Отчет по курсовой работе

по дисциплине «Конструирование программного обеспечения»

|  |  |  |
| --- | --- | --- |
| Выполнили студенты гр. 5130904/30107 |  | Голиков В.С.  Плужник А.Д  Губанов И.А. |
| Руководитель |  | Юркин В. А. |

**Оглавление**

[Введение 3](#_Toc211443074)

[Определение проблемы 3](#_Toc211443075)

[Выработка требований 3](#_Toc211443076)

[Разработка архитектуры и детальное проектирование 4](#_Toc211443077)

[Характер нагрузки на сервис 4](#_Toc211443078)

# Введение

Выполнение курсовой работы заключалось в создании мобильного приложения Placy - социальной сети в виде интерактивной карты, где пользователи по геолокации могут прикреплять свои фотографии, сделанные в режиме реального времени, и делиться ими с друзьями. Идея базируется на таких приложениях как BeReal, Zenly и Locket. Приложение будет написано на языке Kotlin с использованием библиотеки разработки UI – JetpackCompose. Также будет использоваться СУБД PostgreSQL и внешняя зависимость Yandex Maps API. Курсовая работа также включает в себя работу с git хостингом github

Выбор темы проекта и технологического стека

Тема: Создание мобильного приложения – социальной сети на базе BeReal, Zenly, Locket.

Состав команды:

* Голиков Валентин Сергеевич 5130904/30107
* Плужник Анастасия Дмитриевна 5130904/30107
* Губанов Иван Андреевич 5130904/30107

Ссылка на репозиторий: <https://github.com/Nao2705/placy.git>

Технологический стек:

Внешняя зависимость: Yandex Maps API

СУБД: PostgreSQL

Язык: Kotlin

IDE: Android Studio

UI: Jetpack Compose

# Определение проблемы

Проблема: Создание качественного и уникального отечественного продукта в рамках программы импортозамещения ушедших сервисов.

# Выработка требований

Требования (в виде описания пользовательских сценариев):

1. Я путешественник, который хочет делиться фотографиями в режиме реального времени. Я хочу иметь социальную сеть, которая одновременно позволяет мне делиться своим местоположением, фотографией с этого места и комментариями к ней. Я открываю приложение и могу ознакомиться со всеми фотографиями на интерактивной карте. Нажав кнопку «сделать фото», я делаю снимок и добавляю подпись к нему. После нажатия кнопки «разместить» фотография публикуется на месте моей геолокации.
2. Я не знаю куда поехать. Я открываю приложение и просматриваю фотографии других пользователей в интересующих меня местах. Я могу нажать на фотографию и узнать дополнительную информацию (подпись, адрес, открытую информацию об авторе). Я могу создать запрос на добавление в друзья к автору фото, а также добавить фотографию или место в «избранное», чтобы просмотреть в любой момент. Я могу нажать на кнопку профиля и отредактировать информацию о себе, а также просмотреть список друзей и новые заявки

Количество пользователей в сутки: 20 тысяч человек

Период хранения информации: Информация о фото должна храниться в базе 5 лет, чтобы карта мира не забивалась большим количеством фотографий. Учетные записи пользователей хранятся бессрочно.

# Разработка архитектуры и детальное проектирование

## Характер нагрузки на сервис

* **Соотношение R/W нагрузки:**
  + Соотношение Чтение/Запись будет эквивалентно 80/20%, следовательно основная нагрузка придется на чтение информации. Запись будет производиться реже, в основном для публикации новых фотографий с метаданными. Причиной этому является постоянное превалирование всех фотографий над общим количеством фотографий, созданных за день.
* **Объемы трафика:**
* Ежедневный трафик: 20.000 пользователей в сутки. Если каждый из них выполняет в среднем 30-35 запросов на чтение (просмотр фото, масштабирование карты, просмотр профилей) и 2-4 запроса на запись (публикация фото, добавление новых друзей), то в среднем мы получим [20.000 \* (35 + 4)] = 780.000 запросов в сутки (берем параметры по верхней границе)
* Объем данных на запрос: самый «тяжелый »запрос включает в себя показ фото в приложении пользователя, что в среднем составляет 4-5 МБ, остальные запросы возвращают сравнимо меньший объем информации, следовательно, будем считать объемом данных по запросу – 5 МБ
* Общий объем трафика в день: 780.000 \* 5 МБ = 3.5 ТБ (при постоянной ежедневной активности 20.000 пользователей)
* **Объем дисковой системы:**
  + Хранение фотографий: В день публикуется 20 000 пользователей \* 1.5 фото/день = 30 000 фотографий в день. (1-2 фото, берем среднее)
  + Размер одной фотографии: Исходное фото ~4-5 МБ (объем фотографии для социальных сетей) + превью (для карты и ленты) ~132 КБ. ~4.8 МБ на одно размещение (с предзагрузкой данных).
  + Ежедневный прирост: 30 000 фото \* 4.8 МБ = ~141 ГБ в день.
  + Хранение за 5 лет (1825 дней): 141 ГБ/день \* 1825 дней = ~250 ТБ.
  + Хранение данных в PostgreSQL (метаданные, пользователи, социальный граф):

- Метаданные фотографии (геолокация, timestamp, подпись, информация из профиля пользователя): ~1 КБ на запись.

- Данные пользователя (профиль, настройки): ~10 КБ на пользователя.

- Данные социальных действий (лайки, избранное, друзья): ~0.1 КБ на действие.

- Общий объем данных в БД за 5 лет: Метаданные фото: 30 000 записей/день \* 1825 дней \* 1 КБ = ~55 ГБ.

- Данные пользователей: 20 000 пользователей (активных) + учет неактивных (пусть будет 60 000 всего) \* 10 КБ = ~0,763 ГБ.

- Социальные действия: (предположим, 10 действий на фото) 30 000 фото/день \* 10 \* 1825 дней \* 0.1 КБ = ~55 ГБ.

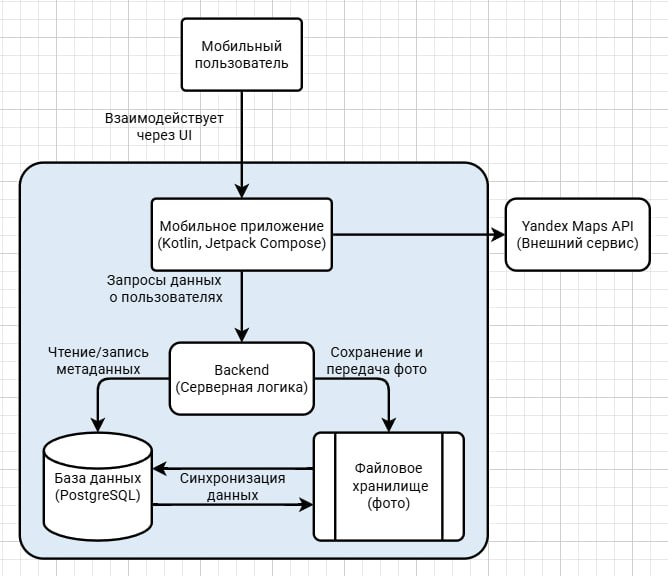
- Итого для БД: ~110 ГБ.

* + **Общее дисковое пространство**: ~300ТБ (с учетом запаса)

## System Context Diagram



## Container Diagram



## Контракты API

Локатор API инструмент для определения местоположения устройств без использования GPS. В качестве источников данных API может принимать:

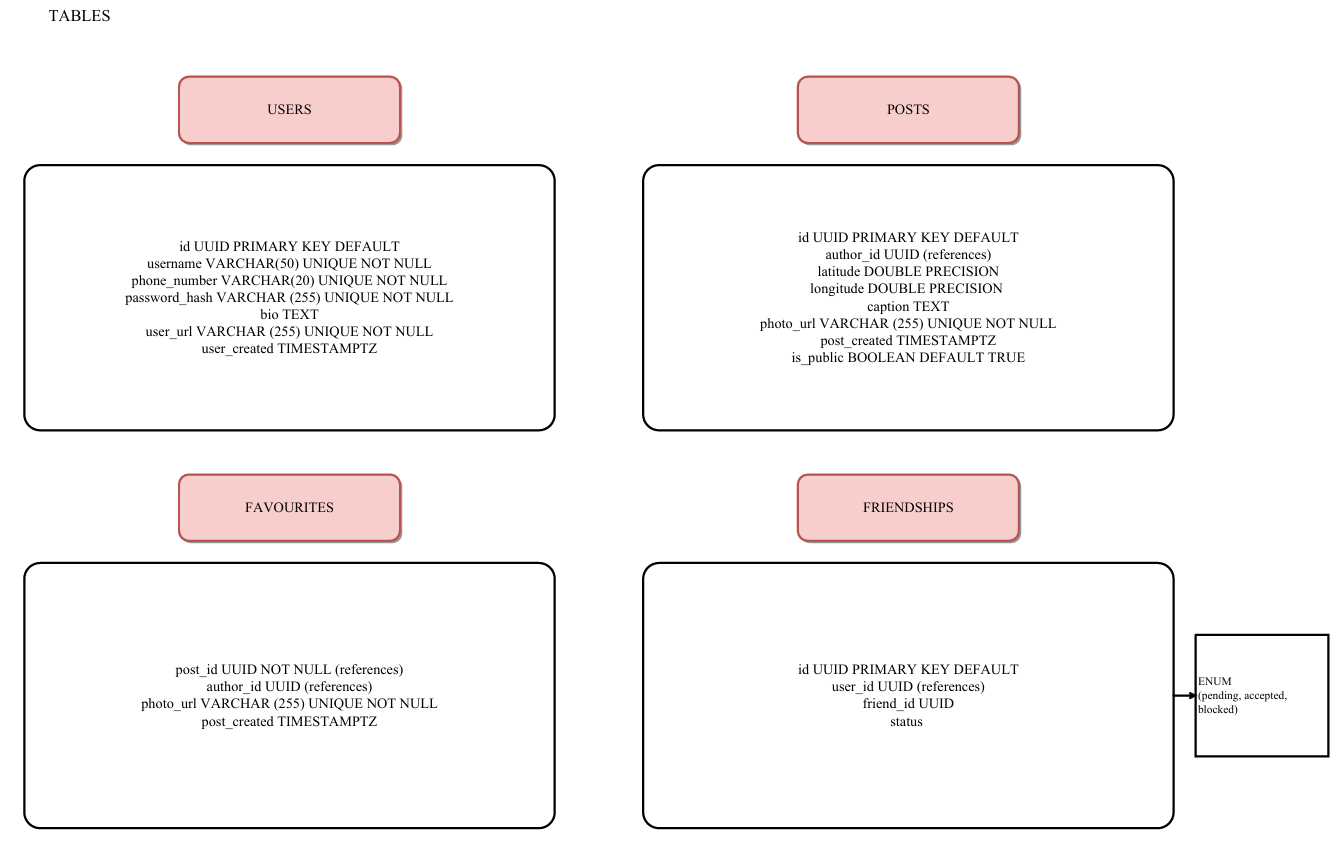
cигналы точек доступа Wi‑Fi;

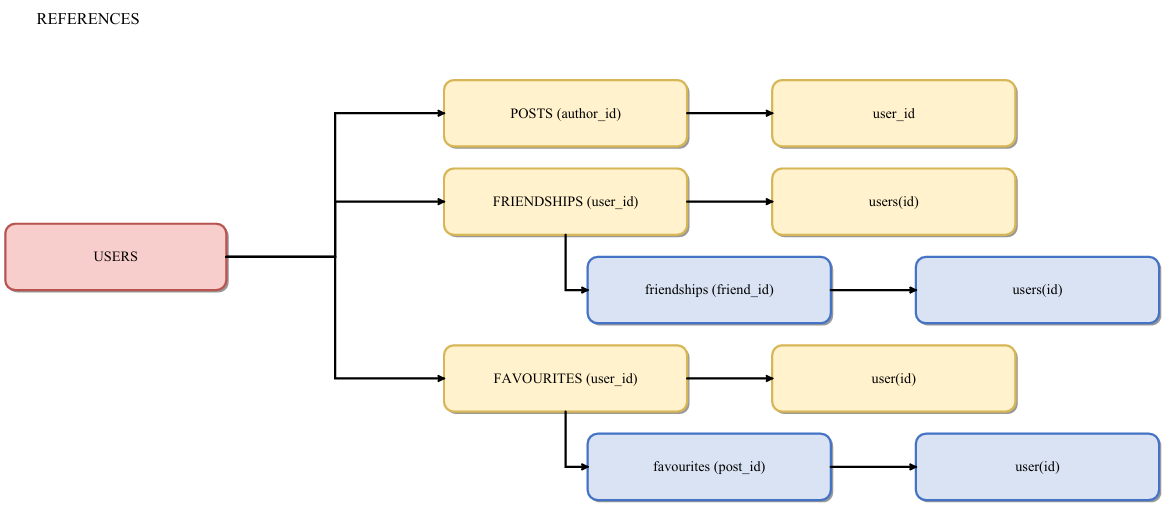
cигналы сетей мобильной связи;

IP-адреса.

В ответ на запрос Локатор API возвращает координаты и точность позиционирования в метрах.

## Схема Базы Данных и нефункциональные требования





**Почему схема выдержит нефункциональные требования:**

**Индексация**

Все ключевые поля для поиска и соединения таблиц индексируются. Это включает внешние ключи, такие как author\_id в таблице posts, и поля для фильтрации, такие как latitude и longitude.

Создан составной индекс для таблицы posts по полям latitude, longitude и created\_at. Это позволяет мгновенно находить все фотографии в заданном радиусе и сортировать их по новизне, что является основным сценарием приложения.

Индексы на полях user\_id в таблицах friendships, favorites и likes обеспечивают быстрый доступ к социальному графу и личным данным пользователя. Уникальные индексы на username и phone\_number гарантируют быструю проверку при регистрации и входе.

**Оптимизация под нагрузку**

Соотношение операций чтения и записи в приложении оценивается как 80% на чтение и 20% на запись. Схема баз данных оптимизирована для такого сценария. Высокая нормализация данных устраняет избыточность и минимизирует объем данных, участвующих в операциях чтения. Это ускоряет выполнение запросов и снижает нагрузку на подсистему ввода-вывода.

Критически важные данные, такие как файлы фотографий, могут быть полностью вынесены за пределы базы данных в объектное хранилище. Это предотвращает ситуацию, когда тяжелые операции с бинарными данными замедляют работу с легковесными метаданными, которые хранятся в PostgreSQL.

**Шардирование**

При значительном росте нагрузки и объема данных базу данных можно разделить на несколько шардов. Позволяет распределить нагрузку между несколькими серверами. Для данного приложения эффективным будет шардирование по географическому признаку, когда данные для разных регионов мира хранятся на разных шардах.

**Репликация**

Для обеспечения отказоустойчивости и повышения производительности база данных может быть реплицирована. Настраивается один главный сервер, который обрабатывает все операции записи, и несколько реплик, которые используются для операций чтения. В случае сбоя одна из реплик автоматически становится новым главным сервером, что минимизирует простой приложения.

## Схема масштабирования при нагрузке

**Сервера**

1. Применяется горизонтальное масштабирование серверов, увеличивая их количество в 10 раз

2. Применяется вертикальное масштабирование серверов, увеличивая объем их оперативной памяти и количество CPU

3. Внедряется Балансировщик нагрузки для равномерного распределения входящего трафика между всеми новыми узлами (серверами)

**База данных**

1. Репликация базы данных: Один сервер выступает как Master (Обрабатывает все операции записи - 20%), а несколько серверов как Slave (80% операций чтения)

2. Шардинг (партиционирование): Данные распределяются на несколько серверов с помощью шардинга, где каждый сервер (или шард) хранит только часть данных.

В реалиях нашего приложения больше подходит вариант с Репликацией, так как в действительности операции чтения/записи будут в соотношении 80/20

**Кэширование (Redis/Memcached)**

Кеширование применяется для снижения нагрузки на БД и ускорения ответа.

1. Внедрить кэширование часто запрашиваемых данных с использованием Redis или Memcached.

2. Shared Session: Для хранения состояния пользователей используется Shared session (общая сессия), при котором состояние хранится во внешнем кеше (Redis), а не на серверах приложений.

**Хранение фотографий**

Поскольку объем фотографий вырастет, необходимо использовать масштабируемое хранилище.

1. Shared Storage: Внедряется общее хранилище (Shared storage) в виде объектного хранилища (например, Yandex Object Storage) для всех загружаемых пользователями фотографий