**Содержание**

[Введение 2](#_Toc194507424)

[Профессия Fullstack разработчик 4](#_Toc194507425)

[Общие требования к программному продукту 6](#_Toc194507426)

[Техническое задание на создание веб-сайта 8](#_Toc194507427)

[Работа над техническим заданием на создание веб-сайта 10](#_Toc194507428)

[Технология гипертекста — язык HTML 10](#_Toc194507429)

[CSS (Cascading Style Sheets) — язык стилей 11](#_Toc194507430)

[SCSS – расширение CSS 11](#_Toc194507431)

[JavaScript – язык динамичных веб-приложений 12](#_Toc194507432)

[TypeScript – надстройка над JavaScript 13](#_Toc194507433)

[Go (Golang) 14](#_Toc194507434)

[ScyllaDB – высокопроизводительная NoSQL-база данных 14](#_Toc194507435)

[Vue.js – фреймворк для фронтенда 15](#_Toc194507436)

[Git 16](#_Toc194507437)

[Docker 17](#_Toc194507438)

[Makefile 17](#_Toc194507439)

[Сроки выполнения 19](#_Toc194507440)

[Вычислительные ресурсы для разработки сайта 20](#_Toc194507441)

[Документы ГОСТы 22](#_Toc194507442)

[Заключение 23](#_Toc194507443)

[Список использованных источников 24](#_Toc194507444)

[Приложения 25](#_Toc194507445)

# **Введение**

В современном мире цифровые технологии оказывают значительное влияние на развитие бизнесов и проектов. Всё больше количество людей осознает важность онлайн-присутствия, так как это позволяет не только расширить аудиторию, но и повысить уровень взаимодействия с клиентами. Особенно это актуально для сферы электронной коммерции, где интернет-магазины предоставляют пользователям удобный доступ к разнообразной продукции.

Изучение особенностей разработки интернет-магазина, в особенности магазинов технически сложных устройств, является актуальной задачей, поскольку грамотный подход к проектированию и технической реализации для комфортного и понятного выбора товара для обычного пользователя играет ключевую роль в конкурентоспособности. Конкуренция требует создания удобных, функциональных и технологически продвинутых решений, обеспечивающих комфортный опыт и эффективное управление сайтом и ассортиментом.

Целью данного проекта является исследование особенностей разработки интернет-магазина, включая анализ ключевых факторов, влияющих на его успешность. В рамках работы необходимо выполнить следующие задачи:

* изучение требований к архитектуре и структуре сайта;
* анализ современных технологий и инструментов для разработки;
* исследование пользовательского опыта и интерфейсных решений;
* оценка методов интеграции баз данных;

Технический аспект проекта включает анализ программных решений, технологий разработки и оптимизации интернет-магазина для повышения его производительности и удобства использования.

Данное исследование направлено на выявление ключевых аспектов успешного проектирования интернет-магазина компьютерных комплектующих, что позволит сформировать рекомендации по созданию эффективного онлайн-ресурса, удовлетворяющего потребности бизнеса и пользователей.

# **Профессия Fullstack разработчик**

Fullstack-разработчик — это универсальный специалист в сфере программирования, обладающий компетенциями как в клиентской, так и в серверной разработке. Он разрабатывает веб-приложения и сайты, охватывая весь процесс создания программного продукта — от интерфейса до серверной логики. В рамках данного дипломного проекта эти навыки будут продемонстрированы на примере веб-ресурса сектора электронной информации муниципальной библиотеки.

Основные обязанности

1. Разработка интерфейсов:

- Проектирование и реализация пользовательских интерфейсов (UI).

- Выполнение адаптивной верстки, обеспечивающей корректное отображение на разных устройствах.

- Использование современных библиотек и фреймворков, включая React, Vue.js, Angular.

2. Разработка серверной части:

- Реализация серверной логики веб-приложений.

- Работа с различными системами управления базами данных (SQL).

- Создание API для связи клиентской и серверной частей приложения.

3. Интеграционные процессы:

- Обеспечение корректного взаимодействия фронтенда и бэкенда.

- Подключение сторонних API и сервисов.

4. Тестирование и устранение ошибок:

- Проведение различных видов тестирования (юнит-тесты, функциональные тесты).

- Анализ и исправление багов в коде.

5. Работа над архитектурой проекта:

- Участие в проектировании архитектуры приложений.

- Взаимодействие с дизайнерами и менеджерами продукта.

Важные навыки

1. Фронтенд-разработка

- Владение HTML, CSS, JavaScript.

- Знание популярных фреймворков (React, Angular, Vue.js и др.).

- Опыт работы с системами контроля версий (Git).

2. Бэкенд-разработка

- Знания языков программирования (Node.js, Python, GO, PHP, Java).

- Опыт работы с серверными технологиями (Express, Django, Spring, Laravel).

- Работа с базами данных (MySQL, PostgreSQL, NoSQL, SQLite).

3. Общее понимание

- Основы DevOps (контейнеризация, CI/CD).

- Основы безопасности веб-приложений.

# **Общие требования к программному продукту**

Создание программного продукта подразумевает соблюдение ряда условий, гарантирующих его высокое качество, стабильную работу и удобство для пользователей. Эти условия можно классифицировать по нескольким ключевым категориям:

1. Требования к функциональности

Программное обеспечение должно эффективно выполнять все назначенные функции, отвечающие потребностям пользователя. При этом условия и ограничения, в рамках которых будет реализована функциональность, должны быть ясно прописаны.

Дизайн интерфейса должен быть простым в освоении, удобным для пользователя и отвечать современным дизайнерским тенденциям.

Совместимость – это ключевой аспект, который подразумевает способность программного обеспечения взаимодействовать с другими системами и инструментами, используемыми пользователями, например, интеграцию с базами данных или другими ПО.

2. Требования, не относящиеся к функциональности.

Эффективность. Программное обеспечение должно обеспечивать необходимую скорость обработки информации и рациональное использование ресурсов, таких как память и процессор. Программа должна демонстрировать высокую степень надежности, обеспечивая корректную и стабильную работу с минимальной вероятностью сбоев и ошибок. К этому относится и эффективная обработка ошибок, а также устойчивое восстановление после их возникновения.

Безопасность. Необходимо учитывать защищенность данных и возможность их несанкционированного доступа. Сюда входят шифрование, аутентификация и авторизация пользователей.

3. Критерии качества.

Совместимость. Программное обеспечение должно корректно работать на различных аппаратных платформах и в разных операционных системах.

Легкость в сопровождении. Код должен быть хорошо структурирован и документирован, что обеспечит его простую поддержку и возможность внесения изменений.

Доступность для тестирования. Разработка должна учитывать возможность удобного тестирования на различных уровнях, включая модульные и интеграционные тесты.

4. Пользовательские требования

Документальное сопровождение. Включает руководство пользователя, технические спецификации и инструкции по установке.

Учёт мнения пользователей. Обратная связь от пользователей поможет в дальнейшем совершенствовании системы.

5. Регуляторные и правовые требования.

Соответствие стандартам и нормам. Продукт должен соответствовать действующим стандартам, таким как ISO и IEEE, а также требованиям законодательства.

Лицензирование и защита авторских прав. Обеспечение правовой защиты интеллектуальной собственности и правильное оформление лицензий на использование стороннего ПО.

Заключение.

Настоящее техническое задание определяет ключевые требования к разработке веб-сайта. Важно учитывать, что в процессе работы возможны изменения и корректировки в зависимости от специфики проекта и потребностей пользователей.

# **Техническое задание на создание веб-сайта**

1. Введение

1.1 Цель создания сайта

Создание веб-сайта с целью для представления товаров и услуг организации клиенту, облегчения взаимодействия продавца и клиента, улучшения узнаваемости компании в интернете.

1.2 Описание проекта

Сайт будет представлять собой интернет-магазин с предоставлением товаров и услуг. Он будет включать интуитивно понятный интерфейс, товары и услуги, контактные данные.

2. Требования к сайту

2.1 Общие требования

- Адаптивный дизайн.

- Удобная навигация и интуитивно понятный интерфейс.

- Быстрая загрузка страниц.

2.2 Функциональные требования

2.2.1 Главная страница

- Отображение услуг.

- Управление сайтом.

- Контактная информация.

2.2.2 Раздел Каталог

- Продукты.

- Переход по разделам.

2.2.3 Раздел Продукт

- Информация о товаре.

- Взаимодействие с товаром.

2.3 Нефункциональные требования

- Высокая скорость загрузки.

- Кроссбраузерная совместимость (Chrome, Firefox, Safari).

3. Технические детали

3.1 Платформа

- Сайт будет разработан: HTML, Javascript, TypeScript, GO, NoSQL

3.3 Дизайн

- Прототипы страниц.

- Разработка индивидуального дизайна.

4. Процесс разработки

4.1 Этапы работы

- Подготовка прототипа и дизайн-макета.

- Разработка функционала.

- Тестирование.

- Запуск сайта.

4.2 Сроки выполнения

- Сроки для каждого этапа разработки.

5. Заключение

Данное техническое задание содержит основные требования и пожелания к созданию веб-сайта. В процессе разработки возможны изменения и уточнения по причине выявленных недочетов и недостатков в ходе работы.

# **Работа над техническим заданием на создание веб-сайта**

В рамках технической реализации проекта мной, как fullstack-разработчиком, были определены основные языки программирования и разметки, необходимые библиотеки, фреймворки, системы управления базами данных (СУБД), а также дополнительное программное обеспечение, требуемое для разработки кодовой базы проекта.

# **Технология гипертекста — язык HTML**

HTML (HyperText Markup Language) – это язык разметки, применяемый для создания веб-страниц и веб-приложений. Он определяет структуру и содержание документа, задавая, как должны отображаться элементы в браузере.

Основные особенности HTML:

1. Теги и элементы. HTML использует теги для обозначения различных элементов страницы. Как правило, они парные: открывающий (<p>) и закрывающий (</p>), между которыми размещается содержимое.

2. Структура документа. Стандартный HTML-документ начинается с объявления типа (<!DOCTYPE html>), включает корневой элемент <html>, содержащий <head> (метаданные, заголовок, подключения CSS и JS) и <body> (непосредственное содержимое страницы, включая текст, изображения и ссылки).

3. Атрибуты. Они предоставляют дополнительную информацию о теге. Например, в теге <img> атрибут src указывает путь к изображению: <img src="image.jpg" alt="Описание">.

4. Семантические элементы. Современные HTML-теги, такие как <header>, <nav>, <article>, <section>, <footer>, улучшают структурированность, доступность и SEO-оптимизацию.

5. Интеграция с CSS и JavaScript. HTML используется совместно с CSS (для стилизации) и JavaScript (для интерактивности).

Заключение

HTML – основа веб-разработки. Понимание его структуры и принципов помогает создавать удобные и хорошо организованные веб-страницы.

# **CSS (Cascading Style Sheets) — язык стилей**

CSS (Cascading Style Sheets) отвечает за оформление HTML-документов, позволяя отделить содержание от визуального представления. Это упрощает разработку и поддержку веб-сайтов.

Основные особенности CSS:

1. Селекторы и правила. CSS использует селекторы для выбора HTML-элементов и применения к ним стилей. Правило включает селектор и набор свойств в фигурных скобках.

2. Каскадность. Если к элементу применяются несколько стилей, приоритет определяется по специфичности селектора и порядку объявления правил.

3. Модель коробки (box model). Определяет отображение элементов: содержимое, отступы (margin), границы (border) и внутренние отступы (padding).

4. Свойства. CSS предлагает широкий набор параметров для стилизации (color, font, display, grid, flex, position и др.).

5. Медиа-запросы. Позволяют адаптировать дизайн под разные устройства и размеры экранов.

Заключение

CSS – ключевой инструмент для создания стильных, адаптивных и удобных интерфейсов. Он обеспечивает полный контроль над внешним видом веб-страниц и работает в связке с HTML и JavaScript.

# **SCSS – расширение CSS**

**SCSS** — это синтаксический сахар для CSS, являющийся частью препроцессора Sass. SCSS добавляет дополнительные возможности, такие как переменные, вложенность и миксины, которые делают CSS более мощным и гибким.

Основные особенности SCSS:

1. Переменные. SCSS позволяет использовать переменные для хранения значений (например, цветов, шрифтов или размеров), что упрощает изменения и повторное использование значений в разных частях стилей.

2. Вложенность. SCSS поддерживает вложенность стилей, что позволяет структурировать CSS, как HTML, делая код более читаемым и логичным.

3. Унаследованные стили. SCSS поддерживает наследование стилей, позволяя создавать базовые стили, которые можно расширять и переопределять.

4. Чистый и организованный код. SCSS помогает структурировать стили, разделять их на файлы и организовывать код, что упрощает поддержку и разработку крупных проектов.

Заключение

SCSS делает процесс стилизации более мощным и гибким, добавляя функциональные возможности, которых нет в обычном CSS. Это улучшает читаемость кода и позволяет легче управлять большими проектами.

# **JavaScript – язык динамичных веб-приложений**

JavaScript – это высокоуровневый язык программирования, применяемый для добавления интерактивности и динамических функций на веб-страницах. Он широко используется как на клиентской, так и на серверной стороне.

Основные особенности JavaScript:

1. Интерактивность. Позволяет реализовать обработку событий, анимации, проверку форм, изменение DOM и другие функции.

2. Объектно-ориентированный подход. Поддерживает работу с объектами и наследование свойств и методов, что упрощает разработку.

3. Динамическая типизация. Переменные могут менять тип данных во время выполнения, что дает гибкость, но требует внимательной работы с кодом.

4. Асинхронность. Использует коллбэки, промисы и async/await для работы с серверными запросами без блокировки выполнения кода.

5. Многофункциональность. JavaScript применяется как для фронтенда, так и для бэкенда.

6. Развитая экосистема. Богатый выбор библиотек и фреймворков (React, Angular, Vue.js) ускоряет разработку сложных приложений.

Заключение

JavaScript – мощный инструмент для веб-разработки, позволяющий создавать интерактивные и удобные приложения. Благодаря своей гибкости и активному сообществу, он остается ключевым языком индустрии.

# **TypeScript – надстройка над JavaScript**

TypeScript (TS) – это надстройка над JavaScript, добавляющая статическую типизацию. Он помогает избежать многих ошибок, улучшает читаемость кода и упрощает разработку крупных приложений.

Основные особенности TypeScript:

1. Статическая типизация. Позволяет объявлять типы переменных, параметров и возвращаемых значений функций, что помогает обнаруживать ошибки еще на этапе написания кода.

2. Совместимость с JavaScript. TypeScript – это надстройка над JavaScript, поэтому любой JS-код является корректным TypeScript-кодом.

3. Продвинутые возможности ООП. Поддерживает интерфейсы, перечисления (enums), модификаторы доступа (public, private, protected) и другие функции для организации кода.

4. Декларации типов. Используются для описания типов данных сторонних библиотек, что делает работу с ними более безопасной и предсказуемой.

5. Поддержка современных стандартов. TypeScript компилируется в чистый JavaScript, поддерживая последние версии ECMAScript.

Заключение

TypeScript делает разработку на JavaScript безопаснее и удобнее, особенно в масштабируемых проектах. Благодаря статической типизации и продвинутым инструментам, он значительно упрощает поддержку кода.

# **Go (Golang)**

Go (Golang) – это язык программирования, разработанный Google для создания надежных, быстрых и многопоточных приложений. Он сочетает простоту, производительность и удобные механизмы работы с конкурентностью.

Основные особенности Go:

1. Высокая производительность. Go компилируется в машинный код, обеспечивая скорость, сравнимую с C и C++.

2. Простота и лаконичность. В Go отсутствуют сложные конструкции, что делает код чистым и понятным.

3. Горутины. Легковесные потоки, которые позволяют эффективно работать с многозадачностью без значительных затрат ресурсов.

4. Сборка мусора. Автоматически управляет памятью, упрощая работу с ресурсами.

5. Кроссплатформенность. Позволяет легко компилировать код под разные операционные системы.

Заключение

Go – это мощный инструмент для создания высоконагруженных сервисов, облачных решений и многопоточных приложений. Его простота и производительность делают его одним из ведущих языков для серверной разработки.

# **ScyllaDB – высокопроизводительная NoSQL-база данных**

ScyllaDB – это распределенная NoSQL-база данных, разработанная как высокопроизводительная альтернатива Apache Cassandra. Она предназначена для работы с огромными объемами данных и высокой нагрузкой.

Основные особенности ScyllaDB:

1. Высокая производительность. Использует асинхронную архитектуру на основе C++, что позволяет обрабатывать миллионы запросов в секунду.

2. Совместимость с Cassandra. Поддерживает CQL (Cassandra Query Language) и может использоваться без изменения существующих приложений.

3. Автоматическое масштабирование. Позволяет легко увеличивать объем хранимых данных и количество запросов без снижения производительности.

4. Низкие задержки. Минимизирует время отклика, что критично для систем реального времени.

5. Экономичность. Требует в 3-5 раз меньше серверов, чем Cassandra, обеспечивая такую же или лучшую производительность.

Заключение

ScyllaDB – это мощная NoSQL-база данных для высоконагруженных систем. Ее высокая скорость, низкие задержки и автоматическое масштабирование делают ее отличным выбором для облачных сервисов, IoT и аналитики.

# **Vue.js – фреймворк для фронтенда**

Vue.js – это JavaScript-фреймворк, предназначенный для создания интерактивных пользовательских интерфейсов. Он сочетает простоту, гибкость и мощные инструменты для разработки современных веб-приложений.

Основные особенности Vue.js:

1. Компонентный подход. Позволяет разбивать интерфейс на независимые, переиспользуемые компоненты.

2. Реактивность. Автоматически обновляет интерфейс при изменении данных, минимизируя необходимость вручную управлять DOM.

3. Гибкость. Поддерживает как простые скрипты, так и сложные архитектуры с Vuex или Pinia для управления состоянием.

4. Интуитивный синтаксис. Простой API и декларативный стиль делают его доступным даже для новичков.

5. Экосистема и поддержка. Vue имеет мощную экосистему инструментов, таких как Vue Router (маршрутизация) и Vue CLI (шаблоны проектов).

Заключение

Vue.js – удобный и производительный фреймворк для создания современных веб-приложений. Его простота и мощные возможности делают его отличным выбором для разработчиков любого уровня.

# **Git**

Git – это распределенная система контроля версий, которая используется для отслеживания изменений в исходном коде и совместной работы над проектами. Она позволяет разработчикам эффективно работать в команде, обеспечивая возможность вернуться к предыдущим версиям кода, отслеживать изменения и управлять ветками.

Основные особенности Git:

1. Распределенная система. Git хранит полные копии репозиториев на каждом компьютере, что позволяет работать с проектом без постоянного подключения к серверу. Это делает Git очень быстрым и надежным.

2. Ветки и слияния. Git позволяет легко создавать ветки для параллельной работы над различными частями проекта. Затем эти ветки можно сливать с основной веткой, что упрощает работу в команде.

3. История изменений. Git позволяет отслеживать все изменения, внесенные в код, с возможностью возвращения к любой предыдущей версии. Это облегчает обнаружение ошибок и откат к стабильным состояниям проекта.

4. Местная и удаленная работа. Git поддерживает работу как с локальными репозиториями, так и с удаленными, что позволяет синхронизировать изменения между участниками команды.

5. Мощные инструменты для слияния. Git предоставляет мощные механизмы для слияния веток, а также для разрешения конфликтов, что делает работу в команде более плавной и организованной.

Заключение

Git — это неотъемлемый инструмент для современной разработки, позволяющий эффективно управлять изменениями в коде и работать в команде. Он упрощает процессы слияния, отслеживания изменений и управления версиями, что делает его стандартом де-факто для большинства разработчиков.

# **Docker**

Docker — это платформа для создания, доставки и запуска приложений в контейнерах. Контейнеры позволяют изолировать приложение с его зависимостями, что упрощает развертывание и масштабирование.

Основные особенности Docker:

Основные характеристики виртуального хостинга:

1. Контейнеризация. Docker изолирует приложение и его зависимости в контейнеры, что делает его переносимым между различными средами (например, между разработкой, тестированием и продакшеном).

2. Легковесность. В отличие от виртуальных машин, контейнеры используют ядро хостовой операционной системы, что делает их более легкими и быстрыми.

3. Docker Hub. Официальный репозиторий Docker, где можно найти образы для популярных приложений и баз данных, а также публиковать свои собственные.

4. Автоматизация развертывания. Docker Compose позволяет легко управлять многоконтейнерными приложениями, создавая описания для всех необходимых сервисов в одном файле.

5. Масштабируемость. Docker помогает управлять инфраструктурой, позволяя масштабировать приложения и обеспечивать их высокую доступность.

Заключение

Docker значительно упрощает разработку и развертывание приложений. Его контейнеры делают приложения гибкими, масштабируемыми и переносимыми между различными средами.

# **Makefile**

Makefile — это файл, используемый системой сборки Make для автоматизации задач в разработке программного обеспечения. Он описывает, как компилировать и собирать проекты, управляя зависимостями и процессами сборки.

Основные особенности Makefile:

1. Автоматизация сборки. Makefile позволяет автоматически компилировать и собирать проекты, избегая необходимости вручную выполнять каждую команду.

2. Управление зависимостями. С помощью Makefile можно определить зависимости между файлами, чтобы система сборки только перезапускала необходимые шаги при изменении исходных файлов.

3. Гибкость. Makefile поддерживает различные команды, сценарии и переменные, позволяя настраивать сборку под разные нужды проекта.

4. Удобство для командной работы. Использование Makefile позволяет всем разработчикам проекта работать с одинаковыми настройками сборки, минимизируя вероятность ошибок.

5. Поддержка различных языков. Makefile используется не только для C/C++ проектов, но и для других языков программирования, таких как Python, JavaScript и др.

Заключение

Makefile является важным инструментом для автоматизации процессов сборки и управления зависимостями. Это удобный и эффективный способ упрощения рабочих процессов и улучшения согласованности в команде разработчиков.

# **Сроки выполнения**

Разработка веб-сайта типа магазин включает несколько этапов, каждый из которых занимает определенное время. Ниже приведены примерные сроки для каждого из них:

1. Планирование и сбор данных (2-4 недели)

- Сбор данных об ассортименте

- Исследование целевой аудитории

- Сбор требований и функционала

2. Создание прототипа и дизайна (2-4 недели)

- Создание визуального дизайна (UI/UX)

3. Разработка (4-8 недель)

- Настройка серверной части и базы данных

- Реализация клиентской части (веб-интерфейса)

- Интеграция необходимых функций

4. Тестирование (2-3 недели)

- Функциональное тестирование

- Тестирование производительности и нагрузки

- Исправление найденных ошибок и недочетов

5. Запуск сайта (1 неделя)

- Настройка хостинга и домена

- Размещение сайта в интернете

Таким образом, общая продолжительность разработки веб-сайта типа магазин может составить от 11 до 20 недель в зависимости от сложности проекта и наличия ресурсов.

# **Вычислительные ресурсы для разработки сайта**

При разработке сайта типа интернет-магазин важно учитывать несколько факторов, связанных с вычислительными ресурсами. Оценка необходимых ресурсов может варьироваться в зависимости от масштабов проекта, предполагаемого трафика и выбранных технологий.

Здесь представлены основные рекомендации, основанные на различных аспектах:

1. Процессор (ЦП):

- Минимум. Двухъядерный процессор (например, Intel Core i3 или аналогичный).

- Рекомендуется. Четырехъядерный процессор (например, Intel Core i5 или аналогичный) для оптимальной работы со средами разработки и запуска локальных серверов.

2. Оперативная память (ОП):

- Минимум. 8 ГБ. Это достаточно для базовой разработки и работы с текстовыми редакторами или IDE.

- Рекомендуется.16 ГБ. Это обеспечит комфортную работу при одновременном запуске нескольких программ.

3. Накопитель (Жесткий диск):

- Минимум: HDD на 500 ГБ. Несмотря на достаточный объем, скорость работы может быть ниже.

- Рекомендуется: SSD на 512 ГБ или более. SSD значительно ускоряет загрузку операционной системы и программ, что очень важно для тяжелых программ.

4. Графическая карта. Для разработки сайта особых требований к графической карте нет, если только не планируется разработка фронтенда с использованием сложной графики или работа с видео.

5. Операционная система:

- Windows, macOS или Linux.

6. Дополнительные устройства:

- Мониторы.

- Клавиатура и мышь.

7. Интернет-соединение. Для разработки и тестирования веб-сайтов необходимо стабильное и быстрое интернет-соединение.

Заключение

Для комфортной работы над сайтом рекомендуется использовать ПК с четырехъядерным процессором, 16 ГБ оперативной памяти, SSD на 512 ГБ и современной операционной системой. Такой компьютер позволит разработчику выполнять свои задачи быстро и эффективно.

# **Документы ГОСТы**

В России процесс создания проектной документации подчиняется множеству стандартов и норм, среди которых значительное место занимают ГОСТы (Государственные стандарты). В данном разделе представлены основные из них, которые оказывают влияние на различные области разработки.

Основные ГОСТы:

1. ГОСТ 2.105-95 - «Единая система конструкторской документации (ЕСКД)». Этот стандарт определяет требования к оформлению конструкторской документации.

2. ГОСТ 19.101-77 - «Документация программных средств. Общие требования и правила оформления». Он устанавливает базовые требования к созданию документации для программного обеспечения.

3. ГОСТ Р 52766-2007 - «Системы менеджмента качества. Требования». Данный стандарт определяет требования к системам управления качеством в организациях.

4. ГОСТ Р ИСО 9001-2015 - аналог международного стандарта ISO 9001, который устанавливает требования для систем управления качеством.

5. ГОСТ 34.301-95 - «Информационная технология. Комплексы программных средств. Общие требования». Этот стандарт описывает требования к программному обеспечению.

Влияние на разработку:

Качество проектирования. Соблюдение ГОСТов способствует высокому качеству проектной документации и конечного продукта.

Соответствие законодательству. Следование этим стандартам необходимо для получения разрешений и гарантирования безопасности.

Упрощение разработки. Четкие регламенты облегчают процесс разработки и согласования проектной документации.

# **Заключение**

Темой дипломного проекта является «Исследование особенностей разработки интернет-магазина компьютерных комплектующих».

Основной целью дипломного проекта было изучение процесса и особенностей создания сайта для организации продажи технически сложных устройств.

Итогом дипломного проекта стала основа сайта, позволяющая клиентам (пользователям) магазина выбирать услуги и товары.

Сайт написан на языке разметки HTML и языках программирования JavaScript, TypeScript, а также GO отвечающий за бэкенд сайта с использованием фреймворка Vue.js.

Проанализировав работу по созданию сайта, можно сделать выводы об тяжелой реализации в одиночку и важности командной работы в серьезных проектах, а также об окупаемости с точки зрения узнаваемости организации. Многие функции существующих интернет-магазинов не были реализованы из-за трудностей проекта. Причинами также являются ограниченные сроки, нехватка знаний в нужном объеме и желание использовать современные инструменты разработки, которые требовали дополнительного изучения материала для их применения. Целью остается как развитие проекта, так и улучшение профессиональных знаний.

Сайт был разработан с целью исследовать грамотную структуру реализации проекта, связывая фронтенд и бэкенд разработку, а также организацию товара в интернет-магазине.

# **Список использованных источников**

1. Книга Jon Duckett HTML и CSS: Разработка и дизайн веб-сайтов" — М.-[Эксмо](https://www.labirint.ru/pubhouse/438/), 2020
2. Книга Zeldman, Jeffrey. Designing With Web Standards. Москва: ИД Питер, 2018.
3. Книга Максим Жашкевич. Язык Go для начинающих. 2020г.
4. Книга Keith, Jeremy. HTML5 for Web Designers. Оренбург: Бета-Пресс, 2021.
5. Книга Bo Ingram. ScyllaDB in Action: Manning, 2024г.
6. Книга Никсон Р.. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 6-е изд. . Москва: ИД Питер, 2024.
7. ГОСТ Р 12.0.004-2015. "Системы менеджмента охраны труда. Общие требования."
8. Руководство: Vuetify. <https://vuetifyjs.com/en/getting-started> /installation /#using-laravel-mix

# **Приложения**

Основное содержание диплома:

<https://github.com/IlyaNagachev/Project-SND>

Скриншоты сайта



Рисунок 1 Шапка сайта

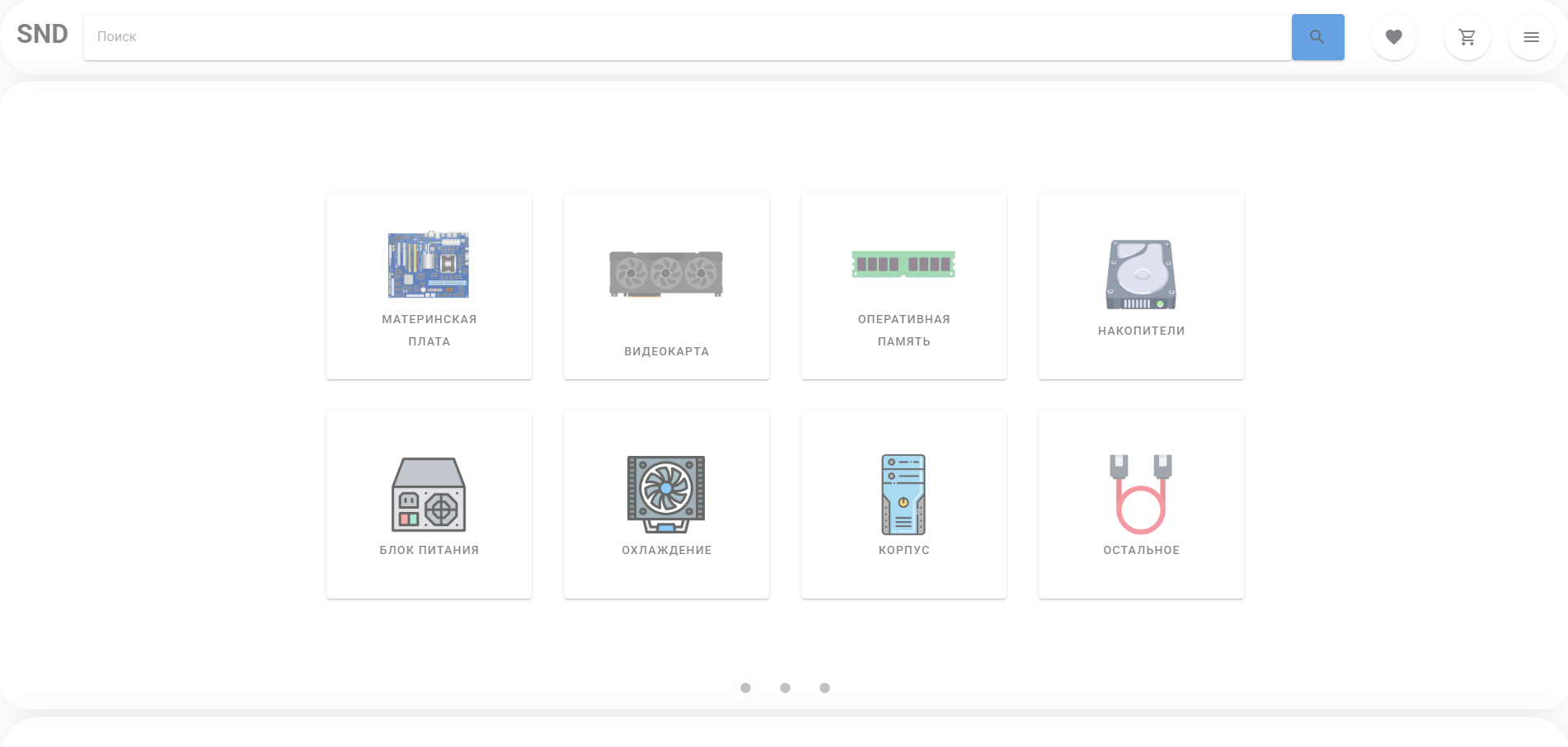


Рисунок 2 Главная страница (1 раздел)



Рисунок 3 Главная страница (2 раздел)

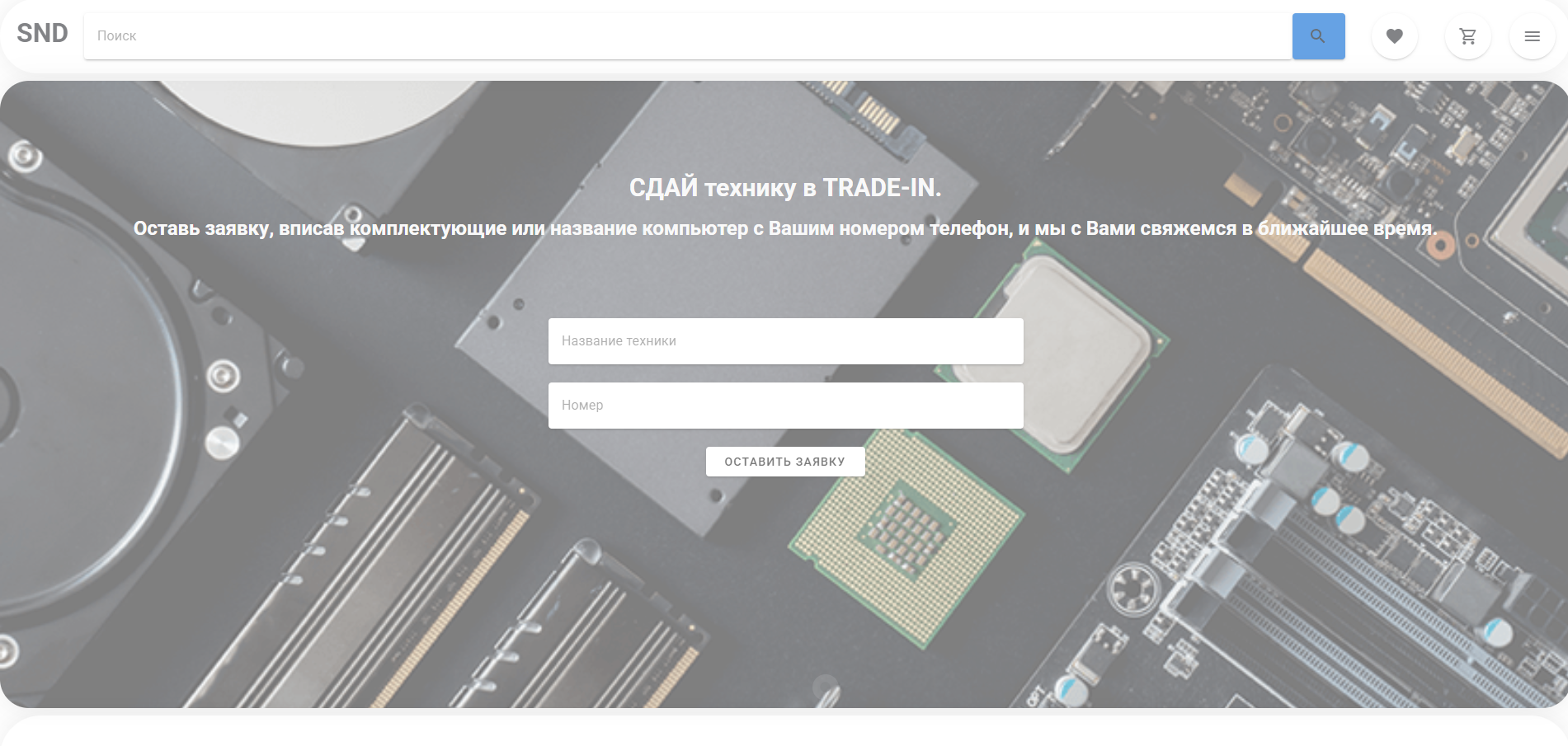


Рисунок 4 Главная страница (3 раздел)

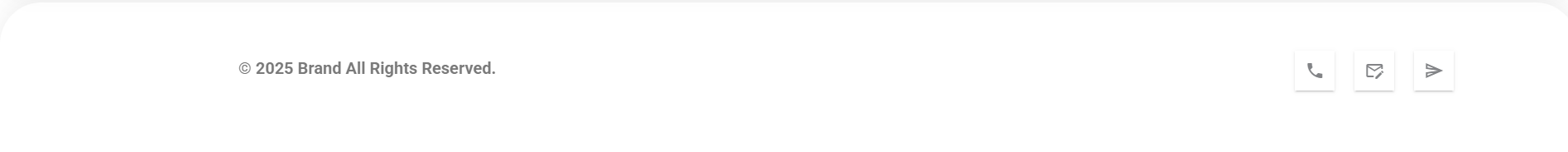


Рисунок 5 Подвал сайта

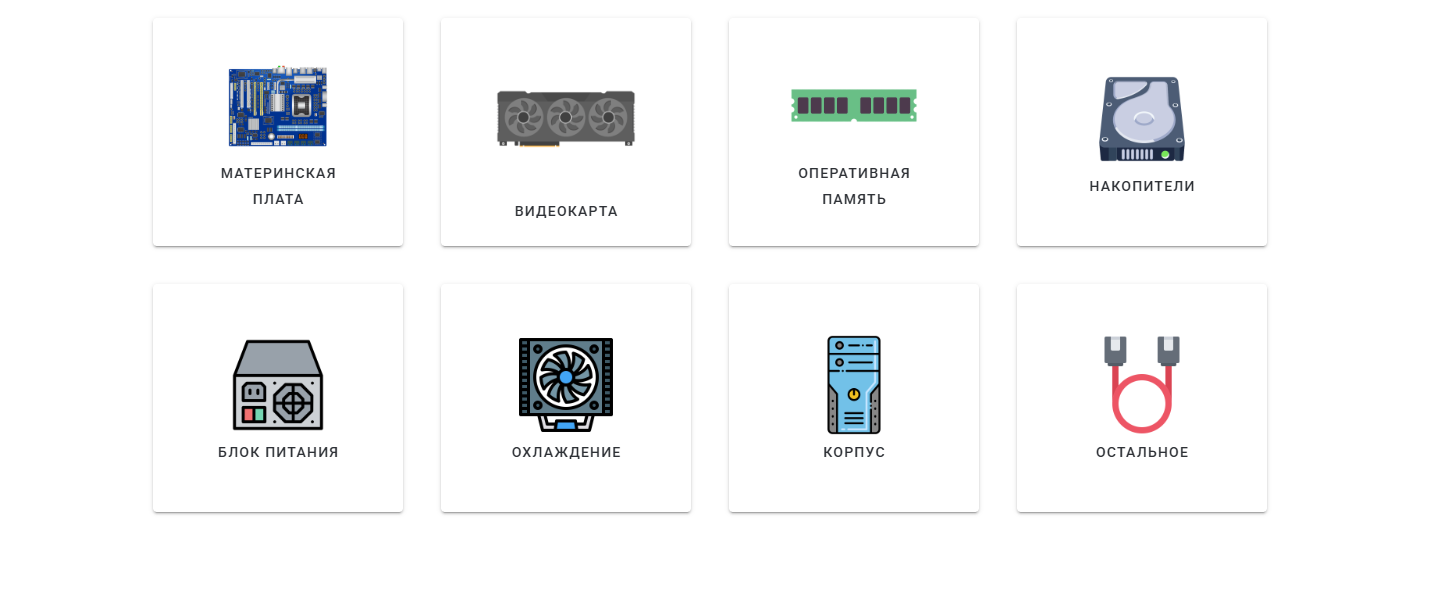


Рисунок 6 Каталог



Рисунок 7 Выбор товара



Рисунок 8 Карточка товара

Фрагменты кода

Makefile

start:  
 docker-compose -f ./docker-compose.yml up --build

index.html

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8">  
 <link rel="stylesheet" href="./style.css">  
 <link rel="icon" href="/favicon.ico">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Welcome to Vuetify 3</title>  
 </head>  
 <body>  
 <div id="app"></div>  
 <script type="module" src="/src/main.ts"></script>  
 </body>  
</html>

Dockerfile

FROM node:lts-alpine as *builder*WORKDIR /app  
COPY package\*.json ./  
RUN npm install  
COPY . .  
RUN npm run build  
  
  
FROM nginx:stable-alpine  
COPY nginx.conf /etc/nginx/conf.d/default.conf  
COPY --from=*builder* /app/dist /usr/share/nginx/html  
COPY --from=*builder* /app/src/assets /usr/share/nginx/html/assets  
EXPOSE 80

Dockerfile-backend

FROM golang:1.23.6-alpine as *build*WORKDIR /snd  
COPY go.mod ./  
COPY go.sum ./  
RUN go mod download  
COPY /backend /snd/backend  
WORKDIR /snd/backend/cmd  
RUN go build -o /snd/snd  
RUN echo test  
  
FROM alpine:latest  
COPY --from=*build* /snd/snd /snd/snd  
COPY --from=*build* /snd/backend/templates /snd/templates/  
COPY --from=*build* /snd/backend/products.json /snd/products.json  
RUN ls -la  
RUN echo test  
  
EXPOSE 8001 10500

Docker-compose.yml

version: '3.8'  
volumes:  
 scylla\_data:  
 driver: local  
networks:  
 test\_network:  
 driver: bridge  
  
services:  
 frontend:  
 ports:  
 - "8005:80"  
 container\_name: frontend  
 networks:  
 - test\_network  
 build: .  
 working\_dir: /app  
 volumes:  
 - .:/app  
 - /app/node\_modules  
  
  
 backend:  
 ports:  
 - "8001:10500"  
 container\_name: backend  
 networks:  
 - test\_network  
 build:  
 dockerfile: Dockerfile-backend  
 entrypoint: ["sh","-c", "sleep 7 && /snd/snd -config='/snd/templates/config.yaml'"]  
  
 stage-scylla:  
 image: scylladb/scylla:latest  
 container\_name: scylla  
 ports:  
 - "9043:9042"  
 - "10000:10000" *# Порт для API* - "7001:7000" *# Порт для межнодового соединения* volumes:  
 - scylla\_data:/var/lib/scylla  
 command: --developer-mode 1  
 networks:  
 - test\_network  
  
 stage-scylla-load-keyspace:  
 container\_name: stage-scylla-load-keyspace  
 image: scylladb/scylla:latest  
 volumes:  
 - ./create\_keyspace.cql:/scylla\_scripts/1.cql  
 - ./initdb.sh:/scylla\_scripts/initdb.sh  
 entrypoint: [ "bash", "/scylla\_scripts/initdb.sh" ]  
 networks:  
 - test\_network

nginx.conf

server {  
 listen 80;  
 server\_name \_ default\_server;  
 root /usr/share/nginx/html;  
 location / {  
 try\_files $uri $uri/ /index.html;  
 }  
}

Header.vue

import App from './App.vue'  
import \* as components from "vuetify/components";  
import \* as directives from "vuetify/directives";  
import '@mdi/font/css/materialdesignicons.css'  
import 'vuetify/styles'  
// Composables  
import { createApp } from 'vue'  
import *router* from './router'  
import { createVuetify } from 'vuetify'  
import {createPinia} from "pinia";  
const vuetify = createVuetify({  
 theme: {  
 defaultTheme: 'light',  
 },  
 components,  
 directives,  
})  
const app = createApp(App)  
app  
.use(vuetify)  
.use(*router*)  
.use(createPinia())  
app.mount('#app')

Footer.vue

<script setup lang="ts">  
  
import {mdiMagnify} from "@mdi/js";  
</script>  
  
<template>  
 <div class="footer">  
 <h1>© 2025 Brand All Rights Reserved.</h1>  
 <div class="footer\_\_button">  
 <v-col cols="auto">  
 <a href=»tel:1234567890″ target="\_blank"><v-btn icon="mdi-phone"></v-btn></a>  
 </v-col>  
 <v-col cols="auto">  
 <a href="https://gmail.com" target="\_blank"><v-btn icon="mdi-email-edit-outline"></v-btn></a>  
 </v-col>  
 <v-col cols="auto">  
 <a href="https://t.me" target="\_blank"><v-btn icon="mdi-send-outline" ></v-btn></a>  
 </v-col>  
 </div>  
 </div>  
</template>  
  
<style scoped lang="scss">  
  
.footer{  
 display:flex;  
 margin-top: 1vh;  
 justify-content: space-between;  
 align-content: center;  
 height: 20vh;  
 border-radius: 50px 50px 0 0;  
 box-shadow: 0px 0px 30px 0px rgba(0, 0, 0, 0.12);  
 background: white;  
 opacity: 0.6;  
 h1{  
 margin-top: 7vh;  
 margin-left: 15vw;  
 font-size: 20px;  
 }  
 &\_\_button{  
 display: flex;  
 flex-direction: row;  
 margin-top: 5vh;  
 margin-right: 15vh;  
 button{  
 border-radius: 0 **!important**;  
 }  
 }  
}  
</style>

MainPage.vue

<template>  
 <Header/>  
 <div>  
 <section style="height: 83.5vh; margin-top: 1vh; border-radius: 35px; box-shadow: 0px 0px 30px 0px rgba(0, 0, 0, 0.12); background: white; opacity: 0.6;">  
 <v-carousel style="height: 100%; border-radius: 35px"  
 show-arrows="hover"  
 cycle  
 hide-delimiter-background  
 :interval="30000"  
 >  
 <v-carousel-item  
 v-for="(slide, i) in slides"  
 :key="i"  
 >  
 <v-sheet  
 height="100%"  
 >  
 <div v-if="i==0" class="d-flex fill-height justify-center align-center">  
 <GlobalCatalog/>  
 </div>  
 <div v-if="i==1" class="d-flex fill-height justify-center align-center">  
 <oldpc-cover/>  
 </div>  
 <div v-if="i==2" class="d-flex fill-height justify-center align-center">  
 <trade-in-application/>  
 </div>  
 </v-sheet>  
 </v-carousel-item>  
 </v-carousel>  
 </section>  
 </div>  
 <Footer/>  
  
</template>  
  
<script setup lang="ts">  
import { mdiMagnify } from '@mdi/js';  
  
import {onMounted, ref} from 'vue';  
import {ProductStore} from "../../store/store";  
import Header from "./Header.vue";  
import {red} from "vuetify/util/colors";  
import CatalogMenu from "../product/catalogMenu.vue";  
import Footer from "./Footer.vue";  
import OldpcCover from "../product/oldpcCover.vue";  
import TradeInApplication from "../product/tradeInApplication.vue";  
import GlobalCatalog from "../product/GlobalCatalog.vue";  
  
const log = ref<string>("")  
const passwd = ref<string>("")  
  
const print = () => {  
 *console*.log(log.value);  
 *console*.log(passwd.value);  
}  
  
  
const slides = [  
 0,  
 1,  
 2,  
 ]  
  
  
const userStore = ProductStore()  
  
onMounted(()=>{  
 userStore.GetProducts()  
})  
</script>

Catalog.vue

<template>  
 <Header/>  
 <div>  
 <section style="height: 120vh; margin-top: 1vh; border-radius: 35px; box-shadow: 0px 0px 30px 0px rgba(0, 0, 0, 0.12); background: white; opacity: 0.6">  
 <div style="display:none">  
 {{selectClass}}  
 </div>  
 <v-virtual-scroll  
 :height="1200"  
 :items="products"  
 >  
 <template class="cardList" v-slot:default="{ item }">  
 <router-link to="./product" @click="productStore.SelectedProducts=item">  
 <div class="card">  
 <img :src="item.img" style="margin-top: 5vh; height: 20vh">  
 <h2>{{item.name}}</h2>  
 <h3>{{item.price}}</h3>  
 </div>  
 </router-link>  
  
 </template>  
 </v-virtual-scroll>  
 </section>  
 </div>  
 <Footer/>  
  
</template>  
  
<script setup lang="ts">  
import Header from "../general/Header.vue";  
import Footer from "../general/Footer.vue";  
import {inject, onMounted, ref} from "vue";  
import {Product} from "../../models/requests";  
import {ProductStore} from "../../store/store";  
  
const selectClass = ref<string>(inject<string>("selectedClass")||"")  
  
  
const products = ref<Product[]>([])  
const productStore = ProductStore()  
  
onMounted(()=> {  
 *console*.log(productStore.Products)  
 *console*.log(selectClass.value)  
 productStore.Products.forEach((value) => {  
 if (value.class == selectClass.value){  
 products.value.push(value);  
 }  
 })  
 *console*.log(products.value)  
})  
  
  
  
  
</script>  
  
  
<style scoped lang="scss">  
.cardList{  
  
}  
.card{  
 color: #2e3136;  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 height: 30vh;  
 gap: 2vh;  
 margin-bottom: 10vh;  
}  
</style>

catalogMenu.vue

<template>  
<div class="catalogMenu">  
 <router-link to="/catalog">  
 <div class="button container">  
 <v-btn @click="openMenu = true; classProduct = 'Материнская плата'" style="height: 25vh; width: 13vw; margin: 1vw">  
 <v-img width="100" height="100" src="../../assets/images/catalog/materinskaja\_plata.png"></v-img>  
 <p>Материнская</p>  
 <p>плата</p>  
 </v-btn>  
 <v-btn @click="openMenu = true; classProduct = 'Видеокарта'" style="height: 25vh; width: 13vw; margin: 1vw">  
 <v-img width="150" height="150" src="../../assets/images/catalog/graphic-card.png"></v-img>  
 <p>Видеокарта</p>  
 </v-btn>  
 <v-btn @click="openMenu = true; classProduct = 'Оперативная память'" style="height: 25vh; width: 13vw; margin: 1vw">  
 <v-img width="150" height="100" src="../../assets/images/catalog/ram.png"></v-img>  
 <p>Оперативная</p>  
 <p>память</p>  
 </v-btn>  
 <v-btn @click="openMenu = true; classProduct = 'Накопители'" style="height: 25vh; width: 13vw; margin: 1vw">  
 <v-img width="100" height="100" src="../../assets/images/catalog/hdd.png"></v-img>  
 <p>Накопители</p>  
 </v-btn>  
 <v-btn @click="openMenu = true; classProduct = 'Блок питания'" style="height: 25vh; width: 13vw; margin: 1vw">  
 <v-img width="100" height="100" src="../../assets/images/catalog/psu.png"></v-img>  
 <p>Блок питания</p>  
 </v-btn>  
 <v-btn @click="openMenu = true; classProduct = 'Охлаждение'" style="height: 25vh; width: 13vw; margin: 1vw">  
 <v-img width="100" height="100" src="../../assets/images/catalog/cooler.png"></v-img>  
 <p>Охлаждение</p>  
 </v-btn>  
 <v-btn @click="openMenu = true; classProduct = 'Корпус'" style="height: 25vh; width: 13vw; margin: 1vw">  
 <v-img width="100" height="100" src="../../assets/images/catalog/case.png"></v-img>  
 <p>Корпус</p>  
 </v-btn>  
 <v-btn @click="openMenu = true; classProduct = 'Остальное'" style="height: 25vh; width: 13vw; margin: 1vw">  
 <v-img width="100" height="100" src="../../assets/images/catalog/other.png"></v-img>  
 <p>Остальное</p>  
 </v-btn>  
 </div>  
 </router-link>  
</div>  
</template>  
  
  
<script setup lang="ts">  
import {inject, provide, ref} from "vue";  
import {mdiMagnify} from "@mdi/js";  
  
const openMenu = inject<boolean>("open")  
const classProduct = inject<string>("selectedClass")  
</script>  
  
<style lang="scss">  
.catalogMenu {  
 display: flex;  
 max-width: 70vw;  
 flex-direction: column;  
 align-items: center;  
 justify-content: center;  
 box-sizing: border-box;  
 margin-left: 10vw;  
}  
.v-btn\_\_content {  
 color: #2e3136;  
 display: flex;  
 flex-direction: column;  
 gap: 1vh;  
}  
</style>

GlobalCatalog.vue

<template>  
  
 <catalogMenu v-if="!openMenu "/>  
 <Catalog v-if="openMenu"/>  
</template>  
  
<script setup lang="ts">  
import {provide, ref} from "vue";  
import CatalogMenu from "./catalogMenu.vue";  
import Catalog from "./catalog.vue";  
  
  
const selectClass = ref<string>("")  
provide("selectedClass", selectClass)  
  
const openMenu = ref<boolean>(false)  
provide("open", openMenu)  
</script>  
  
oldpcCover.vue

<template>  
<div class="oldpcCover">  
 <h1>PC КАК НОВЫЕ</h1>  
 <div class="oldpcCover\_\_variants">  
 <v-btn style="height: 40vh; width: 15vw; margin-right: 2vw">  
 <v-img width="300" height="300" src="../../assets/images/oldpc/1.png"></v-img>  
 <p>i3 rx580</p>  
 </v-btn>  
 <v-btn style="height: 40vh; width: 15vw; margin-right: 2vw">  
 <v-img width="300" height="300" src="../../assets/images/oldpc/2.png"></v-img>  
 <p>i5 rtx2060</p>  
 </v-btn>  
 <v-btn style="height: 40vh; width: 15vw; margin-right: 2vw">  
 <v-img width="250" height="300" src="../../assets/images/oldpc/3.png"></v-img>  
 <p>ryzen 5 gtx1060</p>  
 </v-btn>  
 <v-btn style="height: 40vh; width: 15vw">  
 <v-img width="300" height="300" src="../../assets/images/oldpc/2.png"></v-img>  
 <p>ryzen 5 rtx2060</p>  
 </v-btn>  
 </div>  
 <v-btn style="height: 5vh; width: 13vw">  
 <p>Посмотреть все варианты</p>  
 </v-btn>  
  
</div>  
</template>  
<script setup lang="ts">  
  
import {mdiMagnify} from "@mdi/js";  
</script>  
  
<style lang="scss">  
.oldpcCover{  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 gap: 4vw;  
 button{  
 }  
 .v-btn\_\_content{  
 display:flex;  
 flex-direction:column;  
 gap: 1vh;  
 }  
}  
</style>

ProductPage.vue

<template>  
 <Header/>  
 <div class="productPage">  
 <section class="productPage\_\_photo">  
 <v-carousel style="height: 95%; border-radius: 35px"  
 show-arrows="hover"  
 cycle  
 hide-delimiter-background  
 :interval="30000"  
 >  
 <v-carousel-item ><v-img style="width: 100%" :src="product?.img"></v-img></v-carousel-item>  
 <v-carousel-item><v-img style="width: 100%" :src="product?.img"></v-img></v-carousel-item>  
 <v-carousel-item><v-img style="width: 100%" :src="product?.img"></v-img></v-carousel-item>  
 </v-carousel>  
 </section>  
 <div class="productPage\_\_down">  
 <div class="productPage\_\_down\_\_buy">  
 <v-col cols="auto">  
 <v-btn style="border-radius: 2vh; height: 6vh" icon="mdi-heart" size="large"></v-btn>  
 </v-col>  
 <h1>{{product?.price}}</h1>  
 <v-btn style="border-radius: 2vh; height: 6vh" >Купить</v-btn>  
 </div>  
 <div class="productPage\_\_down\_\_info">  
 <h1>{{product?.name}}</h1>  
 <p>{{product?.description}}</p>  
 <h2>{{product?.specs}}</h2>  
 </div>  
 </div>  
 </div>  
 <Footer/>  
</template>  
  
<style scoped lang="scss">  
.productPage{  
 margin-top: 1vh;  
 border-radius: 35px;  
 box-shadow: 0px 0px 30px 0px rgba(0, 0, 0, 0.12);  
 background: white;  
 &\_\_photo{  
 height: 83.5vh;  
 opacity: 0.6;  
 }  
 &\_\_down{  
 display: flex;  
 align-items: center;  
 justify-content: center;  
 flex-direction: column;  
 gap: 10vh;  
 margin-bottom: 5vh;  
 &\_\_buy{  
 display: flex;  
 align-items: center;  
 gap: 10vw;  
  
 }  
 &\_\_info{  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 gap: 7vh;  
 }  
 }  
  
}  
</style>  
  
<script setup lang="ts" xmlns="http://www.w3.org/1999/html">  
  
import Header from "../general/Header.vue";  
import Footer from "../general/Footer.vue";  
import CatalogMenu from "./catalogMenu.vue";  
import OldpcCover from "./oldpcCover.vue";  
import TradeInApplication from "./tradeInApplication.vue";  
import {onMounted, onUnmounted, ref} from "vue";  
import {Product} from "../../models/requests";  
import {ProductStore} from "../../store/store";  
  
const product = ref<Product>()  
const userStore = ProductStore()  
  
onMounted(()=>{  
 product.value = userStore.SelectedProducts  
 *console*.log(product.value)  
})  
</script>

tradeInApplication.vue

<template>  
<div class="tradeInApplication">  
 <div class="tradeInApplication\_\_form">  
 <h1>СДАЙ технику в TRADE-IN.</h1>  
 <h2>Оставь заявку, вписав комплектующие или название компьютер с Вашим номером телефон, и мы с Вами свяжемся в ближайшее время.</h2>  
 <v-form class="application">  
 <v-text-field style="width: 30vw" label="Название техники" variant="solo"></v-text-field>  
 <v-text-field style="width: 30vw" label="Номер" variant="solo"></v-text-field>  
 <v-btn>  
 <template v-slot:prepend>  
 <p style="margin-left: 0.5vw">Оставить заявку</p>  
 </template>  
 </v-btn>  
 </v-form>  
 </div>  
  
</div>  
</template>  
  
  
<script setup lang="ts">  
  
import {mdiMagnify} from "@mdi/js";  
</script>  
  
  
  
<style lang="scss">  
.tradeInApplication{  
 background-image: url("../../assets/images/tradein/cover.png");  
 background-size: cover;  
 background-position: center;  
 width: 100vw;  
 height: 100vh;  
 &\_\_form{  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 margin-top: 20vh;  
 h1{  
 color: whitesmoke;  
 margin-bottom: 1vh;  
 font-size: 30px;  
 }  
 h2{  
 color: whitesmoke;  
 margin-bottom: 10vh;  
 }  
 .application{  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 }  
 }  
 .v-btn\_\_content{  
 display:flex;  
 gap: 1vh;  
 }  
}  
</style>

requests.ts

export enum Requests {  
 *Registration* = "user/reg",  
 *GetProduct* = "user/get",  
 *GetProductById* = "product/id"  
  
}  
export interface Product{  
 uuid : string,  
 class : string,  
 description : string,  
 img : string,  
 name : string,  
 price : number,  
 specs : Record<string, string>  
}

user.ts

export interface User {  
 name: string  
 password: string  
}  
  
export interface SucceedResponse {  
 msg: string  
}

config.ts

class Config {  
 private readonly prodHost : string  
 private readonly devHost : string  
  
 constructor() {  
 this.prodHost = "http://localhost:8001"  
 this.devHost = "http://localhost:8001"  
 }  
  
 getProdUrl() : string {return this.prodHost}  
 getDevUrl() : string {return this.devHost}  
}  
  
const *config* : Config = new Config()  
  
export {*config*}

network.ts

import *axios*, {AxiosInstance, AxiosRequestConfig, AxiosResponse} from "axios";  
import {*config*} from "./config";  
  
class Network {  
 private userApi : AxiosInstance = axios.create({  
 baseURL: *config*.getProdUrl(),  
 timeout: 1000,  
 headers: {"Content-Type" : "application/json"}  
 });  
  
 UserPost<T>(url : string, payload : Record<string,any>){  
 return this.sendPost<T>(url, payload)  
 }  
  
 private sendPost<T>(url : string, payload : Record<string,any>):Promise<T>{  
 const r\_cfg : AxiosRequestConfig = {  
 timeout : 30000,  
 withCredentials : true  
 }  
 return new Promise<T>((resolve, reject) => {  
 this.userApi.post(url, payload, r\_cfg)  
 .then((response : AxiosResponse) => {  
 resolve(response.data)  
 })  
 .catch((err) => {  
 reject(err)  
 })  
 })  
 }  
  
}  
const *network* : Network = new Network()  
export default *network*

index.ts

import { createRouter, createWebHistory } from 'vue-router'  
import MainPage from '../components/general/MainPage.vue'  
import ProductPage from "../components/product/ProductPage.vue";  
import CatalogPage from "../components/product/GlobalCatalog.vue";  
  
const *router* = createRouter({  
 history: createWebHistory(),  
 routes: [{path: "/main", component: MainPage},  
 {path: "/product", component: ProductPage},  
 {path: "/catalog", component: CatalogPage}],  
})

*router*.isReady().then(() => {  
 *localStorage*.removeItem('vuetify:dynamic-reload')  
})  
  
export default *router*

store.ts

import {SucceedResponse, User} from "../models/user";  
import {defineStore} from "pinia";  
import *network* from "../network/network";  
import {Product, Requests} from "../models/requests";  
  
interface state {  
 Products : Product[],  
 SelectedProducts : Product  
}  
  
export const ProductStore = defineStore("users", {  
 state : () : state => ({  
 Products : <Product[]>{},  
 SelectedProducts: <Product>{}  
 }),  
 actions : {  
 Registration(login : string, password : string) : Promise<SucceedResponse>{  
 return new Promise<SucceedResponse>((resolve, reject) => {  
 *network*.UserPost<SucceedResponse>(Requests.*Registration*, {  
 login : login,  
 password : password  
 })  
 .then((r) => {  
 resolve(r)  
 return(r)  
 })  
 .catch((err) => {  
 reject(err)  
 })  
 })  
 },  
 GetProducts() : Promise<Product[]>{  
 return new Promise<Product[]>((resolve, reject) => {  
 *network*.UserPost<Product[]>(Requests.*GetProduct*, {})  
 .then((r) => {  
 resolve(r)  
 this.Products = r  
 return(r)  
 })  
 .catch((err) => {  
 reject(err)  
 })  
 })  
 },  
  
 GetProduct(uuid : number) : Promise<Product>{  
 return new Promise<Product>((resolve, reject) => {  
 *network*.UserPost<Product>(Requests.*GetProductById*, {uuid : uuid})  
 .then((r) => {  
 resolve(r)  
 return(r)  
 })  
 .catch((err) => {  
 reject(err)  
 })  
 })  
 }  
 }  
})

test.http

POST localhost:8001/user/get  
###  
POST localhost:8001/user/create  
*Content-Type*: application/json  
  
{  
 "uuid" : "2",  
 "description" : "KNihniuojoisgnosijg",  
 "img" : "dggfd",  
 "name" : "GRX 1060",  
 "price" : 200,  
 "specs" : {  
 "Размер" : "230x560",  
 "Производитель" : "Nvidia"  
 }  
}

main.go

package main  
  
import (  
 "encoding/json"  
 "flag"  
 "github.com/rs/zerolog/log"  
 "net/http"  
 "os"  
 snd "snd/backend"  
 "snd/backend/logs"  
 "snd/backend/network"  
 "snd/backend/store"  
 "snd/backend/store/models"  
)  
var (  
 config \*snd.Config  
 configPath = flag.String("config", "./snd/templates/config.yaml", "config File Path")  
 err error  
)  
func init() {  
 flag.Parse()  
 if config, err = snd.NewConfig(configPath); err != nil {  
 panic(err)  
 }  
 if err = logs.NewLogger(config.Logger); err != nil {  
 panic(err)  
 }  
}  
func main() {  
 log.Info().Msg("Hello")  
 var server http.Server  
 var router http.Handler  
 router = network.CreateRouter()  
 server = network.NewServer(config.Network, router)  
 if store.Default, err = store.NewStore(config.Store); err != nil {  
 panic(err)  
 }  
 if p,\_ := store.Default.Products.GetAll(); len(p)<4 {  
 var file []byte  
 if file, err = os.ReadFile("./snd/products.json"); err != nil {  
 panic(err)  
 }  
 var products []models.Product  
 if err = json.Unmarshal(file, &products); err != nil {  
 panic(err)  
 }  
 for \_,p :=range products {  
 if err = store.Default.Products.New(&p); err != nil {  
 panic(err)  
 }  
 }  
 }  
  
 server.ListenAndServe()  
}

logs.go

package logs  
import (  
 "github.com/rs/zerolog"  
 "github.com/rs/zerolog/log"  
 "os"  
)  
type Config struct {  
 Level string `yaml:"level"`  
}  
func NewLogger(config \*Config) (err error) {  
 var level zerolog.Level  
 switch config.Level {  
 case "debug": level = zerolog.*DebugLevel* case "info": level = zerolog.*InfoLevel* case "warning": level = zerolog.*WarnLevel* case "error": level = zerolog.*ErrorLevel* }  
 log.Logger = log.With().Caller().Logger().Level(level).Output(zerolog.ConsoleWriter{Out: os.Stderr})  
 return  
}

config.go

package network  
  
type Config struct {  
 Address string `yaml:"address"`  
 Port int `yaml:"port"`  
}

handlers.go

package network  
  
import (  
 "encoding/json"  
 "fmt"  
 "io"  
 "net/http"  
 "snd/backend/store"  
 "snd/backend/store/models"  
 "strconv"  
)  
func createProduct() http.HandlerFunc {  
 return func(writer http.ResponseWriter, request \*http.Request) {  
 var err error  
 var bytes []byte  
 if bytes, err = io.ReadAll(request.Body); err != nil{  
 writer.Write([]byte("Ошибка" + err.Error()))  
 return  
 }  
 var product \*models.Product  
  
  
 if err = json.Unmarshal(bytes, &product); err != nil{  
 writer.Write([]byte("Ошибка" + err.Error()))  
 return  
 }  
  
 if err = store.Default.Products.New(product); err != nil{  
 writer.Write([]byte("Ошибка" + err.Error()))  
 return  
 }  
 }  
}  
func getProduct() http.HandlerFunc {  
 return func(writer http.ResponseWriter, request \*http.Request) {  
 var err error  
 var products []models.Product  
 if products, err = store.Default.Products.GetAll(); err != nil{  
 writer.Write([]byte("Ошибка" + err.Error()))  
 return  
 }  
  
 var bytes []byte  
  
 if bytes, err = json.Marshal(products); err != nil{  
 writer.Write([]byte("Ошибка" + err.Error()))  
 return  
 }  
  
 writer.Write(bytes)  
 }  
}  
func getProductById() http.HandlerFunc {  
 return func(writer http.ResponseWriter, request \*http.Request) {  
 var err error  
 var product models.Product  
 var in = struct {  
 Uuid int `json:"uuid"`}{}  
 var inbytes []byte  
 if inbytes, err = io.ReadAll(request.Body); err != nil{  
 writer.Write([]byte("Ошибка" + err.Error()))  
 return  
 }  
 if err = json.Unmarshal(inbytes, &in); err != nil{  
 writer.Write([]byte("Ошибка" + err.Error()))  
 return  
 }  
 fmt.Println(in)  
 if product, err = store.Default.Products.Get(strconv.Itoa(in.Uuid)); err != nil{  
 writer.Write([]byte("Ошибка" + err.Error()))  
 return  
 }  
 var bytes []byte  
 if bytes, err = json.Marshal(product); err != nil{  
 writer.Write([]byte("Ошибка" + err.Error()))  
 return  
 }  
 writer.Write(bytes)  
 }  
}

router.go

package network  
  
import (  
 "github.com/go-chi/chi/v5"  
 "github.com/rs/zerolog/log"  
 "net/http"  
 "strings"  
)  
func CreateRouter() http.Handler {  
 var router = chi.NewRouter()  
 router.Use(validateRequest)  
 router.Post("/user/info", userInfo())  
 router.Post("/user/create", createProduct())  
 router.Post("/user/get", getProduct())  
 router.Post("/product/id", getProductById())  
 return router  
}  
func userInfo() http.HandlerFunc {  
 return func(writer http.ResponseWriter, request \*http.Request) {  
 log.Info().Msg(request.RequestURI)  
 }  
}  
func validateRequest(next http.Handler) http.Handler {  
 return http.HandlerFunc( func(w http.ResponseWriter, r \*http.Request) {  
 var origin string  
 var allowed\_headers = strings.Join([]string{  
 "origin",  
 "content-type",  
 }, ", ")  
 if origin = r.Header.Get("Origin"); origin == "" {origin = "\*"}  
 w.Header().Set("Access-Control-Allow-Origin", origin)  
 w.Header().Set("Access-Control-Allow-Credentials", "true")  
 w.Header().Set("Access-Control-Allow-Methods", "GET, POST, HEAD, PATCH, PUT, DELETE, OPTIONS")  
 w.Header().Set("Access-Control-Allow-Headers", allowed\_headers)  
 w.Header().Set("Access-Control-Expose-Headers", "\*")  
  
 if r.Method == http.*MethodOptions* {  
 w.WriteHeader(http.*StatusOK*)  
 return  
 }  
 next.ServeHTTP(w, r)  
 })  
}

server.go

package network  
  
import (  
 "fmt"  
 "net/http"  
)  
func NewServer(config \*Config, h http.Handler) http.Server{  
 return http.Server{  
 Addr: fmt.Sprintf("%s:%d", config.Address, config.Port),  
 Handler: h,  
 }  
}

config.go

package scylla  
  
import (  
 "errors"  
  
 "strings"  
 "time"  
)  
type Config struct {  
 IPAddress string `yaml:"ip\_address"`  
 Keyspace string `yaml:"keyspace"`  
  
 //Tables  
 ProductTable string `yaml:"products\_table"`  
  
 //Settings  
 Port int `yaml:"port"`  
 MaxRetries int `yaml:"max\_retries"`  
 NumCons int `yaml:"num\_cons"`  
 Timeout time.Duration `yaml:"timeout"`  
 IntervalCons time.Duration `yaml:"interval\_cons"`  
 QueryTimeout time.Duration `yaml:"query\_timeout"`  
}  
func(c \*Config) Validate() (err error) {  
 if c == nil {return errors.New("nil mysql config")}  
  
 if c.Keyspace = strings.TrimSpace(c.Keyspace); c.Keyspace == "" {return errors.New("empty Keyspace string")}  
 if c.IPAddress = strings.TrimSpace(c.IPAddress); c.IPAddress == "" {return errors.New("empty IPAddress string")}  
  
 if c.ProductTable = strings.TrimSpace(c.ProductTable); c.ProductTable == "" {return errors.New("empty UserTable string")}  
  
 if c.Port == 0 {return errors.New("empty Port string")}  
 if c.Timeout.Seconds() < 1 {return errors.New("timeout for scylla must be greaten than 1s")}  
  
 return  
}

scylla.go

package scylla  
  
import (  
 "fmt"  
 "github.com/scylladb/gocqlx/v3"  
 "snd/backend/store/models"  
  
 "github.com/gocql/gocql"  
 "github.com/scylladb/gocqlx/v3/table"  
)  
  
type Adapter struct {  
 config \*Config  
 session gocqlx.Session // Use getter for get connection for private methods too  
 tables \*models.Tables  
}  
  
func NewAdapter(config \*Config) (a \*Adapter, err error) {  
 var cluster\_cfg = gocql.NewCluster(config.IPAddress)  
  
 cluster\_cfg.Port = config.Port  
 cluster\_cfg.Consistency = gocql.*Quorum* cluster\_cfg.Keyspace = config.Keyspace  
 cluster\_cfg.Timeout = config.Timeout  
 cluster\_cfg.Timeout = config.QueryTimeout  
 cluster\_cfg.NumConns = config.NumCons  
 cluster\_cfg.ReconnectionPolicy = &gocql.ConstantReconnectionPolicy{  
 MaxRetries : config.MaxRetries,  
 Interval : config.IntervalCons,  
 }  
  
 a = &Adapter{config : config}  
 a.tables = &models.Tables{  
 ProductTable : table.New(productMetadata),  
  
 }  
 if a.session, err = gocqlx.WrapSession(cluster\_cfg.CreateSession()); err != nil {return}  
  
 if err = a.syncTables(); err != nil {return}  
  
 return  
}  
func(a \*Adapter) GetSession() (session gocqlx.Session) {return a.session}  
  
func(a \*Adapter) syncTables() (err error) {  
 if err = a.session.ExecStmt(fmt.Sprintf(`CREATE TABLE IF NOT EXISTS %s (  
 uuid text PRIMARY KEY,  
 class text,  
 name text,  
 price float,  
 img text,  
 description text,   
 specs map<text, text>,  
 )`, a.config.ProductTable)); err != nil {return}  
  
 return  
}  
  
func(a \*Adapter) GetTables() \*models.Tables {  
 if a == nil {return nil}  
 return a.tables  
}

tpl.go

package scylla  
  
import (  
 "github.com/scylladb/gocqlx/v3/table"  
 "snd/backend/store/models"  
)  
  
var productMetadata = table.Metadata{  
 Name : models.*ProductTable*,  
 Columns : []string{"uuid", "class", "name", "price", "img", "description", "specs"},  
 PartKey : []string{"uuid"},  
 SortKey : []string{"uuid"},  
}

models.go

package models  
import "github.com/scylladb/gocqlx/v3/table"  
const *ProductTable* = "products"  
type Tables struct {  
 ProductTable \*table.Table  
}  
type Products []\*Product  
type Product struct{  
 Uuid string `gorm:"uuid" json:"uuid"`  
 Class string `db:"class" json:"class"`  
 Name string `db:"name" json:"name"`  
 Price float32 `db:"price" json:"price"`  
 Img string `db:"img" json:"img"`  
 Description string `db:"description" json:"description"`  
 Specs map[string]string `db:"specs" json:"specs"`  
}

getters.go

package products  
  
import (  
 "encoding/json"  
 "fmt"  
 "github.com/scylladb/gocqlx/v3/qb"  
 "snd/backend/store/models"  
)  
func(s \*Storage) Get(uuid string) (product models.Product, err error){  
 err = s.session.Query(qb.Select(models.*ProductTable*).Where(qb.Eq("uuid")).ToCql()).BindMap(qb.M{"uuid":uuid}).GetRelease(&product)  
 return  
}  
func(s \*Storage) GetAll() (product []models.Product, err error){  
 var m []map[string]interface{}  
 if m, err = s.session.Session.Query(fmt.Sprintf("SELECT \* FROM %s", models.*ProductTable*)).Iter().SliceMap(); err != nil{  
 return  
 }  
 var bytes []byte  
  
 if bytes, err = json.Marshal(m); err != nil{  
 return  
 }  
 if err = json.Unmarshal(bytes, &product); err != nil{  
 return  
 }  
 return  
}

setters.go

package products  
import "snd/backend/store/models"  
func(s \*Storage) New(product \*models.Product) (err error){  
 return s.session.Query(s.Tables.ProductTable.Insert()).BindStruct(product).Exec()  
}

products.go

package products  
  
import (  
 "context"  
 "github.com/scylladb/gocqlx/v3"  
 "snd/backend/store/models"  
)  
type Storage struct {  
 ctx context.Context  
 session gocqlx.Session  
 Tables \*models.Tables  
}  
func NewStorage(ctx context.Context, session gocqlx.Session, tables \*models.Tables) (s \*Storage, err error) {  
 return &Storage{ctx: ctx, session: session, Tables: tables}, nil  
}

config.go

package store  
  
import "snd/backend/store/adapters/scylla"  
type Config struct {  
 Scylla \*scylla.Config `yaml:"scylla"`  
}

store.go

package store  
  
import (  
 "context"  
 "snd/backend/store/adapters/scylla"  
 "snd/backend/store/products"  
)  
func NewStore(config \*Config) (s \*Store, err error) {  
 s = &Store{}  
 if s.scylla, err = scylla.NewAdapter(config.Scylla); err != nil {  
 return  
 }  
 if s.Products, err = products.NewStorage(context.Background(),s.scylla.GetSession(), s.scylla.GetTables()); err != nil {  
 return  
 }  
 return  
}  
type Store struct {  
 scylla \*scylla.Adapter  
 Products \*products.Storage  
}  
var Default \*Store

config.yaml

network:  
 address: "backend"  
 port: 10500  
  
logger:  
 level: "debug"  
  
store:  
 scylla:  
 ip\_address: "scylla"  
 keyspace: "snd\_project"  
 port: 9042  
 timeout: "10s"  
 max\_retries: 5  
 interval\_cons: "3s"  
 query\_timeout: "1m"  
 num\_cons: 3  
 products\_table: "products"