лиц (через их участие);
* оптимизировать, оценить или улучшить рабочий продукт.

## Критерии отнесения к формальной инспекции

В таблице 2 перечислены критерии отнесения к формальной инспекции.

Т а б л и ц а 2 – критерии для различных типов рабочих продуктов

|  |  |  |
| --- | --- | --- |
|  | **Формальная инспекция в случае CRUD-**операций **над** | **Неформальная инспекция в случае CRUD-операций над** |
| **Требования** | >3 требованием | При любом изменении |
| **Документы дизайна** | >3 объектом дизайна внешнего вида готового продукта |
| **Код** | >50 строчками |
| **Тесты** | >5 тестами |

## Перечень ролей участников инспекции и их обязанности

В таблице 3 перечислены роли и обязанности, которые они подразумевают.

Т а б л и ц а 3 - Роли и их описание

|  |  |
| --- | --- |
| **Роль** | **Обязанности** |
| Автор (Coder 1) | Участник, сделавший инспектируемые изменения в существующем рабочем продукте |
| Инспектор, проверяющий (Coder 2) | Участник, ответственный за эффективную проверку инспектируемого рабочего продукта. |

Так как в команде два человека, то роли не фиксированы и менялись в зависимости от того, кто внес изменения в проект (тот выступал в роли автора, соответственно).

## Этапы инспекции

1. Планирование инспекции.

При планировании инспекции коллективно выбирается дата, время, формат (очный или заочный) и платформа (при заочной инспекции) проведения инспекции.

1. Подготовка к инспекции.

Инспектор самостоятельно изучает предоставленный для инспекции рабочий продукт, используя накопленный опыт и стандарты.

1. Собрание по инспекции.

На собрании происходит обсуждение замечаний и рекомендаций инспектора по рабочему продукту. На собрании по инспекции обязательно присутствует инспектор и автор рабочего продукта, требующего инспекции.

1. Завершение инспекции.

Если рабочий продукт требует доработки, то автор фиксирует все замечания и рекомендации инспектора, разрабатывает план предстоящих работ и далее согласовывает его с инспектором. Если рабочий продукт не требует доработки, то инспектор подтверждает слияние рабочей ветки в «develop» ветку.

## Порядок организации

1. **Автор** загружает изменения в репозиторий и приглашает инспектора для проверки.
2. **Инспектор** готовиться к инспекции и составляет протокол ошибок, которые отправляет автору**.**
3. **Автор** исправляет ошибки и отправляет исправленную версию кода инспектору.
4. **Инспектор** проверяет корректность исправлений и подтверждает их, завершая инспекцию.

## Порядок подготовки и проведения

1. **Уведомление:** Автор уведомляет инспектора о назначении проверки за 1-2 дня.
2. **Подготовка:** Инспектору предоставляется от 3 до 7 дней для изучения изменений и составления протокола в зависимости от объема изменений и сложности кода.
3. **Собрание по инспекции:** оперативно обсуждаются ошибки и способы их исправления.
4. **Исправления и проверка:** после исправления ошибок автор отправляет обновленный код инспектору, который проверяет и подтверждает их корректность.

## Перечень статусов и степени важности замечаний

В таблице 4 перечислены замечания и их описания.

Т а б л и ц а 4 - Замечания и их описание

|  |  |
| --- | --- |
| **Статус замечания** | **Описание** |
|  | |
| **Дефект (Defect)** | Ошибка, найденная в коде на другой фазе разработки. |
| **Ошибка (Error)** | Ошибка присутствует в коде на той же фазе разработки. |
| **Комментарий (Comment)** | Предложения, улучшения и рекомендации для будущего релиза. |
| **Замечание для исследования (Investigate)** | Проблема требует дополнительного анализа и не может быть решена сразу. |

В таблице 5 перечислены степени важности замечаний и их описания.

Т а б л и ц а 5 - Степень важности и их описание

|  |  |
| --- | --- |
| **Степень важности** | **Описание** |
|  | |
| **Критическая (Critical)** | Ошибка приводит к невозможности запуска программы. |
| **Особо важная (Major)** | Ошибка влияет на выполнение программы и вызывает сбои. |
| **Средняя (Moderate)** | Ошибка проявляется в специфических условиях и временно. |
| **Мелкая (Minor)** | Ошибка во вводе/выводе данных или валидации данных. |
| **Другие (Other)** | Предложения по улучшению производительности или оптимизации кода. |

## Порядок верификации замечаний

После завершения обсуждения в рамках инспекции:

1. **Автор исправляет ошибки**, отмечая все исправления в протоколе;
2. **Инспектор проверяет исправления**, подтверждая, что все замечания учтены в обновленном коде;
3. После подтверждения исправлений формальная **инспекция считается успешно завершенной**.

# Модель состояний задач

## Перечень возможных состояний задачи и их интерпретация

Каждая задача в процессе разработки отражает этапы рабочего процесса и проходит через определенные состояния. Важно понимать, что каждое состояние имеет свою цель и соответствующие критерии.

Ниже представлен перечень состояний задач для проекта:

1. **New** – новая подзадача.
2. **Analysis** – в процессе анализа. В это состояние подзадачу переводит член команды после того, как начнёт её анализ.
3. **Forward** – в данном случае имеет значение «переданный на дальнейшую разработку». В это состояние задача переводится CCB после анализа при назначении задачи на разработку конкретному члену команды.
4. **Coding** – кодирование. В это состояние задача переводится разработчиком при начале работы по кодированию, связанному с задачей.
5. **Inspected** – проинспектировано. В это состояние задача переводится разработчиком после завершения кодирования и инспектирования изменений рабочего продукта.
6. **Integrated** – интегрировано. Переводится членом команды, осуществляющим интеграцию изменений в основную ветку рабочего продукта после успешной интеграции этих изменений.
7. **Tested** – протестировано. Переводится членом команды, осуществляющим тестирование изменений в рабочий продукт (tester).
8. **Closed** – закрыто. В это состояние задача переводится CCB по результатам отчёта о тестировании сделанных изменений.

## Правила создания новой задачи

1. **Кто может создавать задачи:**
   1. Задачи могут создавать **Coder 1 или Coder 2**, в зависимости от их ответственности в проекте;
   2. Также, в случае критической ошибки или необходимости улучшения процесса, задачи могут инициироваться **Team Lead**.
2. **Когда следует создавать задачу:**
   1. Для исправления ошибок в коде или тестах;
   2. Для добавления новых функций или улучшения существующего функционала;
   3. Для реализации изменений по результатам обсуждений в ходе код-ревью или инспекций;
   4. Для создания задачи в соответствии с фидбеком от QA-инженеров или пользователей системы.
3. **Инструменты для создания задач:**
   1. GitHub Issues, где задачи группируются по релизам и критическим обновлениям;
   2. Используются шаблоны задач для более четкого описания требований и ожидаемых результатов.

## Правила перехода задачи из состояния в состояние

Состояния задач всегда идут последовательно друг за другом, в некоторых случаях пункты могут опускаться или повторяться. Схема перехода из состояния в состояние показана на рисунке 2.

Изображение выглядит как диаграмма, текст, снимок экрана, линия

Автоматически созданное описание

Рисунок 2 – Состояния задачи

# Презентация проекта

На рисунке 3 представлена титульная страница презентации.

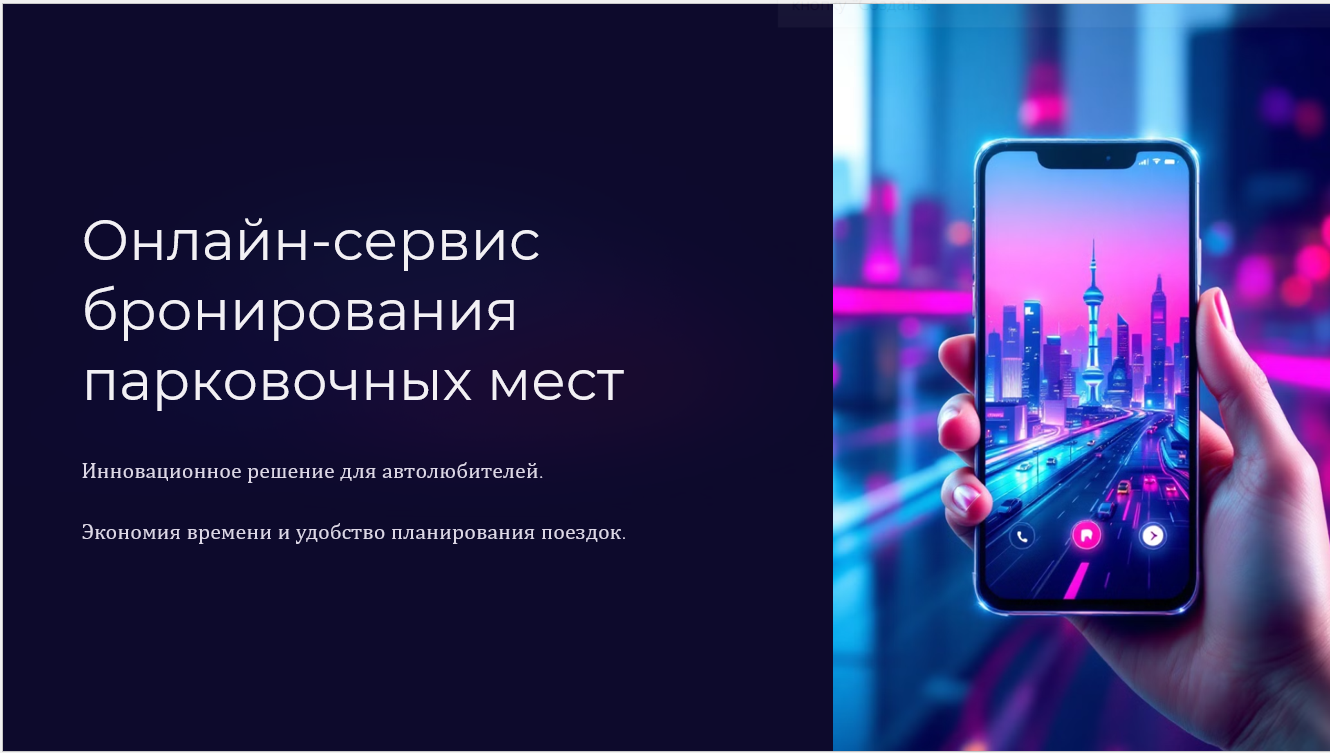


Рисунок 3 - Титульная страница

На рисунке 4 демонстрируется команда разработки проекта.

Изображение выглядит как текст, снимок экрана, видео, СМИ

Автоматически созданное описание

Рисунок 4 - Команда

Описание разрабатываемого проекта представлены на рисунке 5.

Изображение выглядит как текст, Мобильный телефон, снимок экрана, Устройство связи

Автоматически созданное описание

Рисунок 5 – Описание проекта

На рисунке 6 изображены ключевые преимущества проекта.

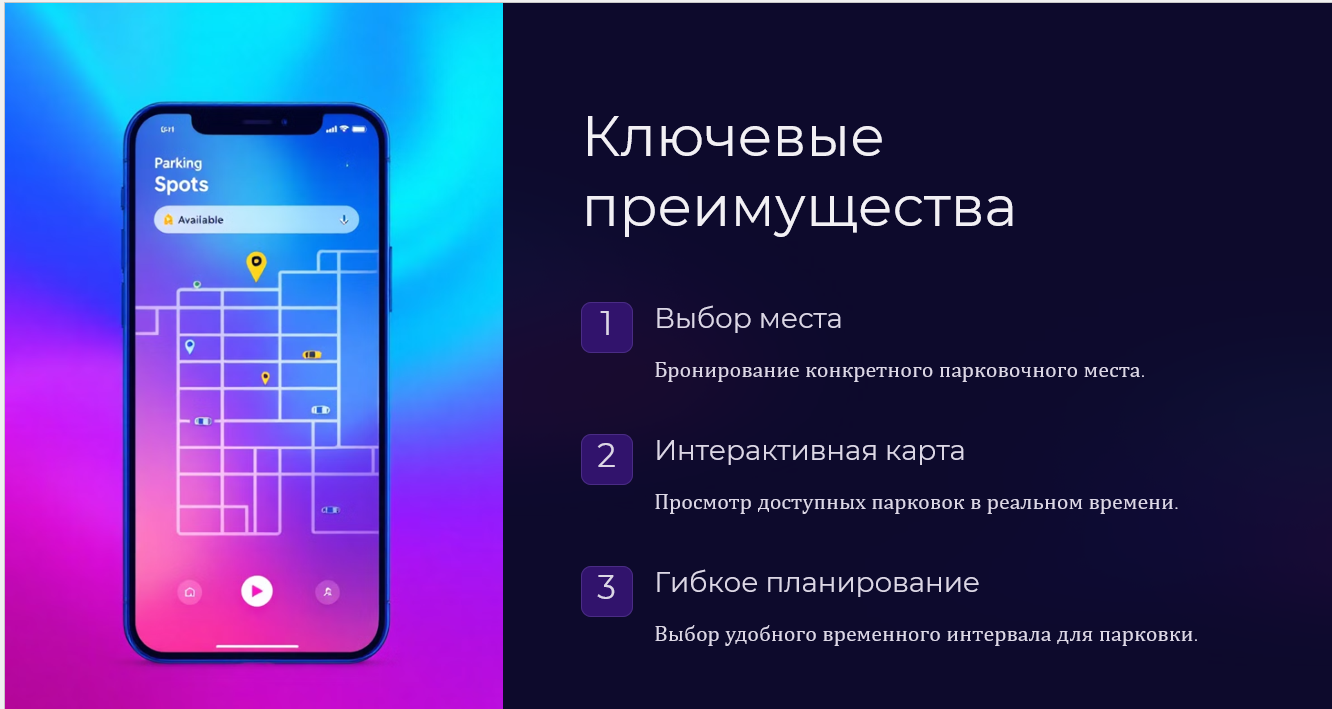


Рисунок 6 – Преимущества проекта

Перспективы развития проекта представлены на рисунке 7.

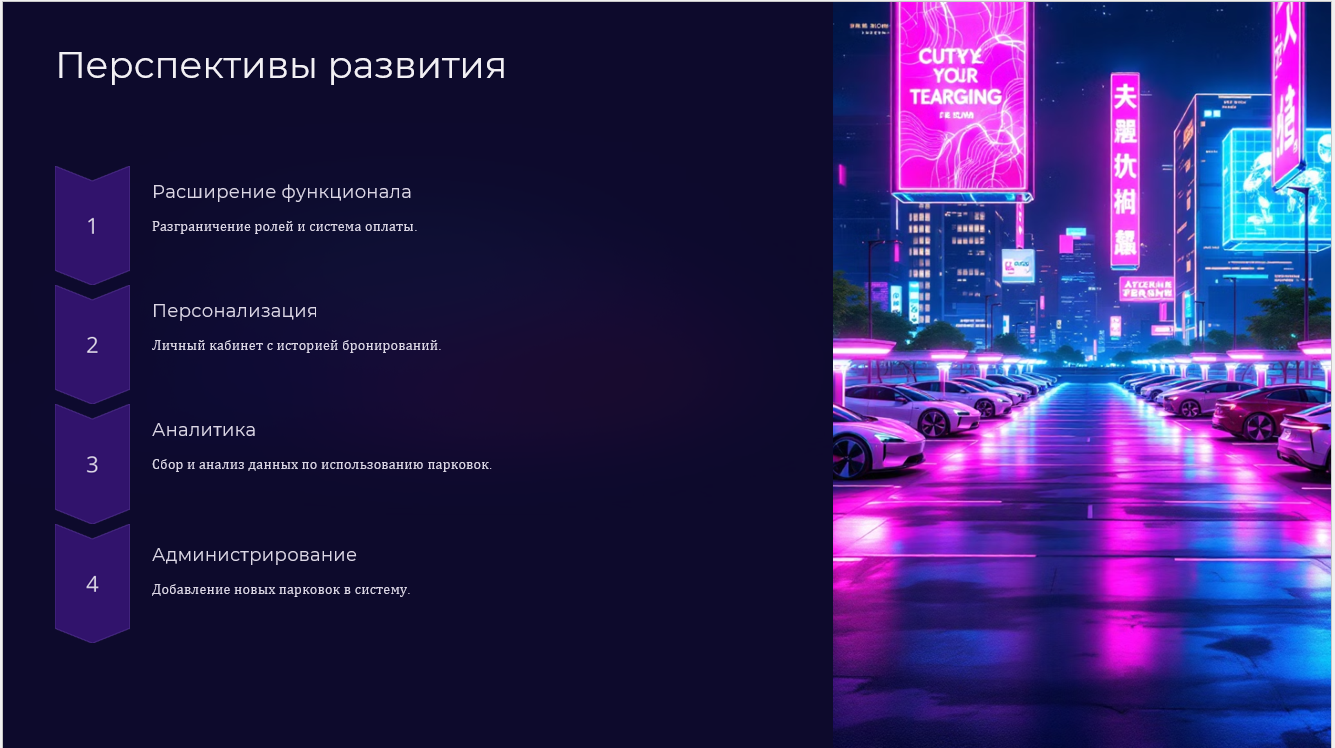


Рисунок 7 – Перспективы развития проекта

Интерфейс проекта представлен на рисунках 8 – 13.

Изображение выглядит как текст, снимок экрана, Мобильный телефон, Мобильное устройство

Автоматически созданное описание

Рисунок 8 – Экран начала работы с приложением

Изображение выглядит как текст, электроника, снимок экрана, Мобильный телефон

Автоматически созданное описание

Рисунок 9 – Экран для регистрации

Изображение выглядит как текст, снимок экрана, Мобильный телефон

Автоматически созданное описание

Рисунок 10 – Экран отображения карты с парковками

Изображение выглядит как текст, электроника, снимок экрана, пульт дистанционного управления

Автоматически созданное описание

Рисунок 11 – Экран отображения карты парковки

Изображение выглядит как текст, электроника, снимок экрана, Мобильный телефон

Автоматически созданное описание

Рисунок 12 – Экран подтверждения выбора места

Изображение выглядит как текст, снимок экрана, Мобильный телефон, Мобильное устройство

Автоматически созданное описание

Рисунок 13 – Экран оповещения об успешном бронировании

# Требования к проекту

Программный продукт «Онлайн-сервис бронирования парковочных мест» состоит из следующих подсистем:

* **User Interface (UI)** – пользовательский интерфейс.
* **Security (S)** – модуль регистрации и авторизации.
* **Booking Subsystem (BS)** – модуль бронирования

## Требования к подсистеме UI

**Требование REQ\_UI\_001**

Пользователю должен быть доступен режим карты, на котором визуализируются результаты работы остальных модулей в хорошем качестве.

**Требование REQ\_UI\_002**

При нажатии на иконку парковки пользователь должен получить новое окно с возможностью посмотреть информацию о выбранной парковке: кол-во свободных мест, стоимость бронирования места за час и доступные для бронирования места.

**Требование REQ\_UI\_003**

При нажатии на доступное для бронирования место пользователь должен получить новое окно с возможностью как посмотреть информацию о выбранном месте, так и изменить временной промежуток для бронирования.

**Требование REQ\_UI\_004**

При нажатии на кнопку «Забронировать» пользователь должен получить новое окно с уведомлением об успешном бронировании места.

## Требования к подсистеме S

**Требование REQ\_S\_001**

Пользователю должна быть предоставлена возможность регистрации в приложении.

**Требование REQ\_S\_002**

Пользователю должна быть предоставлена возможность авторизации в приложении

## Требования к подсистеме BS

**Требование REQ\_BS\_001**

На клиент должен приходить список парковок с координатами, адресом и количеством свободных парковочных мест.

**Требование REQ\_BS\_002**

На клиент должен приходить список парковочных мест с указанием доступности каждого.

**Требование REQ\_BS\_003**

При бронировании конкретного парковочного места должна проходить проверка на его доступность в выбранные временные рамки.

# Архитектура проекта

Архитектура программного обеспечения относится к фундаментальным структурам программной системы и дисциплине создания таких структур и систем. Каждая структура включает элементы программного обеспечения, отношения между ними, а также свойства как элементов, так и отношений. Архитектура программной системы – это метафора, аналогичная архитектуре здания. Он функционирует как план для системы и проекта разработки, в котором излагаются задачи, которые должны быть выполнены командами разработчиков.

Система представляет собой клиент-серверное приложение. На рисунке 14 представлена архитектурно-контекстная диаграмма клиентской части приложения.

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

Рисунок 14 - Архитектурно-контекстная диаграмма клиентской части

На рисунке 15 представлена архитектурно-контекстная диаграмма серверной части приложения.

Изображение выглядит как текст, диаграмма, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 15 - Архитектурно-контекстная диаграмма серверной части

На рисунке 16 представлена Use-cases диаграмма взаимодействия клиента с системой.

Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание

Рисунок 16 - Use-cases диаграмма

На рисунке 17 представлена ER-диаграмма.

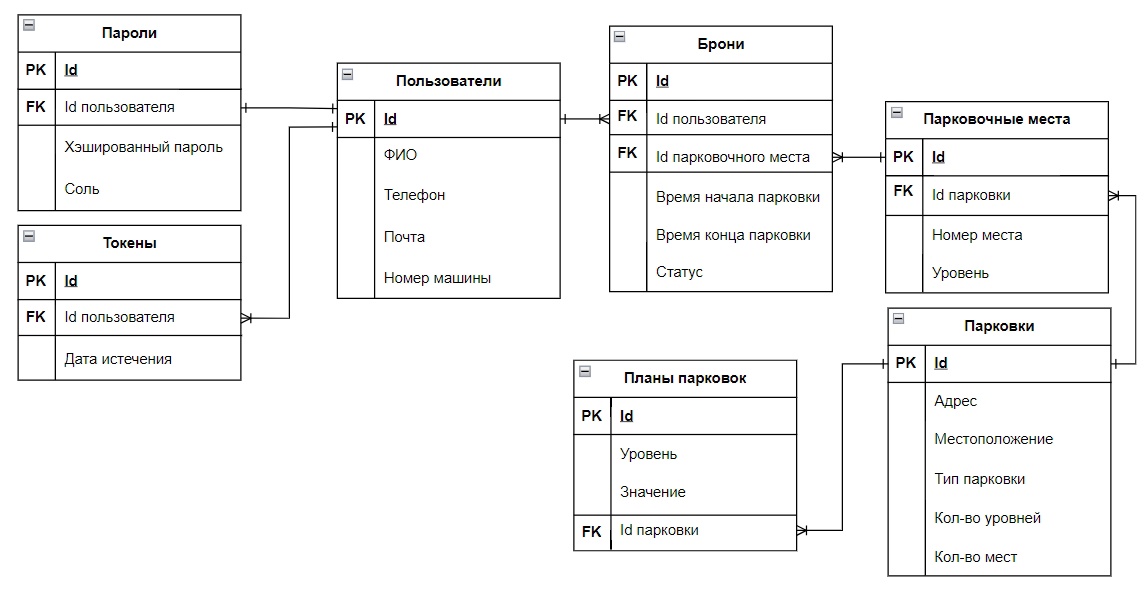


Рисунок 17 – ER-диаграмма

# Измерения проекта

Контроль за производственным процессом и его результатами является ключевым видом деятельности на современном предприятии, производящем программное обеспечение на заказ. В силу специфики такого продукта, как программное обеспечение, для оценки эффективности процесса разработки и качества конечного продукта применяются особые методы. Комплекс мероприятий, направленных на количественную оценку эффективности работы компании, называется программой измерений компании. Программа измерений выполняется как в рамках отдельных проектов, осуществляемых компанией, так и в рамках определённых видов деятельности компании.

**Метрики для оценки эффективности процесса разработки**

project\_size – размер разрабатываемого, сумма количества документов дизайна, количества требований, количества тестов и количества строк кода без комментариев.

error\_count – общее количество найденных ошибок в процессе разработки.

errors – общее количество найденных ошибок на этапе разработки

missed\_errors – общее количество не найденных ошибок на этапе разработки.

**Метрика качества программного продукта**

c\_req – количество требований, e\_req – количество требований в которых допущена ошибка.

c\_design – количество документов дизайна (диаграмм, схем и т.д.), e\_design – количество документов дизайна в которых допущена ошибка.

e\_coding – количество ошибок допущенных в коде.

c\_test – количество тестов, e\_test -количество тестов в которых допущена ошибка.

Каждое из слагаемых может выступать в роли отдельной метрики.

# Перечень задач проекта

**Задачи для подсистемы «UI»**

*Задача TASK\_UI\_001*

Название: Реализация режима карты.

Описание: Реализовать функционал отображения карты в приложении, на которой визуализируются парковки с учетом данных, предоставленных подсистемой BS. Карта должна поддерживать масштабирование, перемещение и быть удобной для пользователя. Качество визуализации должно быть высоким, обеспечивающим четкое отображение всех элементов.

*Задача TASK\_UI\_002*

Название: Окно информации о парковке.

Описание: Реализовать функционал, позволяющий пользователю, при нажатии на иконку любой парковки на карте, попадать на страницу, отображающую информацию о выбранной парковке. Обеспечить показ количества свободных мест, стоимости бронирования за час и списка доступных для бронирования мест. Вся информация должна динамически обновляться в соответствии с данными подсистемы BS.

*Задача TASK\_UI\_003*

Название: Интерфейс выбора и бронирования места.

Описание: Реализовать функционал, позволяющий пользователю, после выбора доступного для бронирования места, открыть страницу с информацией о данном месте. Пользователь должен иметь возможность выбрать или изменить временной промежуток для бронирования. Интерфейс должен быть интуитивно понятным и информативным.

*Задача TASK\_UI\_004*

Название: Уведомление об успешном бронировании.

Описание: Реализовать функционал, позволяющий отображать в интерфейсе окно уведомления о результате бронирования. После нажатия на кнопку «Забронировать», в случае успешного проведения операции, должно отображаться сообщение об успешном бронировании.

**Задачи для подсистемы «S»**

*Задача TASK\_S\_001*

Название: Функционал регистрации.

Описание: Реализовать регистрацию в приложении. В ходе регистрации пользователь должен иметь возможность указать ФИО, номер телефона, адрес электронной почты, регистрационный номера и пароль. Каждое поле должно быть провалидировано.

*Задача TASK\_S\_002*

Название: Функционал авторизации.

Описание: Реализовать авторизацию в приложении. Пользователь должен иметь возможность войти в свою учетную запись, введя почту/телефон и пароль. Основана авторизация должна быть на механизме JWT токенов.

**Задачи для подсистемы «BS»**

*Задача TASK\_BS\_001*

Название: Получение информации о доступных парковках.

Описание: Реализовать получение из бд и отправку на клиент информации о доступных парковках города. Среди информации, отправляемой по каждой парковке должно быть: координаты парковки, адрес парковки, кол-во уровней на парковке и количество мест свободных на выбранный период времени.

*Задача TASK\_BS\_002*

Название: Получение информации о доступных парковочных местах.

Описание: Реализовать получение из бд и отправку на клиент информации о доступных парковочных местах выбранной пользователем парковки. Среди отправляемой информации должно быть: схема парковки, представленная одним или несколькими svg-изображениями, идентификаторы парковочных мест, а также статусы их занятости на выбранный период времени.

*Задача TASK\_BS\_003*

Название: Функционал бронирования парковочного места.

Описание: Реализовать проверку возможности бронирования парковочного места на выбранный период времени, а также сохранение записи в бд в случае успешного бронирования.

# Рекомендации по кодированию

Для создания качественного кода на любом языке программирования, обладающего таким свойствами, как удобочитаемость (readability) и понятность (understandability), необходимо следовать хорошо определённым стандартам и руководствам. Особенно это актуально при коллективной разработке программ.

Любой стандарт кодирования призван определить набор правил, которые способствуют разработке более единообразного кода и минимизации числа общераспространенных ошибок в нем, не ущемляя при этом права разработчика на творчество.

## Рекомендации

1. **Рекомендуется не допускать строки кода длиной более 120 символов.** Это улучшит читаемость кода и позволит не тратить лишнее время на пролистывание ползунка до окончания строки.
2. **При написании условий и циклов рекомендуется использовать фигурные скобки, даже если тело условия/цикла состоит из одного выражения.** Это более явно обозначит тело условия/цикла и при необходимости добавления новой логики избавит от необходимости добавления скобок.
3. **При открытии ресурсов, требующих обязательного закрытия, рекомендуется использовать конструкцию try-with-resources (Java).** Это улучшит читаемость кода и избавит от необходимости добавления блока finally.
4. **При обработке коллекций, требующей несколько операций, рекомендуется использовать возможности библиотеки Stream API (Java).** Это улучшит читаемость кода и сократит кол-во строк.
5. **Рекомендуется использовать конструкции `guard` для проверки условий, которые могут привести к раннему завершению выполнения функции (Swift)**. Это улучшит читаемость кода, делая основную логику более линейной и понятной.
6. **При обработке коллекций рекомендуется использовать замыкания и функции высшего порядка - map, filter, reduce (Swift).** Это может улучшить ясность и краткость кода.

## Требования

1. **Требуется классы и интерфейсы именовать в UpperCamelCase, методы и переменные в – lowerCamelCase, и константы в – SNAKE\_CASE.** Это позволит по стилю нейминга сразу понимать тип конструкции языка.
2. **Требуется в импортах указывать конкретную сущность, не использовать \*.** Это позволит быстрее определять откуда берется та или иная сущность.
3. **Требуется давать осмысленные названия сущностям.** Название сущности должно давать о понимании о её предназначении в коде.
4. **Для полей, описывающих состояние класса, требуется использовать как можно более строгий модификатор доступа.** Это позволит обеспечить инкапсуляцию и защитить внутреннюю логику.

## Запреты

1. **Запрещается использование ‘force unwrap’, так как это может привести к крашам приложения.** Вместо этого лучше использовать безопасное разворачиваниее – ‘optional binding’ или ‘nil-coalescing’ (Swift).
2. **Запрещается оставлять ресурсы без закрытия.** Все открытые ресурсы нужно закрывать, если они больше не используются.
3. **Запрещается использовать магические числа и строки.** Все значения должны быть вынесены в константы или конфигурацию.

# Список литературы

1. Гриняк В.М. Лекции по дисциплине «Технологии коллективной промышленной разработки информационных систем». Электронный вариант.