**II Глава**

**2.1. СУБД PostgreSQL**

PostgreSQL — это объектно-реляционная система управления базами данных (ОРСУБД, ORDBMS), основанная на POSTGRES, Version 4.2 — программе, разработанной на факультете компьютерных наук Калифорнийского университета в Беркли. В POSTGRES появилось множество новшеств, которые были реализованы в некоторых коммерческих СУБД гораздо позднее.

PostgreSQL — СУБД с открытым исходным кодом, основой которого был код, написанный в Беркли. Она поддерживает большую часть стандарта SQL и предлагает множество современных функций:

* сложные запросы
* внешние ключи
* триггеры
* изменяемые представления
* транзакционная целостность
* многоверсионность

Кроме того, пользователи могут всячески расширять возможности PostgreSQL, например создавая свои

* типы данных
* функции
* операторы
* агрегатные функции
* методы индексирования
* процедурные языки

А благодаря свободной лицензии, PostgreSQL разрешается бесплатно использовать, изменять и распространять всем и для любых целей — личных, коммерческих или учебных.

PostgreSQL не просто реляционная, а объектно-реляционная СУБД. Это даёт ему некоторые преимущества над другими SQL базами данных с открытым исходным кодом, такими как MySQL, MariaDB и Firebird.

Фундаментальная характеристика объектно-реляционной базы данных — это поддержка пользовательских объектов и их поведения, включая типы данных, функции, операции, домены и индексы. Это делает PostgreSQL невероятно гибким и надежным. Среди прочего, он умеет создавать, хранить и извлекать сложные структуры данных.

**2.1.1. Отличительные характеристики PostgreSQL**

Существует обширный список типов данных, которые поддерживает Постгрес. Кроме числовых, с плавающей точкой, текстовых, булевых и других ожидаемых типов данных (а также множества их вариаций), PostgreSQL так же имеет поддержку uuid, денежного, перечисляемого, геометрического, бинарного типов, сетевых адресов, битовых строк, текстового поиска, xml, json, массивов, композитных типов и диапазонов, а также некоторых внутренних типов для идентификации объектов и местоположения логов. MySQL, MariaDB и Firebird тоже имеют некоторые из этих типов данных, но только PostgreSQL поддерживает их все.

*PostgreSQL имеет поддержку таких типов и структур данных как:*

* *Сетевые адреса*

PostgreSQL обеспечивает хранение разных типов сетевых адресов. Тип данных CIDR следует соглашению для сетевых адресов IPv4 и IPv6. Вот несколько примеров:

* 192.168.100.128/25
* 10.1.2.3/32
* 2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128
* ::ffff:1.2.3.0/128

Также для хранения сетевых адресов доступен тип данных INET, используемый для IPv4 и IPv6 хостов, где подсети являются необязательными. Тип данных MACADDR может использоваться для хранения MAC-адресов для идентификации оборудования, таких как 08-00-2b-01-02-03.

У MySQL и MariaDB тоже есть INET функции для конвертации сетевых адресов, но они не предоставляют типы данных для внутреннего хранения сетевых адресов. У Firebird тоже нет типов для хранения сетевых адресов.

* *Многомерные массивы*

Поскольку PostgreSQL — это объектно-реляционная база данных, массивы значений могут храниться для большинства существующих типов данных. Сделать это можно путём добавления квадратных скобок к спецификации типа данных для столбца или с помощью выражения ARRAY. Размер массива может быть задан, но это необязательно.

MySQL, MariaDB, и Firebird так не умеют. Чтобы хранить такие массивы значений в традиционных реляционных базах данных, придется использовать обходной путь и создавать отдельную таблицу со строками для каждого из значений массива.

* *Геометрические данные*

Геоданные быстро становятся основным требованием для многих приложений. PostgreSQL уже давно поддерживает множество геометрических типов данных, таких как точки, линии, круги и многоугольники. Один из этих типов – PATH, он состоит из множества последовательно расположенных точек и может быть открытым (начальная и конечная точки не связаны) или закрытым (начальная и конечная точки связаны).

* *Поддержка JSON*

Поддержка JSON в PostgreSQL позволяет перейти к хранению schema-less данных в SQL базе данных. Это может быть полезно, когда структура данных требует определённой гибкости: например, если в процессе разработки структура всё ещё меняется или неизвестно, какие поля будет содержать объект данных.

Тип данных JSON обеспечивает проверку корректности JSON, который позволяет использовать специализированные JSON операторы и функции, встроенные в PostgreSQL для выполнения запросов и манипулирования данными. Также доступен тип JSONB — двоичная разновидность формата JSON, у которой пробелы удаляются, сортировка объектов не сохраняется, вместо этого они хранятся наиболее оптимальным образом, и сохраняется только последнее значение для ключей-дубликатов. JSONB обычно является предпочтительным форматом, поскольку требует меньше места для объектов, может быть проиндексирован и обрабатывается быстрее, так как не требует повторного синтаксического анализа.

В MySQL 5.7.8 и MariaDB 10.0.1 была добавлена поддержка встроенных объектов JSON. Но, хотя существует множество функций и операторов для JSON, которые теперь доступны в этих базах данных, они не индексируются так, как JSONB в PostgreSQL. Firebird пока что не присоединился к тренду и поддерживает объекты JSON только в виде текста.

* *Создание нового типа*

Если вдруг так случится, что обширного списка типов данных PostgreSQL вам окажется недостаточно, вы можете использовать команду CREATE TYPE, чтобы создать новые типы данных, такие как составной, перечисляемый, диапазон и базовый.

Поскольку они не являются объектно-реляционными, MySQL, MariaDB и Firebird не предоставляют такую мощную функциональность.

**2.1.2. Обоснование выбора PostgreSQL.**

У PostgreSQL множество возможностей. Созданный с использованием объектно-реляционной модели, он поддерживает сложные структуры и широкий спектр встроенных и определяемых пользователем типов данных. Он обеспечивает расширенную ёмкость данных и заслужил доверие бережным отношением к целостности данных.

**2.2. ORM технология. Hibernate и JPA.**

Поскольку без поддержки баз данных не обходится практически ни одно современное приложение, то вопросам взаимодействия с ними уделяется пристальное внимание во всех учебниках по программированию. К сожалению, о такой полезной вещи, как ORM, в учебниках описывают редко.

**2.2.1. ORM технология**

Большая часть языков программирования, которые используются сегодня в индустрии программирования, относятся к объектно-ориентированным языкам. Это значит, что программист, создавая программы на них, оперирует с объектами - некоторыми абстрактными сущностями, имеющими некоторые свойства и позволяющими применять по отношению к себе некоторые методы. При всём при этом базы данных, которые используются для хранения информации об этих объектах, являются реляционными.

При создании приложения на Java и работы с базой данных ,необходимо понимать крайне важную деталь: существует огромная разница между объектной моделью и реляционной.

СУБД даёт нам информацию в табличном формате, в то время, как Java даёт информацию в виде некоего графа объектов.

ORM-решения имеют много преимуществ, по сравнению с "ручной" работой с базами данных в проекте. Они позволяют оптимизировать количество запросов к базе данных, но при этом избежать загрузки избыточных на данный момент для приложения данных. Кроме того, благодаря единому для всех используемых СУБД API-интерфейсу, который предоставляет ORM-фреймворк, в случае необходимости смены СУБД по желанию заказчика или просто из-за чрезмерного роста количества данных очень легко можно перейти с одной СУБД на другую, поскольку все SQL-диалекты уже реализованы в фреймворке его разработчиками, и программисту, использующему данный фреймворк.

**2.2.2. Hibernate и JPA**

JPA (Java Persistence API) это спецификация Java EE и Java SE, описывающая систему управления сохранением java объектов в таблицы реляционных баз данных в удобном виде. Сама Java не содержит реализации JPA, однако есть существует много реализаций данной спецификации от разных компаний (открытых и нет). Это не единственный способ сохранения java объектов в базы данных (ORM систем), но один из самых популярных в Java мире.

Hibernate — библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения (object-relational mapping — ORM). Она представляет собой свободное программное обеспечение с открытым исходным кодом (open source), распространяемое на условиях GNU Lesser General Public License. Данная библиотека предоставляет легкий в использовании каркас (фреймворк) для отображения объектно-ориентированной модели данных в традиционные реляционные базы данных. Hibernate совместима с JSR-220/317 и предоставляет стандартные средства JPA.

Hibernate является одним из самых востребованных ORM фреймворков для Java, потому что:

1. Hibernate устраняет множество повторяющегося кода, который постоянно преследует разработчика при работе с JDBC. Скрывает от разработчика множество кода, необходимого для управления ресурсами и позволяет сосредоточиться на бизнес логике.
2. Hibernate поддерживает XML так же как и JPA аннотации, что позволяет сделать реализацию кода независимой.
3. Hibernate предоставляет собственный мощный язык запросов (HQL), который похож на SQL. Стоит отметить, что HQL полностью объектно-ориентирован и понимает такие принципы, как наследование, полиморфизм и ассоциации.
4. Hibernate — широко распространенный open source проект. Благодаря этому доступны тысячи открытых статей, примеров, а так же документации по использованию фреймворка.
5. Hibernate легко интегрируется с другими Java EE фреймворками, например, Spring Framework поддерживает встроенную интеграцию с Hibernate.
6. Hibernate поддерживает ленивую инициализацию используя proxy объекты и выполняет запросы к базе данных только по необходимости.
7. Hibernate поддерживает разные уровни cache, а следовательно может повысить производительность.
8. Важно, что Hibernate может использовать чистый SQL, а значит поддерживает возможность оптимизации запросов и работы с любым сторонним вендором БД и его фичами.

Файл конфигурации Hibernate содержит в себе данные о базе данных и необходим для инициализации SessionFactory. В .xml файле необходимо указать вендора базы данных или JNDI ресурсы, а так же информацию об используемом диалекте, что поможет hibernate выбрать режим работы с конкретной базой данных.

Существует четыре способа конфигурации работы с Hibernate :

* используя аннотации;
* hibernate.cfg.xml;
* hibernate.properties;
* persistence.xml.

Самый частый способ конфигурации: через аннотации и файл persistence.xml, что касается файлов hibernate.properties и hibernate.cfg.xml, то hibernate.cfg.xml главнее (если в приложение есть оба файла, то принимаются настройки из файла hibernate.cfg.xml). Конфигурация аннотациями, хоть и удобна, но не всегда возможна, к примеру, если для разных баз данных или для разных ситуаций хотите иметь разные конфигурацию сущностей, то следует использовать xml файлы конфигураций.

**2.3. Сборщик Java проектов Maven**

Maven - это инструмент для сборки Java проекта: компиляции, создания jar, создания дистрибутива программы, генерации документации. Простые проекты можно собрать в командной строке. Если собирать большие проекты с командной строки, то команда для сборки будет очень длинной, поэтому её иногда записывают в bat/sh скрипт. Но такие скрипты зависят от платформы. Для того чтобы избавиться от этой зависимости и упростить написание скрипта используют инструменты для сборки проекта.

Для платформы Java существуют два основных инструмента для сборки: Ant, Maven и Gradle.

**2.3.1. Терминология Maven**

Вся структура проекта описывается в файле pom.xml (POM – Project Object Model), который должен находиться в корневой папке проекта. Ключевым понятием Maven является артефакт — это, по сути, любая библиотека, хранящаяся в репозитории. Это может быть какая-то зависимость или плагин.

* Зависимости — это те библиотеки, которые непосредственно используются в вашем проекте для компиляции кода или его тестирования.
* Плагины же используются самим Maven'ом при сборке проекта или для каких-то других целей (деплоймент, создание файлов проекта для Eclipse и др.).
* В самом начале работы с Maven, пользователь непременно столкнется с таким понятием как архетип. Архетип — это некая стандартная компоновка файлов и каталогов в проектах различного рода (веб, swing-проекты и прочие). Другими словами, Maven знает, как обычно строятся проекты и в соответствии с архетипом создает структуру каталогов.

Как правило, название артефакта состоит из названия группы, собственного названия и версии. К примеру Spring будет иметь вот такое название в среде Maven: org.springframework.spring:2.5.5. Последний домен означает всегда artifactId, все, что перед ним – groupId.

**2.3.2. Основные преимущества Maven**

Независимость от OS. Сборка проекта происходит в любой операционной системе. Файл проекта один и тот же.

* Управление зависимостями. Редко какие проекты пишутся без использования сторонних библиотек(зависимостей). Эти сторонние библиотеки зачастую тоже в свою очередь используют библиотеки разных версий. Maven позволяет управлять такими сложными зависимостями. Что позволяет разрешать конфликты версий и в случае необходимости легко переходить на новые версии библиотек.
* Возможна сборка из командной строки. Такое часто необходимо для автоматической сборки проекта на сервере (Continuous Integration).
* Хорошая интеграция со средами разработки. Основные среды разработки на java легко открывают проекты которые собираются c помощью Maven. При этом зачастую проект настраивать не нужно - он сразу готов к дальнейшей разработке.
* Как следствие - если с проектом работают в разных средах разработки, то Maven удобный способ хранения настроек. Настроечный файл среды разработки и для сборки один и тот же - меньше дублирования данных и соответственно ошибок.
* Декларативное описание проекта.

**2.4. Контейнер сервлетов Tomcat**

Tomcat (в старых версиях — Catalina) — контейнер сервлетов с открытым исходным кодом, разрабатываемый Apache Software Foundation. Реализует спецификацию сервлетов, спецификацию JavaServer Pages (JSP) и JavaServer Faces (JSF). Написан на языке Java.

Tomcat позволяет запускать веб-приложения, содержит ряд программ для самоконфигурирования.

Основные каталоги Tomcat:

* Папка bin: хранит различные скрипты, в частности, для запуска, перезагрузки и т.д.
* Папка conf: хранит конфигурационные файлы, наиболее важным из которых является файл server.xml, который определяет основную часть конфигурации.
* Папка logs: папка по умолчанию для хранения лог-файлов.
* Папка webapps: папка, где хранятся собственно файлы приложений.

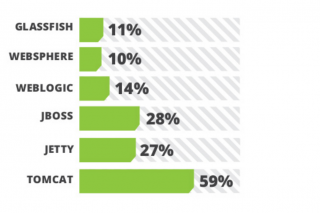
Таким образом, все файлы приложений будут помещаться в папку webapps.

Tomcat используется в качестве самостоятельного веб-сервера, в качестве сервера контента в сочетании с веб-сервером Apache HTTP Server, а также в качестве контейнера сервлетов в серверах приложений JBoss и GlassFish.

Tomcat нужен для работы Java сервера с применением сервлетов. Cервелеты это аналог тех же php скриптов. На сервер Tomcat от клиентов приходят запросы. В зависимости от них сервер запустит те или иные сервелеты, которые сформируют ответы в виде текстовых файлов. Чаще всего это html страницы.

**2.4.1. Сравнение с другими серверами приложений**

Доступные сервера, обладающие только базовым функционалом, являются наиболее используемыми, как показал Отчет Производительности Разработчиков:



*Рисунок 2.1. Отчет производительности разработчиков о серверах приложений.*

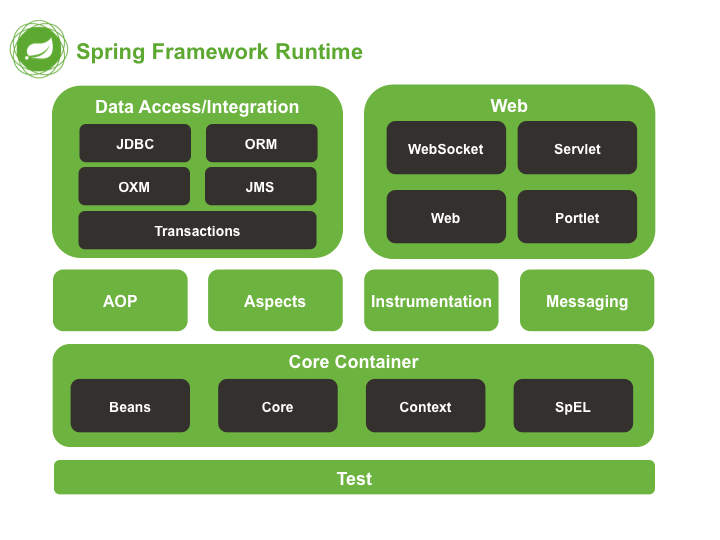
Отчёт составлялся на основе таких критерий как :

* Простота загрузки и установки;
* Настоящие метрики производительности;
* Поддержка инструментов;
* Конфигурация сервера.

JBoss и GlassFish - полномасштабные серверы приложений Java EE, включая контейнер EJB и все другие функции этого стека. С другой стороны, Tomcat имеет меньшую площадь памяти (~ 60-70 МБ), в то время как серверы Java EE весом в сотни мегабайт. Tomcat очень популярен для простых веб-приложений или приложений с использованием фреймворков, таких как Spring, которые не требуют полного Java EE-сервера. Администрирование сервера Tomcat, возможно, проще, поскольку число движущихся частей меньше.

**2.5. Spring Framework. MVC и Security.**

Spring Framework обеспечивает комплексную модель разработки и конфигурации для современных бизнес-приложений на Java - на любых платформах. Ключевой элемент Spring - поддержка инфраструктуры на уровне приложения: основное внимание уделяется "водопроводу" бизнес-приложений, поэтому разработчики могут сосредоточиться на бизнес-логике без лишних настроек в зависимости от среды исполнения.



*Рисунок 2.2. Основные компоненты Spring Framework.*

В настоящее время, под термином "Spring" часто подразумевают целое семейство проектов. В большинстве своем, они развиваются и курируются компанией Pivotal и силами сообщества. Ключевые проекты семейства Spring это:

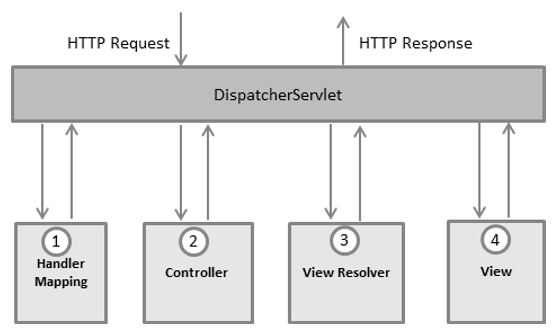
* IoC (Inversion of Control) контейнер;
* AOP-фреймворк (включая интеграцию с AspectJ);
* Data Access фреймворк;
* Transaction management;
* MVC-фреймворк;
* Remote Access фреймворк;
* Batch processing;
* Фреймворк аутентификации и авторизации;
* Remote Management;
* Messaging-фреймворк;
* Testing-фреймворк.

**2.5.1. Spring MVC**

Фреймворк Spring MVC обеспечивает архитектуру паттерна Model — View — Controller (Модель — Отображение (далее — Вид) — Контроллер) при помощи слабо связанных готовых компонентов. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними.

* *Model (Модель)* инкапсулирует (объединяет) данные приложения, в целом они будут состоять из POJO («Старых добрых Java-объектов», или бинов).
* *View (Отображение, Вид)* отвечает за отображение данных Модели, — как правило, генерируя HTML, которые мы видим в своём браузере.
* *Controller (Контроллер)* обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид.

Вся логика работы Spring MVC построена вокруг DispatcherServlet, который принимает и обрабатывает все HTTP-запросы (из UI) и ответы на них. Рабочий процесс обработки запроса DispatcherServlet'ом проиллюстрирован на следующей диаграмме:



*Рисунок 2.3. Процесс обработки запроса DispatcherServlet'ом.*

Последовательность событий, соответствующая входящему HTTP-запросу:

* После получения HTTP-запроса DispatcherServlet обращается к интерфейсу HandlerMapping, который определяет, какой Контроллер должен быть вызван, после чего, отправляет запрос в нужный Контроллер.
* Контроллер принимает запрос и вызывает соответствующий служебный метод, основанный на GET или POST. Вызванный метод определяет данные Модели, основанные на определённой бизнес-логике и возвращает в DispatcherServlet имя Вида (View).
* При помощи интерфейса ViewResolver DispatcherServlet определяет, какой Вид нужно использовать на основании полученного имени.
* После того, как Вид (View) создан, DispatcherServlet отправляет данные Модели в виде атрибутов в Вид, который в конечном итоге отображается в браузере.

Все вышеупомянутые компоненты, а именно, HandlerMapping, Controller и ViewResolver, являются частями интерфейса WebApplicationContext extends ApplicationContext, с некоторыми дополнительными особенностями, необходимыми для создания web-приложений.

**2.5.2. Spring Security**

Spring Security это Java/JavaEE framework, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework. Проект был начат Беном Алексом (Ben Alex) в конце 2003 года под именем «Acegi Security», первый релиз вышел в 2004 году. Впоследствии проект был поглощён Spring'ом и стал его официальным дочерним проектом. Впервые публично представлен под новым именем Spring Security 2.0.0 в апреле 2008 года.

Ключевые объекты контекста Spring Security:

* SecurityContextHolder, в нем содержится информация о текущем контексте безопасности приложения, который включает в себя подробную информацию о пользователе(Principal) работающем в настоящее время с приложением. По умолчанию SecurityContextHolder используетThreadLocal для хранения такой информации, что означает, что контекст безопасности всегда доступен для методов исполняющихся в том же самом потоке. Для того что бы изменить стратегию хранения этой информации можно воспользоваться статическим методом класса SecurityContextHolder.setStrategyName(String strategy). Более подробно SecurityContextHolder
* SecurityContext, содержит объект Authentication и в случае необходимости информацию системы безопасности, связанную с запросом от пользователя.
* Authentication представляет пользователя (Principal) с точки зрения Spring Security.
* GrantedAuthority отражает разрешения выданные пользователю в масштабе всего приложения, такие разрешения (как правило называются «роли»), например ROLE\_ANONYMOUS, ROLE\_USER, ROLE\_ADMIN.
* UserDetails предоставляет необходимую информацию для построения объекта Authentication из DAO объектов приложения или других источников данных системы безопасности. Объект UserDetailsсодержит имя пользователя, пароль, флаги: isAccountNonExpired, isAccountNonLocked, isCredentialsNonExpired, isEnabled и Collection — прав (ролей) пользователя.
* UserDetailsService, используется чтобы создать UserDetails объект путем реализации единственного метода этого интерфейса

Процесс аутентификации :

1. Пользователю будет предложено войти в систему предоставив имя (логин или email) и пароль. Имя пользователя и пароль объединяются в экземпляр класса UsernamePasswordAuthenticationToken(экземпляр интерфейса Authentication) после чего он передается экземпляру AuthenticationManager для проверки.
2. В случае если пароль не соответствует имени пользователя будет выброшено исключение BadCredentialsException с сообщением “Bad Credentials”.
3. Если аутентификация прошла успешно возвращает полностью заполненный экземпляр Authentication.
4. Для пользователя устанавливается контекст безопасности путем вызова метода SecurityContextHolder.getContext().setAuthentication(…), куда передается объект который вернул AuthenticationManager.

**2.6. Rest API**

API расшифровывается как Application Programming Interface. Это набор правил, позволяющий программам «общаться» друг с другом. Разработчик создает API на сервере и позволяет клиентам обращаться к нему.

REST – это архитектурный подход, определяющий, как API должны выглядеть. Читается как «Representational State Transfer». Этому набору правил и следует разработчик при создании своего приложения. Одно из этих правил гласит, что при обращении к определенному адресу, вы должны получать определенный набор данных (ресурс).

Каждый адрес маршрутом, пакет данных - запросом, в то время как результирующий ресурс – ответом.

**2.6.1. Структура запроса**

Важно понимать структуру запроса:

* Маршрут отправки
* Тип метода
* Заголовки
* Тело (или данные)
* ***Маршрут отправки***

Маршрут - это адрес, по которому отправляется ваш запрос. Root-endpoint - это точка приема запроса на стороне сервера (API).

Путь определяет запрашиваемый ресурс. Это что-то вроде автоответчика, который просит нажать 1 для одного сервиса, 2 для другого и так далее. Для понимания того, какие именно пути доступны, следует просмотреть документацию.

Последняя часть маршрута – это параметры запроса. Технически запросы не являются частью REST-архитектуры, но на практике сейчас всё строится на них. Параметры запроса позволяют использовать в запросе наборы пар «ключ-значение». Они всегда начинаются знаком вопроса. Каждая пара параметров после чего разделяется амперсантом.

JSON – JavaScript Object Notation – общий формат для отправки и приема данных посредством REST API.

* ***Тип метода***

Тип метода - обозначает тип производимого запроса, де-факто он является спецификацией операции, которую должен произвести сервер. Всего существует пять типов запросов:

* GET – используется для получения со стороны севера определенного ресурса. Если вы производите этот запрос, сервер ищет информацию и отправляет ее вам назад. По сути, он производит операцию чтения на сервере. Дефолтный тип запросов.
* POST – нужен для создания определенного ресурса на сервере. Сервер создает в базе данных новую сущность и оповещает вас, был ли процесс создания успешным. По сути, это операция создания.
* PUT и PATCH – используются для обновления определенной информации на сервере. В таком случае сервер просто изменяет информацию существующих сущностей в базе данных и оповещает об успехе выполнения операции.
* DELETE – как и следует из названия, удаляет указанную сущность из базы или сигнализирует об ошибке, если такой сущности в базе не было.
* ***Заголовки***

Заголовки используются, чтобы предоставить информацию как клиенту, так и серверу. Вообще, их можно использовать для много чего – пример – та же самая аутентификация и авторизация. Найти список доступных заголовком можно на официальной странице MDN. Заголовки представляют из себя пары ключей-значений.

* **Статус-коды и возможные ошибки**

HTTP определяет различные коды ответов для указания клиенту различной информации об операциях. Ваше REST приложение могло бы эффективно использовать все доступные HTTP-коды, чтобы помочь клиенту правильно настроить ответ. Далее представлен список кодов ответов HTTP:

* 200+: запрос успешен
* 300+: запрос перенаправлен на другой маршрут
* 400+: ошибка на стороне клиента
* 500+: ошибка на стороне сервера
* **Версии API**

Время от времени разработчики обновляют свои API. Порой обновления могут быть такими сильными, что разработчик желает выпустить релиз новой версии. В таком случае, если ваше приложение ломается, это происходит по причине, что вы писали код с учетом старого компонента, тогда как новый несколько отличается в плане реализации.

Запросить текущую версию API можно двумя путями.

* Через маршрут
* Через заголовок

**2.7. Инструменты использующиеся в разработке Web модуля приложения.**

В Web части приложения использовался фреймворк Angular. Для упрощения и ускорения работы использовался npm менеджер пакетов. За автоматизацию сборки модулей в единые пакеты был использован Webpack. Pug и Sass препроцессоры предоставили возможность для быстрой верстки.

**2.7.1. NPM**

Npm (node package manager) — это менеджер пакетов Node.js. В npm имеется более полумиллиона пакетов, что делает его самым большим в мире репозиторием кода, написанного на одном языке. Это позволяет говорить о том, что в npm можно найти пакеты, предназначенные для решения практически любых задач.

Изначально npm создавался как система управления пакетами для Node.js, но в наши дни он используется и при разработке фронтенд-проектов на JavaScript. Для взаимодействия с реестром npm используется одноимённая команда, которая даёт разработчику огромное количество возможностей:

* *Загрузка пакетов*

С помощью команды npm можно загружать пакеты из реестра. Ниже мы рассмотрим примеры её использования.

* *Установка всех зависимостей проекта*

Если в проекте имеется файл package.json, то установить все зависимости этого проекта можно такой командой: *npm install*. Эта команда загрузит всё, что нужно проекту, и поместит эти материалы в папку node\_modules, создав её в том случае, если она не существует в директории проекта.

* *Установка отдельного пакета*

Отдельный можно установить следующей командой: npm install <package-name>

Часто можно видеть, как эту команду используют не в таком вот простом виде, а с некоторыми флагами:

* Флаг --save позволяет установить пакет и добавить запись о нём в раздел dependencies файла package.json, который описывает зависимости проекта. Эти зависимости используются проектом для реализации его основного функционала, они устанавливаются в ходе его развёртывания на сервере (после выхода npm 5 записи об устанавливаемых пакетах в разделе зависимостей делаются автоматически, и без использования этого флага).
* Флаг --save-dev позволяет установить пакет и добавить запись о нём в раздел, содержащий перечень зависимостей разработки (то есть — пакетов, которые нужны в ходе разработки проекта, вроде библиотек для тестирования, но не требуются для его работы) файла package.json, который называется devDependencies.
* *Обновление пакетов*

Для обновления пакетов служит следующая команда: npm update

Получив эту команду, npm проверит все пакеты на наличие их новых версий, и, если найдёт их новые версии, соответствующие ограничениям на версии пакетов, заданным в package.json, установит их.

Обновить можно и отдельный пакет: npm update <package-name>

Эти и многие другие возможности npm делают его необходимым в современном мире разработке приложений.

**2.7.2. Webpack**

В большинстве языков программирования есть возможность импортирования кода из одного файла в другой. Изначально в JS такой возможности не было, потому что этот язык разрабатывался только для исполнения в браузере, без доступа к файловой системе на клиентской машине (по причинам безопасности). Так что долгое время для организации JS-кода в нескольких файлах требовалось загружать каждый файл с глобально доступными переменными.

Webpack - это инструмент для сборки модулей в единые пакеты, имеющий доступ к файловой системе. Получающиеся пакеты совместимы с браузером, которому не нужен доступ к файловой системе. В нашем случае бандлер нужен для поиска всех выражений require (имеющих ошибочный, с точки зрения браузера, JS-синтаксис) и замены на настоящее содержимое каждого требуемого файла. В финале получается единый JS-файл без выражений require.

**2.7.3. Pug**

Pug - это препроцессор HTML и шаблонизатор, который был написан на JavaScript для Node.js. Pug — ребрендинг Jade. Был переименован из-за жалобы некоего одноименного бренда, не связанного с IT. Pug это именно то средство, которое предоставляет вам возможность написания разметки совершенно по новому, с целым рядом преимуществ по сравнению с обычным HTML.

Три основные черты Pug:

* Простые теги;
* Добавление атрибутов в теги;
* Блоки текста.

Полезные функции, которые делают разметку умнее. Среди них:

* JavaScript

Pug реализован на JavaScript, по этому использовать JavaScript в Pug довольно просто.

* Циклы

Pug использует прекрасный синтаксис для написания циклов, так что вам не придётся прибегать к JavaScript.

* Интерполирование

В строке создаётся конструкция ${...}, внутри которой вы можете поместить любую переменную или выражение.

* Миксины

Миксины, они как функции, они принимают параметры в качестве входных данных и генерируют соответствующую разметку. Миксины оглашаются с помощью ключевого слова mixin.

**2.7.4. SASS**

Sass — это метаязык на основе CSS, предназначенный для увеличения уровня абстракции CSS кода и упрощения файлов каскадных таблиц стилей.

Язык Sass имеет два синтаксиса:

* sass — отличается отсутствием фигурных скобок, в нём вложенные элементы реализованы с помощью отступов;
* SCSS (Sassy CSS) — использует фигурные скобки, как и сам CSS.

Написание CSS само по себе интересное занятие, но когда таблица стилей становится огромной, то становится и сложно её обслуживать. И вот в таком случае нам поможет препроцессор. Sass позволяет использовать функции недоступные в самом CSS, например, переменные, вложенности, миксины, наследование и другие приятные вещи, возвращающие удобство написания CSS.

Препроцессор обрабатывает ваш Sass-файл и сохраняет его как простой CSS-файл, который можно использовать на любом сайте.

Полезные функции SASS, которые делают таблицы стилей удобными в использовании:

* Переменные

Переменные - это способ хранения информации, которую используется на протяжении написания всех стилей проекта. Можно хранить в переменных цвета, стеки шрифтов или любые другие значения CSS, которые нужно использовать. Чтобы создать переменную в Sass нужно использовать символ $.

* Вложенности

Sass позволяет вкладывать CSS селекторы таким же образом, как и в визуальной иерархии HTML. Но чрезмерное количество вложенностей делает документ менее читабельным и воспринимаемым, что считается плохой практикой.

* Фрагментирование

Создавать фрагменты Sass-файла, которые будут содержать в себе небольшие отрывки CSS, которые можно будет использовать в других Sass-файлах. Это отличный способ сделать CSS модульным, а также облегчить его обслуживание. Фрагмент — это простой Sass-файл, имя которого начинается с нижнего подчеркивания, например, \_partial.scss. Нижнее подчеркивание в имени Sass-файла говорит компилятору о том, что это только фрагмент и он не должен компилироваться в CSS. Фрагменты Sass подключаются при помощи директивы @import.

* Расширение/Наследование

Это одна из самых полезных функций Sass. Используя директиву @extend можно наследовать наборы свойств CSS от одного селектора другому. Это позволяет держать Sass-файл в «чистоте». Класс-шаблон - особый тип классов, который выводится только при использовании расширения - это позволит сохранить ваш скомпилированный CSS чистым и аккуратным.

* Математические операторы

Использовать математику в CSS очень полезно. Sass имеет несколько стандартных математических операторов, таких как +, -, \*, / и %.

**2.8. Инструменты использующиеся в разработке Mobile модуля приложения**

Mobile часть приложения была сделана под операционную систему Android на языке Java. Для работы с сетью использовалась, довольно популярная среди Android-разработчиков, библиотека Retrofit. Mobile приложение не будет хранить все данные на сервере. В режиме offline данные будут хранится в компактной встраиваемой СУБД SQLite.

**2.8.1. Android**

Android — операционная система для смартфонов, планшетов, электронных книг, цифровых проигрывателей, наручных часов, фитнес-браслетов, игровых приставок, ноутбуков, нетбуков, смартбуков, телевизоров и других устройств (в 2015 году появилась поддержка автомобильных развлекательных систем и бытовых роботов).

Основана на ядре Linux и собственной реализации виртуальной машины Java от Google. Изначально разрабатывалась компанией Android Inc., которую затем купила Google. Впоследствии Google инициировала создание альянса Open Handset Alliance (OHA), который сейчас занимается поддержкой и дальнейшим развитием платформы. Android позволяет создавать Java-приложения, управляющие устройством через разработанные Google библиотеки. Android Native Development Kit позволяет портировать библиотеки и компоненты приложений, написанные на Си и других языках.

Android стала самой революционной операционной системой благодаря разработке по революционной методологии SCRUM. В 86 % смартфонов, проданных во втором квартале 2014 года, была установлена операционная система Android. На конференции для разработчиков в мае 2017 года Google объявила, что за всю историю Android было активировано более 2 млрд Android-устройств.

Важно понимать, что само приложение для Android пишется на Java (а теперь ещё и на Kotlin), а среда разработки выбирается по вкусу. То, что можно сделать на Android Studio (или в его старшем брате IntelliJ IDEA), можно сделать и на Eclipse и в других редакторах. Хотя с 2016 официальная поддержка Eclipse прекратилась.

**2.8.2. Retrofit**

Retrofit — это известная среди Android-разработчиков библиотека для сетевого взаимодействия, некоторые даже считают её в каком-то роде стандартом. Причин для такой популярности масса: библиотека отлично поддерживает REST API, легко тестируется и настраивается, а запросы по сети с её помощью выполняются совсем просто.

Для работы с Retrofit необходимы следующие три класса:

* Model класс, который используется как модель JSON
* Интерфейсы, которые определяют возможные HTTP операции
* Класс Retrofit.Builder — экземпляр, который использует интерфейс и API Builder, чтобы задать определение конечной точки URL для операций HTTP

Каждый метод интерфейса представляет собой один из возможных вызовов API. Он должен иметь HTTP аннотацию (GET, POST и т. д.), чтобы указать тип запроса и относительный URL. Возвращаемое значение завершает ответ в Call-объекте с типом ожидаемого результата.

Retrofit позволяет сделать полноценный REST-клиент, который может выполнять POST, GET, PUT, DELETE. Для обозначения типа и других аспектов запроса используются аннотации. Например, для того, чтобы обозначить, что требуется @GET запрос, нам нужно написать перед методом GET, для @POST запроса POST, и так далее.

**2.8.3. SQLite**

Платформа Android имеет встроенный инструментарий для управления базой данных sqlite3. SQLite - это замечательный продукт, который успел завоевать признание во всем мире и получить множество наград. Интересно отметить, что iOS (iPhone, iPod, iPad) также используют базы данных SQLite. А недавно и Microsoft присоединилась к данному решению и телефоны Windows Phone 8 также работают с SQLite.

SQLite - это проект с открытыми исходными кодами, поддерживающий стандартные возможности обычной SQL: синтаксис, транзакции и др. Занимает очень мало места - около 250 кб. Домашняя страница SQLite: http://www.sqlite.org.

Основную функциональность по работе с базами данных предоставляет пакет android.database. Функциональность непосредственно для работы с SQLite находится в пакете android.database.sqlite.

База данных в SQLite представлена классом android.database.sqlite.SQLiteDatabase. Он позволяет выполнять запросы к бд, выполнять с ней различные манипуляции.

Класс android.database.sqlite.SQLiteCursor предоставляет запрос и позволяет возвращать набор строк, которые соответствуют этому запросу.

Класс android.database.sqlite.SQLiteQueryBuilder позволяет создавать SQL-запросы.

Сами sql-выражения представлены классом android.database.sqlite.SQLiteStatement, которые позволяют с помощью плейсхолдеров вставлять в выражения динамические данные.

Класс android.database.sqlite.SQLiteOpenHelper позволяет создать базу данных со всеми таблицами, если их еще не существует.

В SQLite применяется следующая система типов данных:

* INTEGER: представляет целое число, аналог типу int в java
* REAL: представляет число с плавающей точкой, аналог float и double в java
* TEXT: представляет набор символов, аналог String и char в java
* BLOB: представляет массив бинарных данных, например, изображение, аналог типу int в java

Сохраняемые данные должны представлять соответствующие типы в java.

**2.9. Cреда разработки IntelliJ Idea**

Одна из самых умных и удобных сред разработки для Java, включающая поддержку всех последних технологий и фреймворков. IntelliJ IDEA предоставляет инструменты для продуктивной работы и идеально подходит для создания коммерческих, мобильных и веб-приложений.

Представленную программу следует рассматривать как интеллектуальную среду разработки, понимающую код. В процессе его написания программистом она занимается построением синтаксического дерева, определением особенностей размещенных ссылок, анализом возможных путей исполнения операторов и передачи данных. Основываясь на полученных результатах, программа обращает внимание специалиста на существующие ошибки и самостоятельно устраняет их, предоставляет варианты автоматического дополнения кода. Благодаря указанным особенностям она избавляет пользователя от повседневной рутины и позволяет ему сконцентрироваться на более важных задачах.

Данная программа помогает специалисту экономить время вследствие глубокого анализа контекста и удаления неподходящих вариантов. Эта и другие детали обеспечивают повышение уровня продуктивности пользователя, одновременно позволяя ему получать больше удовольствия от деятельности.

Как правило, специалисты, использующие для работы указанную программу, пишут на Java. В то же время достаточно востребованными языками считаются Scala, Groovy. Наиболее часто данный продукт применяется для разработки веб-приложений и приложений для Android. Что касается существующих лицензий, покупателям предлагается приобрести один из двух вариантов программы. Первый из них является полнофункциональной редакцией, цена которой определяется категорией заказчика. В частности, образовательные учреждения и разработчики проектов с открытым исходным кодом могут получить ее совершенно бесплатно. Кроме того, купить intellij idea специалистам удастся в урезанном варианте. Он не поддерживает ряд языков и интеграцию с определенными инструментами. Заказать программу физические и юридические лица способны на основном сайте компании-разработчика либо у одного из дистрибьюторов. Вид лицензии непосредственно зависит от программного обеспечения и типа покупателя. Это важно принимать во внимание.

IntelliJ IDEA Ultimate - полная версия самой умной среды разработки для создания коммерческих приложений предоставляет такие возможности как:

* Умное автодополнение, инструменты для анализа качества кода, удобная навигация, расширенные рефакторинги и форматирование для Java, Groovy, Scala, HTML, CSS, JavaScript, CoffeeScript, ActionScript, LESS, XML и многих других языков.
* Поддержка всех популярных фреймворков и платформ, включая Java EE, Spring Framework, Grails, Play Framework, GWT, Struts, Node.js, AngularJS, Android, Flex, AIR Mobile и многих других.
* Интеграция с серверами приложений, включая Tomcat, TomEE, GlassFish, JBoss, WebLogic, WebSphere, Geronimo, Resin, Jetty и Virgo.
* Инструменты для работы с базами данных и SQL файлами, включая удобный клиент и редактор для схемы базы данных.
* Интеграция с коммерческими системами управления версиями Perforce, Team Foundation Server, ClearCase, Visual SourceSafe.
* Инструменты для запуска тестов и анализа покрытия кода, включая поддержку всех популярных фреймворков для тестирования.

**2.9.1.Cравнение с другими IDE**

* **Eclipse**

Eclipse работает довольно медленно, поскольку упирается корнями в довольно старое «железо» и старые версии JVM. Даже сегодня она кажется медлительной, особенно если использовать много плагинов.

Часть расходов ресурсов Eclipse можно отнести на счёт её встроенного инкрементного компилятора, который запускается каждый раз при загрузке файла или обновлении кода.

Экосистема плагинов Eclipse — это одновременно сильная сторона этой IDE и одна из главных её проблем. Именно из-за несовместимости плагинов порой «падают» целые сборки, и программистам приходится начинать работу сначала.

* **NetBeans**

Редактор NetBeans поддерживает языки, обнаруживает ошибки в то время, когда печатаете, и помогает с помощью всплывающих подсказок и «умным» автодополнением кода. По субъективному ощущению IDE справляется с этой задачей быстрее, чем Eclipse, но несколько медлительнее IntelliJ IDEA.

PostgreSQL 9.6 <https://postgrespro.ru/docs/postgresql/9.6/preface>

Чем PostgreSQL лучше других SQL баз данных с открытым исходным кодом. Часть 1 <https://habr.com/ru/post/282764/>

Собеседование по Java EE — Hibernate Framework <http://javastudy.ru/interview/jee-hibernate-questions-answers/>

Hibernate: ORM вообще и в Java в частности <https://www.kv.by/archive/index2008401108.htm>

Руководство по Maven <http://www.apache-maven.ru/>

Apache Maven — основы <https://habr.com/ru/post/77382/>

Структура Apache Tomcat <https://metanit.com/java/javaee/2.2.php>

Apache Tomcat - <https://ru.wikipedia.org/wiki/Apache_Tomcat>

Дебаты об Отличном Сервере Приложений Java c Tomcat, Jboss, GlassFish, Jetty и Liberty Profile <https://habr.com/ru/company/otus/blog/343890/>

Introduction to Spring Framework <https://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/overview.html>

Как писать на Spring в 2017 <https://habr.com/ru/post/333756/>

Spring MVC — основные принципы <https://habr.com/ru/post/336816/>

Spring Security <https://spring.io/projects/spring-security>

Краткий обзор Spring Security <https://habr.com/ru/post/203318/>

КАК ПРАВИЛЬНО РАБОТАТЬ С REST API <https://itvdn.com/ru/blog/article/rest-api-18>

REST API Best Practices <https://habr.com/ru/post/351890/>

Руководство по Node.js, часть 4: npm, файлы package.json и package-lock.json <https://habr.com/ru/company/ruvds/blog/423703/>

Объясняем современный JavaScript динозавру <https://habr.com/ru/company/mailru/blog/340922/>

Туториал по Jade для начинающих <https://habr.com/ru/post/278109/>

SASS - <https://ru.wikipedia.org/wiki/Sass>

SASS основы - <https://sass-scss.ru/guide/>

Android <https://ru.wikipedia.org/wiki/Android>

Android Подготовка к разработке <http://developer.alexanderklimov.ru/android/android1.php>

Используем Retrofit 2 в Android-приложении <https://habr.com/ru/post/429058/>

Работа с базами данных SQLite <https://metanit.com/java/android/14.5.php>

SQLite <http://developer.alexanderklimov.ru/android/sqlite/>

ПРОГРАММА INTELLIJ IDEA: ПРЕИМУЩЕСТВА, ОСОБЕННОСТИ, ПРАВИЛА ПОЛЬЗОВАНИЯ <http://bayguzin.ru/main/uroki/soft/programma-intellij-idea-preimushchestva-osobennosti-pravila-polzovaniya.html>

IntelliJ IDEA <https://jetbrains.ru/products/idea/>

Eclipse, NetBeans или IntelliJ IDEA? Выбираем IDE для Java-разработки <https://javarush.ru/groups/posts/1642-eclipse-netbeans-ili-intellij-idea-vihbiraem-ide-dlja-java-razrabotki>