Постановка MVP0

- Обязательно рассмотреть в функциональных требованиях базовую функциональность: лайк/дизлайк анкеты, хранение и редактирование профилей, матчинг пар, нотификации.
- Хранение и раздача медиаконтента (в анкетах пользователей много фотографий и иногда короткие видео)
- Желательно все-таки добавить в скоуп real-time чат между совпавшими парами и разобраться как правильно хранить переписку.

MVP0

- Профиль пользователя (создание/редактирование/удаление)
- Лента рекомендаций
- Система лайков
- Система сопоставлений (matches), перебора
- Чаты совместимых пользователей
- Обработка медиа фото-видео

Общая статистика

В сутках 24*3600 = 86400 секунд
± 100M MAU (Monthly active users) и ±10M DAU (Daily active users)

Значение	Параметр + Источник значения параметра	URL
100 млн	Из постановки MVP0 выше MAU	
10 млн (~10%)	Из постановки MVP0 выше DAU	
5 млн (~50%)	Количество пользователей в пиковое время (21:00-22:00)	https://www.huffingtonpost.co.uk/entry/how-to-get-tinder-matches_n_56a78f4be4b0172c659422da
25 минут	Среднее время использования сервиса в день	https://www.businessofapps.com/data/tinder-statistics/ и https://ogury.com/
0.4 млрд	Количество свайпов в день	https://www.businessofapps.com/data/tinder-statistics/
0.75 млрд	Пиковое количество свайпов за день	https://www.bbc.com/news/business-52743454

Значение	Параметр + Источник значения параметра	URL
3 млн	Количество мэтчей в день	https://www.businessofapps.com/data/tinder-statistics/
9.92	Среднее количество просматриваемых страниц	https://www.similarweb.com/website/tinder.com/
10 минут	Среднее время использования сервиса за 1 посещение	https://www.similarweb.com/website/tinder.com/#overview

Предположим, что 60% совместимых пользователей начинают общение в чате и пишут в среднем по 15 сообщений:

3 млн * 0.6 = 1.8 млн чатов/день

1.8 млн * 15 = 27 млн сообщений/день

Также предположим, что за день:

- для прочтения новых сообщений в чат заходят в среднем 5 раз;
- информацию в своем профиле изменяет 0.5% активных пользователей (60 тыс.);
- регистрируется 100 тыс. новых пользователей.

Сетевой трафик

При использовании сайта воспроизводилась прокрутка ленты и общение в чате.
Использование сайта 1 пользователем за 1 минуту:

Статика - 600 КБ; Динамика - 120 КБ; Изображения - 3 МБ.

Суточный трафик

При среднем времени использования сервиса в день равным 25 минутам и количестве пользователей, ежедневно пользующиеся сервисом, равным 10 млн:
Статика - 600 КБ * 25 * 10**6 ≈ 150 000 Гбайт/сутки

Динамика - 120 КБ * 25 * 10**6 ≈ 30 000 Гбайт/сутки

Изображения - 3 МБ * 25 * 10**6 ≈ 3 * 1024кб* 25 * 10**6 ≈ 768 000 Гбайт/сутки

Пиковое потребление

В пиковое время (21:00-22:00) количество пользователей составляет 6 млн. Пусть среднее время использования сервиса за этот час составляет 10 минут на пользователя и использование сервиса распределено равномерно. Тогда для 60 пользователей пик составит 10 одновременных сессий. Для 6 млн - 1 млн.
Статика - 600 КБ * 1 000 000/ 60 ≈ 76 Гбит/сек

Динамика - 120 КБ * 1 000 000 / 60 ≈ 15 Гбит/сек

Изображения - 3 МБ * 1 000 000/ 60 ≈ 391 Гбит/сек

RPS по типам запросов

Профили
Запросы на создание и изменение профиля:

(100000 + 60000) / (24 * 60 * 60) ≈ 2 RPS

Профиль пользователя должен подгружаться при каждом свайпе:

400*10**6 / (24 * 60 * 60) ≈ 4630 RPS

Главная страница сервиса

Тут самый подробный и негативный сценарий с учетом промежуточных служебных запросов:

Пусть пользователь заходит в приложение или стартовую страницу сервиса, он видит: статический контент, это - 1-2 запроса, стат картинки - например лого сервиса, 1 запрос. Это от cdn или подобного своего сервиса, например с балансера. Далее - видит превью своего фото - 1 запрос, свою анкету - 1 запрос, кол-во сообщений - пусть 1 запрос, кол-во запросов или входящих свайпов-лайков-чего-то там. -пусть 1 запрос, рандомное превью ленты, пусть с 3 рандомными участниками - 3 запроса на фото + 3 запроса анкеты. Итого ~ 13 запросов.

400*10**6 *13/ (24 * 3600) ≈ 60190 RPS

Далее рассматриваем без промежуточных запросов как выше, для абстракции на данных, которые непосредственно относятся к процессу.

Свайпы
Запросы на запись:

400*10**6 / (24 * 60 * 60) ≈ 4630 RPS

Мэтчи/Сопоставления
Запросы на запись:

3*10**6 / (24 * 60 * 60) ≈ 35 RPS

Получать список мэтчей необходимо один раз за один заход пользователя.
Поэтому запросы на получение, исходя из 12 млн активной аудитории:

12*10**6 / (24 * 60 * 60) ≈ 139 RPS
Рекомендации
При очередном свайпе необходимо выдать следующего рекомендованного человека. Рекомендательная система будет основана на свайпах и профилях. Сервис рекомендаций будет выдавать по несколько рекомендуемых людей за раз, чтобы заранее высчитывать рекомендации. Выдавать будет по 4 профиля, чтобы рекомендации были актуальны.
Запросы на получение:

400*10**6 / 4 / (24 * 60 * 60) ≈ 1157 RPS
Чаты
Запросы на запись:

2.1*10**6 / (24 * 60 * 60) ≈ 24 RPS

Получать список чатов также необходимо один раз за один заход пользователя.
Поэтому запросы на получение:

12*10**6 / (24 * 60 * 60) ≈ 139 RPS
Сообщения
Запросы на запись:

42*10**6 / (24 * 60 * 60) ≈ 486 RPS

Запросы на получение, исходя из среднего количества заходов в чаты:

12*10**6 * 5 / (24 * 60 * 60) ≈ 694 RPS
Фотографии
Фотографии подгружаются при каждом свайпе.
С учетом среднего количества фото у пользователя равного 3:

400*10**6 * 3 / (24 * 60 * 60) ≈ 13889 RPS

Итоговая таблица

Сущность	RPS
Профили пользователей	4632
Свайпы	4630
Мэтчи	174
Рекомендации	1157
Чаты	163
Сообщения	1180
Фотографии	13889

Сущность	RPS
Главная страница	60190

Хранение медиа-контента.

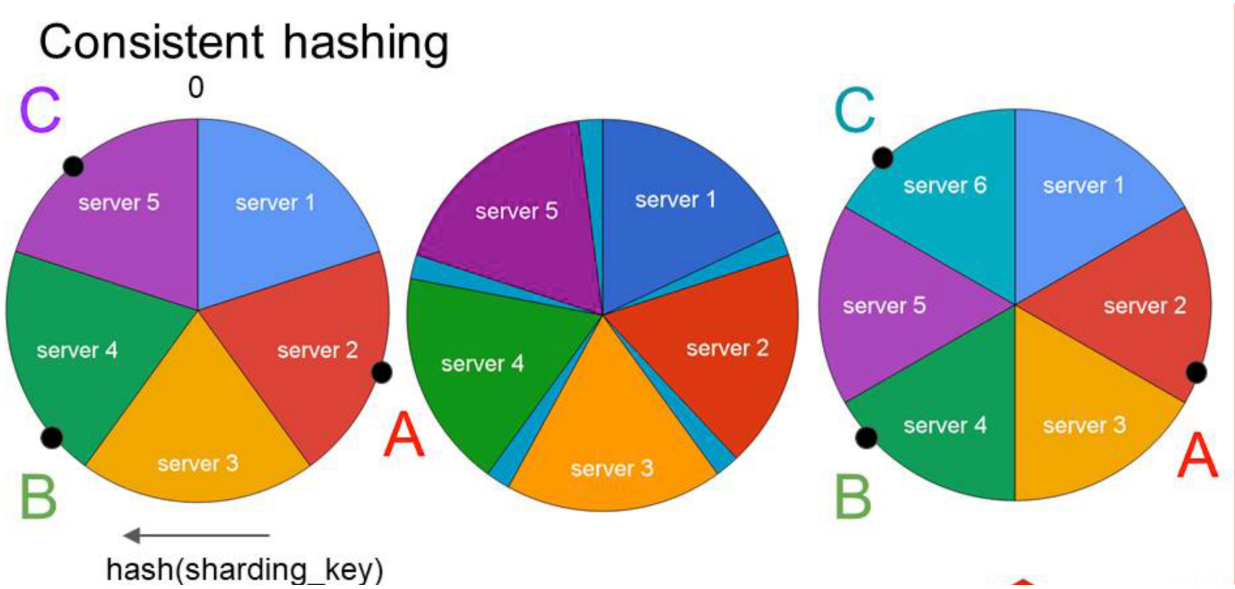
Рассмотрим на примере хранения фотографий. Для видео- аналогичный принцип.

Посмотрим на конкурентов https://habr.com/ru/companies/oleg-bunin/articles/340976/ возьмем некоторые уже используемые решения.

Логикой управляет модуль Media Service.
Фотографии хранятся в Storage Area Network.
Это большие SHD, которые ориентированы на хранение больших объемов данных. Они представляют собой полки с дисками, которые смонтированы на конечные отдающие машины по оптике. Т.о. мы имеем какой-то пул машин, достаточно небольшой, и эти SHD, которые прозрачны для нашей отдающей логики.
В теории можно использовать облако как альтернативу, но нужно прорабатывать вопрос со скоростью возврата фото/видео по запросу + возможные проблемы с доступностью.

У нашего сервиса специфика такая же, - человек регистрируется, заливает фотографию, далее начинает активно смотреть других людей, лайкать их, его активно показывают другим людям. Потом он находит себе пару или не находит пару, это уж как получится, и на какое-то время перестает пользоваться сервисом. В этот момент, когда он пользуется сервисом, его фотки очень горячие — они востребованы, их просматривает очень много людей. Как только он перестает это делать, достаточно быстро он выпадает из таких интенсивных показов другим людям, как были раньше, и его фотки практически не запрашиваются.

Т.е. у нас очень маленький горячий dataset. Но при этом за ним прямо очень много запросов. И решение тут - добавить кэш. Перед Storage Area Network добавляем слой mediascache. Это - кэширующий proxy. Кэширующий слой представляет собой машины с быстрыми локальными дисками (SSD) Далее идет слой хранения, Caches layer, на котором у нас расположены кластера из пар машин, которые друг друга резервируют, асинхронно копируются файлы с одного на другой при любом изменении.
Для распределения фотографий используется consistent hashing.



Кэш логически разделен на три слоя, это условно три директории в файловой системе:

1. Это буфер, куда попадают только что загруженные из proxy фотографии.
2. Это горячий кэш, в котором хранятся активно запрашиваемые сейчас фотографии.
3. И холодный кэш, куда постепенно фотографии выталкиваются из горячего, когда к ним приходит меньше request'ов.

Управление кэшем реализовано в модуле Media Service. Используется двойная запись – в RAMDisk access.log и БД Redis. Указывается относительный путь до фото, которую он сейчас обслужил и раздел где она была обслужена. + hash пути в Redis. Т.е. там может быть написано «photo 1» и дальше или буфер, или горячий кэш, или холодный кэш. Также выполняется сопоставление с id пользователя. Ведутся счетчики активно запрашиваемых фотографий, и они передвигаются в горячий кэш, где бы они ни лежали.

В среднем у пользователя в профиле находится 3 фотографии. Ограничим размер оригинального изображения до 1МБ. Остальные изображения ограничим следующими размерами:

Соотношение	Размер
640x800	100 КБ
320x400	40 КБ
172x216	15 КБ
84x106	5 КБ

Примерный вес всех фотографий:

(1 МБ + 100 КБ + 40 КБ + 15 КБ + 5 КБ) * 100*10**6 * 3 ≈ 357.5*10**6 МБ ≈ 358 ТБ

Выбор СУБД

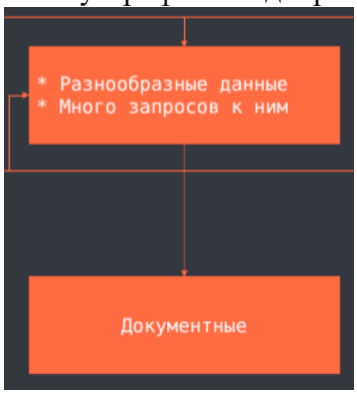
Примечание: *если сложно со знаниями-спецами по конкретному виду БД, можно реализовывать через РСУБД, тот же Posgree позже определяясь по ситуации с нагрузкой и необходимостью перехода на что-то специфичное nosql из описаний ниже.*

БД **Users** - Данные пользователей, информацию о фотографиях, чаты и сообщения будем хранить в PostgreSQL, как в наиболее надежной и функциональной реляционной БД. Кроме того РСУБД поддерживает ACID для обеспечения сохранности пользовательских данных.

Сессионные данные, свайпы, мэтчи, «последнюю» историю переписки **Message_last** будем хранить в хранилище типа key-value, например - Redis из-за его высокой скорости работы. Также Redis поддерживает неблокирующую master-slave репликацию. Кроме того, такие хранилища легко поддаются горизонтальному масштабированию, имеют низкую латентность обращения к данным, лучше чем РСУБД справляются с длинными последовательностями данных. «Последняя история переписки» - например глубиной в 2-3 мес.

Logs , Message, Notif - колоночная БД, так как для логов операции чтения более частые чем запись, + для постов пользователя может обеспечиваться доступность и устойчивость к разделению. Подходят для хранения шаблонов уведомлений.

Profile Профили или анкеты тоже храним в документарной бд – это сокращенная версия основного профиля, содержит сокращённый набор полей – см описание полей ниже. Так как в сервисе кол-во операций чтения больше, чем записи, пользователи чаще просматривают анкеты, ищут, свайпят. Поэтому сокращенные анкеты можно хранить в документарной бд, создаются они сразу с основным профилем. А если нужно обновить что-то в них, такая модель хранения позволяет делать update, если в процессе появятся просадки по производительности то можно удалить старую структуру полностью и создать новую – чтобы не нагружать операцией update колоночную БД. Кроме того, в анкету-профиль подобраны данные, которые изменяются относительно редко.



Типы и объемы данных

Примечание: Описаны основные модели данных для сокращения объема

Users – полная информация о пользователе

Поле	**Тип данных**

user_id	int
first_name	string(64)
last_name	string(64)
email	string(64)
biograph	string(512)
gender	string(1)
age	int
city	string(100)
education	string(128)
hobbies	string(512)

Размер записи одного пользователя

8 + 64 + 64 +64+ 512 + 1 + 8 + 100 + 128 + 512 = 1461 Б

Исходя из статистики в 100 млн зарегистрированных пользователей:

1461 Б * 100 *10**6 = 146,1 * 10**9 Б ≈ 146 ГБ

Profile – анкета, сокращенная информация о пользователе

Поле	**Тип данных**
user_id	int
first_name	string(64)
gender	string(1)
age	int
city	string(100)
education	string(128)

Размер записи одного пользователя

8 + 64 + 1 + 8 + 100 + 128+1 = 309 Б

Исходя из статистики в 100 млн зарегистрированных пользователей:

309 Б * 100 *10**6 = 30,9 * 10**9 Б ≈ 31 ГБ

Session – сессионные данные

Поле	**Тип данных**
session_id	int
user_id	int
value	string(64)
token	string(512)
expires	datetime

Размер записи одной сессии

8 + 8 + 64 + 512 + 8 = 600 Б

Исходя из статистики в 100 млн зарегистрированных пользователей:

600 Б * 100 *10**6 = 60 * 10**9 Б ≈ 60 ГБ

Swipes – свайпы

Поле	**Тип данных**
swip_id	int
user_src_id	int
user_trg_id	int
like	boolean

Размер таблицы для 1 свайпа:

8 + 8 + 8 + 1 = 25 Б

С учетом 0.4 млрд свайпов в день:

25 Б * 0.4* 10**9 * 30 = 10**9 Б * 30 ≈ 30 ГБ/мес

Matches – мэтчи, сопоставления

Поле	**Тип данных**
swip_id	int
user_src_id	int
user_trg_id	int

Размер таблицы для 1 мэтча:

8 + 8 + 8 = 24 Б

С учетом 3 млн мэтчей в день:

24 Б * 3*10**6 * 30 = 72*10**6 Б * 30 ≈ 279.4 ГБ/мес

Message – сообщения пользователей

Поле	**Тип данных**
message_id	int
chat_id	int
user_id_from	int
user_id_to	int
content	string(1024)
updated	datetime
created	datetime

Примем среднюю длину сообщения равную 30 Б.
Размер таблицы для 1 сообщения:

8 + 8 + 8 + 8 + 1024 + 8 + 8 = 1072 Б

С учетом 42 млн сообщений в день:

1072 Б * 42*10**6 * 30 = 1 331.9 Гб /мес

Chat – чаты пользователей

Поле	**Тип данных**
message_id	int
user_id_1	int
user_id_2	int

Размер записи для 1 чата:

8 + 8 + 8 = 24 Б

С учетом 2.1 млн чатов в день:

24 Б * 2.1*10**6 * 30 = 50.4*10**6 Б * 30 ≈ 1.2 ГБ/мес

Сервисы.

Примечание: *Для всех сервисов предусмотрено многократное резервирование – обозначено как сдвоенные фигуры, слои – layer, фигуры с двойной рамкой.*
В некоторых сервисах для снижения плотности компоновки не указана БД logs, но ее наличие подразумевается.

Firewalls - блок безопасности сетевого-прикладного уровней. Фильтрация пакетов, проверка содержимого – загружаемых фото-видео. Защита от DDOS, хотя могут использоваться и внешние провайдеры для защиты от DDOS

Auth Service – обеспечивает аутентификацию-авторизацию пользователей, контроль сессий.

Gateways layer – уровень «умных» шлюзов-маршрутизаторов. Обеспечивает back-end for front-end. На этом уровне реализуется первичная балансировка трафика – классификация трафика по источнику web-mob, по гео-зонам. На уровне балансировщиков, например nginx, реализована функция cdn возврата статического контента.
Gateways обеспечивают формирование полной информации для разделов клиентского интерфейса, выполняют оркестрацию обращений к сервисам. Например, по client_id из сессионных данных получают профиль, анкету клиента, мэтчи и сообщения, возвращая все это в соответствующий UI.
Между шлюзами предусмотрена очередь для постановки несрочных задач и снижения нагрузки на сервисы backend.

Balancer layer – второй уровень балансировки запросов от back-front уровня шлюзов к сервисам back-end. Представляет собой многократно резервированный пул балансировщиков распределяющий нагрузку на back-end. Может работать как реверс-перенаправляя запросы сервисов back между друг-другом.

Работоспособность сервисов определяется через взаимный ping тип heartbeat.

Notifications service – сервис уведомлений, обеспечивает маршрутизацию уведомлений по внешним сервисам. Содержит шаблоны уведомлений, очереди уведомлений и worker – сервисы, вычитывающий очереди и взаимодействующие с внешними сервисами. Notifications получает события из message-broker, например Kafka, и формирует уведомления по приоритетам срочности или выполняет постановку в очередь рассылки. Может использоваться и для уведомления о работе сервисов путем рассылки уведомлений по событиям модуля мониторинга. Может взаимодействовать с User Service для получения контактов рассылки.

ETL – mvp0 реализация сервиса глубокой аналитики данных сервисов. Содержит DWH-систему обработки данных, систему архивного хранения данных Storage Area Network. Также взаимодействует с модулем ML для построения аналитических моделей. Данные получает из брокера сообщений, накапливает-обрабатывает в DWH и вытесняет после обработки в Storage Area Network.

Message broker – брокер сообщений, обеспечивает асинхронную обработку событий сервисов back и данных в топиках. Например Kafka.

Monitoring – сервис мониторинга, собирает статистику и события по работе из топиков брокера и логов сервисов. Сохраняет, формирует отображение через ELK и события уведомлений.

Полный профиль пользователя

User service – реализует функции регистрации пользователя, хранения полных пользовательских данных, структура описана выше.
Также сервис реализует сохранение гео-данных пользователя, если от него было согласие на передачу геоданных из приложения или сайта. Сохраняется в БД для геоданных, например в PosGIS
В сервисе выполняется ведение черных списков пользователей, на основе дизлайков, жалоб, отметок в интерфейсе.

Анкеты пользователей.

User profile – реализует логику формирования данных для анкет пользователей. Является источником данных для сервисов мэтчинга, свайпа, поиска.
Сервис является источником «быстрых» анкет из документарной БД, рассмотренной выше. В сервисе предусмотрен кэш «горячих» анкет, которые запрашиваются наиболее часто. Ведется счетчик кол-ва запросов. Как только уровень запросов к анкете или активность пользователя в поиске, рейтинг падает – анкета удаляется из кэша. Активность пользователя можно узнавать из топика брокера сообщений (например Kafka) или из API сервиса поиска и сервиса рейтинга через балансировщик.
Если пользователь меняет данные анкеты, можно удалять существующую и создавать новую с изменениями, см выше в описании структуры данных.

Поиск пользователей

Search service – реализует поиск. Для поиска по анкетам пользователей строятся индексы наиболее популярных запросов вида «отобрази всех объектов пола-возраста с

интересами» или «все объекты пола-возраста учившиеся в» или «все объекты пола-возраста живущие в». При этом можно ввести ограничение на совсем общие запросы типа – отобрази всех мужчин или всех женщин, чтобы избежать парсинг, перебор базы. Также через Customer Dev можно исследовать аудиторию, анализировать логи поиска, выбирая в индекс популярные запросы.
Сервис поиска возвращает user_id, далее gateway выполняет запрос анкеты, пользователь либо смотрит, либо выполняет запрос на контакт – мэтчинг. В случае взаимного мэтчинга возвращаются данные из полного профиля сервиса User.
Для поиска близких по интересам пользователей можно использовать QuadTree

Для хранения гео-данных пользователя используется специализированная БД для хранения геоданных, например PosGIS. В PostGIS для хранения точек есть отдельный тип Point, а также два базовых типа Geometry и Geography.

Сервис Geohash – используется для поиска точки на карте, или набора каких-то точек. Использует одноименный алгоритм geohash чтобы оптимально разбить карту на отдельные квадраты, каждому из которых задавать адрес. При этом большие квадраты разбиваются на более мелкие с постепенным увеличением адреса последних. Итоговый адрес ячейки записывается в виде строки. Чтобы найти точки поблизости, достаточно изучить ячейки с общим префиксом. При этом для безопасности пользователей точность размазана до нескольких сотен метров, чтобы избежать компрометации пользователей.

Сервис рекомендаций.

Работает на основе ML/DS моделей, например классификации-кластеризации по региону, интересам, возрастному диапазону. Также ориентируется на информацию о поиске, мэтчинге, лайках пользователя, может использовать что-то типа
https://github.com/jeffmli/TinderAutomation
Через сервис User Profile получает данные о активных анкетах из «горячего» кэша, из сервиса Rating – лайки пользователей, из сервиса Users - информацию о черных списках.

Сервис мэтчинга и свайпа.

Они объединены в один сервис, так как схожая модель данных и схожий принцип обработки этих данных.
Очереди используются для обработки мэтчинга и свайпа, например один пользователь свайпнул, другой стал неактивным, вышел из приложения – событие помещается в очередь, также формируется событие в брокер для сервиса уведомлений.
Активные свайпы хранятся в key-value бд, история свайпов и мэтчингов – в колоночной бд.

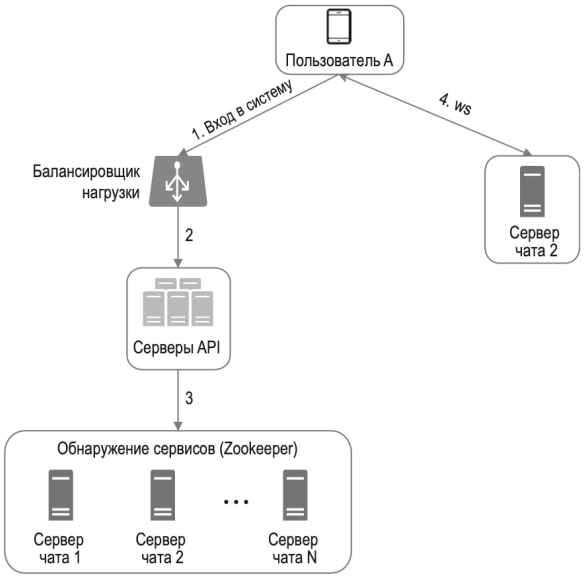
Сервис рейтинга (лайков).

Сервис реализует учет отметок «лаков» на профилях пользователей и фото. Сохраняет в key-value БД, формируется событие в брокер для сервиса уведомлений. Событие «лафк» получает от сервисов в которых есть соответствующий элемент в UI, например из фото, видео, анкеты – получает асинхронно через топик брокера либо если будет нужно синхронное отображение и обработка, то через настройку маршрута в слое балансировщиков.

Обработка сообщений.

Message – сервис реализует логику обработки сообщений, работу чата. Реализует механизм обнаружения сервисов чата, предлагая пользователю лучший для него в текущий момент сервис с учетом нагрузки, географического положения, емкости сервера. Может использовать решение Apache Zookeeper. Реализует маршрутизацию сообщений по чатам.
Websocket-chat-manager – обеспечивает взаимодействие по протоколу WebSocket. Совместно с Message выполняет синхронизацию чатов на устройствах пользователя.
Online status checker – проверяет статус пользователя через отправку «пульса» запроса-состояния клиентского подключения.
Для обновления информации по статусам, состоянию чатов, очередности сообщений используется очередь.

Пример логики работы



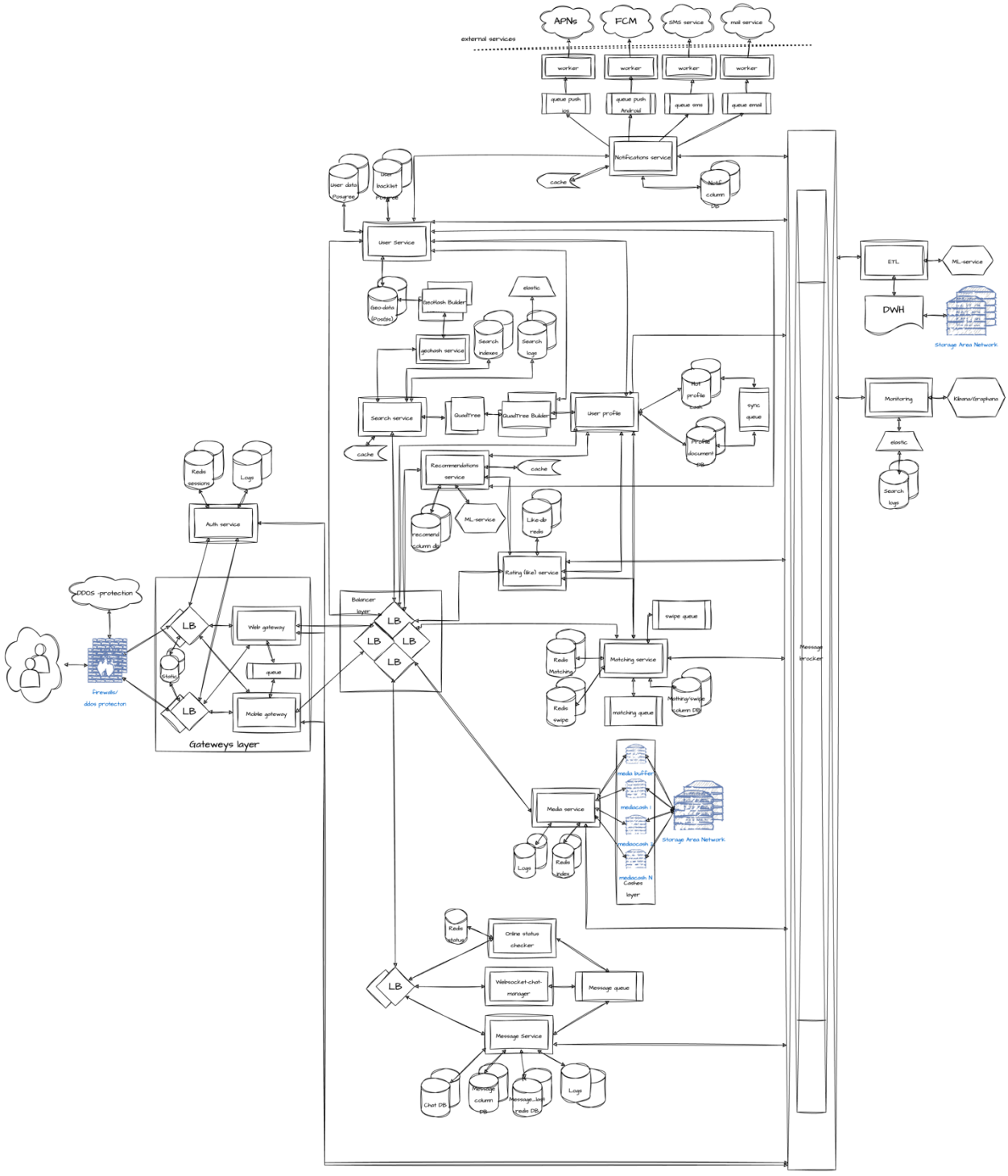
1. Пользователь А отправляет мгновенное сообщение на сервер чата 1.
2. Сервер чата1 получает ID сообщения из генератора идентификаторов.
3. Сервер чата 1 передает сообщение в очередь синхронизации сообщений.
4. Сообщение записывается в хранилище типа «ключ–значение».

Стэк сервисов

Нагруженные сервисы back-end разрабатываются на Golang. MVP сервисов, модели данных, обработка логов – на Python
Протоколы взаимодействий grpc, rest.

Front-end: JS/React, моб клиенты на соответствующем стэке типа Swift/Kotlin.

Общая схема сервиса


APNs
FCM
SMS service
mail service
external services
worker
worker
worker
worker
queue push ios
queue push Android
queue sms
queue email
Notifications service
cache
User Service
Search service
cache
GuadTree
GuadTree Builder
User profile
Hot profile cash
sync queue
Recommendations service
cache
ML-service
Like-db redis
Rating (like) service
Auth service
Redis sessions
Logs
DDOS -protection
Firewall/ ddos protecton
LB
Web gateway
queue
Mobile gateway
static
Gateweys layer
Balancer layer
swipe queue
Redis Matching
Redis swipe
Matching service
matching queue
Message broocker
Media service
Redis
Logs
Storage Area Network
Online status checker
Redis status
Websocket-chat-manager
Message queue
Message Service
Chat DB
Logs
ETL
ML-service
DWH
Storage Area Network
Monitoring
Kibana/Graphana
elastic
Search logs