**Geekbrains**

**Разработка Backend-сервиса для управления задачами с использованием Spring Framework и микросервисной архитектуры.**

IT-специалист:

Программист Java. Цифровые профессии

Ямпольский И.В.

Санкт-Петербург

2024

Оглавление

[1. Введение 3](#_Toc161756712)

[1.1 Постановка проблемы и актуальность темы 3](#_Toc161756713)

[1.2 Цели и задачи проекта 4](#_Toc161756714)

[1.3 Основные этапы разработки проекта. 6](#_Toc161756715)

[2. Разработка проекта 8](#_Toc161756716)

[2.1 Планирование и проектирование 8](#_Toc161756717)

[**2.1.1** **Требования к сервису** 8](#_Toc161756718)

[**2.1.2** **Проектирование баз данных для микросервисов** 10](#_Toc161756719)

[**2.1.3** **Определение структуры API** 10](#_Toc161756720)

[2.2 Настройка проекта 10](#_Toc161756721)

[**2.2.1** **Создание проекта** 10](#_Toc161756722)

[**2.2.2** **Подключение и настройка зависимостей** 10](#_Toc161756723)

[2.3 Разработка моделей данных 10](#_Toc161756724)

[**2.3.1** **Создание сущностей для представления в базе данных** 10](#_Toc161756725)

[**2.3.2** **Определение связей между сущностями** 10](#_Toc161756726)

[2.4 Разработка API контроллеров 10](#_Toc161756727)

[**2.4.1** **Создание контроллеров для обработки HTTP-запросов от клиентов** 10](#_Toc161756728)

[**2.4.2** **Определение эндпоинтов для создания, чтения, обновления и удаления задач.** 10](#_Toc161756729)

[**2.4.3** **Реализация логики обработки запросов в контроллерах** 10](#_Toc161756730)

[2.5 Разработка сервисного слоя 10](#_Toc161756731)

[**2.5.1** **Создание сервисов для обработки бизнес-логики, связанной с управлением задачами** 10](#_Toc161756732)

[**2.5.2** **Выделение повторяющейся логики в сервисы для повышения модульности и повторного использования** 10](#_Toc161756733)

[2.6 Настройка доступа к данным 10](#_Toc161756734)

[**2.6.1** **Настройка репозиториев Spring Data JPA для взаимодействия с базой данных.** 10](#_Toc161756735)

[**2.6.2** **Использование репозиториев для выполнения операций CRUD с объектами.** 10](#_Toc161756736)

[2.7 Аутентификация и авторизация 10](#_Toc161756737)

[**2.7.1** **Настройка Spring Security для обеспечения безопасности сервиса.** 10](#_Toc161756738)

[**2.7.2** **Реализация механизмов аутентификации и авторизации пользователей.** 10](#_Toc161756739)

[2.8 Тестирование 10](#_Toc161756740)

[**2.8.1** **Создание юнит-тестов для проверки отдельных компонентов сервиса.** 10](#_Toc161756741)

[**2.8.2** **Написание интеграционных тестов для проверки взаимодействия различных компонентов** 10](#_Toc161756742)

[**2.8.3** **Запуск автоматических тестов для обеспечения качества кода** 10](#_Toc161756743)

[2.9 Документирование 10](#_Toc161756744)

[**2.9.1** **Создание документации API с помощью Swagger** 10](#_Toc161756745)

[**2.9.2** **Написание комментариев к коду и пояснительной документации для разработчиков** 11](#_Toc161756746)

[2.10 Развертывание и мониторинг 11](#_Toc161756747)

[**2.10.1** **Развертывание приложения на сервере или в облачной среде** 11](#_Toc161756748)

[**2.10.2** **Настройка мониторинга для отслеживания производительности и доступности сервиса.** 11](#_Toc161756749)

[3. Заключение 12](#_Toc161756750)

[4. Список используемой литературы 13](#_Toc161756751)

[Пример 13](#_Toc161756752)

[5. Приложения 14](#_Toc161756753)

1. **Введение**
   1. **Постановка проблемы и актуальность темы**

Сегодня все большее число компаний и коллективов стремятся предложить свои услуги и продукты огромному числу пользователей в различных регионах нашей планеты. Некоторые из них предлагают свои сервисы в десятках стран мира, которые вовсе не находится рядом друг с другом географически.

Капиталистическое устройство мира прежде всего диктует требование к непрерывному увеличению прибыли, которое, в том числе, достигается за счет привлечения новых пользователей, и расширении имеющихся услуг.

Такое положение вещей вызывает необходимость решения проблем, связанных с обеспечением:

* непрерывности в доставке контента и услуг.
* поддержания должного уровня отказоустойчивости и масштабируемости.
* реализации механизмов по управлению сложностью.
* работы над ускорением разработки и доставки изменений потребителю.
* легкости во внедрении нового функционала и его тестированию.
* выбора оптимальных инструментов для решения поставленных задач.

С появлением мобильных приложений, интернета вещей (IoT), облачных вычислений и других технологических трендов, рынок столкнулся с потребностью решения данных проблем, что способствовало росту популярности микросервисов.

Микросервисная архитектура - подход к разработке программного обеспечения, при котором приложение разбивается на небольшие, автономные сервисы, каждый из которых выполняет определенную функцию. Эти сервисы могут быть развернуты, масштабированы и обновлены независимо друг от друга.

Можно выделить ряд причин, по которым микросервисная архитектура остается актуальной:

* **Гибкость и масштабируемость:** микросервисы позволяют легко масштабировать и изменять отдельные компоненты приложения без необходимости внесения изменений в остальные. Это способствует гибкости и ускоряет процесс развертывания новых функций.
* **Управление сложностью:** разделение приложения на небольшие компоненты упрощает его понимание, разработку и обслуживание. Каждый микросервис имеет ограниченную область ответственности, что уменьшает сложность отладки и разработки.
* **Технологическая гетерогенность:** микросервисы позволяют использовать различные технологии и языки программирования для каждого сервиса в зависимости от его потребностей. Это позволяет командам разработки выбирать наиболее подходящие инструменты для решения конкретных задач.
* **Легкая заменяемость и масштабируемость:** микросервисы могут быть легко заменены или обновлены без влияния на другие части системы. Это упрощает поддержку и развертывание приложений в условиях высокой нагрузки.
* **Распределенная разработка:** микросервисы позволяют командам разработки работать независимо друг от друга, что способствует распределенной разработке и ускоряет процесс разработки.
* **Улучшенная отказоустойчивость:** при наличии микросервисов отказ одного из них не приведет к полному отказу всей системы. Это повышает отказоустойчивость и надежность системы в целом.

Такие гиганты рынка как **Netflix** и **Amazon** сыграли значительную роль в продвижении микросервисной архитектуры. Их опыт и практики стали примеров для многих компаний в области разработки и обслуживания микросервисов.

* 1. **Цели и задачи проекта**

Основной целью проекта является закрепление знаний, полученных в процессе обучения на платформе, и успешном их применении на практике в процессе написания кода. Более глубокое знакомство с микросервисной архитектурой и способами обеспечения ее безопасности.

Результатом данного дипломного проекта должен стать разработанный Backend-сервис для управления задачами с использованием Spring Framework и микросервисной архитектуры, который предоставляет следующие возможности:

* Регистрация пользователя
* Аутентификация пользователя
* Авторизация пользователя
* Создание задачи
* Редактирование задачи
* Удаление задачи
* Просмотра списка задач
* Получение информации о задаче
* Фильтрация и сортировка задач

Достижения поставленных целей и результата потребует использования таких инструментов как:

* Java Development Kit (JDK): необходим для компиляции и выполнения Java-кода.
* Spring Boot: позволяет создавать самостоятельные, готовые к запуску приложения с минимальной конфигурацией.
* Spring Framework: предоставляет широкий набор инструментов и функциональности для разработки Java-приложений.
* Spring Data JPA: упрощает доступ к данным в базе данных через Java Persistence API.
* Spring MVC: предоставляет архитектурный шаблон для построения веб-приложений.
* Spring Security: предоставляет мощные средства для обеспечения безопасности вашего приложения.
* Spring AOP: улучшение модульности кода путем выделения повторяющихся аспектов, таких как логирование, транзакции, аутентификация, и их отделения от основной логики приложения.
* Spring Integration: предоставляет интеграционные шаблоны и компоненты для разработки сложных интеграционных приложений. Упрощает интеграцию различных систем, сервисов и приложений через различные протоколы, сообщения и API.
* Maven: Это инструмент для управления зависимостями и сборки проекта.
* СУБД MySQL: хранение данных приложения.
* IntelliJ IDEA: среда разработки с удобной интеграцией Spring и Spring Boot.
* DataGrip: среда разработки для работы с базами данных.
* Docker: инструмент запуска изолированных контейнеров для различных приложений.
* Prometheus: система мониторинга и алертинга.
* Grafana: система визуализации данных и мониторинга.
  1. **Основные этапы разработки проекта.**

**Планирование и проектирование:**

* Определение требований к сервису управления задачами.
* Проектирование базы данных для хранения задач и связанных с ними данных.
* Определение структуры API для взаимодействия с клиентами.

**Настройка проекта Spring:**

* Создание нового проекта Spring Boot.
* Настройка зависимостей для подключения Spring Web, Spring Data JPA, Spring Security и других необходимых библиотек.

**Разработка моделей данных:**

* Создание сущностей для представления задач и других объектов в базе данных.
* Определение связей между сущностями.

**Разработка API контроллеров:**

* Создание контроллеров для обработки HTTP-запросов от клиентов.
* Определение эндпоинтов для создания, чтения, обновления и удаления задач.
* Реализация логики обработки запросов в контроллерах.

**Разработка сервисного слоя:**

* Создание сервисов для обработки бизнес-логики, связанной с управлением задачами.
* Выделение повторяющейся логики в сервисы для повышения модульности и повторного использования.

**Настройка доступа к данным:**

* Настройка репозиториев Spring Data JPA для взаимодействия с базой данных.
* Использование репозиториев для выполнения операций CRUD с объектами.

**Аутентификация и авторизация:**

* Настройка Spring Security для обеспечения безопасности сервиса.
* Реализация механизмов аутентификации и авторизации пользователей.

**Тестирование:**

* Создание юнит-тестов для проверки отдельных компонентов сервиса.
* Написание интеграционных тестов для проверки взаимодействия различных компонентов.
* Запуск автоматических тестов для обеспечения качества кода.

**Документирование:**

* Создание документации API с помощью Swagger.
* Написание комментариев к коду и пояснительной документации для разработчиков.

**Развертывание и мониторинг:**

* Развертывание приложения на сервере или в облачной среде.
* Настройка мониторинга для отслеживания производительности и доступности сервиса.

1. **Разработка проекта**
   1. **Планирование и проектирование**

Планирование и проектирование являются важными этапами в разработке любого продукта или проекта. Основными причинами, по которым его действительно стоит осуществлять, являются:

**Определение целей и требований:** планирование позволяет определить цели проекта и требования к нему. Это помогает всем участникам понять, что должно быть достигнуто в результате, и выработать стратегию для достижения этих целей.

**Предотвращение проблем и ошибок:** проектирование позволяет выявить потенциальные проблемы и ошибки на ранних этапах разработки, когда их исправление гораздо проще и дешевле, чем в более поздние фазы проекта. Это помогает снизить риски и улучшить качество конечного продукта.

**Оптимизация ресурсов:** планирование позволяет оптимизировать использование ресурсов, таких как время, бюджет и человеческие ресурсы. Это помогает избежать излишних затрат и неэффективного использования ресурсов.

**Обеспечение согласованности:** проектирование позволяет создать структурированный и согласованный план действий для всех участников проекта. Это помогает избежать путаницы и неопределенности, а также обеспечивает единое понимание целей и задач проекта.

**Улучшение коммуникации и сотрудничества:** Планирование и проектирование способствуют улучшению коммуникации и сотрудничества между участниками проекта. Это помогает им работать в едином направлении и синхронизировать свои усилия для достижения общих целей.

**Повышение эффективности и производительности:** хорошо спланированный и продуманный проект обычно более эффективен и производителен, чем тот, который разрабатывается на ходу. Планирование и проектирование помогают предвидеть и учесть различные аспекты проекта, что способствует его успешному завершению в срок и в рамках бюджета.

* + 1. **Требования к сервису**

Основные функциональные требования, предъявляемые к проекту:

Аутентификация и авторизация:

* Доступ к функциональности управления задачами должен быть защищен аутентификацией и авторизацией.
* Различным пользователям могут быть предоставлены различные уровни доступа в зависимости от их ролей и прав.

Создание задачи:

* Пользователь должен иметь возможность создавать новые задачи, указывая их название, описание, сроки выполнения и другие связанные данные.

Просмотр списка задач:

* Сервис должен предоставлять возможность просмотра списка всех задач.
* Пользователь должен иметь возможность просматривать задачи

Просмотр деталей задачи:

* Пользователь должен иметь возможность просмотра подробной информации о конкретной задаче, включая название, описание, статус, сроки выполнения и другие связанные данные.

Обновление информации о задаче:

* Пользователь должен иметь возможность обновлять информацию о задаче, такую как название, описание, статус, сроки выполнения и другие атрибуты.

Удаление задачи:

* Пользователь должен иметь возможность удалять задачи из системы, если они больше не нужны или были ошибочно созданы.

Фильтрация и поиск задач:

* Сервис должен предоставлять возможность фильтрации и поиска задач по различным критериям, таким как название, статус, сроки выполнения и другие атрибуты.

Управление статусом задачи:

* Пользователь должен иметь возможность изменять статус задачи, например, помечать задачу как "выполненную", "в процессе выполнения" или "отмененную".

Журналирование и мониторинг:

* Сервис должен вести журнал всех операций, выполняемых с задачами, для обеспечения прозрачности и отслеживания изменений.
* Может быть необходим мониторинг производительности и доступности сервиса для обеспечения надежной работы.

К нефункциональным требованиям стоит отнести:

Производительность:

* Сервис должен обеспечивать быструю и отзывчивую работу, чтобы пользователи могли эффективно управлять своими задачами.
* Максимальное время ответа API не должно превышать, например, 200 миллисекунд при нагрузке до 1000 запросов в секунду.

Масштабируемость:

* Сервис должен быть масштабируемым и способным обрабатывать увеличение числа пользователей и объема данных без существенного снижения производительности.
* Должна быть обеспечена горизонтальная масштабируемость, позволяющая добавлять новые экземпляры сервиса для распределения нагрузки.

Надежность:

* Сервис должен быть надежным и иметь высокую доступность, чтобы минимизировать время простоя и предотвратить потерю данных.
* Время непланового простоя системы должно быть менее 1 часа в год.

Безопасность:

* Сервис должен обеспечивать защиту данных и конфиденциальность информации пользователей.
* Должны применяться современные методы шифрования для защиты данных в покое и в движении.
* Доступ к административным функциям должен быть ограничен и контролируем.

Совместимость:

* Сервис должен быть совместим с различными браузерами и устройствами, чтобы пользователи могли получить к нему доступ с различных платформ.
* API сервиса должно быть совместимо с существующими инструментами и клиентскими приложениями.

Технологические требования:

* Сервис должен быть разработан с использованием современных технологий и стандартов разработки, чтобы обеспечить долгосрочную поддержку и расширяемость.
* Должны использоваться инструменты и практики DevOps для автоматизации развертывания, тестирования и мониторинга сервиса.

Удобство использования:

* Интерфейс пользователя должен быть интуитивно понятным и легко навигируемым, чтобы пользователи могли быстро освоить его и начать работу без дополнительного обучения.
  + 1. **Проектирование баз данных для микросервисов**

Сама суть микросервисной архитектуры состоит в том, что каждый отдельный сервис представляет собой приложение, которое выполняет одну единственную задачу, и содержит в себе все необходимое, включая экземпляр базы данных.

Таким образом для сервиса, управляющего пользователями, база данных будет состоять фактически из одной таблицы, которая будет отвечать за хранение нах пользователей. Набор минимальный набор атрибутов будет следующим:

* id (идентификатор пользователя, первичный ключ)
* username (имя пользователя)
* email (адрес электронной почты пользователя)
* password (пароль пользователя)
* active (состояние ученой записи пользователя)
* roles (роль пользователя в системе)

Следуя данной логике, для микросервиса, управляющего задачами, атрибуты для таблицы, в которой будут храниться задачи примут вид:

* id (идентификатор задачи, первичный ключ)
* title (заголовок задачи)
* description (описание задачи)
* status (статус задачи)
* ownerId (id владельца)
* created (дата создания)
* lastUpdate (дата последнего обновления)
* finished (дата завершения)

В связи с тем, что работа по созданию базы данных и работой с ней будет целиком и полностью возложена на Spring Data JPA, нам потребуется только описаться классы и задать необходимые аннотации в них.

* + 1. **Определение структуры API**

Определение структуры API играет ключевую роль в разработке программного обеспечения и взаимодействии между компонентами системы, способствует эффективной интеграции, облегчает разработку клиентских приложений и обеспечивает безопасность и надежность системы. Данный этап позволяет добиться следующего:

Ясного понимания возможностей сервиса:

* Определение структуры API позволяет разработчикам и клиентам четко понимать, какие операции могут быть выполнены с помощью API, какие данные могут быть переданы и какие результаты ожидаются.

Стандартизации взаимодействия:

* Хорошо определенная структура API позволяет стандартизировать взаимодействие между различными компонентами системы. Это упрощает интеграцию и обмен данными между различными сервисами и приложениями.

Облегчения разработки и поддержки клиентских приложений:

* Заранее определенная структура API упрощает процесс разработки клиентских приложений, так как разработчики знают, как обращаться к API и какие данные ожидать в ответе. Это также облегчает поддержку клиентских приложений при изменении API.

Безопасности:

* Определение структуры API позволяет реализовать механизмы аутентификации и авторизации, контролировать доступ к различным ресурсам и обеспечивать безопасность обмена данными между клиентом и сервером.

Облегчения процессов документирования и обучения:

* Хорошо определенная структура API облегчает процесс создания документации для API, что помогает пользователям понять, как использовать сервис, и обучиться работе с ним.

Улучшение масштабируемости:

* Хорошо спроектированная и структурированная API облегчает масштабирование системы, так как позволяет легко добавлять новые функции и изменять существующие без значительных изменений в клиентских приложениях.

Понимание важности данного этапа позволило на этапе проектирования приложения определить основу структуры API для микросервисов.

Для микросервиса управления пользователями:

**GET /api/users:**

* Описание: Получение списка всех пользователей.
* Ответ: Список пользователей в формате JSON.
* Код ответа: 200 OK

**GET /api/users/{id}:**

* Описание: Получение информации о пользователе по его идентификатору.
* Параметры пути: id - идентификатор пользователя.
* Ответ: Информация о пользователе в формате JSON.
* Код ответа: 200 OK (если пользователь найден), 404 Not Found (если пользователь не найден)

**GET /api/users/find/{username}:**

* Описание: Поиск пользователя по его имени пользователя.
* Параметры пути: username - имя пользователя.
* Ответ: Информация о пользователе в формате JSON.
* Код ответа: 200 OK (если пользователь найден), 404 Not Found (если пользователь не найден)

**POST /api/users:**

* Описание: Создание нового пользователя.
* Тело запроса: Данные нового пользователя в формате JSON.
* Ответ: Информация о созданном пользователе в формате JSON.
* Код ответа: 201 Created (если пользователь успешно создан), 409 Conflict (если пользователь уже существует)

**PUT /api/users/{id}:**

* Описание: Обновление информации о пользователе по его идентификатору.
* Параметры пути: id - идентификатор пользователя.
* Тело запроса: Данные пользователя для обновления в формате JSON.
* Ответ: Обновленная информация о пользователе в формате JSON.
* Код ответа: 200 OK (если пользователь успешно обновлен), 404 Not Found (если пользователь не найден)

**DELETE /api/users/{id}:**

* Описание: Удаление пользователя по его идентификатору.
* Параметры пути: id - идентификатор пользователя.
* Ответ: Пустое тело ответа.
* Код ответа: 204 No Content (если пользователь успешно удален), 404 Not Found (если пользователь не найден)

Для микросервиса управления задачами:

**GET /api/tasks/user/{id}:**

* Описание: Получение списка задач, принадлежащих определенному пользователю.
* Параметры пути: id - идентификатор пользователя.
* Ответ: Список задач пользователя в формате JSON.
* Код ответа: 200 OK

**GET /api/tasks:**

* Описание: Получение списка всех задач.
* Ответ: Список всех задач в формате JSON.
* Код ответа: 200 OK

**GET /api/tasks/{id}:**

* Описание: Получение информации о задаче по ее идентификатору.
* Параметры пути: id - идентификатор задачи.
* Ответ: Информация о задаче в формате JSON.
* Код ответа: 200 OK (если задача найдена), 404 Not Found (если задача не найдена)

**POST /api/tasks:**

* Описание: Создание новой задачи.
* Тело запроса: Данные новой задачи в формате JSON.
* Ответ: Информация о созданной задаче в формате JSON.
* Код ответа: 201 Created (если задача успешно создана), 400 Bad Request (если запрос некорректен)

**PUT /api/tasks/{id}:**

* Описание: Обновление информации о задаче по ее идентификатору.
* Параметры пути: id - идентификатор задачи.
* Тело запроса: Данные задачи для обновления в формате JSON.
* Ответ: Обновленная информация о задаче в формате JSON.
* Код ответа: 200 OK (если задача успешно обновлена), 404 Not Found (если задача не найдена)

**DELETE /api/tasks/{id}:**

* Описание: Удаление задачи по ее идентификатору.
* Параметры пути: id - идентификатор задачи.
* Ответ: Пустое тело ответа.
* Код ответа: 204 No Content (если задача успешно удалена), 404 Not Found (если задача не найдена)
  1. **Настройка проекта**
     1. **Создание проекта**
     2. **Подключение и настройка зависимостей**
  2. **Разработка моделей данных**
     1. **Создание сущностей для представления в базе данных**
     2. **Определение связей между сущностями**
  3. **Разработка API контроллеров**
     1. **Создание контроллеров для обработки HTTP-запросов от клиентов**
     2. **Определение эндпоинтов для создания, чтения, обновления и удаления задач.**
     3. **Реализация логики обработки запросов в контроллерах**
  4. **Разработка сервисного слоя**
     1. **Создание сервисов для обработки бизнес-логики, связанной с управлением задачами**
     2. **Выделение повторяющейся логики в сервисы для повышения модульности и повторного использования**
  5. **Настройка доступа к данным**
     1. **Настройка репозиториев Spring Data JPA для взаимодействия с базой данных.**
     2. **Использование репозиториев для выполнения операций CRUD с объектами.**
  6. **Аутентификация и авторизация**
     1. **Настройка Spring Security для обеспечения безопасности сервиса.**
     2. **Реализация механизмов аутентификации и авторизации пользователей.**
  7. **Тестирование**
     1. **Создание юнит-тестов для проверки отдельных компонентов сервиса.**
     2. **Написание интеграционных тестов для проверки взаимодействия различных компонентов**
     3. **Запуск автоматических тестов для обеспечения качества кода**
  8. **Документирование**
     1. **Создание документации API с помощью Swagger**
     2. **Написание комментариев к коду и пояснительной документации для разработчиков**
  9. **Развертывание и мониторинг**
     1. **Развертывание приложения на сервере или в облачной среде**
     2. **Настройка мониторинга для отслеживания производительности и доступности сервиса.**

# Заключение

В заключение необходимо включить следующее:

1. Краткие и ёмкие теоретические и практические выводы, которые были получены во время анализа теоретической базы и практического исследования.
2. Оценка проведённого исследования, описание его результатов.
3. Практическая значимость работы, рекомендации и планы на дальнейшие исследования.
4. Общий итог — достижение цели, выполнение задач, доказательство гипотезы.
5. Предложения по совершенствованию объекта исследования.

# Список используемой литературы

Здесь нужно будет указатьсписок используемой литературы, ссылки на все ресурсы, которые нужны были для создания проектной работы.

Основные правила оформления использованной литературы и ресурсов:

1. Каждый источник упоминается единожды, независимо от того, насколько часто на него ссылаются.
2. Список литературы оформляется в алфавитном порядке по фамилии автора, сначала русскоязычная литература, затем иностранная, далее интернет-сайты.
3. Библиографическая запись обязательно включает:
   * Фамилию автора или фамилии их группы, инициалы (при наличии).
   * Название статьи, книги, справочника, закона, иного документа.
   * Населённый пункт, в котором был издан источник, наименование издательства.
   * Год публикации.
   * Число страниц.

## Пример

* *Книга: Автор. Название книги. Город: Издательство, Год.*
* *Статья: Автор. "Заголовок статьи." Название журнала Том, номер (Год): страницы.*

# Приложения

В **приложения** обычно входят артефакты, получившиеся в процессе создания проекта:

1. Объёмные графики и таблицы, которые не помещаются на лист А4.
2. Длинные математические формулы и расчёты по ним.
3. Характеристики аппаратуры, которая использовалась для проведения исследования.
4. Авторские методики.
5. Вспомогательный материал: тесты, карточки, схемы, рисунки.
6. Материалы, полученные на предприятии: отчёты, прочие документы.