МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ ЭНЕРГЕТИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ В.И.ЛЕНИНА»

Факультет Заочного и вечернего обучения
(полное наименование факультета)

Кафедра Программного обеспечения компьютерных систем
(полное наименование выпускающей кафедры)

Направление подготовки (специальность) 09.03.04 Программная инженерия
(код, наименование направления подготовки / специальности)

Направленность (профиль) образовательной программы
Разработка программно-информационных систем
(наименование направленности (профиля) образовательной программы / специализации)

УТВЕРЖДАЮ

Заведующий кафедрой

_____ Косяков С.В.
(подпись) *(Ф.И.О.)*

«____»_____20___ г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к выпускной квалификационной работе

на тему **«Разработка веб-приложения для заказа ритуальных памятников»**

Обучающийся: _____ Шулайкин Д.А.
(подпись) *(Ф.И.О.)*

Руководитель: ___к.т.н., доцент _____ Зубков В.П.
(уч. степень, уч. звание) *(подпись)* *(Ф.И.О.)*

Консультанты:_____ _____ _____
(уч. степень, уч. звание) *(подпись)* *(Ф.И.О.)*

Иваново 2021

РЕФЕРАТ

Объем 39 с., 1 кн. , 14 рис., 0 табл., 2 источн., 2 прил.

ВЕБ-РАЗРАБОТКА, АНАЛИЗ ТРЕБОВАНИЙ, ПРОЕКТИРОВАНИЕ, ВЕБ-ПРИЛОЖЕНИЕ, REACTJS, NODEJS, GRAPHQL, MONGODB, KUBERNETES

Объектом работы является процесс разработки веб-приложения для организации по продажам и изготовлению памятников.

Цель работы – создать веб-приложение, позволяющее автоматизировать бизнес процессы организации по продажам и изготовлению памятников..

Методы, используемые в работе: фиксация результатов анализа предметной области и проектирования архитектуры системы при помощи диаграмм UML.

Результаты работы: комплект документации к системе, проект системы, разработанное веб-приложение.

Область применения результатов: коммерческое использование информационной системы.

Макет оформления реферата на иностранном (английском) языке:

ABSTRACT

Volume 39 p., 1 b., 14 fig., 0 tabl., 2 sourc., 2 append.

WEB DEVELOPMENT, REQUIREMENTS ANALYSIS, DESIGN, WEB APPLICATION, REACTJS, NODEJS, GRAPHQL, MONGODB, KUBERNETES

The object of the work is the process of developing e-commerce web application for memorial workshop organization.

The aim of the work is to create a web application that allows you to streamline business processes in the memorial workshop organization.

Methods: fixing the results of domain analysis and system architecture design using UML diagrams.

Results: set of documentation for the system, developed web application.

Application field: commercial use of the information system.

Определения, обозначения сокращения

БД - База данных ОС - Операционная система ПО - Программное обеспечение СУБД - Система управления базами данных РОС - proof of concept ЯП - язык программирования REST - Representational state transfer SPA - Single page application k8s - Kubernetes

Содержание

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ ЭНЕРГЕТИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ В.И.ЛЕНИНА»

Факультет Заочного и вечернего обучения
(полное наименование факультета)

Кафедра Программного обеспечения компьютерных систем
(полное наименование выпускающей кафедры)

Направление подготовки (специальность) 09.03.04 Программная инженерия
(код, наименование направления подготовки / специальности)

Направленность (профиль) образовательной программы
Разработка программно-информационных систем
(наименование направленности (профиля) образовательной программы / специализации)

УТВЕРЖДАЮ
Заведующий кафедрой

_____ Косяков С.В.
(подпись)
«____»_____20___ г.

З А Д А Н И Е
на выполнение выпускной квалификационной работы

обучающемуся **Шулайкину Дмитрию Алексеевичу**

1. Тема работы: Разработка веб-приложения для заказа ритуальных памятников, утвержденная приказом ректора от « 20 » апреля 2020 г. № *310а*-3.

2. Исходные данные к работе: задание согласованное с руководителем.

3. Содержание работы (перечень вопросов, подлежащих разработке):
– Введение
– Разработка и анализ требований
– Проектирование
– Разработка
– Тестирование
– Заключение

4. Графический материал: диаграмма вариантов использования, диграмма классов предметной области, диаграмма сущностей базы данных, диаграмма сборки, диаграмма развертывания.

Руководитель _____ Зубков В.П.
(подпись) *(Ф.И.О.)*

Обучающийся _____ Шулайкин Д.А.
(подпись) *(Ф.И.О.)*

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Ивановский государственный энергетический университет
имени В.И. Ленина»

Факультет _____Заочного и вечернего обучения_____
(полное наименование факультета)

Кафедра _____Программного обеспечения компьютерных систем_____
(полное наименование выпускающей кафедры)

Направление подготовки (специальность) _____09.03.04 Программная инженерия_____
_____
(код, наименование направления подготовки / специальности))

Направленность (профиль) образовательной программы_____
_____Разработка программно-информационных систем_____
(наименование направленности (профиля) образовательной программы / специализации)

КАЛЕНДАРНЫЙ ПЛАН
подготовки выпускной квалификационной работы (ВКР)

№ п/п	Наименование этапов работы[1]	Срок выполнения	
		начало	окончание
1	Разработка и анализ требований	15.03.2021[2]	18.03.2021
2	Проектирование и разработка	19.03.2021	24.03.2021
3	Тестирование и отладка	25.03.2021	28.03.2021[3]
4	Анализ и при необходимости корректировка проектных решений в соответствии с заданием на ВКР с учетом результатов производственных практик	20.05.2021[4]	17.06.2021
5	Оформление ВКР и графического материала, подготовка к процедуре защиты	18.06.2021[5]	27.06.2021
6	Процедура защиты ВКР	28.06.2021[6]	28.06.2021[7]
7	Завершение процедуры государственной итоговой аттестации: информационная и техническая подготовка результатов ВКР, в том числе к размещению в ЭИОС и библиотеке ИГЭУ	29.06.2021	30.06.2021[8]

Руководитель _____ Зубков В.П.
(подпись) *(Ф.И.О.)*

Обучающийся _____ Шулайкин Д.А.
(подпись) *(Ф.И.О.)*

1 *Наименования этапов ВКР должны быть согласованы с заданием на ВКР*
2 *Дата начала преддипломной практики*
3 *Дата окончания преддипломной практики*
4 *Дата начала ГИА*
5 *Дата начала подготовки к процедуре защиты*
6 *Дата начала работы ГЭК по приказу*
7 *Дата окончания работы ГЭК по приказу*
8 *Последний день ГИА согласно графику учебного процесса*

Введение

Одной из наиболее активно развивающихся отраслей в нашей стране является отрасль ритуальных услуг, в том числе изготовление ритуальных памятников. В данной сфере до сих пор нет четко выраженного лидера, велика доля малого бизнеса. Гораздо чаще люди обращаются за услугой оффлайн.

Внедрение цифровых технологий как никогда актуально в этой отрасли из-за пандемии COVID-19. В России, как и во всем мире, выросла смертность, с этим нетрудно связать увеличение числа клиентов и заказов в сфере ритуальных услуг. Памятник обычно ставят через год после погребения усопшего, а пандемия началась ровно год назад, это значит что в этом году ожидается резкий подъем продаж. Также из-за карантинов и новых правил самоизоляции какое-то время не могли работать офисы и торговые точки. После внедрения цифровых технологий сотрудники организаций этой отрасли смогут работать удаленно, а клиенты получать услугу дистанционно.

Цель данной работы – разработать систему для оформления заказов ритуальных памятников, которая поможет решить проблемы поставленные заказчиком, которая в то же время будет универсальным и гибким решением, которое может подходить и другим компаниям в этой сфере.

Постановка задач

В рамках разработки приложения необходимо решить множество задач. Основными из которых являются:

1. Проанализировать предметную область, проблемы и требования заказчика
2. Определить основные функциональные возможности и варианты использования системы
3. Выбрать архитектуру и технологии разработки наиболее подходящие для разработки системы
4. Спроектировать и сконструировать систему используя знания полученные в ходе анализа
5. Протестировать и выполнить деплой системы

Глава 1. Разработка и анализ требований

В этой главе будет рассмотрена предметная область, а также анализированы требования к разрабатываемой системе. Также будут определены основные варианты использования.

1.1 Заказчик

Заказчиком системы является небольшая компания по изготовлению и продаже памятников из гранита. Данную компанию можно охарактеризовать как семейный бизнес. Владельцы бизнеса, также являются и управляющими компанией. Тем не менее в этой компании есть некоторое количество наемных рабочих, которые выполняют роль менеджеров по продажам. Всего у компании есть три точки продаж в одной области.

Следовательно, система будет разрабатываться для конкретного заказчика, значит требования должны учитывать проблемы и желания конкретного заказчика. Тем не менее, было решено разрабатывать универсальную систему, для небольших салонов по изготовлению и продаже памятников.

У заказчика еще нет интернет сайта и магазина так как у него нет проблем с привлечением клиентов оффлайн.

1.2 Предметная область

Предметная область смоделирована с помощью Диаграммы классов UML, которая представлена на рисунке 1.

Главной сущностью в предметной области заказа памятников является заказ. Основная информация в заказе это информация о товарах и клиенте, данные усопшего. Помимо этого также есть, например, информация о менеджере и отделении ответственных за этот заказ.

Главное отличие заказа в рассматриваемой предметной области от других бизнесов связанных с торговлей является то, что памятник часто заказывают индивидуально и выбирают каждую часть отдельно:

- Памятник состоит из множества частей: стеллы, подставки, цветника, надгробия, и т.д.
- Также на памятник наносят индивидуальное оформление, гравировку или фотокерамику
- В дополнение к памятнику могут также заказать металлический забор, ворота, столик, скамейку, а так же всевозможные изделия из гранита
- К каждому элементу заказа предлагается целый ряд обязательных или опциональных услуг которые зависят как от свойств самого элемента, так и от желания клиента

Таким образом, получается огромное число возможных вариантов заказа, в то время как в обычных интернет магазинах у одного товара как правило есть всего один или несколько вариантов а также есть четкая цена. Для типовых недорогих памятников есть готовые комплекты с фиксированными ценами, но там уже гораздо меньше возможностей индивидуального оформления.

Каждая часть памятника и каждый элемент заказа описывается в документе “Наряд-Заказ”, также вместе с ним оформляется документ “Договор”,

который фиксирует итоговую цену а также описывает права и обязанности бизнеса и покупателя.

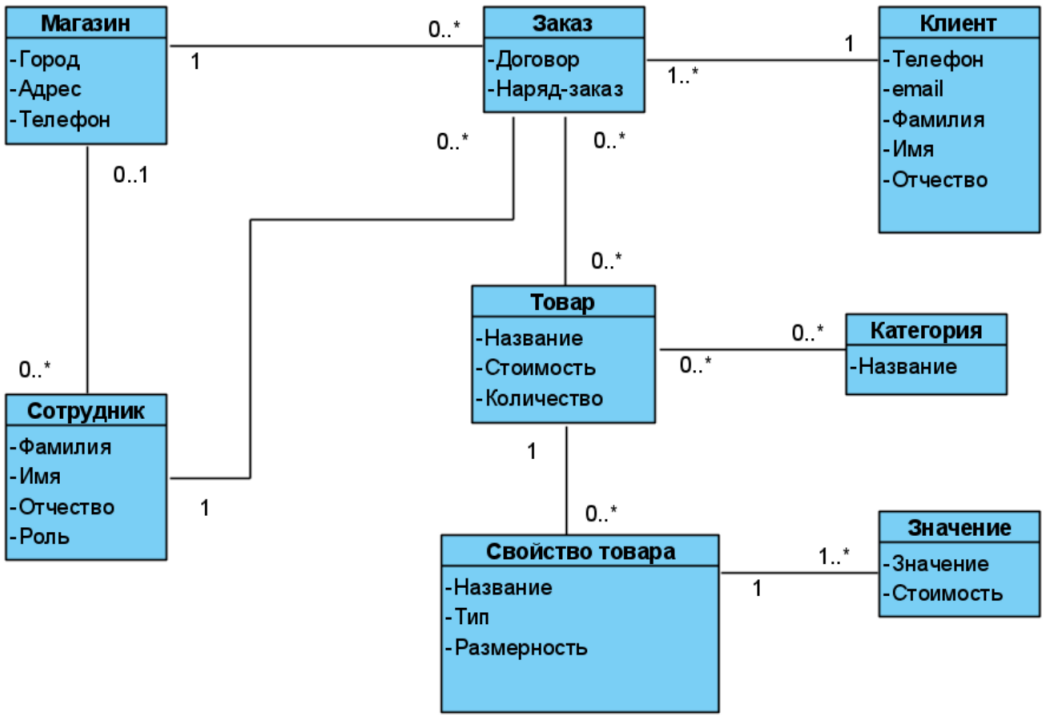


Рис. 1: Диаграмма классов предметной области.

1.3 Основные бизнес правила

- Чтобы совершить заказа, клиент обязан прийти в отделение компании
- Клиент может оплатить заказ только наличными или банковским переводом с карты на карту в отдельных случаях
- Изготовление стеллы начинается после того как заказ была оплачен

1.4 Проблемы

- Ошибки в заказах, так как они оформляются от руки
- Менеджеры по продажам плохо разбираются в вопросах предметной области, например выбирают слишком маленькую подставку для слишком большой стелы

- Для управления расценками и каталогом используется простой текстовый документ и описывает только самые типовые модели
- Продавцы часто не знают сколько будет стоить изготовить памятник с произвольным, не типовым оформлением
- Управляющим сложно следить за всеми заказами и поставками, так как сейчас эта информация передается устно и записывается только в excel таблицу

1.5 Варианты использования

Основные варианты использования системы описаны с помощью диаграммы вариантов использования в нотации UML. Диаграмма представлена на рисунке 2:

1. Оформление заказа, добавление информации о стелле и об усопшем
2. Поиск заказа - по номеру телефона и номеру заказа
3. Информация о наличии товара на складе и резервирование
4. Печать документов

1.6 Ограничения

- Поддержка только русского языка, так как у конкретного заказчика есть магазины только в одной области в России. Также предметная область специфична для России и стран СНГ.
- Заказчик и другие потенциальные клиенты являются представителями малого бизнеса Следовательно системой будут пользоваться ограниченное число пользователей, и магазинов в системе будет магазинов не так много. Это накладывает ограничение как и на трафик, который будет маленьким, так и на процесс. Так как в маленьких магазинах чаще всего нет четкого разделения ролей между продавцами.

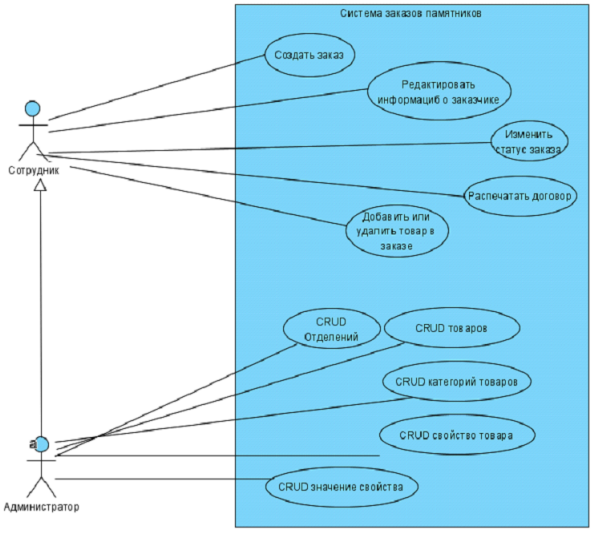


Рис. 2: Диаграмма вариантов использования.

- В одном заказе может быть только одна стелла. Это ограничение нужно для упрощения логики обработки заказов.

1.7 Постановка целей перед системой

Основная цель системы – решать проблемы основного заказчика, а именно приложение должно:

1. Сократить время необходимое на оформление и обработку заказов на 30%
2. Сократить количество ошибок в заказах на 50%
3. Сократить время необходимое на обучение сотрудника до 1 недели.

Глава 2. Проектирование

В этой главе будут рассмотрены основные проектные решения выполненные при проектировании и разработки системы для заказа памятников. При выборе технологии, я руководствовался в основном тремя факторами:

1. популярность и размер сообщества;
2. гибкость и масштабируемость;
3. открытая код и лицензия.

При проектировании я пытался сделать систему как можно более гибкой.

2.1 Выбор платформы

Основным пользователем приложения являются сотрудники бизнеса, основным устройством которых являются либо моноблок, либо ноутбук.

Таким образом следует разрабатывать либо настольное приложение, либо веб-приложение.

В качестве платформы для разработки был выбран веб, на это имеется две основные причины:

1. Стоимость разработки гораздо ниже чем у desktop приложений. Это обусловлено развитием сообщества библиотек, фреймворков, инструментов и приложений с открытым исходным кодом, а именно наличием большого количества готовых компонентов для решения типовых задач бизнес.
2. Универсальность, почти все платформы поддерживают веб, так что приложение не привязано к ОС. Так что в теории заказчик может отказаться от ОС MS Windows и перейти на linux чтобы сократить издержки.

2.2 Выбор архитектуры приложения

Приложение будет иметь традиционную монолитную трехслойную архитектуру и будет использовать технологии стека MERNG: MongoDB, Express, ReactJS, NodeJS, GraphQL. Слои перечислены ниже:

1. Слой данных - сервер баз данных MongoDB
2. Слой API - сервер express-graphql-apollo
3. Слой представления - SPA ReactJS клиент

Монолитная архитектуру была выбрана в первую очередь из-за ограниченности ресурсов. Микросервисная архитектуру в первую очередь нужна когда проект слишком большой, и его проще разбить на отдельные сервисы, с целью распараллелить разработку на несколько команд разработчиков.

Также микросервисная архитектура упрощает поддержку и увеличивает гибкость при деплое приложения.

Но так как разрабатывать приложение для заказа памятников будет только один человек, использовать микросервисную архитектуру особого смысла не имеет.

2.3 Выбор языка программирования

Основным языком программирования в вебе является JavaScript, а также TypeScript. Также сейчас развивается WebAssembly, он позволяет компилировать почти любой ЯП в низкоуровневый код который понимает браузер. В теории это позволяет писать приложения который быстрее работают и загружаются. Однако прирост производительности тут не очень велик, по сравнению с увеличением стоимости разработки. Поэтому выбор был сделан в пользу TypeScript.

2.4 Выбор технологий backend

В качестве протокола был выбран GraphQL в первую очередь из-за его популярности а также желания, его изучить, и освоить навыки разработки приложений с его применением. Также он неплохо подходит для приложения.

Основные преимущества GraphQL:

- возможность писать гибкие запросы и получать только те данные которые нужны
- встроенная проверка типов времени выполнения

Однако у GraphQL протокола есть главные недостаток, он заставляет проектировать схему запросов. Проектирование которой требует хорошего знания предметной области и GraphQL [1].

Также кроме этого традиционные REST протоколы выигрывают в производительности в несколько раз, за счет того что GraphQL вносит много своих издержек производительности, а также его сложнее кешировать на уровне API.

Однако вопросы производительности GraphQL выходят за рамки поставленных ограничений.

Помимо основного backend, в приложении также есть дополнительный backend для потенциального интернет магазина клиента. Прямо сейчас у заказчика нет своего интернет магазина, но для того чтобы расширить гибкость и функционал системы было принято решение добавить REST endpoint, а также в рамках работы создать POC клиента интернет магазина.

Диаграмма архитектуры представлена на рисунке 3

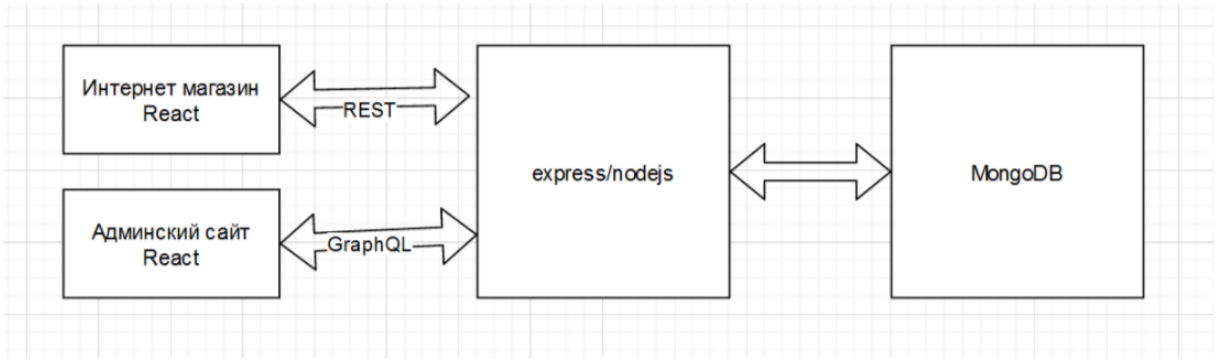


Рис. 3: Диаграмма сущностей базы данных относящихся к функционалу создания заказа

2.5 Выбор технологий frontend

В качестве клиента выступает SPA endpoint. Для разработки SPA будет использоваться библиотека ReactJS. А также фреймворк Apollo Client React. В качестве набора компонентов будет использоваться отдельная библиотека разработанная в рамках данной ВКР специально для этого приложения. Но данная библиотека спроектирована универсально для любых приложений, и в теории может использоваться в других приложениях в будущем.

2.6 Выбор инструментов разработки

Для организации кода и NPM модулей будет использоваться моно-репозиторий и инструмент lerna, который помогает их связывать.

Моно-репозиторий эффективен в небольших командах, где инженерам приходится одновременно работать сразу с несколькими приложениями и проектами.

Когда приложение начинало разрабатываться TypeScript еще не умел поддерживать проекты с несколькими tsconfig.json файлами. А web-клиент созданный с помощью Create React App не поддерживает импортирование кода из внешних пакетов. В моем случае, самым лучшим решением для локальной разработки стало использование lerna в качестве инструмента управления моно-репозиторием.

В главном репозитории моего проекта есть 4 пакета:

1. Пакет core - содержит общую логику, которая используется как на сервере, так и на клиенте
2. Пакет graphql - содержит типы нужные для взаимодействия клиента и сервера.
3. Пакет api - private пакет GraphQL API сервера
4. Пакет web - private пакет SPA клиента созданный с помощью Create React App

Особенно интересен пакет graphql, внутри него находится схема graphql, и по этой схеме генерируются типы TypeScript, которые затем используются во всех других проектах.

2.7 Проектирование базы данных

Проектирование схемы базы данных будет осуществляться с помощью библиотеки ODM Mongoose. Основными коллекциями являются

1. orders - заказы, схема представлена на рисунке 4
2. users - сотрудники магазина, управляющие и продавцы
3. shops - отделения, офисы, точки продаж памятников
4. products - абстрактная информация о продукте, его свойствах и значениях, которые возможно заказать, схема представлена на рисунке 5
5. customers - клиенты

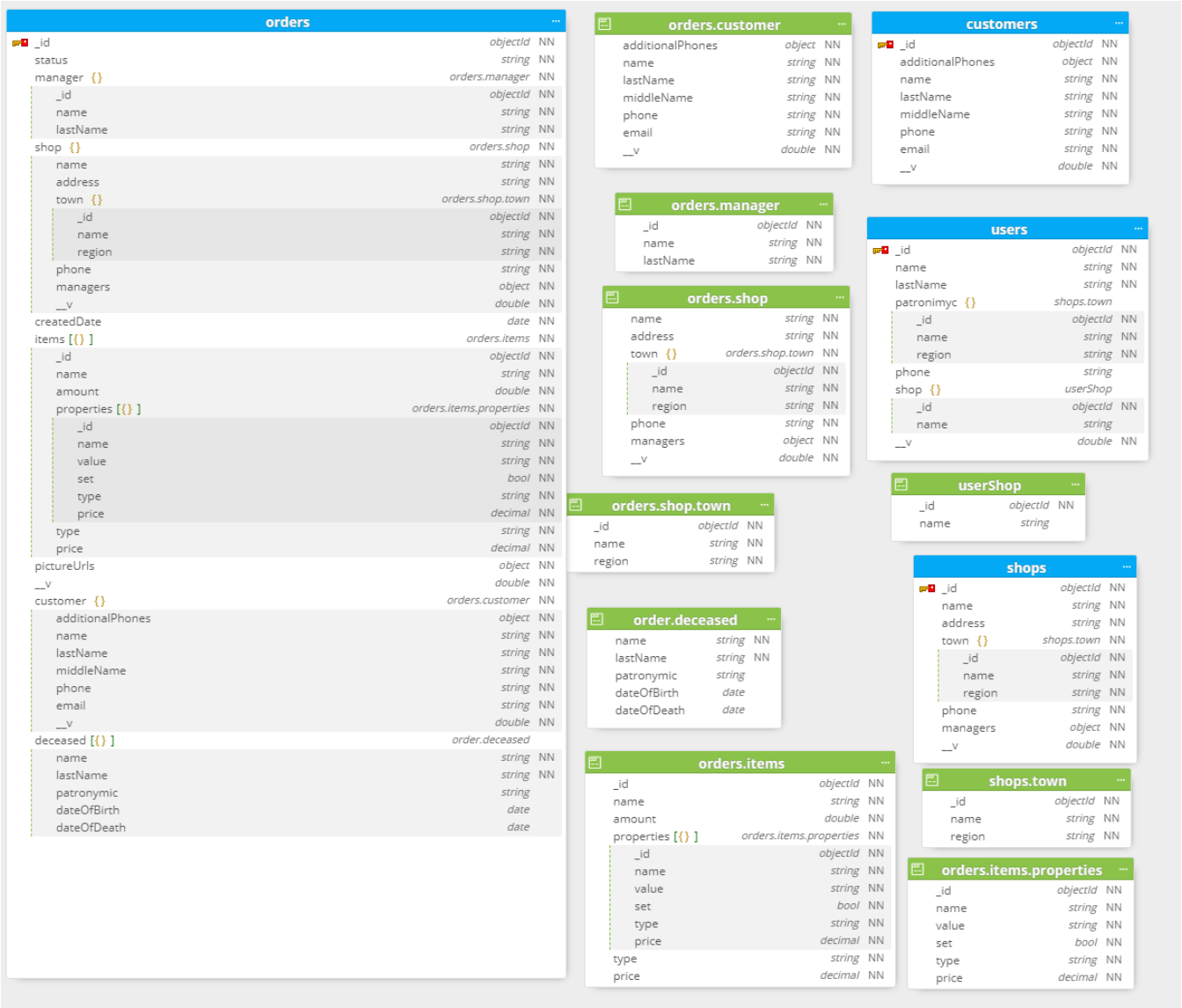


Рис. 4: Диаграмма сущностей базы данных относящихся к функционалу создания заказа

2.8 Макет интерфейса

Изначально макет интерфейса обсуждался напрямую с заказчиком, был сделан акцент на функционал форм, а также процесс добавления предметов в заказ и изменение состояния заказа.

Макет интерфейса программы разрабатывался в приложении Figma. Был разработан типовой дизайн компонентов интерфейса, а также несколько основных страниц как на настольном браузере, так и в мобильной версии.

Пример одной страницы представлен на рисунке 6.

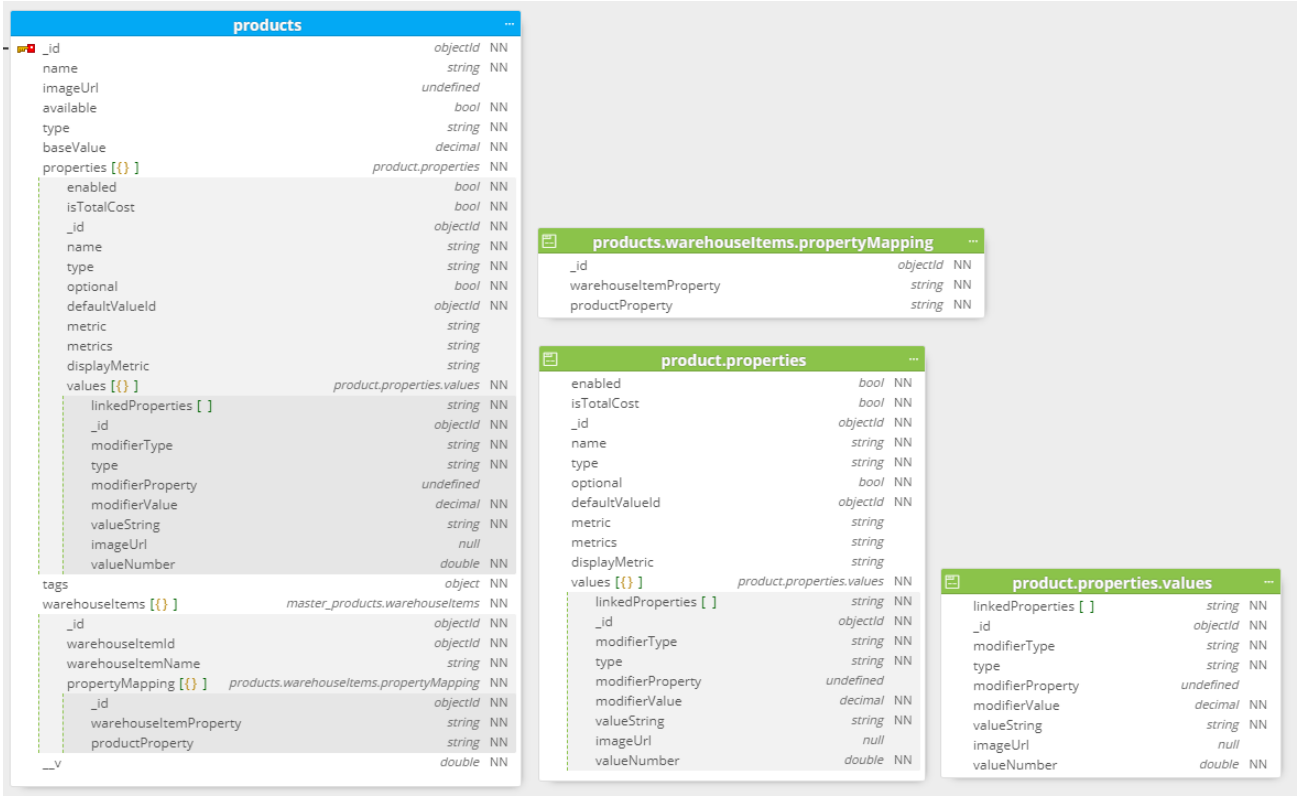


Рис. 5: Диаграмма сущностей базы данных относящихся к функционалу каталога

Глава 3. Разработка

3.1 Разработка собственного набора компонентов

3.1.1 Обоснование

Так как разработка проекта выполняется в рамках выпускной квалификационной работы, было решено создать свой набор компонентов, с целью исследования возможностей библиотеки ReactJS а также получения новых навыков.

При использовании одной сторонней библиотеки возникает проблема ограниченности выбора компонентов. При использовании нескольких сторонних библиотек компонентов возникает проблема их несовместимости и более того они выглядят некрасиво.

Такой библиотека поможет сделать приложение более консистентным

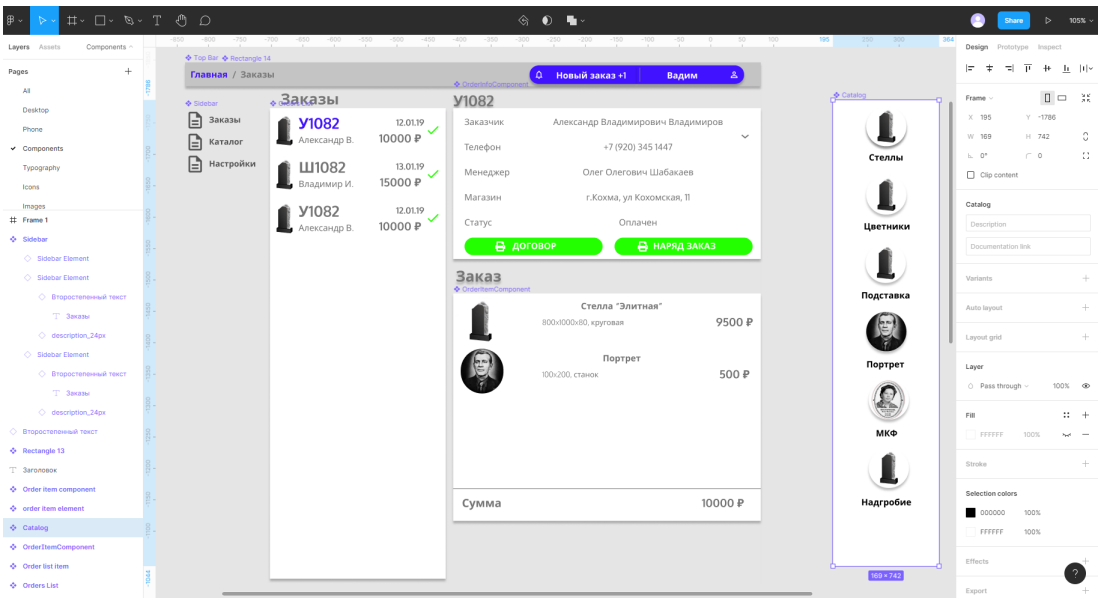


Рис. 6: Пример макета одной из страниц приложения.

3.1.2 Разработка

Для разработки использовался фреймвок styled-components.

Который позволяет создавать стилизованные react компоненты с помощью backticks. Это библиотека использует JSS, при этом используется синтаксис похожий на CSS [2].

В процессе разработки было решено, что для экономии времени и ресурсов, все компоненты будут в одном пакете, который будет содержать как низкоуровневые стилизованные компоненты, так и высокоуровневые компоненты с вспомогательной логикой.

В идеальном мире, нужно было разбить это на два разных пакета и сделать их независимыми, но было решено сделать переход к этой архитектуре постепенным.

3.1.3 Результат

Библиотека компонентов опубликована со свободной лицензией в публичный NPM registry с именем @dmitriiqq/react-kit.

Глава 4. Разработка мобильного приложения

В рамках курса по разработки мобильных приложений был создан Proof-of-Concept мобильный клиент для системы на платформе ReactNative.

Данный клиент был предназначен для пользователей с ролью администратора, для того чтобы быть всегда в курсе изменений происходящих в системе.

Например иметь всю информацию о заказах, также был реализован функционал редактирования каталога.

4.1 Цели и задачи разработки приложения

Приложение должно выполнять вспомогательные функции для управления системой, чтобы облегчить ее администрирование, когда нет под рукой компьютера.

4.2 Портрет целевой аудитории

Целевой аудиторией являются пользователи системы для заказа памятников, управляющие бизнесом индивидуальные предприниматели, малый бизнес. Они работают 24/7, всегда нужно решать какие то вопросы, даже когда под рукой нет компьютера.

4.3 Краткое описание мобильного приложения

Приложение позволяет администраторам системы достигать целей по управлению системой заказа памятников: управлять персоналом, магазинами и каталогом изделий.

Варианты использования представлены в нотации UML на диаграмме 7.

4.4 Результат разработки

Для разработки я использовал платформу ReactNative, так как она позволяет использовать уже существующий код написанный для веб-клиента для бизнес логики программы и общения с API.

Рис. 7: Диаграмма сборки проекта

Единственное что нужно добавить - это слой представления. Он был реализован с помощью набора готовых компонентов react-ui-kitten. Основные экраны приложения показаны на рисунке 8.

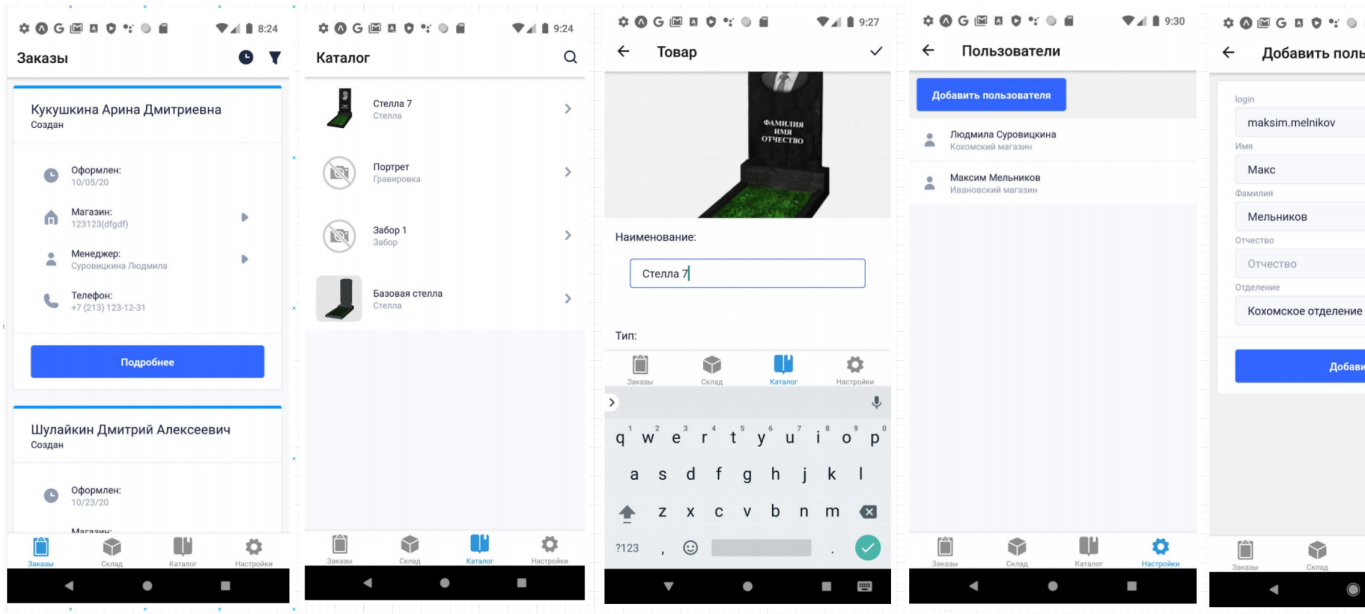


Рис. 8: экраны приложения

Глава 5. Тестирование

В данной главе описываются методы которые были применены при тестировании приложения:

1. Тестирование методом черного ящика
2. Тестирование методом белого ящика
3. Интеграционное тестирование

5.1 Тестирование методом черного ящика

При тестировании методом черного ящика тестирующая система ничего не знает об исходном коде тестируемой системы. Для тестирования методом черного ящика рассмотрим форму добавления информации о заказчике.

Эта форма удовлетворяет вариант использования «Редактировать информацию о заказчике». Интерфейс формы показан на рисунке 9.

Определим классы эквивалентности:

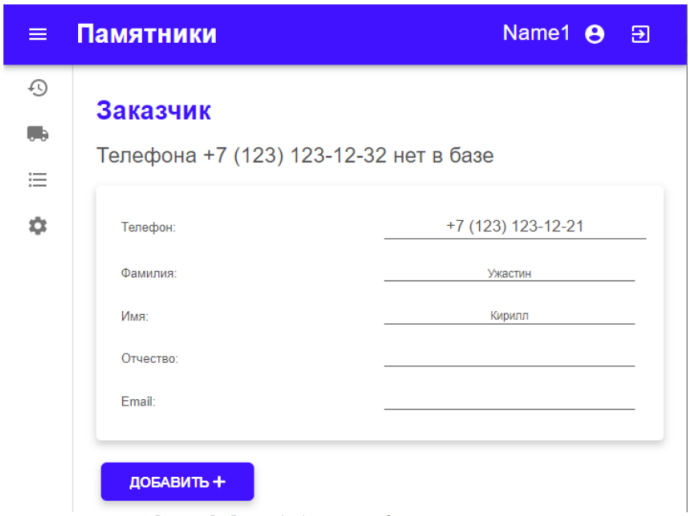
1. Телефон должен быть в формате +7 (999) 99-99-99. То есть начинаться с +7, вместо 9 могут быть любые другие цифры от 0 до 9.
2. email должен быть в формате example@email.com. Где example, email, com могут быть любыми строками содержащими только символы латинского алфавита, цифры и спец символы
3. Фамилия, имя, отчество могут содержать только символы русского алфавита
4. Фамилия, имя, отчество начинаются с большой буквы
5. Телефон, фамилия, имя являются обязательными полями

Так как выходным данным метода валидации является true/false у нас есть всего два класса эквивалентности: допустимый класс и недопустимый класс Для составления классов эквивалентности по спецификации можно рассмотреть следующие ситуации относящиеся к допустимому классу:

1. Все поля формы заполнены и содержат допустимые значения;
2. Заполнены только минимально необходимые поля («телефон», «фамилия», «имя»), и эти поля содержат допустимые значения;
3. Заполнены только минимально необходимые поля и поле «email», и эти поля содержат допустимые значения;
4. Заполнены только минимально необходимые поля и поле «отчество», и эти поля содержат допустимые значения.

Все остальные ситуации относятся к недопустимому классу:

1. Одно или несколько обязательных полей не заполнены
2. Одно или несколько из заполненных полей содержат недопустимые значения



Фигура 5: Форма добавления информации о заказчике

Рис. 9: Диаграмма сборки проекта.

Для реализации unit-тестирования используется тестовый фреймворк jest. Форма добавления информации о клиенте использует метод validateCustomer который возвращает true или false.

Согласно методу черного ящика тестировщик не знает как устроен этот метод, а только что этот метод принимает в качестве входных параметров и

отдает на выход. При написании тестов тестировщик полагается только на классы эквивалентности. Классы эквивалентности показаны в таблице 10.

Значение входных данных	Значение выходных данных	Тип ошибки	Примечание
'Имя', 'Фамилия', '+7 (933) 333-33-33', 'Отчество', 'email@domain.com'	true	-	Все поля заполнены корректно
'Имя', 'Фамилия', '+7 (933) 333-33-33'	true	-	Только необходимые поля заполнены корректно
'Имя', 'Фамилия', '+7 (933) 333-33-33', 'Отчество'	true	-	Только необходимые поля и отчество заполнены корректно
'Имя', 'Фамилия', '+7 (933) 333-33-33', undefined, 'email@domain.com'	true	-	Только необходимые поля и email заполнены корректно
'Имя', 'Фамилия', '8 (933) 333-33-33'	false	«телефон» заполнен некорректно	«телефон» должен начинаться с +7
'Имя имя', 'Фамилия', '+7 (933) 333-33-33'	false	«имя» заполнено некорректно	«имя» не может содержать пробельные символы
'имя', 'Фамилия', '+7 (933) 333-33-33'	false	«имя» заполнено некорректно	«имя» должно начинаться с заглавной буквы
'sdfsdf', 'Фамилия', '+7 (933) 333-33-33'	false	«имя» заполнено некорректно	«имя» должно содержать только буквы русского алфавита
'Имя1', 'Фамилия', '+7 (933) 333-33-33'	false	«имя» заполнено некорректно	«имя» должно содержать только буквы русского алфавита
'Имя', '', '+7 (933) 333-33-33'	false	Обязательное поле «фамилия» не заполнено	-
'Имя', 'Фамилия', ''	false	Обязательное поле «телефон» не заполнено	-
'', 'Фамилия', '+7 (933) 333-33-33'	false	Обязательное поле «имя» не заполнено	-

Рис. 10: классы эквивалентности

5.2 Тестирование методом белого ящика

В качестве метода для демонстрации тестирования был выбран метод getPropertyCost, который используется в варианте использования “Добавить товар в заказ”. Этот метод вычисляет стоимость отдельного свойства памятника, например размер или толщина. Интерфейс формы добавит в заказ

показан на рисунке 11.

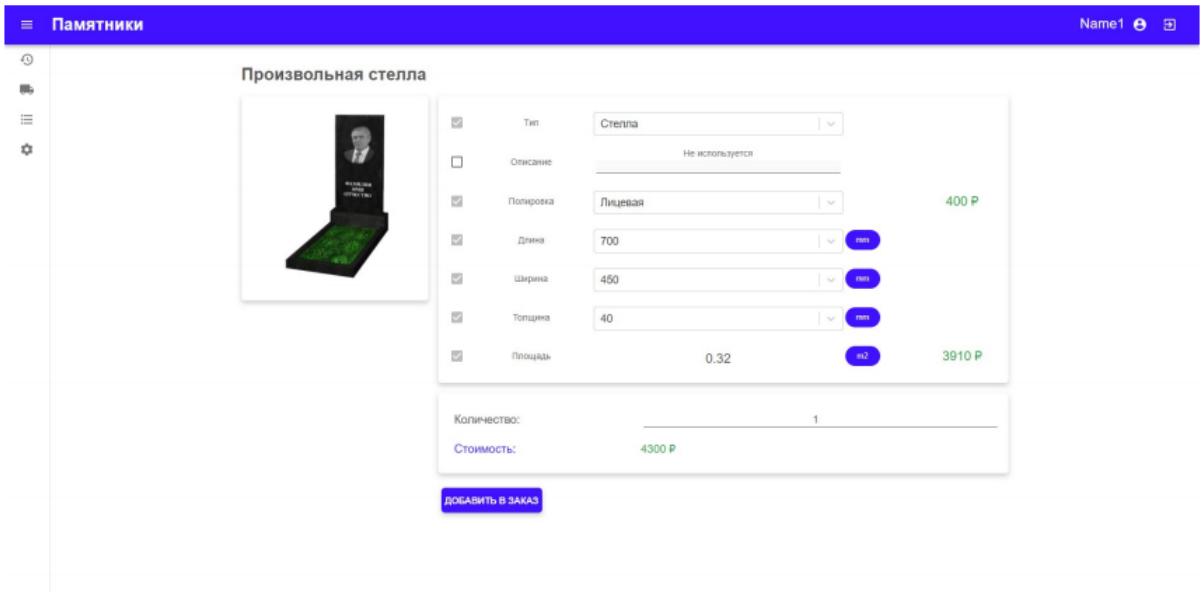


Рис. 11: форма вычисления стоимости

Программа получает на вход описание свойства property, значение для которого нужно вычислить стоимость, и уже вычисленные на данный момент свойства.

Так как стоимость одного свойства может зависеть от другого свойства, как например стоимость полировки памятника может зависеть от площади памятника. При вызове функции гарантируется что все зависимости свойства уже вычислены и находятся в массиве computedProperties.

```
export const getPropertyCost = (
  property: MasterProductProperty,
  value: MasterProductPropertyValue,
  computedProperties: PropertyResponse[] = []
  ): number => { // 1


  if (!value) { // 2
    throw new ValidationError("No value provided"); // 3
  }
```

```
const { modType, modifierValue } = value; // 4
const modifierValueFloat = Number(modifierValue.toString());

if (modType === ModifierType.Constant) { // 5
  return modifierValueFloat; //9
} else if (modType === ModifierType.CostFromValue) { // 6
  if (!value.valueNumber) { // 10
    throw // 12
      new ConfigurationError(‘...expected number property‘)
  }
  return modifierValueFloat * value.valueNumber; // 11
} else if (modType === ModifierType.ProcentToPropertyCost) { // 7
  const modifierProperty = computedProperties
    .find(p => p.propertyName === value.modifierProperty);
  if (!modifierProperty) { // 13
    throw new Error("Failed to link..."); // 15
  }
    return value.modifierValue * modifierProperty.cost / 100; // 14
} else if (modType === ModifierType.MultiplyPropertyCost) { // 8
  const modifierProperty = computedProperties
  .find(p => p.propertyName === value.modifierProperty);
  const modValueFloat = Number(value.modifierValue.toString()) || 1;
  if (!modProperty) { // 16
    throw new Error("Failed to link..."); // 19
  }
  if (!value.valueNumber) { // 17
    throw new Error("Failed to get valueNumber"); // 20
  }
  return value.valueNumber
    * modalueFloat
    * Number(modifierProperty.cost); // 18
}
```

```
    return 0; // 21
}
```

Диаграмма УГП представлена на рисунке 12

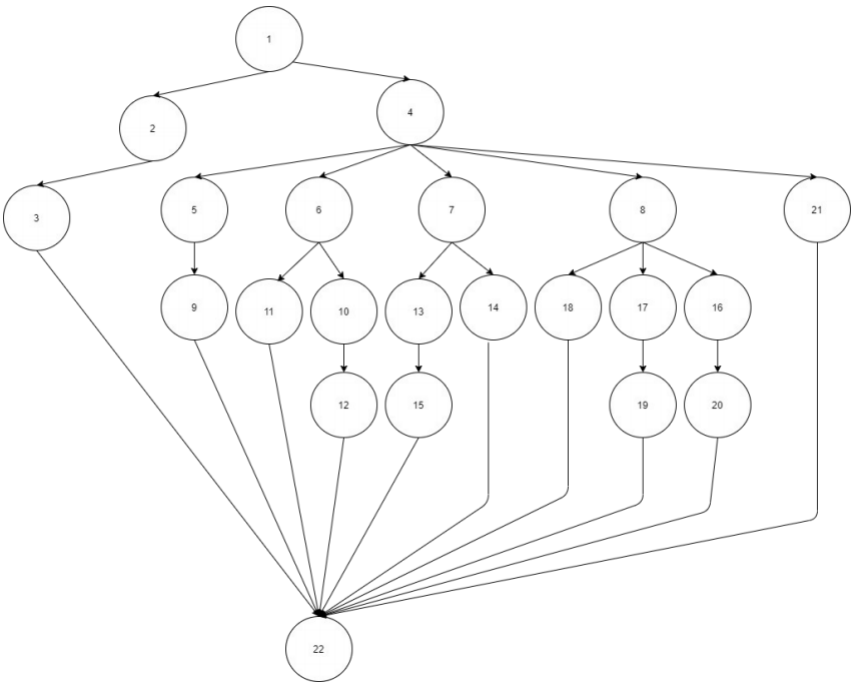


Рис. 12: Диаграмма управляющего графа программы

Далее вычисляется цикломатическая сложность:

$$V(G) = 30 \text{ дуг} - 22 \text{ узла} + 2 = 10$$

Далее строится базовое множество независимых линейных путей

1. 1,2,3,22
2. 1,4,5,9,22
3. 1,4,6,11,22
4. 1,4,6,10,12,22
5. 1,4,7,13,15,22

6. 1,4,7,14,22
7. 1,4,8,18,22
8. 1,4,8,17,19,22
9. 1,4,8,16,20,22
10. 1,4,21,22

5.3 Интеграционное тестирование

Для демонстрации интеграционного тестирования возьмем функционал создания записи клиента. Будем тестировать серверную часть, а именно общение сервера и базы данных.

Так как в тесте участвуют сразу два компонента, сервер и база данных, такое тестирование называют интеграционным. Воспользуемся тестовым фреймворком jest. В качестве метода для тестирования используем метод createCustomer.

```
  export const createCustomer = async (
  _: any,
  params: MutationCreateCustomerArgs
): Promise<Customer> => {
  const {
    name,
    lastName,
    middleName,
    phone,
    email,
    additionalPhones,
  } = params.input;

  if (!validateCustomer(params.input)) {
    throw new Error("Customer validation failed");
  }
```

```
  const document = await customerModel.create({
    name,
    lastName,
    middleName,
    phone,
    email,
    additionalPhones,
  });

  return getCustomerView(document);
};
```

Этот метод общается с базой данных, он использует модель сущности Mongoose, для интеграционного тестирования нам потребуется развернуть базу данных mongo.

```
  import {
    customerByPhone,
    createCustomer,
  } from "../resolvers/CustomersResolver";

  const dbHandler = require("./inMemoryDB");

  const getCustomer = (
    name: string,
    lastName: string,
    phone: string,
    middleName?: string,
    email?: string
  ) => ({
    name,
    middleName,
    lastName,
```

```
  phone,
  email,
  additionalPhones: [],
});

beforeAll(async () => await dbHandler.connect());

afterEach(async () => await dbHandler.clearDatabase());

afterAll(async () => await dbHandler.closeDatabase());

describe("customer ", () => {
  it("can be created correctly", async () => {
    const createCustomerParams = getCustomer(
      "Имя",
      "Фамилия",
      "+7 (933) 333-33-33",
      "Отчество",
      "email@domain.com"
    );

    const customer = await createCustomer(null, {
      input: createCustomerParams,
    });

    expect(customer!.id).toBeDefined();
    expect(customer!.name).toBe("Имя");
    expect(customer!.lastName).toBe("Фамилия");
    expect(customer!.middleName).toBe("Отчество");
    expect(customer!.phone).toBe("+7 (933) 333-33-33");
    expect(customer!.email).toBe("email@domain.com");

    const newCustomer = await customerByPhone(null, {
```

```
      phone: "+7 (933) 333-33-33",
    });

    expect(newCustomer!.id).toBe(customer.id);
    expect(newCustomer!.name).toBe("Имя");
    expect(newCustomer!.lastName).toBe("Фамилия");
    expect(newCustomer!.middleName).toBe("Отчество");
    expect(newCustomer!.phone).toBe("+7 (933) 333-33-33");
    expect(newCustomer!.email).toBe("email@domain.com");
  });

  it("validates user phone", async () => {
    await expect(async () => {
      const createCustomerParams = getCustomer(
        "Имя",
        "Фамилия",
        "8 (933) 333-33-33"
      );
      await createCustomer(null, { input: createCustomerParams });
      return true;
    }).rejects.toThrow();
  });
});
```

Данная проблема решается с помощью пакета mongodb-memory-server, который позволяет запустить базу данных mongo внутри оперативной памяти компьютера только на время тестов. Логика запуска базы данных общая для всех тестов и находится в файле inMemoryDB.js.

```
const mongoose = require('mongoose');
const { MongoMemoryServer } = require('mongodb-memory-server');

const mongod = new MongoMemoryServer();
```

```
module.exports.connect = async () => {
    const uri = await mongod.getUri();

    const mongooseOpts = {
        useNewUrlParser: true,
        autoReconnect: true,
        reconnectTries: Number.MAX_VALUE,
        reconnectInterval: 1000
    };

    await mongoose.connect(uri, mongooseOpts);
};

module.exports.closeDatabase = async () => {
    await mongoose.connection.dropDatabase();
    await mongoose.connection.close();
    await mongod.stop();
};

module.exports.clearDatabase = async () => {
    const collections = mongoose.connection.collections;

    for (const key in collections) {
        const collection = collections[key];
        await collection.deleteMany();
    }
};
```

Глава 6. Сборка проекта

Итогом сборки проекта должен стать один Docker image, в котором должны быть упакованы файлы полученные при сборке пакетов web и api. В

качестве базового образа используется обычный образ node:14.

Перед тем, как начать финальную сборку, нужно убедиться что пакеты core, graphql были опубликованы в NPM registry. Так как эти пакеты должны будут потом установиться внутрь Docker image при установке зависимостей api.

Сборка docker image проходит в два этапа:

1. Сначала создается временный контейнер в который копируются весь код из репозитория нужный для сборки, потом выполняются все скрипты build во всех проектах моно-репозитория.

2. Полученные артефакты сборки web и api копируются в финальный образ докера. Внутри финального образа устанавливаются зависимости api.

Полученный Docker образ нужно в дальнейшем загрузить в docker registry. Полная диаграмма сборки показана на рисунке 13.

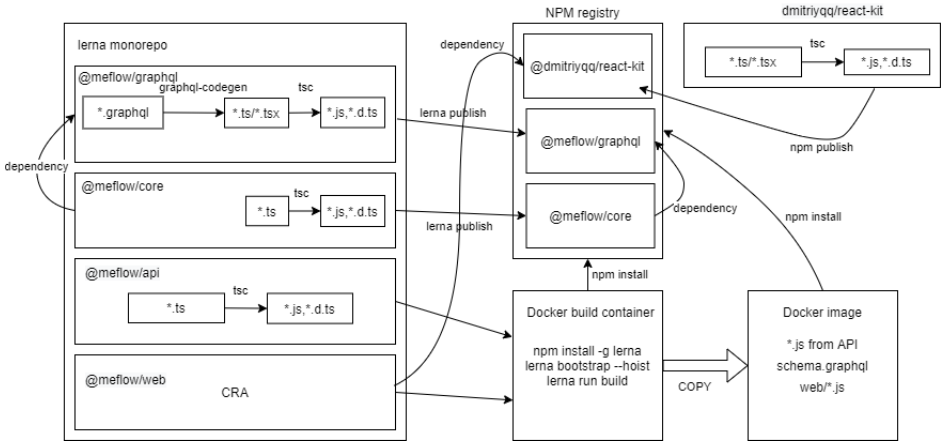


Рис. 13: Диаграмма сборки проекта.

Глава 7. Разворачивание приложения

Самый простой, универсальный и популярный способ развернуть docker приложение это k8s. Для самого приложения я использую простой deployment, так как мое приложение не использует состояния.

Кроме самого приложения также разворачивается сервис ClusterIP который выполняет роль LoadBalancer внутри кластера, а также nginx-ingress который позволяет обрабатывать внешние запросы. Также в kubernetes должен быть Issuer сертификатов чтобы поддерживать протокол TLS.

Полная диаграмма k8s показана на рисунке 14

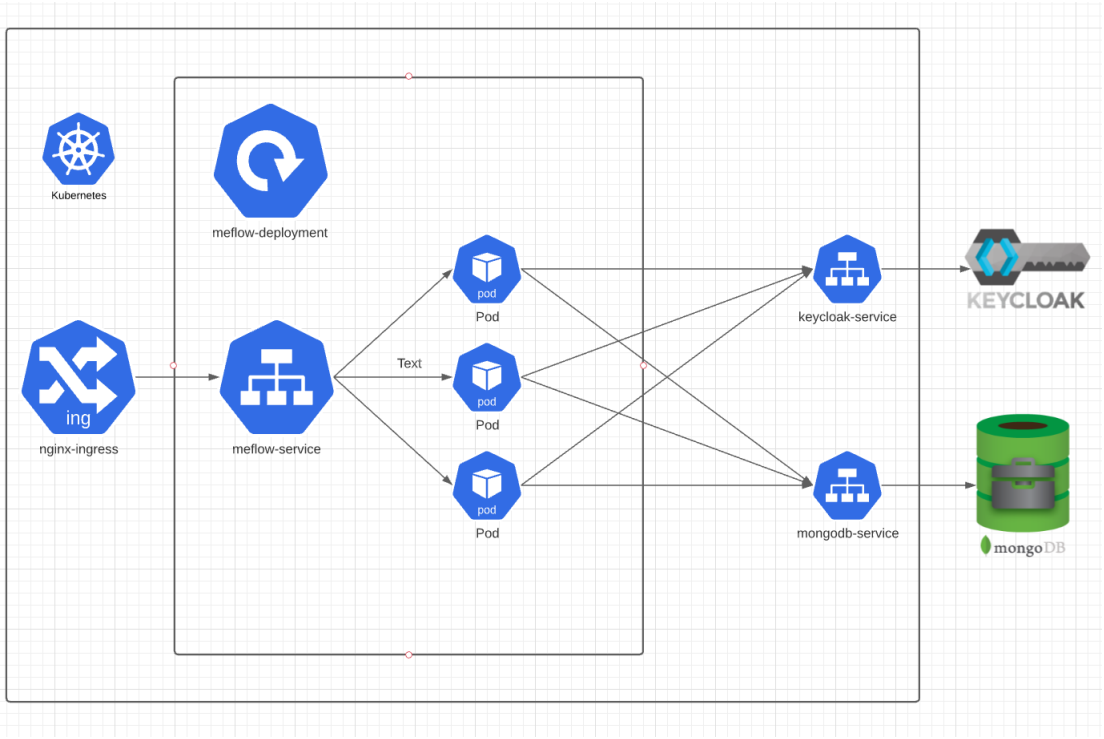


Рис. 14: Диаграмма разворачивания проекта.

Заключение

Целью данной работы являлась разработка веб-приложения для заказа памятников.

Была проведена работа по анализу требований, проектированию и разработке веб-приложения для заказа памятников. Организовано тестирование и развертывание приложение.

Разработан универсальная библиотека компонентов ReactJS используемая в веб приложении.

А также были созданы POC интернет-магазина и приложения для мобильных телефонов на базе Android и iOS способное взаимодействовать с

системой, как альтернативный клиент для некоторых вариантов использования.

В результате было получено готовое приложение которое решает проблемы и удовлетворяет потребностям заказчика.

Список литературы

[1] Introduction to Apollo Server [Электронный ресурс]. Режим доступа: https://www.apollographql.com/docs/apollo-server/. – Заглавие с экрана. – (Дата обращения 13.06.2021)

[2] Styled Components Documentation [Электронный ресурс]. Режим доступа: https://styled-components.com/docs. – Заглавие с экрана. – (Дата обращения 13.06.2021)