Санкт-Петербургский государственный университет

Направление: 02.03.02 «Фундаментальная информатика и информационные технологии

ООП: Программирование и информационные технологии

**ОТЧЕТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ**

**Тема задания:** Описание реализации серверной части приложения для тренировки ходьбы людей, страдающих заболеваниями нижних конечностей.

**Выполнил:** Панюшин Даниил Васильевич – студент группы 19Б12-пу бакалавриата.

**Руководитель научно-исследовательской работы:** Раевская Анастасия Павловна, доцент кафедры математической теории экономических решений Санкт-Петербургского Государственного Университета, кандидат физико-математических наук.

Санкт-Петербург

2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

Глава 1. Язык программирования 5

1.1 Golang 5

1.2 Python 6

1.3 Java 6

1.4 Выбор 8

Глава 2. Система управления базами данных 9

2.1 Классификация 9

2.2 Реляционные системы управления базами данных 9

2.3 Системы управления базами данных типа ключ-значение 11

2.4 Документоориентированные системы управления базами данных 12

2.5 Колоночные системы управления базами данных 13

2.6 Графовые системы управления базами данных 13

2.7 Выбор 14

Глава 3. Фреймворки 15

3.1 Потребность в фреймворках 15

3.2 Spring 15

3.3 Hibernate 17

3.4 Тесты 18

3.5 Системы сборки 20

3.6 Инструменты документирования application programming interface 22

4.1 Среда разработки 23

4.2 Графический интерфейс для работы с системой управления базами данных 24

4.3 Инструменты тестирования application programming interface 24

ЗАКЛЮЧЕНИЕ 25

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26

# ВВЕДЕНИЕ

Практически каждое приложение в современном мире использует сервера для различных целей. Бэкенд определяет логику хранения и обработки данных, важных для конкретного бизнеса, осуществляет рендер веб страниц и многое другое. Наш проект по созданию приложения для тренировки ходьбы людей, страдающих заболеваниями нижних конечностей не исключение. Фронтенд и мобильное приложение не могут обмениваться информацией сами по себе – удалённый сервер служит необходимой прослойкой для работы с данными, полученными от пользователей и визуализируемыми на страницах фронтенда.

Проведение исследовательской работы необходимо с целью определения и описания методов и технологий, которые могут быть использованы для реализации вышеупомянутых задач.

На текущий момент бекенд сервисы практически любого назначения могут быть реализованы на многих языках программирования, таких как *Java, Python, Golang, PHP, .NET* и т.д. Популярность различных языков проиллюстрирована на Рисунке 1.

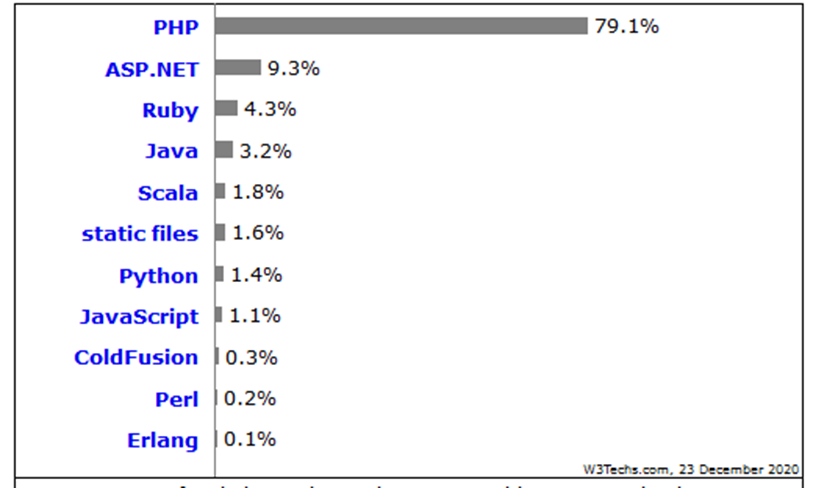


Рисунок 1 - Популярность различных языков программирования в сфере веб-разработки

Для каждого из вышеперечисленных языков программирования написаны фреймворки[[1]](#footnote-1), обладающие как достоинствами, так и недостатками, исходящими как от конкретной реализации, так и от особенностей языка.

Помимо языка программирования очень важно выбрать подходящую систему управления базами данных (или СУБД). Так как данные, хранимые, собираемые и обрабатываемые различными приложениями имеют разную структуру, то от выбора базы зависит скорость и эффективность работы конечного продукта. Подробнее о разновидностях СУБД будет рассказано позже.

Сводка самых популярных СУБД с точки зрения практического внедрения или использования предоставлена на Рисунке 2.

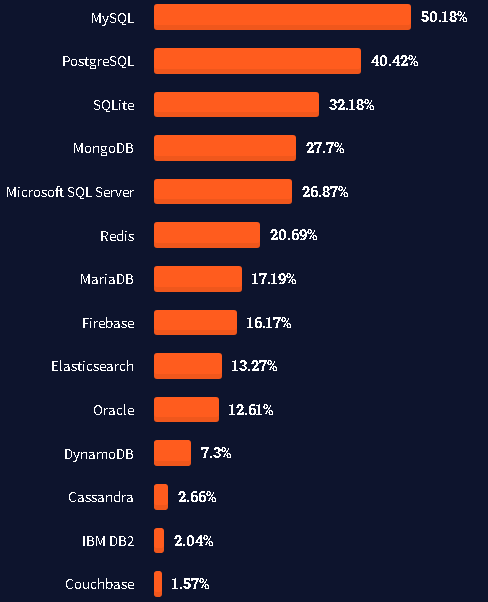


Рисунок 2 - Наиболее популярные СУБД

Так же будет рассказано об инструментах, используемых для облегчения тестирования и написания системы.

# Глава 1. Язык программирования

## Golang

*Golang* или просто *Go* - компилируемый многопоточный язык программирования, разработанный внутри компании Google. Go позволяет создавать высокоэффективные программы, исполняющиеся на современных распределённых системах и многоядерных процессорах. Этот язык обладает строгой статической типизацией, имеет достаточно простой синтаксис, основанный на *C*, но существенно доработанный, с большим количеством синтаксического сахара[[2]](#footnote-2).

Ключевые достоинства языка:

* Go компилируется в один небольшой двоичный файл с нулевыми зависимостями, что делает его очень быстрым и обеспечивает кроссплатформенное[[3]](#footnote-3) развертывание.
* Скорость работы – модель параллельной работы и масштабируемости использования ЦПУ[[4]](#footnote-4) позволяет Go получить хорошую скорость работы, особенно при параллельных вычислениях.
* Для веб-разработки в большинстве случаев не требуются сторонние библиотеки – язык включает собственную библиотеку веб шаблонизации.
* Строгая статическая типизация[[5]](#footnote-5) данного языка важна в крупных приложениях.
* Go позволяет быстро разрабатывать масштабируемые и безопасные веб-приложения.
* Поддержка - язык разработан компанией Google, имя которой говорит само за себя, так что поддержка и обновление языка будет происходить стабильно в ближайшие годы. К тому же, сообщество пользователей языка постоянно растёт, одной из причин чего является простота освоения.

## Python

Python – высокоуровневый интерпретируемый мультипарадигменный[[6]](#footnote-6) язык программирования общего назначения с открытым исходным кодом, обладающий динамической[[7]](#footnote-7)строгой типизацией и автоматическим управлением памятью.

Наиболее активно данный язык применяется в сферах машинного обучения, DevOps[[8]](#footnote-8) и веб-разработки.

Основные преимущества:

* Язык обладает простым минималистическим синтаксисом, что делает его простым в освоении.
* Огромное количество всевозможных библиотек, начиная с библиотек для машинного обучения и заканчивая библиотеками для написания веб или мобильных приложений.
* Python позволяет быстро писать рабочий код с необходимым функционалом, что особенно полезно при работе со строгими временными ограничениями.

Главным же недостатком языка Python является низкая скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках.

## Java

*Java* – строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle).

Особенность языка Java – использование Java-машины (или, сокращённо, JVM[[9]](#footnote-9)), которая транслирует java-код в специальный байт-код, что позволяет запускать приложение на любой компьютерной архитектуре, поддерживающая JVM.

Java используется для создания: мобильных приложений, веб приложений, видеоигр, приложений для работы с базами данных и прочее.

Основные достоинства:

* Кроссплатформенность – как было сказано выше, JVM позволяет писать приложения на Java для широкого спектра платформ.
* Простота синтаксиса по сравнению с C/C++.
* Безопасность. При создании платформы Java создатели проделали большую работу по её защите. Обойти или взломать механизмы крайне сложно. Наглядным примером механизма безопасности Java является использование классов, использующих цифровую подпись, а, следовательно, полные права предоставляются только при полном доверии автору класса.
* Производительность. Изначально она вызывала вопросы. Новые версии динамических компиляторов Java не уступают традиционным из других платформ. Мощный прирост скорости обработки даёт оптимизация тех фрагментов кода, которые исполняется чаще. При необходимости, те или иные приёмы оптимизации включаются или отменяются JIT-компилятором[[10]](#footnote-10).
* Надёжность. Программы на Java стабильно работают в любых условиях. Компилятор способен выявить ошибки ещё до выполнения кода, то есть на ранних стадиях. Контроль выполнения позволяет предотвратить сбои в памяти. Сами указатели можно применять не везде, а только там, где это необходимо.
* Динамичность и адаптируемость. При необходимости можно добавить в библиотеки новые объекты и методы. При этом нет необходимости изменять приложения, использующие данные библиотеки.
* Эффективные и удобные сетевые возможности. Имеется библиотека для передачи данных по самым распространённым протоколам. Существует механизм удалённого вызова методов.
* Поддержка. Язык пользуется популярностью на протяжении более двадцати лет и за это время вокруг него сформировалось значительное сообщество, способное дать ответ практически на любой вопрос касательно данного языка.

Недостатки Java:

* Платное коммерческое использование.
* Отсутствует нативный дизайн. Для создания интерфейса пользователя на компьютере для java-приложения нет ни одного java-инструмента, поэтому разработчики используют сторонние инструменты и библиотеки.
* Синтаксис Java всё ещё сложен. Это приводит к необходимости создания сложных и объёмных структур кода.
* Отсутствие поддержки корутин[[11]](#footnote-11).

## Выбор

Рассмотрев приведённые выше языки программирования автор данной научной работы сделал выбор в пользу языка Java для написания логики связи мобильного приложения или фронтенда с бекендом.

Причинами такого решения являются надёжность и безопасность, предоставляемая языком Java. Это необходимо для нашего проекта, так как имеет место обработка приватных данных клиентов.

Другой причиной такого решения стоит отметить поддержку данного языка, которая сильно облегчает работу с ним.

Но, пожалуй, главной причиной выбора Java является наличие крайне эффективного и популярного фреймворка Spring, созданного для создания как веб-приложений, так и многого другого. Подробнее об этой технологии будет рассказано позже.

# Глава 2. Система управления базами данных

## 2.1 Классификация

Существует два основных вида баз данных: SQL[[12]](#footnote-12) (так же известные как реляционные) базы и NoSQL. Вторые, в свою очередь делятся на следующие подтипы по принципу хранения данных: колоночные, ключ-значение, документные, графовые.

## 2.2 Реляционные системы управления базами данных

Реляционная база данных – набор данных с предопределёнными связями между ними. Данные организованы в виде множества таблиц, состоящих из столбцов и строк. Строка такой таблицы – набор связанных значений, характеризующих один объект или, по-другому, сущность. В каждом столбце таблицы могут храниться данные только одного типа. Строки таблиц помечены уникальным идентификатором, называемым первичным ключом. Строки нескольких таблиц могут быть связаны с помощью первичных ключей. Управление реляционными базами данных осуществляется с помощью специализированного языка SQL. При выполнении запросов к базе и получения доступа к данным не требуется реорганизовывать данные.

Классические реляционные базы данных чаще всего применяются для создания продуктов, использующих Online Transaction Processing - принцип организации базы данных, при котором система работает с большим потоком малообъёмных транзакций[[13]](#footnote-13). Клиенты таких систем должны получать ответы от системы с минимальным временем отклика. Так же, ввиду использования транзакций, являющихся атомарными[[14]](#footnote-14) операциями, должна существовать возможность откатить любые операции в рамках одной транзакции.

Реляционные СУБД чаще всего применяются при работе с большим количеством сущностей (представленных в базе таблицами), соединенных различными видами связей: один-к-одному, один-ко-многим, многие-ко-многим.

Наиболее популярными реляционными СУБД являются Oracle Database, Microsoft SQL Server, PostgreSQL, MySQL.

*Oracle Database* - СУБД, написанная компанией Oracle, преднзначенная для облачнростых сред и может быть размещена на одном или нескольких серверах. Написана на языках Assembly, C, C++.

Достоинства:

* Обрабатывает огромные данные.
* Очень высокая степень надёжности. Каждая транзакция изолирована от других.

Недостатки:

* Является платным продуктом, что может быть слишком дорогим решением для небольших компаний.
* Высокая требовательность к аппаратным ресурсам.

*Microsoft SQL Server* - СУБД, созданная компанией Microsoft, и изначально привязанная к операционной системе Windows. Впоследствии была адаптирована и для использования на Linux. Написана на языках C, C++.

Достоинства:

* Простота использования.
* Скорость.
* Стабильность.
* Наличие графического интерфейса.
* Синергия с другими продуктами Microsoft.

Недостатки:

* Является платным продуктом, что может быть слишком дорогим решением для небольших компаний.
* Способна задействовать все возможные ресурсы платформы, на которой запущена.

*PotgreSQL* - масштабируемая объектно-реляционная кроссплатформенная СУБД.

Написана на языке C.

Достоинства:

* Позволяет работать с огромными данными.
* Полная SQL совместимость.
* Поддержка json формата.
* Множество встроенных функций.
* Доступен ряд интерфейсов, в том числе и графических.
* Бесплатная. В том числе и для коммерческого использования.

Недостатки:

* Туманная документации.
* Сложная конфигурация.
* Низкая производительность при выполнении запросов чтения или пакетных запросов.
* Проблемы с поиском провайдера для хостинга из-за не высокой популярности и трудности настройки.

*MySQL* – СУБД с открытым исходным кодом. Написана на языках C, C++.

Достоинства:

* Простота использования.
* Широкий функционал.
* Отличная документация.
* Надёжность.
* Оптимальное использование аппаратных средств.
* Позволяет работать с огромными данными.
* Скорость работы. MySQL пренебрегает некоторыми стандартами СУБД, что позволяет получить прирост в производительности.
* Возможность взаимодействия с другими СУБД.
* Доступен ряд интерфейсов, в том числе и графических.

Недостатки:

* Некоторые операции реализованы менее недежно чем в других СУБД.
* Застой в разработке данной СУБД.
* Отсутствие встроенной поддержки XML[[15]](#footnote-15) и OLAP.
* Для бесплатной версии доступна только платная поддержка.

## 2.3 Системы управления базами данных типа ключ-значение

Базы данных основанные на принципе «ключ-значение» хранят данные как совокупность пар «ключ-значение». Ключ служит уникальным идентификатором. Ключи и значения могут представлять что-угодно: от элементарных до составных объектов. Принцип работы таких баз основан на использовании хеш-таблиц, в которых хранятся уникальные ключи и указатели на конкретные объекты данных.

Достоинства:

* Высокая разделяемость данных.
* Превосходное горизонтальное масштабирование, недостижимое базами других типов.

Недостатки:

* Плохая согласованность данных.
* Отсутствие стандартных возможностей баз данных (атомарность транзакций, согласованность данных при одновременном выполнении нескольких транзакций). Подобный функционал должен быть предоставлен приложением, использующим такую СУБД.
* При очень большом количестве данных могут возникать проблемы с уникальностью ключей.

Самыми популярными базами данных типа «ключ-значение» являются Riak и Dynamo от Amazon.

## 2.4 Документоориентированные системы управления базами данных

СУБД, специально предназначенные для хранения иерархических структур документов. В основе таких баз лежа т документные хранилища, имеющие древовидную структуру, начинающуюся с корневого узла и содержащую несколько внутренних и листовых узлов. Листовые узлы хранят данные, заносящиеся в индексы при добавлении документов. Это позволяет находить путь по искомых данных даже при сложной структуре дерева.

Подобные СУБД применяются в различных областях, как, например, документный поиск, издательское дело и прочее.

Достоинства:

* Возможность возвращать в выборке части большого количества документов без полной из загрузки в оперативную память. Документоориентированные базы хранят не только сами данные, но и, так называемые метаданные, хранящие информацию о структуре самого содержимого.
* Облегчённая система добавления полей в JSON документы из-за отсутствия жесткой схемы.

Недостатки:

* Отсутствие связей между данными.

Самыми популярными СУБД такого типа являются Apache CouchDB и MongoDB.

## 2.5 Колоночные системы управления базами данных

Основная идея колоночных СУБД в том, чтобы хранить данные не по строкам, как это делают традиционные базы (то есть реляционные, ведь именно они на протяжении являлись стандартом хранения информации), а по колонкам. Таким образом классические для SQL-клиента таблицы являются совокупностью колонок, каждая из которых по сути представляет собой таблицу из одного поля.

СУБД такого типа наиболее широка применяются в аналитических системах.

Достоинства:

* Высокая скорость выполнения операций выборки и агрегации, особенно если нет необходимости получать все атрибуты сущности.
* Возможность сильной компрессии данных. Данные в колонке чаще всего одного типа, чего нельзя сказать про данные в одной строке.

Недостатки:

* Медленное выполнение операций записи.
* Не подходят для транзакционных систем.
* Наличия сложностей для разработчика. Технология относительно новая и поэтому не так хороша развита, как, например, реляционные СУБД.

Самыми популярными СУБД колоночного типа являются BigTable от Google и Cassandra от HBase.

## 2.6 Графовые системы управления базами данных

Графовая СУБД использует графовые структуры для хранения и обработки информации. Другими словами, данные представлены в виде графов и их обобщений, что более естественно и основанно на той же логике, с которой мы сталкиваемся с этими данными в реальной жизни. Такой подход позволяет выполнять семантические запросы к графовой базе.

Достоинства:

* Универсальность. В графовых СУБД можно хранить и реляционные, и документарные и сложные семантические данные.
* Модель построения базы может меняться в процессе развития приложения без изменения архитектуры структуры запросов.

Недостатки:

* Низкая производительность при незначительном количестве связей и больших объёмах данных.
* На данный момент практически нет графовых СУБД, хорошо работающих в приложениях с параллельной архитектурой.

Самыми популярными графовыми СУБД являются Neo4j и Microsoft Azure Cosmos DB.

## 2.7 Выбор

Стоит ещё раз упомянуть, что выбор СУБД в первую очередь зависит от специфики приложения и тех данных, с которыми ему придётся работать.

Рассмотрев все типы баз данных из приведённой выше классификации, был сделан выбор в пользу документоориентированной СУБД. Это решение обусловлено тем, что наше приложение будет собирать статистику по большому количеству пользователей, для чего должна существовать возможность хорошего горизонтального масштабирования так как объём данных значительно увеличиваться со временем. Статистические данные, полученные от каждого пользователя не имеют большого числа связей между собой ввиду своей специфики. Также, стоит упомянуть, что при использовании различного аппаратного обеспечения, а именно различных трекеров активности[[16]](#footnote-16) пользователей, серверной стороной приложения будут приниматься данные различной семантики, для чего документоориентированные СУБД отлично подходят ввиду отсутствия жесткой схемы. Ещё одной причиной выбора в пользу вышеупомянутой базы данных является ускоренная обработка выборок по частям документов без их полной загрузки в оперативную память. Действительно, статистика, отображаемая пользователю, обрабатывается по частям, например, график дневного сердечного ритма может быть отрисован отдельно от графика числа шагов на интервал времени.

# Глава 3. Фреймворки

## 3.1 Потребность в фреймворках

Использование фреймворков, является неотъемлемой частью современной разработки практически любого приложения. Данное программное обеспечение не только облегчает разработку и объединение воедино элементов большого проекта, но и помогает унифицировать процесс написания большой части кода, что позволяет самым разным людям достаточно просто дорабатывать один и тот же продукт.

Фреймворк, являясь чем-то большим чем библиотека, может включать в себя:

* вспомогательные программы;
* библиотеки кода;
* язык сценариев;
* другое программное обеспечение, облегчающее разработку и интеграцию различных компонентов проекта.

Так как для нашего проекта языком программирования был выбран Java, то рассмотрим наиболее популярные фреймворки именно для этого языка.

## 3.2 Spring

Spring, разработанный организацией Apache, является одной из самых широко используемых платформ Java. Этот фреймворк представляет собой композицию из нескольких программных проектов, таких как Spring Boot, Spring MVC, Spring Data, Spring Security и др.

Все эти элементы данной технологии строятся вокруг Spring Core. Оно реализует принцип Inversion of Control / Dependency Injection, или, переводя на русский язык, инверсия управления и внедрение зависимостей. Чтобы объяснить этот принцип нужно понять, что такое зависимость. Допустим у нас имеется два класса, один из которых содержит своим атрибутом экземпляр второго класса. В таком случае можно сказать, что первый класс зависит от второго класса. Определяя ссылку из первого класса на экземпляр второго класса, мы внедряем зависимость. Реализация инверсии управления подразумевает что классы, написанные в приложении, не определяют свои зависимости внутри своего же кода – они получают их извне. В приведённом выше примере это может быть достигнуто если в конструктор (или метод-сеттер) первого класса помимо остальных параметров будет передаваться так же и экземпляр второго класса. В Spring Core это реализовано с помощью контейнера внедрения зависимостей – при написании класса он добавляется в так называемый контекст приложения, откуда может быть подан экземпляр этого класса в качестве зависимости в один или несколько других классов, так же объявленных в контексте. При этом, с точки зрения кода, внедрение зависимости может происходить как автоматически (например, с помощью аннотаций[[17]](#footnote-17)), так и через непосредственное обращение к менеджеру контекста.

Далее стоит поговорить про Spring Boot, с помощью которого реализуется REST[[18]](#footnote-18) архитектура нашего приложения. Данный элемент фреймворка Spring обладает следующими преимуществами:

* Скорость и простота разработки.
* Автоматическая конфигурация компонентов Spring приложений.
* Встроенные серверы развёртывания: Tomcat, Jetty, Undertow.
* Отсутствие XML конфигурации.
* Облегчённая работа с различными СУБД.
* Отличная интеграция с экосистемой Spring.
* Большое сообщество, способное ответить на различные вопросы, касающиеся написания и поддержания кода.

Недостатков Spring Boot тоже не лишён:

* Большой размер файлов развёртывания.
* Не подходит для масштабных монолитных проектов.
* Трудоёмкость преобразования проекта на чистом Spring в проект Spring Boot.

Отдельного внимания заслуживает Spring Security – мощный инструмент создания гибкого аппарата аутентификации[[19]](#footnote-19) и авторизации[[20]](#footnote-20) приложения. Является де-факто стандартом организации систем безопасности Spring проектов. Как и все остальные элементы Spring, настоящая мощь Spring Security проявляется в простоте его расширения для удовлетворения пользовательских потребностей.

Достоинства:

* Встроенные методы шифрования паролей.
* Аутентификация In-Memory. Полезно на стадии разработки и тестирования, так как позволяет работать с данными, временно хранящимися лишь в оперативной памяти.
* LDAP-аутентификация. Lightweight Directory Access Protocol – протокол аутентификации пользователей в организациях, позволяющий определять структуру пользователей и назначать им права.
* Управление сессиями. Механизмы создания и контроля при входе в систему и уничтожения при выходе сеансов пользователя.
* Механизм распознавания, позволяющий пользователям не вводить учётные данные при каждом посещении сервиса.
* OAuth 2.0. Open Authorization 2.0 – стандарт проверки прав пользователей с помощью сервиса авторизации. Одним из его возможностей является возможность аутентификации через учётные записи Google, Facebook и прочее.

В нашем приложении использована именно экосистема Spring, так как на данный момент она является наиболее развитой и популярной, а значит с её помощью можно будет быстро и надёжно реализовать функционал практически любой сложности.

## 3.3 Hibernate

Для подключения к различным базам данных в Javaпридумали стандарт JDBC[[21]](#footnote-21), который позволяет работатть по единым правилам с различными базами данных. Для этого требуется лишь установка драйвера определённой базы по стандарту JDBC. Такой подход не является самым оптимальным, так как при увеличении масштаба приложения появляется много однообразного и запутанного кода, что усложняет и, соответственно, замедляет работу над продуктом.

Hibernate – библиотека для языка программирования Java, изначально предназначенная для решения задач объектно-реляционного отображения, то есть для работы с реляционными СУБД. Является самой популярной реализацией Java Persistence API[[22]](#footnote-22)[[23]](#footnote-23). В последствии Hibernate получил обновление, в котором появилась возможность работать с NoSQL базами данных, для чего был добавлен OGM – движок для работы с нереляционными СУБД, аналогичный по функционалу ORM[[24]](#footnote-24).

ORM и OGM предоставляют удобный интерфейс, представляющий сущности из базы данный в виде классов, а объекты сущностей в виде экземпляров классов. Такое представление позволяет автоматически строить запросы и извлекать данные, что значительно ускоряет процесс разработки, ввиду отсутствия необходимости ручного написания SQL или JDBC кода.

В нашем проекте использована именно технология Hibernate OGM ввиду того, что MongoDB это документоориентированная база данных.

## 3.4 Тесты

Тестирование – проверка соответствия реальных и ожидаемых результатов работы программы, проводимая на ограниченном наборе тестов. Тестирование является неотъемлемый этапом создания любого программного обеспечения. Он настолько важен, что появились особые техники разработки, основывающиеся на тестах, например, TDD[[25]](#footnote-25).

Тестирование бывает:

* Модульное. Тесты для проверки модулей системы по отдельности.
* Системное. Тесты высокого уровня для проверки работы крупного куска приложения.
* Регрессионные. Проверка влияния новых элементов системы на ранее написанные.
* Функциональное. Тестирование соответствия приложения требованиям, заявленных в спецификациях, документации и пр. Делятся в свою очередь на:
  + Тесты «белого ящика». Проверка соответствия приложения требованиям, учитывая внутреннее устройство системы.
  + Тесты «черного ящика». Проверка соответствия приложения требованиям, без учёта внутреннего устройства системы.
* Тестирование производительности. Определение скорости работы приложения под определённой нагрузкой.
* Нагрузочное тестирование. Проверка системы на устойчивость при различных нагрузках. Так же позволяют находить пик производительности корректной работы приложения.
* Стресс-тестирование. Проверка работоспособности системы при нештатных нагрузках. Позволяют определить пик производительности системы, при котором она продолжает функционировать.
* Тестирование локализации приложения.
* Юзабилити тестирование. Поверка удобства использования системы пользователями.

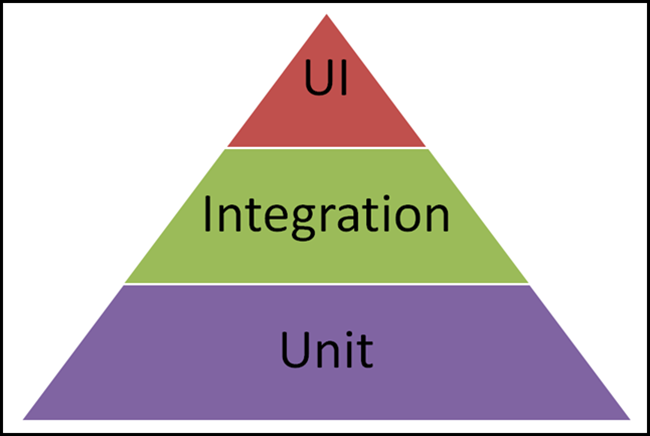
На практике существует так называемая пирамида тестирования Майка Кона, в соответствии с упрощённой версией которой, представленной на Рисунке 3, тестирование делится на три части:

Рисунок 3 - Упрощённая пирамида тестирования Майка Кона

* Unit тестирование – модульное тестирование, применяемое в различных слоях приложения и тестирующее наименьшую делимую логику проекта. Такие тесты максимально изолированы от внешней логики, то есть имитируют стандартное поведение остальной части программы.
* Интеграционное тестирование – проверяет крупные элементы системы. Так же интеграционные тесты применяются для проверки работы с внешним элементом системы. Ввиду своей тяжеловесности их количество меньше чем unit-тестов.
* UI-тестирование – проверка работы пользовательского интерфейса. Данные тесты затрагивают логику на всех уровнях приложения. Являются самими тяжеловесными и из-за этого проверяют в основном лишь ключевые элементы системы.

Данный подзаголовок имеет такое название ввиду того, что для тестирования Spring приложений используется не один фреймворк, а сразу два и более: Junit, Mockito, AssertJ и др. Стоит отметить удобство использования фреймворка Spring и на этом этапе разработки – Spring Boot Starter Test включает в себя и управляет версиями всех вышеперечисленных библиотек. Рассмотрим подробнее некоторые из этих библиотек:

* Junit – фреймворк модульного тестирования для языка Java. Содержит аннотации, используемые для определения функций, выполняемых перед тестами, во время тестов и после них. Так же имеет assertions – так называемые утверждения, используемые для сравнения ожидаемого результата с полученным.
* Mockito – фреймворк для работы с заглушками. При тестировании кода тестируемому коду часто необходимо передавать экземпляры классов, которыми он должен пользоваться в процессе работы. Поведение таких объектов должно быть полностью предсказуемым. Данный фреймворк позволяет создавать такие объекты, называемые моками, буквально в одну строчку кода. Для моков можно переопределить поведение требуемым образом с необходимой степенью детальности. Например, можно установить методу некоторого мок-объекта возвращать целое число 10 при получении на вход строки «Hello» (естественно поддерживается работа с произвольными структурами данных).
* AssertJ – библиотека для написания более плавных и лаконичных assertions. Так же это программное обеспечение позволяет генерировать утверждения, основанные на структуре конкретных классов.

## 3.5 Системы сборки

Системы сборки необходимы при Java-разработке так как они используются на всех стадиях работы: от создания структуры проекта и подключения библиотек, до развертывания продукта на сервере.

Две основные самые популярные системы сборки проектов:

* Gradle. Система сборки с открытым исходным кодом и использующая специфический язык, основанный на языке Groovy. Данная система определяет порядок выполнения задач с помощью ациклического графа. Используется Google в качестве системы сборки Android проектов.
* Maven. Система сборки, использующая XML-файлы для описания проекта, его зависимостей, необходимых плагинов и порядка сборки. Для задач упаковки и компиляции существуют заранее определённые цели. Данная система используется в основном для языка Java, но поддерживает также Scala, Ruby, C# и пр.

Существуют фундаментальное отличие вышеперечисленных систем. Производительность обоих программ позволяет параллельно выполнять многомодульные сборки. Тем не менее Gradle позволяет выполнять инкрементные сборки, благодаря проверки обновления задач. Если задача обновлена, то она не выполняется, что даёт гораздо меньшее время сборки. Так же Gradle позволяет: инкрементно компилировать классы Java, избегать компиляции для Java, использовать своё API для инкрементальных подзадач, использовать демон[[26]](#footnote-26) компилятора, значительно ускоряющий компиляцию.

Достоинства Gradle не заканчиваются на производительности:

* Он имеет возможность безопасного кеширования. Gradle хранит метаданные репозитория вместе с кэшированными зависимостями, обеспечивая то, что несколько проектов не перезапишут друг друга. Наличие кеша на основе контрольной суммы позволяет синхронизировать кеш с репозиториев.
* Gradle позволяет определять пользовательские правила указания версии для динамической зависимости и разрешать конфликты версий.
* Облегчённая работа с составными сборками, позволяя работать со специальными и постоянными составными сборками, а также объединять различные сборки и импортировать составную сборку в Eclipse[[27]](#footnote-27) или IntelliJ IDEA[[28]](#footnote-28).
* Gradle имеет полностью настраиваемую модель выполнения. Создание произвольного дерева задач предлагает расширенные возможности для упорядочивания задач и финализаторов.
* Возможность использования пользовательских дистрибутивов.
* Gradle также позволяет настраивать среды сборки на основе версий без необходимости настраивать их вручную.

Ввиду приведённых выше достоинств системы сборки Gradle было решено использовать именно его.

## 3.6 Инструменты документирования application programming interface

Польза от недокументированного программного обеспечения весьма сомнительна, ведь даже разработчик продукта спустя время может взглянуть на свой же код и задаться вопросом как он работает. Особенно остро вопрос документации стоит при проектировании API, так как без знания структуры дерева URL-адресов и понимания способа формирования компонент запроса невозможно работать с данным программным интерфейсом. Документирование API позволяет клиентам понимать способ использования сервиса.

Для создания документации используется Swagger и спецификация OpenAPI. Swagger – инструмент для автоматической генерации документации API в виде json. Библиотека Springdoc, используемая совместно со Swagger, предоставляет удобный графический интерфейс для как визуализации сгенерированной документации, так и для создания различных запросов к сервису. Так же есть функционал для создания клиента или сервера по спецификации API Swagger, для чего используется Swagger-Codegen. Swagger использует аннотации для генерации документации.

Спецификация OpenAPI (OAS) определяет стандартный, не зависящий от языка интерфейс для RESTful API, который позволяет людям и компьютерам понимать возможности сервиса без доступа к его исходному коду. При правильном использовании потребитель может понимать и взаимодействовать с удаленной службой при минимальном количестве логики реализации.

Глава 4. Средства разработки

Правильный выбор средств разработки способен значительно облегчить работу любого разработчика. Мощное инструментальное программное обеспечение обладают широким спектром возможностей, необходимых как во время написания кода, так и в момент его тестирования. Именно средства разработки являются тем, что видит перед собой большинство программистов большую часть времени создания проекта.

## 4.1 Среда разработки

Данное программное обеспечения включает в себя редактор текста, компилятор или интерпретатор (для интерпретируемых языков), средства автоматизации сборки и отладчик.

Существует множество инструментов, не специализирующихся на каком-то конкретном языке программирования, например, VSCode. Такие средства удобны в проектах, содержащих код на различных языках. В нашем приложении с серверной стороны использован один язык – Java. Ввиду этого целесообразно использовать инструменты, специализирующиеся именно на этом языке. Рассмотрим две самых популярных среды разработки на языке Java: IntelliJ IDEA и Eclipse:

* IntelliJ IDEA предлагает поддержку ряда языков, включая Java, Kotlin, Groovy, Clojure, Scala и других. Включает в себя такие функции, как расширенное прогнозирование, анализ кода и интеллектуальное завершение кода, а также набор подключаемых модулей и расширений для настройки IDE[[29]](#footnote-29) в соответствии с вашими потребностями, если вы не удовлетворены ее готовыми функциями. набор.
* Универсальное и гибкое решение Eclipse было разработано для удовлетворения потребностей сложных корпоративных проектов и разработки приложений для встроенных систем. Помимо Java, он предлагает поддержку различных языков программирования, а также популярных фреймворков. Благодаря множеству подключаемых модулей, подробной документации и большому сообществу разработчиков Eclipse по праву завоевала множество преданных поклонников.

Сравнение IntelliJ IDEA и Eclipse:

* IntelliJ IDEA бесплатна только в версии Community, обладающей ограниченным функционалом, по сравнению с Ultimate версией. Eclipse полностью бесплатна.
* Eclipse более оптимально работает с оперативной памятью.
* IntelliJ IDEA обладает большим количеством плагинов чем Eclipse.
* Бо́льшие возможности кастомизации интерфейса в IntelliJ IDEA.

Ввиду приведённого выше сравнения было решено использовать IntelliJ IDEA. Стоит отметить, что выбор между двумя этими инструментами во многом зависит от предпочтений самого разработчика.

## 4.2 Графический интерфейс для работы с системой управления базами данных

Так как в качестве базы данных для нашего проекта была выбрана MongoDB, то стоит рассмотреть официальное программное обеспечение, разработанное создателями выбранной СУБД, а именно MongoDB Compass. Этот инструмент позволяет просматривать базы данных, коллекции и отдельные документы, интерактивно создавать запросы, управлять существующими документами и создавать конвейеры агрегации. При работе над проектом данное программное обеспечение используется для отладки работы с базой данных и заполнением её тестовой информацией.

## 4.3 Инструменты тестирования application programming interface

При разработке API необходимо иметь возможность отправлять HTTP запросы на разрабатываемый сервер. Это можно делать программно. При таком подходе приходится писать новый код для выполнения каждого запроса, что очень долго при необходимости отправления большого количества запросов различного содержания на различные адреса, обрабатываемые приложением. Но существует и другой подход – использование графических инструментов, позволяющих отправлять запросы и обрабатывать ответы от серверов.

Один из таких инструментов мы и используем - Postman. Подробнее о его преимуществах:

* Бесплатный
* Интуитивно понятный и простой в использовании
* Поддерживает разные виды API (REST, SOAP, GraphQL)
* Простая интеграция
* Поддержка ручного и автоматического тестирования
* Позволяет сохранять созданные запросы
* Позволяет гибко настраивать параметры запросов

# ЗАКЛЮЧЕНИЕ

Данная работа описывает различные инструменты, используемые для реализации как задуманного приложения, так и приложений, схожих с ним. Не стоит настаивать на использовании именно вышеприведённого стека технологий для написания подобных продуктов, ибо выбор зависит от специфики проекта и, во многом, от способностей и знаний разработчиков. Именно поэтому другие программисты могут как соглашаться с нашим выбором инструментарного программного обеспечения, так и выступать против него.

Так как данный проект является во многом обучающим, то в случае ошибочного выбора используемых технологий будут сделаны соответствующие выводы и переписана неудачная часть приложения уже с учётом выявленных проблем.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальная документация Java – URL: https://www.oracle.com/java/
2. Официальная документация Golang – URL: https://go.dev
3. Официальная документация Python – URL: https://www.python.org
4. Описание различных СУБД опытными разработчиками – URL: https://habr.com/ru/post/579248/
5. Официальная документация MongoDB – URL: https://www.mongodb.com/docs/
6. Официальная документация Oracle Database – URL: https://docs.oracle.com/en/database/oracle/oracle-database/index.html
7. Официальная документация MySQL – URL: https://dev.mysql.com/doc/
8. Официальная документация PostgreSQL – URL: https://www.postgresql.org/docs/
9. Официальная документация Microsoft SQL Server – URL: https://docs.microsoft.com/ru-ru/sql/sql-server/?view=sql-server-ver15
10. Официальная документация Spring – URL: https://spring.io/projects
11. Официальная документация Hibernate – URL: https://hibernate.org/orm/documentation/6.0/
12. Официальная документация Gradle – URL: https://docs.gradle.org/current/userguide/userguide.html
13. Официальная документация Maven – URL: https://maven.apache.org/guides/index.html
14. Официальная документация Swagger – URL: https://swagger.io/docs/
15. Официальная документация IntelliJ IDEA – URL: https://www.jetbrains.com/help/idea/getting-started.html
16. Официальная документация Eclipse – URL: https://www.eclipse.org/documentation/

1. Программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта. [↑](#footnote-ref-1)
2. Синтаксические возможности, применение которых не влияет на поведение программы, но делает использование языка более удобным для человека. [↑](#footnote-ref-2)
3. Способность программного обеспечения работать с несколькими аппаратными платформами или операционными системами. [↑](#footnote-ref-3)
4. Центральное процессорное устройство. [↑](#footnote-ref-4)
5. Типизация, при которой конечные типы переменных и функций устанавливаются на этапе компиляции. [↑](#footnote-ref-5)
6. Язык программирования, поддерживающий несколько парадигм программирования, например, объектно-ориентированное программирование или процедурное. [↑](#footnote-ref-6)
7. Типизация, при которой все типы выясняются уже во время выполнения программы. [↑](#footnote-ref-7)
8. Методология автоматизации технологических процессов сборки, настройки и развёртывания программного обеспечения. [↑](#footnote-ref-8)
9. Java Virtual Machine [↑](#footnote-ref-9)
10. JIT-компиляция - технология увеличения производительности программных систем, использующих байт-код, путём компиляции байт-кода в машинный код или в другой формат непосредственно во время работы программы. [↑](#footnote-ref-10)
11. Облегченный поток исполнения кода, который может работать поверх основных потоков программы. [↑](#footnote-ref-11)
12. Structured Query Language - язык запросов, применяющийся для управления реляционными базами данных. [↑](#footnote-ref-12)
13. Группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными. [↑](#footnote-ref-13)
14. Единый, неделимый [↑](#footnote-ref-14)
15. eXtensible Markup Language - расширяемый язык разметки. [↑](#footnote-ref-15)
16. Это могут быть различные фитнес-браслеты или специально медицинское оборудование, сохраняющее статистику на смартфон пациента. [↑](#footnote-ref-16)
17. В языке Java специальная форма синтаксических метаданных, которая может быть добавлена в исходный код. [↑](#footnote-ref-17)
18. Representational State Transfer – передача состояния представления. Это архитектурный стиль взаимодействия компонентов распределённой системы в компьютерной сети. [↑](#footnote-ref-18)
19. Процедура проверки подлинности. [↑](#footnote-ref-19)
20. Процедура предоставления определенному лицу или группе лиц прав на выполнение определенных действий. [↑](#footnote-ref-20)
21. Java DataBase Connectivity - платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД. [↑](#footnote-ref-21)
22. API – Application Programmer Interface – программный интерфейс приложения. [↑](#footnote-ref-22)
23. Java Persistence API (JPA) - спецификация API Java, предоставляющая возможность сохранять в удобном виде Java-объекты в базе данных. [↑](#footnote-ref-23)
24. Object Relation Mapping - технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». [↑](#footnote-ref-24)
25. Test-driven development - техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам. [↑](#footnote-ref-25)
26. Демон – компьютерная программа, запускаемая системой и работающая в фоновом режиме без прямого взаимодействия с пользователем. [↑](#footnote-ref-26)
27. Eclipse – свободная интегрированная среда разработки модульных и кроссплатформенных приложений. [↑](#footnote-ref-27)
28. IntelliJ IDEA – интегрированная среда разработки на многих языках. [↑](#footnote-ref-28)
29. IDE (Integrated development environment) – интегрированная среда разработки. [↑](#footnote-ref-29)