## 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Проанализировав необходимый к реализации функционал, были выработаны требования к проектируемой системе. Для реализации системы было решено разбить архитектуру приложения на несколько составных модулей. Это позволит вносить изменения в существующий код приложения, которые не будут затрагивать остальные модули системы, а воздействовать лишь на свой блок. Тем самым обеспечим архитектуру не только модульностью, но и масштабируемостью.

Масштабируемость – это способность системы расширяться, наращивать функционал и увеличивать свою производительность.

Мобильное приложение разрабатывается под операционную систему Android. Перед началом разработки очень важно выбрать версию операционной системы, под которую разрабатывается приложение. Операционная система Android является системой с обратной совместимостью. Это означает, что приложения, которые были написаны используя старую версию SDK (Software Development Kit), гарантированно будут поддерживаться на последней версии системы, а также на версиях системы, которые будут выходить в будущем. Ознакомившись с данными, иллюстрирующими распространенность версий ОС мобильных устройств на данный момент, выбор был сделан в пользу версии Android 5.0, которая используются на 80.2% устройств (рисунок 2.1).

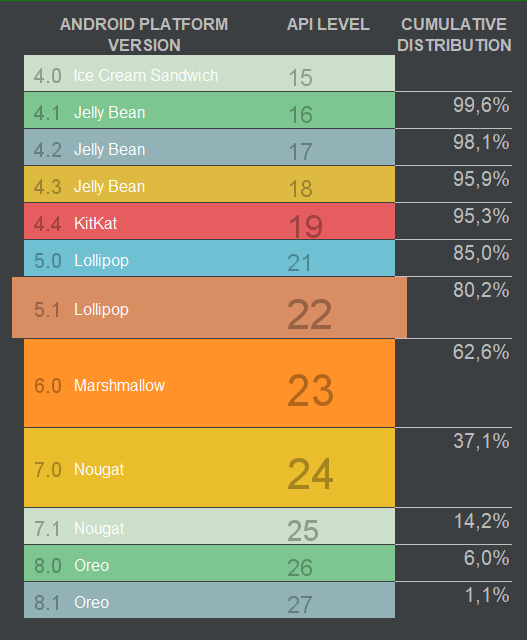


Рисунок 2.1 – Распространённость версий ОС Android

В разрабатываемом мобильном приложении можно выделить несколько составных модулей(блоков):

* модуль ресурсов;
* модуль представлений;
* модуль данных;
* модуль обработки событий;
* модуль взаимодействия с API.

*Модуль ресурсов* представляет собой файлы в формате XML, которые использует операционная система Android для хранения следующих видов данных:

* описание анимаций приложения;
* данные приложения (цвета приложения, стили приложения, строковые данные, размеры и отступы);
* иконки и изображения;
* описания меню;
* разметка экранов пользовательского интерфейса.

Этот модуль обеспечивает приложению гибкость настройки. Так, можно настроить поддержку нескольких языков в приложении, создав для каждого языка свой строковый ресурс или быстро заменить описанные выше ресурсы, не вмешиваясь в исходный код приложения.

*Модуль представлений* – это модуль, содержащий в себе всё необходимое, для отображения пользовательского интерфейса и управления им. Ключевым компонентом пользовательского интерфейса в Android является activity (активность). Одна активность обычно представляет собой один экран приложения, на котором пользователь может взаимодействовать с ним. Активности описываются в виде Java-класса, унаследованного от системного класса android.app.Activity, переопределяя его методы. Также этот модуль контролирует состояние этих активностей с помощью наблюдения за их жизненным циклом. Для инициализации компонентов активности, модуль представлений обращается к ресурсам, описанным в модуле ресурсов.

*Модуль данных* – является модулем, который хранит в себе модели данных в виде Java-объектов. Этот модуль предоставляет другим модулям структурированные данные о некотором объекте и методы для управления этими данными. Модели данных не зависят от пользовательского интерфейса, а лишь предоставляют для него данные к отображению. В некоторых моделях данных описаны части бизнес-логики приложения, которая затрагивает только поведение той модели, в которой она описана.

*Модуль обработки событий* взаимодействует с модулем представлений, модулем данных и модулем взаимодействия с API, отлавливая события, которые необходимо в дальнейшем обработать определённым образом. Такое поведение обеспечивается благодаря методам обратного вызова, описанных в других модулях. Эти методы вызываются при возникновении некоторого события (нажатие кнопки пользователем, ответ от сервера на посланный запрос, поворот экрана, закрытие приложения).

*Модуль взаимодействия с API*: представляет собой интерфейс, который обращается по протоколу HTTP к поставщику контента за необходимой информацией и получает её в формате JSON. Затем, через взаимодействие с модулем обработки событий обрабатывает полученные данные и преобразует ее в вид POJO (Plain Old Java Objects) для модели данных. После того как обновятся модели данных, происходит взаимодействие с модулем представлений через модуль обработки событий. Модуль представлений в свою очередь обновляет содержимое пользовательского интерфейса. Для реализации этого модуля используется сторонняя библиотека Retrofit, которая позволяет описать лишь интерфейс взаимодействия с API, без необходимости реализовывать обмен данными по протоколу HTTP стандартными средствами Java.

Разрабатываемое программное средство подразумевает под собой не только мобильное приложение, но также и серверную часть, обслуживающую его. Приложения под Android традиционно разрабатываются на языке программирования Java. Для удобства и совместимости, серверная часть также будет разработана на этом языке программирования.

Серверная часть приложения отвечает за хранение и предоставление пользовательской информации, такой как имя пользователя, его пароль, история просмотров и оценок пользователя. Также посредством обращения пользовательского мобильного приложения к веб-серверу, происходит авторизация пользователя в системе.

Серверная часть реализуется с использованием Spring Framework. Spring – универсальный фреймворк с открытым исходным кодом для Java-платформы. Spring можно рассматривать как целый набор мини-фреймворков, большинство из которых могут работать независимо друг от друга.

Использование Spring и его мини-фреймворков при реализации серверной части программного средства позволит реализовать все модули, необходимые для работы приложения:

* Spring Security для авторизации пользователя;
* Spring Boot для простого и быстрого разворачивания приложения на сервере;
* Spring Data для работы с базой данных приложения.

Серверная часть содержит в себе следующие модули:

* веб-сервер приложения;
* модуль обработки запросов;
* модуль бизнес-логики;
* модуль работы с БД;
* база данных;

*Веб-сервер приложения* представляет собой контейнер сервлетов с открытым исходным кодом Apache Tomcat. Выбор именно этого веб-сервера обусловлен тем, что он наиболее совместим с Java, и считается стандартом в разработке веб-приложений на этом языке программирования. Веб-сервер получает запросы по протоколу HTTP и передаёт эти запросы модулю обработки запросов.

*Модуль обработки запросов* получает запрос от веб-сервера и проверяет возможность обработки этого запроса. Если же запрос валиден и обработать его можно, то модуль обработки запросов извлекает из него всю необходимую информацию (например параметры) и передает дальнейшее управление обработкой запроса модулю бизнес-логики. Этот модуль также отвечает за отдачу информации обратно клиенту (если требуется). После обработки запроса Spring MVC автоматически формирует ответ клиенту в сконфигурированном заранее на сервере формате, например в формате JSON, который отлично подходит под задачи передачи объектов приложения между двумя модулями. Этот модуль является RESTful сервисом. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

*Модуль бизнес-логики* предназначен для выполнения конкретных операцийнад данными приложения, взаимодействуя при этом с модулем базы данных. Этот модуль представляет собой слой, который находится по середине между слоем сервиса (модуль обработки запросов) и слоем данных (модуль базы данных). Поэтому в нем находится реализация всех функций, которые должны присутствовать в серверной части программного средства, разрабатываемого в рамках дипломного проекта. Здесь описаны процессы авторизации пользователя в приложении, получение информации о просмотрах и оценках, а также формируются индивидуальные рекомендации к просмотру для пользователя.

*Модуль работы с базой данных* взаимодействует с базой данных приложения, модифицируя или извлекая данные для других слоев. Предоставляет другим модулям интерфейс, через который они могут общаться с базой данных, не зная ничего о её структуре и наполнении. Такая архитектура позволяет изменять структуру базы данных, не изменяя бизнес-логику и архитектуру других модулей и не нарушая целостности системы. При изменении структуры данных необходимо лишь изменить интерфейс, который предоставляется данным модулем другим слоям для взаимодействия с данными из базы. В этом модуле используется Spring Data, которая позволяет еще больше упростить процесс работы с базой данных, реализуя доступ к сложным SQL-запросам через простое описание в названии метода, который используется для вызова этого запроса.

*База данных -* набор сведений, хранящихся некоторым упорядоченным способом. В серверной части приложения было решено использовать MySQL в качестве СУБД. Плюсы использования именно этой СУБД:

* полнофункциональная версия является полностью бесплатной;
* может запускаться на любой ОС;
* MySQL поддерживает большинство функционала SQL;
* большое количество функций, обеспечивающих безопасность, которые поддерживается по умолчанию;
* MySQL легко работает с большими объемами данных и легко масштабируется;
* упрощение некоторых стандартов позволяет MySQL значительно увеличить производительность.

База данных приложения содержит в себе несколько таблиц, связанных между собой и описывающих информацию о пользователях, а также данные их личных профилей, такие как историю просмотров и оценок. Пароли пользователей хранятся в зашифрованном виде, чтобы обеспечить конфиденциальность информации, а также защитить пользователей от утечки личных данных.

Обе части архитектуры приложения (пользовательское мобильное приложение и серверная часть) реализованы в рамках архитектуры MVC (Model-View-Controller). Этот шаблон проектирования предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

Под компонентом следует понимать некую отдельную часть кода, каждая из которых играет одну из ролей: Контроллера, Модели или Представления, где Модель служит для извлечения и манипуляций данными приложения, Представление отвечает за видимое пользователю отображение этих данных, а Контроллер управляет всем процессом взаимодействия компонентов между собой, обрабатывая события. Такая архитектура позволяет модифицировать каждый компонент системы независимо друг от друга.

*Поставщик информации* - также является частью структурной модели программного средства, однако представляет собой сторонний ресурс, предоставляющий доступ по ключу к данным о медиаконтенте (API), который можно получить в формате JSON.При необходимости, модуль взаимодействия с API обращается на этот ресурс, отсылая ему запрос определенного формата по протоколу HTTP. Поставщик информации обрабатывает этот запрос и отсылает клиенту ответ, содержащий запрашиваемую информацию о медиаконтенте.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.304 С1.