Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №1
по дисциплине «Экономика программной инженерии»

Вариант:
https://supabase.com

Преподаватель:
Блохина Елена Николаевна

Выполнил:
Карташев Владимир
Группа: P3415

Санкт-Петербург, 2025 г.

Оглавление

Текст задания

Для выданного веб-проекта: https://supabase.com

1. Сформировать набор функциональных требований для разработки проекта.
2. Оценить трудоемкость разработки проекта наивным методом.
3. Оценить трудоемкость разработки проекта методом PERT (Project Evaluation and Review Technique**).** Нарисовать сетевую диаграмму взаимосвязи работ и методом критического пути рассчитать минимальную продолжительность разработки. Предложить оптимальное количество разработчиков и оценить срок выполнения проекта.
4. Оценить размер проекта методом функциональных точек, затем, исходя из предположения, что собранной статистики по завершенным проектам нет, рассчитать трудоемкость методом COCOMO II (Обновленная таблица количества строк на точку для разных языков программирования)
5. Оценить размер проекта методом оценки вариантов использования (Use Case Points). Для расчета фактора продуктивности PF использовать любой свой завершенный проект с известными временными трудозатратами, оценив его размер методом UCP.
6. Сравнить полученные результаты и сделать выводы.

ВВЕДЕНИЕ

Данный документ описывает функциональные требования к системе **SUPABASE** и предоставляет анализ трудоемкости разработки наивным методом и методами PERT (Project Evaluation and Review), COCOMO II, UCP (Use Case Points)

SUPABASE представляет собой открытую *Backend-as-a-Service* платформу на базе *PostgreSQL* с модульной архитектурой из независимых сервисов, объединённых через API Gateway. Для создания минимально жизнеспособной версии критически необходимы четыре компонента: *PostgreSQL* с расширениями, *PostgREST* для автогенерации *API*, *GoTrue* для аутентификации и *Kong* в качестве шлюза, обеспечивающего маршрутизацию и валидацию *JWT токенов*.

ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

EPIC-1: Базовая инфраструктура

TASK-1.1: Необходимо создать Docker Compose конфигурацию для всех сервисов платформы (network, volumes, …) с поддержкой health checks

TASK-1.2: Необходимо создать систему управления переменными окружения через .env файлы и реализовать DNS-резолвинг по именам контейнеров.

TASK-1.3: Необходимо реализовать механизм управления секретами через конфигурационный PaaS Consul

TASK-1.4: Необходимо настроить централизованное логирование через Docker logging driver

TASK-1.5: Необходимо разрешить внешний доступ на серверной машине только к API Gateway Kong

TASK-1.6: Необходимо написать Markdown документацию по развертыванию инфраструктуры, создать скрипт локальной автоматической сборки с поддержкой UNIX (Bash, WSL) и Windows (PowerShell)

EPIC-2: PostgresSQL Database

TASK-2.1: Необходимо развернуть PostgreSQL версии 18+ в Docker контейнере, установить параметры производительности PostgreSQL (shared_buffers, …)

TASK-2.2: Необходимо настроить административные привилегии с полным доступом

TASK-2.3: Необходимо создать схемы: auth, storage, public с правильными permissions

TASK-2.4: Необходимо включить Row Level Security для базы данных

TASK-2.5: Необходимо настроить логическую репликацию с wal_level=logical

TASK-2.6: Необходимо создать publication для отслеживаемых таблиц

EPIC-3: API Gateway (Kong)

TASK-3.1: Необходимо развернуть Kong Gateway Docker image с единой точкой входа на порту 8000 и создать декларативную YAML-конфигурацию для Kong

TASK-3.2: Необходимо настроить маршрутизацию:

/rest/v1/*	для	PostgREST
/auth/v1/*	для	GoTrue
/storage/v1/*	для	Storage API
/realtime/v1/*	для	Realtime Server
/functions/v1/*	для	Edge Functions

TASK-3.3: Необходимо установить JWT plugin с алгоритмом HMAC-SHA256

TASK-3.4: Необходимо установить ACL plugin для проверки ролей (anon, authenticated, service_role)

TASK-3.5: Необходимо настроить установку заголовков для бэкенд-сервисов из JWT claims

TASK-3.6: Необходимо установить CORS plugin с настройкой политик доступа

TASK-3.7: Необходимо настроить rate limiting plugin для защиты от DDoS

TASK-3.8: Необходимо создать health check endpoint для мониторинга бэкенд-сервисов по конечным точкам

EPIC-4: Authentication Service (GoTrue)

TASK-4.1: Необходимо развернуть GoTrue Docker image

TASK-4.2: Необходимо создать таблицу auth.users в PostgreSQL

TASK-4.3: Необходимо реализовать регистрацию через email и password с Argon2 хешированием

TASK-4.4: Необходимо реализовать magic link для верификации email и настроить отправку confirmation email через SMTP с помощью почтового сервиса

TASK-4.5: Необходимо реализовать механизм генерации JWT токенов с claims (role, user_id, exp, aud) с использованием HMAC-SHA256 алгоритма для подпиши JWT

TASK-4.6: Необходимо реализовать refresh токены со сроком жизни 30 дней и их обработку через отдельную конечную точку с query параметром ?grant_type=refresh_token

TASK-4.7: Необходимо реализовать passwordless аутентификацию через magic links

TASK-4.8: Необходимо интегрировать OAuth 2.0 с провайдером Google

TASK-4.9: Необходимо интегрировать OAuth 2.0 с провайдером GitHub

TASK-4.10: Необходимо создать database trigger для автоматического создания записи в public.profiles с обработкой случая незавершенной операции регистрации

EPIC-5: REST API Generation

TASK-5.1: Необходимо развернуть PostgREST Docker image

TASK-5.2: Необходимо настроить автоматическую рефлексию схемы PostgreSQL

TASK-5.3: Необходимо реализовать генерацию REST конечных точек для всех публичных таблиц со следующими правилами:

GET метод для чтения данных

POST метод для создания записей

PATCH метод для обновления записей

DELETE метод для удаления записей

Также необходимо создать маршруты в формате /rest/v1/{table_name}

TASK-5.4: Необходимо реализовать извлечение роли, email и user_id из JWT токена

TASK-5.5: Необходимо реализовать SET LOCAL role перед каждым запросом и установку request.jwt.claims через SET LOCAL

TASK-5.6: Необходимо реализовать автоматическое применение Row Level Security политик

TASK-5.7: Необходимо реализовать query параметры для фильтрации, пагинации и осуществления комбинированных условий (логические И и ИЛИ)

TASK-5.8: Необходимо реализовать возврат результатов в JSON формате

EPIC-6: Row Level Security

TASK-6.1: Необходимо включить и настроить Row Level Security для всех таблиц с пользовательскими данными

TASK-6.2: Необходимо создать политики для операций:

USING политику для INSERT операций

WITH CHECK политику для INSERT, UPDATE, DELETE операций

TASK-6.3: Необходимо создать политики для ролей: anon (публичный доступ), authenticated (аутентифицированные пользователи), service_role (администрирование без ограничений)

TASK-6.4: Необходимо документировать шаблоны политик:

USING (auth.uid() = user_id)

WITH CHECK (auth.uid() = user_id)

USING (true) для anon

EPIC-7: JavaScript SDK

TASK-7.1: Необходимо создать npm пакет @supabase/supabase-js

TASK-7.2: Необходимо реализовать функцию createClient(url, anonKey)

TASK-7.3: Необходимо создать модуль auth в клиенте

TASK-7.4: Необходимо реализовать методы авторизации:

auth.signUp({email, password, options})

auth.signInWithPassword({email, password})

auth.signInWithOAuth({provider})

auth.signOut()

TASK-7.5: Необходимо реализовать автоматическое сохранение JWT в localStorage и автоматическое обновление JWT через refresh token

TASK-7.6: Необходимо реализовать метод auth.getUser() для получения текущего пользователя и метод auth.getSession() для получения текущей сессии

TASK-7.7: Необходимо создать модуль from() для работы с данными

TASK-7.8: Необходимо реализовать query builder с методом select()

TASK-7.9: Необходимо реализовать методы работы с записями:

insert() для создания записей

update() для обновления записей

delete() для удаления записей

eq(), neq(), gt(), gte(), lt(), lte(), like(), ilike(), in() для фильтрации

order() для сортировки

range(from, to) для пагинации

single() для получения одной записи

maybeSingle() для опциональной записи

TASK-7.10: Необходимо реализовать автоматическое добавление Authorization header

TASK-7.11: Необходимо добавить TypeScript типизацию для всех методов

TASK-7.12: Необходимо реализовать обработку ошибок в формате {data, error}

TASK-7.13: Необходимо реализовать retry logic

EPIC-8: Storage API

TASK-8.1: Необходимо создать Storage API сервис на Java 25 Spring Framework

TASK-8.2: Необходимо интегрировать S3-совместимое хранилище (AWS SDK)

TASK-8.3: Необходимо развернуть MinIO для локальной разработки

TASK-8.4: Необходимо создать в PostgreSQL таблицу

file_info {

id,

status (UPLOADING, UPLOADED, UPDATED, DELETED)

type (будущий bucket),

path (filename внутри bicket),

original_name

}

TASK-8.5: Необходимо создать CRUD REST API операции для работы с S3 хранилищем и file_info таблицей внутри PostgreSQL используя Hibernate ORM фреймворк.

Политика изменения объекта в S3 хранилище и а запиши file_info:

Перед началом загрузки должна создаваться запись в таблице file_info с status=UPLOADING и id сущности file_info браться как fileName в S3 с добавлением разрешения файла (“.png”);

По завершении загрузки должен меняться статус сущности file_info на UPLOADED;

При редактировании содержимого файла необходимо обновлять файл в S3 хранилище и устанавливать status=UPLOADED в сущности file_info;

При удалении файла необходимо менять status=DELETED в сущности file_info и удалять его файл из S3 хранилища.

Все операции должны производиться в режиме транзакции кроме операции добавления.

TASK-8.6: Необходимо реализовать генерацию signed URLs с настраиваемым expiresIn

EPIC-9: Storage SDK

TASK-9.1: Необходимо создать модуль storage в JavaScript SDK для работы с файлами

TASK-9.2: Необходимо реализовать метод upload(path, file, options) с поддержкой File и Blob объектов

TASK-9.3: Необходимо реализовать автоматическое определение content-type через MIME type

TASK-9.4: Необходимо реализовать параметр upsert для перезаписи существующих файлов

TASK-9.5: Необходимо реализовать upload progress через callback onUploadProgress с процентами загрузки

TASK-9.6: Необходимо реализовать методы:

download(path) для скачивания файлов с возвратом Blob

remove(paths) для удаления одного или массива файлов

list(path, options) для листинга файлов с параметрами limit и offset для пагинации

getPublicUrl(path) для получения постоянного URL к файлу

createSignedUrl(path, expiresIn) для генерации временных подписанных URLs

TASK-9.7: Необходимо реализовать автоматическое добавление JWT токена в Authorization header для всех запросов

TASK-9.8: Необходимо добавить TypeScript типизацию для Storage модуля с интерфейсами для options и responses

TASK-9.9: Необходимо реализовать обработку ошибок прав доступа с понятными сообщениями

TASK-9.10: Необходимо реализовать retry logic для failed uploads с exponential backoff

EPIC-10: Realtime Server

TASK-10.1: Необходимо создать Realtime сервер на Elixir/Phoenix Framework для WebSocket соединений

TASK-10.2: Необходимо подключиться к PostgreSQL через логическую репликацию используя replication protocol

TASK-10.3: Необходимо создать replication slot с именем supabase_realtime для чтения WAL

TASK-10.4: Необходимо установить wal2json extension в PostgreSQL для парсинга WAL записей в JSON

TASK-10.5: Необходимо реализовать слушание изменений в WAL через replication connection в режиме streaming

TASK-10.6: Необходимо реализовать парсинг WAL записей в формат {table, operation, old_record, new_record, timestamp}

TASK-10.7: Необходимо реализовать применение RLS политик для фильтрации подписчиков с проверкой прав на уровне строк

TASK-10.8: Необходимо создать функцию для проверки прав доступа подписчика к конкретной строке через SELECT RLS политику

TASK-10.9: Необходимо реализовать отправку событий только авторизованным подписчикам на основе их JWT токенов

TASK-10.10: Необходимо создать WebSocket endpoint /realtime/v1/websocket с поддержкой wss:// протокола

TASK-10.11: Необходимо реализовать обработку сообщений: phx_join для подписки, phx_leave для отписки, phx_reply для подтверждений

TASK-10.12: Необходимо реализовать валидацию JWT токена из query параметра apikey при установке WebSocket соединения

TASK-10.13: Необходимо реализовать подписку через channel с именем postgres_changes:{schema}:{table}

TASK-10.14: Необходимо реализовать отправку событий с типами: INSERT, UPDATE, DELETE с полными данными

TASK-10.15: Необходимо включать old (предыдущие значения) и new (новые значения) в payload события

TASK-10.16: Необходимо реализовать graceful shutdown с уведомлением всех подключенных клиентов

EPIC-11: Realtime SDK

TASK-11.1: Необходимо создать модуль channel() в JavaScript SDK для управления real-time подписками

TASK-11.2: Необходимо реализовать установку WebSocket соединения к /realtime/v1/websocket с автоматическим upgrade с HTTP

TASK-11.3: Необходимо реализовать передачу JWT токена через apikey query параметр в WebSocket URL

TASK-11.4: Необходимо реализовать методы:

channel(topic) для создания канала с указанием schema:table

on('postgres_changes', options, callback) для подписки на изменения

TASK-11.5: Необходимо реализовать фильтры по:

event: INSERT, UPDATE, DELETE, * (все события)

table через параметр table в options

schema через параметр schema в options (по умолчанию public)

метод subscribe() для активации подписки с возвратом Promise

TASK-11.6: Необходимо реализовать метод unsubscribe() для отмены подписки с закрытием канала

TASK-11.7: Необходимо реализовать a1w 3d 4h

ОЦЕНКА ТРУДОЕМКОСТИ РАЗРАБОТКИ НАИВНЫМ МЕТОДОМ

№	Название	Описание	Оценка мин. (часы)	Оценка макс. (часы)
1	**Подготовка**			
1.1	Требования	Сбор, анализ, уточнение требований, определение критериев приемки, изучение архитектуры Supabase	40	80
1.2	Архитектура	Выбор технологического стека, проектирование микросервисной архитектуры, составление контрактов API и схем данных	60	120
1.3	Онбординг	Погружение разработчиков в контекст платформы, знакомство с требованиями, архитектурой, изучение PostgreSQL RLS и логической репликации	24	40
1.4	Окружение	Создание страниц документации, высокоуровневых эпиков в трекере, репозиториев, настройка базовой инфраструктуры, CI/CD, выдача доступов	40	80
2	**Разработка Core компонентов**			
2.1	EPIC-1: Базовая инфраструктура			
2.1.1	Docker Compose конфигурация	Создание docker-compose.yml с network, volumes, health checks для всех сервисов	16	24
2.1.2	Управление окружением	Настройка .env файлов, DNS-резолвинг, интеграция с Consul для секретов	12	20
2.1.3	Логирование	Настройка централизованного логирования через Docker logging driver	8	12
2.1.4	Документация и скрипты	Написание Markdown документации, создание Bash и PowerShell скриптов для автоматической сборки	12	20
2.2	EPIC-2: PostgreSQL Database			
2.2.1	Развертывание PostgreSQL 17+	Настройка Docker контейнера, параметров производительности (shared_buffers, work_mem, etc)	8	16
2.2.2	Настройка схем и permissions	Создание схем auth, storage, public с правильными правами доступа	8	12
2.2.3	Row Level Security	Включение RLS, создание базовых функций для работы с JWT claims	12	20

2.2.4	Логическая репликация	Настройка wal_level=logical, создание publication для отслеживаемых таблиц	8	16
2.3	EPIC-3: API Gateway (Kong)			
2.3.1	Развертывание Kong	Настройка Kong Gateway в Docker, создание декларативной YAML конфигурации	12	20
2.3.2	Настройка маршрутизации	Конфигурация маршрутов для всех сервисов (/rest/v1/*, /auth/v1/*, /storage/v1/*, /realtime/v1/*, /functions/v1/*)	16	24
2.3.3	JWT plugin	Установка и настройка JWT plugin с HMAC-SHA256, валидация токенов	12	16
2.3.4	ACL и CORS plugins	Настройка ACL для ролей (anon, authenticated, service_role), CORS политик	8	12
2.3.5	Rate limiting и health checks	Настройка rate limiting plugin, создание health check endpoints	8	12
2.4	EPIC-4: Authentication Service (GoTrue)			
2.4.1	Развертывание GoTrue	Настройка GoTrue Docker image, создание таблицы auth.users	8	12
2.4.2	Email/Password регистрация	Реализация регистрации с Argon2 хешированием, создание REST endpoints	16	24
2.4.3	Email верификация	Реализация magic links, настройка SMTP, отправка confirmation emails	20	32
2.4.4	JWT токены	Механизм генерации JWT с claims (role, user_id, exp, aud), HMAC-SHA256	12	20
2.4.5	Refresh токены	Реализация refresh токенов со сроком 30 дней, endpoint с grant_type=refresh_token	12	16
2.4.6	OAuth интеграция	Интеграция OAuth 2.0 с Google и GitHub провайдерами	24	40
2.4.7	Database triggers	Создание trigger для автоматического создания public.profiles, обработка ошибок	8	12
2.5	EPIC-5: REST API Generation (PostgREST)			
2.5.1	Развертывание PostgREST	Настройка PostgREST Docker image, автоматическая рефлексия схемы PostgreSQL	8	12
2.5.2	REST endpoints генерация	Генерация endpoints для всех таблиц с GET, POST, PATCH, DELETE методами	12	20

2.5.3	JWT интеграция	Извлечение роли, email, user_id из JWT, SET LOCAL role и request.jwt.claims	12	20
2.5.4	RLS применение	Автоматическое применение Row Level Security политик к запросам	8	16
2.5.5	Query параметры	Реализация фильтрации, пагинации, комбинированных условий (AND/OR)	16	24
2.5.6	JSON сериализация	Настройка возврата результатов в JSON формате	4	8
2.6	EPIC-6: Row Level Security			
2.6.1	Включение RLS	Включение Row Level Security для всех пользовательских таблиц	4	8
2.6.2	Политики для операций	Создание USING политик для SELECT, WITH CHECK для INSERT/UPDATE/DELETE	16	24
2.6.3	Политики для ролей	Создание политик для anon, authenticated, service_role ролей	12	20
2.6.4	Документация политик	Документирование шаблонов политик с примерами использования	8	12
2.6.5	Тестирование RLS	Создание тестовых сценариев для проверки работы политик	12	20
2.7	EPIC-7: JavaScript SDK			
2.7.1	Создание npm пакета	Инициализация @supabase/supabase-js, настройка build системы	8	12
2.7.2	Client инициализация	Реализация createClient(url, anonKey), базовая архитектура клиента	8	12
2.7.3	Auth модуль	Создание auth модуля с методами signUp, signInWithPassword, signInWithOAuth, signOut	20	32
2.7.4	JWT управление	Автоматическое сохранение JWT в localStorage, обновление через refresh token	12	20
2.7.5	Session методы	Реализация getUser() и getSession() для получения текущего пользователя	8	12
2.7.6	Database модуль	Создание from() модуля для работы с данными	8	12
2.7.7	Query builder	Реализация query builder с select(), insert(), update(), delete()	24	40
2.7.8	Фильтрация и операции	Реализация eq(), neq(), gt(), gte(), lt(), lte(), like(), ilike(), in()	16	24

2.7.9	Дополнительные методы	Реализация order(), range(), single(), maybeSingle()	12	16
2.7.10	TypeScript типизация	Добавление TypeScript типов для всех методов и интерфейсов	16	24
2.7.11	Error handling	Реализация обработки ошибок в формате {data, error}	16	24
2.7.12	Retry logic	Реализация retry logic с exponential backoff	8	12
3	**Разработка Extended компонентов**			
3.1	EPIC-8: Storage API			
3.1.1	Spring Boot проект	Создание Storage API сервиса на Java 21 Spring Framework	12	20
3.1.2	S3 интеграция	Интеграция AWS SDK для работы с S3-совместимым хранилищем	16	24
3.1.3	MinIO развертывание	Настройка MinIO для локальной разработки	8	12
3.1.4	Database схема	Создание таблицы file_info с полями id, status, type, path, original_name	8	12
3.1.5	CRUD операции	Реализация REST API для CRUD операций с использованием Hibernate ORM	24	40
3.1.6	Транзакционная логика	Реализация политики изменения статусов (UPLOADING, UPLOADED, UPDATED, DELETED)	20	32
3.1.7	Signed URLs	Реализация генерации signed URLs с настраиваемым expiresIn	12	16
3.1.8	Error handling	Обработка ошибок загрузки, удаления, транзакционных откатов	12	16
3.2	EPIC-9: Storage SDK			
3.2.1	Storage модуль	Создание модуля storage в JavaScript SDK	8	12
3.2.2	Upload метод	Реализация upload(path, file, options) с File и Blob поддержкой	16	24
3.2.3	Content-type определение	Автоматическое определение MIME type, параметр upsert	8	12
3.2.4	Upload progress	Реализация onUploadProgress callback с процентами загрузки	12	16
3.2.5	Download и remove	Реализация download(path) и remove(paths) методов	12	16
3.2.6	List и URL методы	Реализация list(), getPublicUrl(), createSignedUrl()	16	24
3.2.7	TypeScript типизация	Добавление типов для Storage	8	12

		модуля с интерфейсами		
3.2.8	Error handling и retry	Обработка ошибок прав доступа, retry logic для failed uploads	12	16
3.3	EPIC-10: Realtime Server			
3.3.1	Elixir/Phoenix проект	Создание Realtime сервера на Elixir/Phoenix Framework	24	40
3.3.2	Логическая репликация	Подключение к PostgreSQL через replication protocol	20	32
3.3.3	Replication slot	Создание replication slot supabase_realtime, установка wal2json	12	20
3.3.4	WAL streaming	Реализация слушания изменений в WAL через streaming connection	24	40
3.3.5	WAL парсинг	Парсинг WAL записей в формат {table, operation, old_record, new_record, timestamp}	20	32
3.3.6	RLS фильтрация	Применение RLS политик для фильтрации подписчиков	24	40
3.3.7	Права доступа	Создание функции проверки прав доступа к строкам через SELECT RLS	16	24
3.3.8	WebSocket endpoint	Создание /realtime/v1/websocket endpoint с wss:// протоколом	12	20
3.3.9	Phoenix Channels	Реализация Phoenix Channels protocol (phx_join, phx_leave, phx_reply)	16	24
3.3.10	JWT валидация	Валидация JWT из query параметра apikey	8	12
3.3.11	Channel подписки	Реализация подписок через postgres_changes:{schema}:{table}	12	20
3.3.12	Event broadcasting	Отправка событий INSERT, UPDATE, DELETE с old/new значениями	16	24
3.3.13	Graceful shutdown	Реализация graceful shutdown с уведомлением клиентов	8	12
3.4	EPIC-11: Realtime SDK			
3.4.1	Channel модуль	Создание модуля channel() в JavaScript SDK	8	12
3.4.2	WebSocket соединение	Установка WebSocket соединения с автоматическим HTTP upgrade	12	20
3.4.3	JWT передача	Передача JWT через apikey query параметр	4	8
3.4.4	Channel методы	Реализация channel(topic) и	16	24

		on('postgres_changes', options, callback)		
3.4.5	Фильтры	Реализация фильтров по event, table, schema	12	16
3.4.6	Subscribe/Unsubscribe	Реализация subscribe() и unsubscribe() с Promise	12	16
3.4.7	Auto reconnection	Автоматическое переподключение с exponential backoff	16	24
3.4.8	Подписки восстановление	Сохранение и восстановление подписок после reconnect	12	16
3.4.9	State management	Управление состоянием: CONNECTING, OPEN, CLOSING, CLOSED	8	12
3.4.10	TypeScript типизация	Добавление generic типов для old и new данных	8	12
3.4.11	Дополнительные методы	Реализация removeAllChannels() и error handling с retry	8	12
4	**Тестирование**			
4.1	Unit тестирование	Unit тесты для всех модулей SDK, сервисов, утилит	60	100
4.2	Интеграционное тестирование	Интеграционные тесты компонентов системы, Auth flow, CRUD операций, Storage, Realtime	80	120
4.3	E2E тестирование	End-to-end тестирование полных сценариев через Playwright/Cypress	60	100
4.4	Performance тестирование	Load testing через k6, определение пределов производительности	40	60
4.5	Security тестирование	Security audit, SQL injection защита, JWT hijacking, RLS bypass attempts	40	80
4.6	Исправление багов	Выявление и исправление критических багов после тестирования	80	120
5	**Документация**			
5.1	Getting Started guide	Пошаговое руководство по началу работы, установке, первым шагам	16	24
5.2	API Reference	Полная документация всех API endpoints и SDK методов	40	60
5.3	Authentication Guide	Подробное руководство по аутентификации с примерами	20	32
5.4	Database Guide	Руководство по работе с базой данных, CRUD операции, query builder	20	32
5.5	Storage Guide	Руководство по работе с файлами,	16	24

		upload/download, signed URLs		
5.6	Realtime Guide	Руководство по real-time подпискам, примеры использования	16	24
5.7	RLS Guide	Подробное руководство по Row Level Security с примерами политик	24	40
5.8	Self-Hosting Guide	Инструкции по самостоятельному развертыванию платформы	20	32
5.9	Troubleshooting и FAQ	Раздел решения проблем и часто задаваемых вопросов	16	24
5.1	Video tutorials	Создание видео-туториалов для основных сценариев использования	40	60
6	Релиз и внедрение			
6.1	Production инфраструктура	Настройка Kubernetes, Helm charts, StatefulSets, Deployments	60	100
6.2	High Availability	Настройка PostgreSQL репликации, failover через Patroni	40	60
6.3	Backup и Recovery	Настройка автоматических backups через pgBackRest, PITR	24	40
6.4	Monitoring производственный	Настройка production monitoring: Prometheus, Grafana, Alertmanager	40	60
6.5	Security hardening	SSL/TLS для всех соединений, mutual TLS, network policies	32	50
6.6	CI/CD pipelines	Настройка GitHub Actions для автоматического тестирования и деплоя	24	40
6.7	Alpha тестирование	Тестирование с ограниченной группой пользователей, сбор feedback	40	80
6.8	Beta тестирование	Расширенное тестирование с большей группой, исправление недочетов	60	100
6.9	Финальный релиз	Подготовка к production релизу, миграция данных, запуск	40	60
6.10	Поддержка	Выстраивание процесса технической поддержки, обучение специалистов	40	60

ОЦЕНКА ТРУДОЕМКОСТИ РАЗРАБОТКИ МЕТОДОМ PERT

Формулы:

Оптимальная оценка = $\frac{\text{Оптимистичная} + \text{Пессимистичная}}{2}$

Ожидаемое время (T_i) = $\frac{\text{Оптимистичная} + 4\times\text{Оптимальная} + \text{Пессимистичная}}{6}$

Среднеквадратичное отклонение ($СКО_i$) = $\frac{\text{Пессимистичная} - \text{Оптимистичная}}{6}$

№	Название	Описание	Оптимист. (часы)	Пессимист. (часы)	Оптим. (часы)	T_i	$СКО_i$
1	**Подготовка**						
1.1	Требования	Сбор, анализ, уточнение требований, определение критериев приемки, изучение архитектуры Supabase	40	80	60	60	7
1.2	Архитектура	Выбор технологического стека, проектирование микросервисной архитектуры, составление контрактов API и схем данных	60	120	90	90	10
1.3	Онбординг	Погружение разработчиков в контекст платформы, знакомство с требованиями, архитектурой, изучение PostgreSQL RLS и логической репликации	24	40	32	32	3
1.4	Окружение	Создание страниц документации, высокоуровневых эпиков в трекере, репозиториев, настройка базовой инфраструктуры, CI/CD, выдача доступов	40	80	60	60	7
2	**Разработка Core компонентов**						
2.1	EPIC-1: Базовая инфраструктура						
2.1.1	Docker Compose конфигурация	Создание docker-compose.yml с network, volumes, health checks для всех сервисов	16	24	20	20	1
2.1.2	Управление окружением	Настройка .env файлов, DNS-резолвинг, интеграция с Consul для секретов	12	20	16	16	1

2.1.3	Логирование	Настройка централизованного логирования через Docker logging driver	8	12	10	10	1
2.1.4	Документация и скрипты	Написание Markdown документации, создание Bash и PowerShell скриптов для автоматической сборки	12	20	16	16	1
2.2	EPIC-2: PostgreSQL Database						
2.2.1	Развертывание PostgreSQL 17+	Настройка Docker контейнера, параметров производительности (shared_buffers, work_mem, etc)	8	16	12	12	1
2.2.2	Настройка схем и permissions	Создание схем auth, storage, public с правильными правами доступа	8	12	10	10	1
2.2.3	Row Level Security	Включение RLS, создание базовых функций для работы с JWT claims	12	20	16	16	1
2.2.4	Логическая репликация	Настройка wal_level=logical, создание publication для отслеживаемых таблиц	8	16	12	12	1
2.3	EPIC-3: API Gateway (Kong)						
2.3.1	Развертывание Kong	Настройка Kong Gateway в Docker, создание декларативной YAML конфигурации	12	20	16	16	1
2.3.2	Настройка маршрутизации	Конфигурация маршрутов для всех сервисов (/rest/v1/*, /auth/v1/*, /storage/v1/*, /realtime/v1/*, /functions/v1/*)	16	24	20	20	1
2.3.3	JWT plugin	Установка и настройка JWT plugin с HMAC-SHA256, валидация токенов	12	16	14	14	1
2.3.4	ACL и CORS plugins	Настройка ACL для ролей (anon, authenticated, service_role), CORS политик	8	12	10	10	1
2.3.5	Rate limiting	Настройка rate limiting	8	12	10	10	1

	и health checks	plugin, создание health check endpoints					
2.4	EPIC-4: Authentication Service (GoTrue)						
2.4.1	Развертывание GoTrue	Настройка GoTrue Docker image, создание таблицы auth.users	8	12	10	10	1
2.4.2	Email/Password регистрация	Реализация регистрации с Argon2 хешированием, создание REST endpoints	16	24	20	20	1
2.4.3	Email верификация	Реализация magic links, настройка SMTP, отправка confirmation emails	20	32	26	26	2
2.4.4	JWT токены	Механизм генерации JWT с claims (role, user_id, exp, aud), HMAC-SHA256	12	20	16	16	1
2.4.5	Refresh токены	Реализация refresh токенов со сроком 30 дней, endpoint с grant_type=refresh_token	12	16	14	14	1
2.4.6	OAuth интеграция	Интеграция OAuth 2.0 с Google и GitHub провайдерами	24	40	32	32	3
2.4.7	Database triggers	Создание trigger для автоматического создания public.profiles, обработка ошибок	8	12	10	10	1
2.5	EPIC-5: REST API Generation (PostgREST)						
2.5.1	Развертывание PostgREST	Настройка PostgREST Docker image, автоматическая рефлексия схемы PostgreSQL	8	12	10	10	1
2.5.2	REST endpoints генерация	Генерация endpoints для всех таблиц с GET, POST, PATCH, DELETE методами	12	20	16	16	1
2.5.3	JWT интеграция	Извлечение роли, email, user_id из JWT, SET LOCAL role и request.jwt.claims	12	20	16	16	1
2.5.4	RLS применение	Автоматическое применение Row Level Security политик к запросам	8	16	12	12	1
2.5.5	Query	Реализация фильтрации,	16	24	20	20	1

	параметры	пагинации, комбинированных условий (AND/OR)					
2.5.6	JSON сериализация	Настройка возврата результатов в JSON формате	4	8	6	6	1
2.6	EPIC-6: Row Level Security						
2.6.1	Включение RLS	Включение Row Level Security для всех пользовательских таблиц	4	8	6	6	1
2.6.2	Политики для операций	Создание USING политик для SELECT, WITH CHECK для INSERT/UPDATE/DELETE	16	24	20	20	1
2.6.3	Политики для ролей	Создание политик для anon, authenticated, service_role ролей	12	20	16	16	1
2.6.4	Документация политик	Документирование шаблонов политик с примерами использования	8	12	10	10	1
2.6.5	Тестирование RLS	Создание тестовых сценариев для проверки работы политик	12	20	16	16	1
2.7	EPIC-7: JavaScript SDK						
2.7.1	Создание npm пакета	Инициализация @supabase/supabase-js, настройка build системы	8	12	10	10	1
2.7.2	Client инициализация	Реализация createClient(url, anonKey), базовая архитектура клиента	8	12	10	10	1
2.7.3	Auth модуль	Создание auth модуля с методами signUp, signInWithPassword, signInWithOAuth, signOut	20	32	26	26	2
2.7.4	JWT управление	Автоматическое сохранение JWT в localStorage, обновление через refresh token	12	20	16	16	1
2.7.5	Session методы	Реализация getUser() и getSession() для получения текущего пользователя	8	12	10	10	1
2.7.6	Database модуль	Создание from() модуля для работы с данными	8	12	10	10	1

2.7.7	Query builder	Реализация query builder с select(), insert(), update(), delete()	24	40	32	32	3
2.7.8	Фильтрация и операции	Реализация eq(), neq(), gt(), gte(), lt(), lte(), like(), ilike(), in()	16	24	20	20	1
2.7.9	Дополнительные методы	Реализация order(), range(), single(), maybeSingle()	12	16	14	14	1
2.7.10	TypeScript типизация	Добавление TypeScript типов для всех методов и интерфейсов	16	24	20	20	1
2.7.11	Error handling	Реализация обработки ошибок в формате {data, error}	16	24	20	20	1
2.7.12	Retry logic	Реализация retry logic с exponential backoff	8	12	10	10	1
3	**Разработка Extended компонентов**						
3.1	EPIC-8: Storage API						
3.1.1	Spring Boot проект	Создание Storage API сервиса на Java 21 Spring Framework	12	20	16	16	1
3.1.2	S3 интеграция	Интеграция AWS SDK для работы с S3-совместимым хранилищем	16	24	20	20	1
3.1.3	MinIO развертывание	Настройка MinIO для локальной разработки	8	12	10	10	1
3.1.4	Database схема	Создание таблицы file_info с полями id, status, type, path, original_name	8	12	10	10	1
3.1.5	CRUD операции	Реализация REST API для CRUD операций с использованием Hibernate ORM	24	40	32	32	3
3.1.6	Транзакционная логика	Реализация политики изменения статусов (UPLOADING, UPLOADED, UPDATED, DELETED)	20	32	26	26	2
3.1.7	Signed URLs	Реализация генерации signed URLs с настраиваемым expiresIn	12	16	14	14	1
3.1.8	Error handling	Обработка ошибок загрузки, удаления,	12	16	14	14	1

		транзакционных откатов					
3.2	EPIC-9: Storage SDK						
3.2.1	Storage модуль	Создание модуля storage в JavaScript SDK	8	12	10	10	1
3.2.2	Upload метод	Реализация upload(path, file, options) с File и Blob поддержкой	16	24	20	20	1
3.2.3	Content-type определение	Автоматическое определение MIME type, параметр upsert	8	12	10	10	1
3.2.4	Upload progress	Реализация onUploadProgress callback с процентами загрузки	12	16	14	14	1
3.2.5	Download и remove	Реализация download(path) и remove(paths) методов	12	16	14	14	1
3.2.6	List и URL методы	Реализация list(), getPublicUrl(), createSignedUrl()	16	24	20	20	1
3.2.7	TypeScript типизация	Добавление типов для Storage модуля с интерфейсами	8	12	10	10	1
3.2.8	Error handling и retry	Обработка ошибок прав доступа, retry logic для failed uploads	12	16	14	14	1
3.3	EPIC-10: Realtime Server						
3.3.1	Elixir/Phoenix проект	Создание Realtime сервера на Elixir/Phoenix Framework	24	40	32	32	3
3.3.2	Логическая репликация	Подключение к PostgreSQL через replication protocol	20	32	26	26	2
3.3.3	Replication slot	Создание replication slot supabase_realtime, установка wal2json	12	20	16	16	1
3.3.4	WAL streaming	Реализация слушания изменений в WAL через streaming connection	24	40	32	32	3
3.3.5	WAL парсинг	Парсинг WAL записей в формат {table, operation, old_record, new_record, timestamp}	20	32	26	26	2
3.3.6	RLS фильтрация	Применение RLS политик для фильтрации подписчиков	24	40	32	32	3

3.3.7	Права доступа	Создание функции проверки прав доступа к строкам через SELECT RLS	16	24	20	20	1
3.3.8	WebSocket endpoint	Создание /realtime/v1/websocket endpoint с wss:// протоколом	12	20	16	16	1
3.3.9	Phoenix Channels	Реализация Phoenix Channels protocol (phx_join, phx_leave, phx_reply)	16	24	20	20	1
3.3.10	JWT валидация	Валидация JWT из query параметра apikey	8	12	10	10	1
3.3.11	Channel подписки	Реализация подписок через postgres_changes:{schema}:{table}	12	20	16	16	1
3.3.12	Event broadcasting	Отправка событий INSERT, UPDATE, DELETE с old/new значениями	16	24	20	20	1
3.3.13	Graceful shutdown	Реализация graceful shutdown с уведомлением клиентов	8	12	10	10	1
3.4	EPIC-11: Realtime SDK						
3.4.1	Channel модуль	Создание модуля channel() в JavaScript SDK	8	12	10	10	1
3.4.2	WebSocket соединение	Установка WebSocket соединения с автоматическим HTTP upgrade	12	20	16	16	1
3.4.3	JWT передача	Передача JWT через apikey query параметр	4	8	6	6	1
3.4.4	Channel методы	Реализация channel(topic) и on('postgres_changes', options, callback)	16	24	20	20	1
3.4.5	Фильтры	Реализация фильтров по event, table, schema	12	16	14	14	1
3.4.6	Subscribe/Unsubscribe	Реализация subscribe() и unsubscribe() с Promise	12	16	14	14	1
3.4.7	Auto reconnection	Автоматическое переподключение с exponential backoff	16	24	20	20	1
3.4.8	Подписки	Сохранение и	12	16	14	14	1

	восстановление	восстановление подписок после reconnect					
3.4.9	State management	Управление состоянием: CONNECTING, OPEN, CLOSING, CLOSED	8	12	10	10	1
3.4.10	TypeScript типизация	Добавление generic типов для old и new данных	8	12	10	10	1
3.4.11	Дополнительные методы	Реализация removeAllChannels() и error handling с retry	8	12	10	10	1
4	**Тестирование**						
4.1	Unit тестирование	Unit тесты для всех модулей SDK, сервисов, утилит	60	100	80	80	7
4.2	Интеграционное тестирование	Интеграционные тесты компонентов системы, Auth flow, CRUD операций, Storage, Realtime	80	120	100	100	7
4.3	E2E тестирование	End-to-end тестирование полных сценариев через Playwright/Cypress	60	100	80	80	7
4.4	Performance тестирование	Load testing через k6, определение пределов производительности	40	60	50	50	3
4.5	Security тестирование	Security audit, SQL injection защита, JWT hijacking, RLS bypass attempts	40	80	60	60	7
4.6	Исправление багов	Выявление и исправление критических багов после тестирования	80	120	100	100	7
5	**Документация**						
5.1	Getting Started guide	Пошаговое руководство по началу работы, установке, первым шагам	16	24	20	20	1
5.2	API Reference	Полная документация всех API endpoints и SDK методов	40	60	50	50	3
5.3	Authentication Guide	Подробное руководство по аутентификации с примерами	20	32	26	26	2
5.4	Database Guide	Руководство по работе с базой данных, CRUD	20	32	26	26	2

		операции, query builder					
5.5	Storage Guide	Руководство по работе с файлами, upload/download, signed URLs	16	24	20	20	1
5.6	Realtime Guide	Руководство по real-time подпискам, примеры использования	16	24	20	20	1
5.7	RLS Guide	Подробное руководство по Row Level Security с примерами политик	24	40	32	32	3
5.8	Self-Hosting Guide	Инструкции по самостоятельному развертыванию платформы	20	32	26	26	2
5.9	Troubleshooting и FAQ	Раздел решения проблем и часто задаваемых вопросов	16	24	20	20	1
5.1	Video tutorials	Создание видео-туториалов для основных сценариев использования	40	60	50	50	3
6	**Релиз и внедрение**						
6.1	Production инфраструктура	Настройка Kubernetes, Helm charts, StatefulSets, Deployments	60	100	80	80	7
6.2	High Availability	Настройка PostgreSQL репликации, failover через Patroni	40	60	50	50	3
6.3	Backup и Recovery	Настройка автоматических backups через pgBackRest, PITR	24	40	32	32	3
6.4	Monitoring производственный	Настройка production monitoring: Prometheus, Grafana, Alertmanager	40	60	50	50	3
6.5	Security hardening	SSL/TLS для всех соединений, mutual TLS, network policies	32	50	41	41	3
6.6	CI/CD pipelines	Настройка GitHub Actions для автоматического тестирования и деплоя	24	40	32	32	3
6.7	Alpha тестирование	Тестирование с ограниченной группой пользователей, сбор feedback	40	80	60	60	7
6.8	Beta	Расширенное	60	100	80	80	7

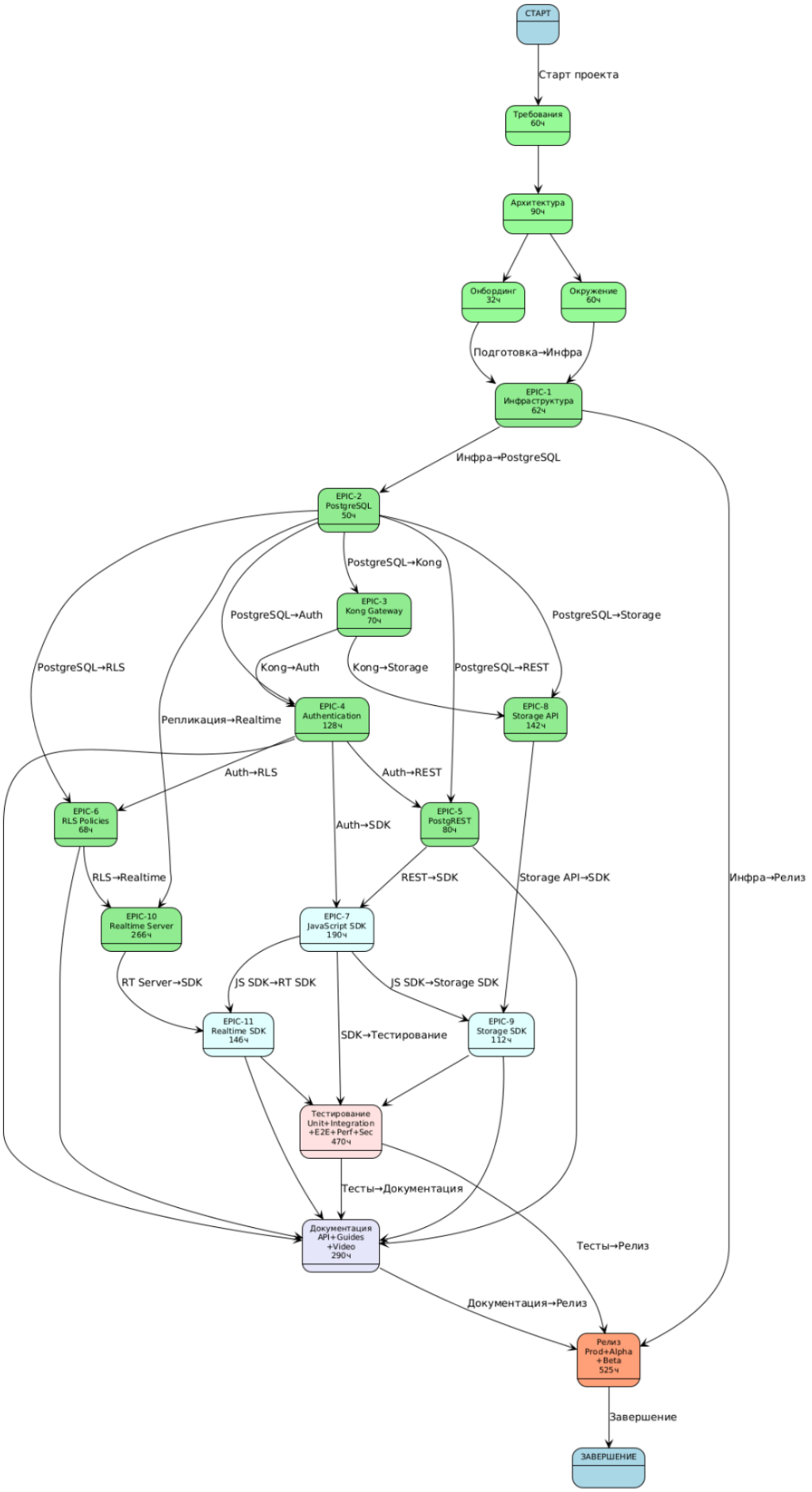
	тестирование	тестирование с большей группой, исправление недочетов					
6.9	Финальный релиз	Подготовка к production релизу, миграция данных, запуск	40	60	50	50	3
6.10	Поддержка	Выстраивание процесса технической поддержки, обучение специалистов	40	60	50	50	3

Предсказанное количество потраченных человеко-часов (ΣT_i): **2857**

Среднеквадратичное отклонение для оценки трудоемкости ($\sqrt{\Sigma СКО_i^2}$): **29,33**

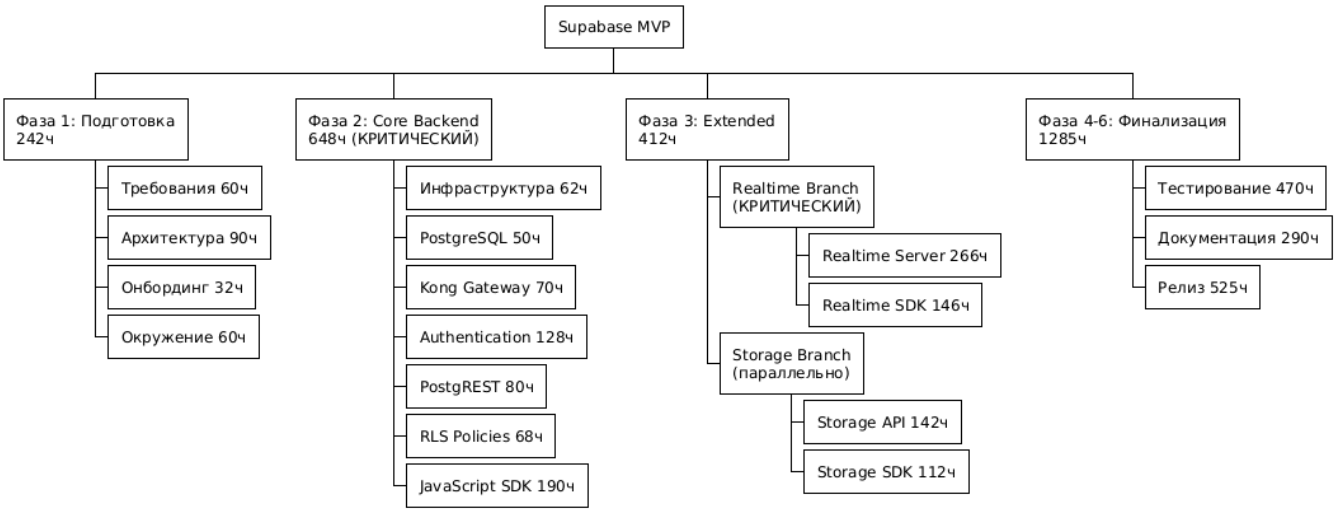
Суммарная трудоемкость проекта с вероятность 95% ($\Sigma T_i + 2 \cdot \sqrt{\Sigma СКО_i^2}$): **2916**

Сетевая диаграмма (клик)



Метод критического пути

Критический путь - это последовательность задач, которые нельзя выполнить параллельно:



Разраб.	Крит. путь (часы)	Параллельные (часы)	Календ. часы (часы)	Срок (мес)
1	2587	329	2916	~ 18.2
2	1294	164	1458	~ 9.1
3	862	110	972	~ 6
4	647	82	729	~ 4.6
5	517	66	583	~ 3.6
6	431	55	486	~ 3

Оптимальное количество разработчиков - 4, так как при большем количестве срок работы уменьшается в меньшем объеме, а цена возрастает

Состав команды

Backend Lead	- PostgreSQL, PostgREST, RLS - Архитектура и технические решения - Code Review
DevOps Engineer	- Docker, Kong Gateway, CI/CD - Инфраструктура, мониторинг - Production deployment
Backend Developer	- GoTrue Authentication Service - Storage API, Realtime Server - Микросервисы на Spring/Elixir
Frontend/SDK Developer	- JavaScript SDK (Auth, Database, Query Builder) - Storage SDK, Realtime SDK - TypeScript типизация, документация

ОЦЕНКА РАЗМЕРА ПРОЕКТА МЕТОДОМ ФУНКЦИОНАЛЬНЫХ ТОЧЕК

функциональные блоки
1. Authentication Service (GoTrue)
2. REST API Generation (PostgREST)
3. Row Level Security (RLS)
4. JavaScript SDK
5. Storage API
6. Storage SDK
7. Realtime Server
8. Realtime SDK
9. API Gateway (Kong)
10. PostgreSQL Database
11. Инфраструктура (Docker, CI/CD)
12. Документация и поддержка

Типы транзакций
- EI (External Input) - ввод данных извне
- EO (External Output) - вывод данных с обработкой
- EQ (External inQuiry) - запросы без обработки

Типы данных
- ILF (Internal Logical File) - внутренние логические файлы
- EIF (External Interface File) - внешние интерфейсные файлы

Параметры
- DET (Data Element Types) - типы элементов данных - поля, элементы
- FTR (File Types Referenced) - количество ссылок на файлы - сколько БД трогает таблиц или система файлов
- RET (Record Element Types) - типы записей

№	Функционал	Тип	DET	FTR	Данные	RET	Сложн. транзакций	Сложн. данных	UFP
1	Authentication Service (GoTrue)								**31**
	Регистрация пользователей	EI	8	2	ILF	2	Average	Low	4
	Авторизация (Email/Password)	EI	6	2	ILF	2	Low	Low	3
	Email верификация	EI	5	1	ILF	1	Low	Low	3
	JWT генерация и валидация	EO	12	3	ILF	2	High	Low	7
	Refresh токены	EI	4	1	ILF	1	Low	Low	3
	OAuth интеграция (Google, GitHub)	EI	10	3	EIF	2	High	Average	6
	Database triggers (profiles)	EO	6	2	ILF	1	Average	Low	5
2	REST API Generation (PostgREST)								**30**
	Автоматическая генерация REST endpoints	EO	15	5	ILF	4	High	High	10
	JWT интеграция и role switching	EI	8	2	ILF	2	Average	Low	4

	Query параметры (фильтрация, пагинация)	EQ	20	6	ILF	5	High	High	12
	JSON сериализация	EO	5	1	ILF	1	Low	Low	4
3	Row Level Security (RLS)								**19**
	Создание политик для операций	EI	12	3	ILF	3	High	Average	6
	Политики для ролей (anon/auth/service)	EI	10	3	ILF	3	Average	Average	5
	Применение RLS в запросах	EO	8	2	ILF	2	Average	Low	5
	Тестирование политик	EQ	6	2	ILF	2	Low	Low	3
4	JavaScript SDK								**42**
	Client initialization (createClient)	EI	4	1	ILF	1	Low	Low	3
	Auth модуль (signUp, signIn, signOut)	EI	12	3	ILF	2	High	Low	6
	JWT управление (localStorage, refresh)	EO	8	2	ILF	2	Average	Low	5
	Database модуль (from, select, insert)	EI	15	4	ILF	3	High	Average	7
	Query builder (filter, order, range)	EQ	18	5	ILF	4	High	High	12
	Error handling	EO	6	1	ILF	1	Low	Low	4
	TypeScript типизация	EO	10	2	ILF	2	Average	Low	5
5	Storage API (Spring Boot)								**22**
	S3 интеграция (upload, download)	EI	10	3	EIF	2	High	Average	6
	CRUD операции с файлами	EI	12	4	ILF	3	High	Average	6
	Транзакционная логика (статусы)	EO	8	3	ILF	2	Average	Low	5
	Signed URLs генерация	EO	6	2	ILF	1	Average	Low	5
6	Storage SDK (JavaScript)								**15**
	Upload с progress tracking	EI	8	2	ILF	2	Average	Low	4
	Download и remove методы	EI	6	2	ILF	2	Low	Low	3
	List и URL методы	EQ	8	2	ILF	2	Average	Low	4
	Content-type определение	EO	4	1	ILF	1	Low	Low	4
7	Realtime Server (Elixir/Phoenix)								**43**
	Логическая репликация PostgreSQL	EI	12	4	EIF	3	High	High	8
	WAL streaming и парсинг	EO	15	5	ILF	4	High	High	10
	RLS фильтрация событий	EO	10	3	ILF	3	High	Average	7
	WebSocket endpoint	EI	8	2	ILF	2	Average	Low	4
	Phoenix Channels protocol	EO	10	3	ILF	2	High	Low	7
	Event broadcasting (INSERT/UPDATE/DEL)	EO	12	4	ILF	3	High	Average	7
8	Realtime SDK (JavaScript)								**22**
	WebSocket соединение	EI	6	2	ILF	2	Low	Low	3
	Channel подписки	EI	10	3	ILF	2	High	Low	6
	Фильтры по event/table/schema	EQ	8	2	ILF	2	Average	Low	4
	Auto reconnection с backoff	EO	7	2	ILF	1	Average	Low	5
	State management	EO	5	1	ILF	1	Low	Low	4
9	API Gateway (Kong)								**18**

	Маршрутизация запросов	EO	12	4	ILF	3	High	Average	7
	JWT plugin и валидация	EI	8	2	ILF	2	Average	Low	4
	ACL и CORS plugins	EI	6	2	ILF	2	Low	Low	3
	Rate limiting	EO	5	1	ILF	1	Low	Low	4
10	PostgreSQL Database								**16**
	Схемы (auth, storage, public)	EI	8	3	ILF	3	Average	Average	5
	RLS базовые функции	EO	10	3	ILF	2	High	Low	7
	Логическая репликация setup	EI	6	2	ILF	2	Average	Low	4
11	Инфраструктура								**16**
	Docker Compose конфигурация	EI	8	3	ILF	2	Average	Low	4
	CI/CD pipeline	EO	10	3	ILF	2	High	Low	7
	Логирование и мониторинг	EO	8	2	ILF	2	Average	Low	5
12	Документация и поддержка								**18**
	API Reference (автогенерация)	EO	15	5	ILF	4	High	High	10
	Getting Started, Guides	EQ	12	3	ILF	3	Average	Average	5
	FAQ и Troubleshooting	EQ	6	1	ILF	1	Low	Low	3

Общее количество UFP:
31 + 30 + 19 + 42 + 22 + 15 + 43 + 22 + 18 + 16 + 16 + 18 = 292 UFP

ОЦЕНКА РАЗМЕРА ПРОЕКТА МЕТОДОМ COCOMO II

Перевод функциональных точек в строки кода (SLOC)

Сервис	**Технология / Язык**	**SLOC/FP avg**	**UFP**
PostgreSQL Database	SQL	21	16
API Gateway (Kong)	Lua (внутри Kong) конфигурация - YAML/Declarative	47 (аналог JS)	18
Authentication Service (GoTrue)	Go	50 (аналог Java)	31
REST API Generation (PostgREST)	Haskell	50 (аналог Java)	30
JavaScript SDK	JavaScript / TypeScript	47	42
Storage API	Java (Spring)	53	22
Storage SDK	JavaScript / TypeScript	47	15
Realtime Server	Elixir	50 (аналог Java)	43
Realtime SDK	JavaScript / TypeScript	47	22
Инфраструктура	Bash / PowerShell Docker Compose (YAML)	24 (аналог Perl)	16
Документация	Не код => не учитываем		18

Исключаем документацию из UFP: $292 - 18 = 274$ UFP

$SLOC/FP = \frac{21\times16+47\times18+50\times31+50\times30+47\times42+53\times22+47\times15+50\times43+47\times22+24\times16}{274} = 42.5$

$SLOC = 274 \times 42.5 = 11645$

$KLOC = 11.645$

$Effort = 2.94 \times (KLOC)^{1.0997} = 43.7$ человеко-месяцев $= 6992$ человека-часов

Ответ: 6992 человеко-часов

ОЦЕНКА РАЗМЕРА ПРОЕКТА МЕТОДОМ USE CASE POINTS

$UCP = UUCW \times TCF \times ECF$

UUCW (Unadjusted Use Case Weight) - невзвешенный вес вариантов использования
TCF (Technical Complexity Factor) - фактор технической сложности
ECF (Environmental Complexity Factor) - фактор сложности окружения

Трудозатраты = $Effort = UCP \times PF$ (где PF - фактор продуктивности)

Use Cases:
Simple (1-3 транзакций): 0

Average (4-7 транзакций): 13

Complex (>7 транзакций): 2

Веса:

Simple UC: 5 весовых единиц
Average UC: 10 весовых единиц
Complex UC: 15 весовых единиц

$UUCW = 0UC \times 5 + 13UC \times 10 + 2UC \times 15 = 0 + 130 + 30 = 160$

Actors:

- Разработчик - человек через JavaScript SDK (GUI) = Complex = 3
- Конечный пользователь - человек через GUI (веб-приложение) = Complex = 3
- Мобильное прилоежние - Приложение через REST API (Protocol) = 2
- PostgreSQL - Система через логическую репликацию (API) = 1
- S3/MiniO - Система через AWS SDK (API) = 1
- OAuth провайдеры - Внешние системы (Google, GitHub) через API = 1

Actor Weight:

Simple - API: 1
Average - UI/Protocol: 2
Complex - GUI: 3

$Actor\ Weight = 3 + 3 + 2 + 1 + 1 + 1 = 11$
UUCP (Нескорректированные UCP) = $UUCW + Actor\ Weight = 160 + 11 = 171$

В пример - курсач

Use Cases:
Simple (1-3 транзакций):

- UP-03 Переход в Личный Кабинет (4 шага, но 1-2 транзакции: проверка авторизации + показ страницы)
- AP-01 Переход в Панель Администратора (4 шага, аналогично)

Average (4-7 транзакций):

- UP-01 Регистрация (11 шагов: валидация, проверка email, создание записи, генерация токена и т.д. ≈ 5-6 транзакций)
- UP-02 Авторизация (11 шагов: валидация, поиск пользователя, проверка пароля, генерация токена ≈ 4-5 транзакций)
- UP-05 Удаление аккаунта (6 шагов: показ, удаление, очистка данных ≈ 3-4 транзакции)

Complex (>7 транзакций):

- UP-04 Изменение данных аккаунта (11 шагов: валидация, создание новой записи, деактивация старой, обновление статусов, генерация нового токена ≈ 7-8 транзакций)
- AP-02 Заморозка/Блокировка/Удаление (10 шагов: поиск, показ списка, выбор, открытие профиля, действие ≈ 8+ транзакций с разными путями)
- MP-01 Перевод денежных средств (16 шагов: поиск, выбор, проверка валют, конвертация, запись транзакции ≈ 10+ транзакций)

Веса:

Simple UC: 5 весовых единиц
Average UC: 10 весовых единиц
Complex UC: 15 весовых единиц

$$UUCW = 2UC \times 5 + 3UC \times 10 + 3UC \times 15 = 10 + 30 + 45 = 85$$

Actors:

- Клиент (обычный) - человек через GUI (веб-приложение) = Complex = 3
- Бизнес-клиент - человек через GUI = Complex = 3
- Администратор системы - человек через GUI = Complex = 3

Actor Weight:

Simple - API: 1
Average - UI/Protocol: 2
Complex - GUI: 3

$$Actor\ Weight = 3 + 3 + 3 = 9$$

UUCP (Нескорректированные UCP) $= UUCW + Actor\ Weight = 85 + 9 = 94$

Technical Complexity Factor:

Фактор	Описание	Вес	Оценка	Итог
T1	Распределенная система	2	5	10
T2	Время отклика	1	5	5
T3	Эффективность для пользователя	1	5	5
T4	Сложная внутренняя обработка	1	5	5
T5	Повторное использование кода	1	4	4
T6	Простота установки	0.5	5	2.5
T7	Простота использования	0.5	5	2.5
T8	Переносимость	2	5	10
T9	Простота изменений	1	5	5
T10	Параллельное использование	1	5	5
T11	Специальные функции безопасности	1	5	5
T12	Прямой доступ для третьих сторон	1	5	3
T13	Требования к обучению	1	2	2

$TFactor = \Sigma(\text{Вес} \times \text{Оценка}) = 66$
$TCF = 0.6 + (0.01 \times TFactor) = 0.6 + 0.66 = 1.26$

Environmental Complexity Factor:

Фактор	Описание	Вес	Оценка	Итог
E1	Знакомство с проектом	1.5	4	6
E2	Опыт работы в приложениях	0.5	4	2
E3	Опыт ООП	1	5	5
E4	Способности ведущего аналитика	0.5	4	2
E5	Мотивация	1	5	5
E6	Стабильность требований	2	5	10
E7	Работники на неполную ставку	-1	1	-1
E8	Сложность языка программирования	-1	3	-3

$EFactor = \Sigma(\text{Вес} \times \text{Оценка}) = 26$
$ECF = 1.4 + (-0.03 \times EFactor) = 1.4 - 0.78 = 0.62$
$UCP = UUCP \times TCF \times ECF = 171 \times 1.26 \times 0.62 = 133.56 = 134$

Technical Complexity Factor:

Фактор	Описание	Вес	Оценка	Итог
T1	Распределенная система	2	5	10
T2	Время отклика	1	4	4
T3	Эффективность для пользователя	1	4	4
T4	Сложная внутренняя обработка	1	4	4
T5	Повторное использование кода	1	3	3
T6	Простота установки	0.5	4	2
T7	Простота использования	0.5	4	2
T8	Переносимость	2	4	8
T9	Простота изменений	1	4	4
T10	Параллельное использование	1	5	5
T11	Специальные функции безопасности	1	5	5
T12	Прямой доступ для третьих сторон	1	3	3
T13	Требования к обучению	1	2	2

$TFactor = \Sigma(\text{Вес} \times \text{Оценка}) = 56$
$TCF = 0.6 + (0.01 \times TFactor) = 0.6 + 0.56 = 1.16$

Environmental Complexity Factor:

Фактор	Описание	Вес	Оценка	Итог
E1	Знакомство с проектом	2	3	6
E2	Опыт работы в приложениях	1	3	3
E3	Опыт ООП	1	3	3
E4	Способности ведущего аналитика	1	3	3
E5	Мотивация	1	4	4
E6	Стабильность требований	2	4	8
E7	Работники на неполную ставку	1	4	4
E8	Сложность языка программирования	1	3	3

$EFactor = \Sigma(\text{Вес} \times \text{Оценка}) = 34$
$ECF = 1.4 + (-0.03 \times EFactor) = 1.4 - 1.02 = 0.38$

$$UCP = UUCP \times TCF \times ECF = 94 \times 1.16 \times 0.38 = 41$$

$$PF = \frac{\text{Фактические часы}}{UCP} = \frac{350}{41} = 8.53 \text{ часов/UCP}$$

Применение PF:
Общее время разработки $= UCP \times PF + CONST$
CONST - доп работы - пусть будет 240

$$E = 102 \times 8.53 + 240 = 1383$$

Выводы по работе

Результат затрат человеко-часов разными методами:

Наивный метод = 2857
PERT = 2916
COCOMO-II = 6992
UCP = 1383

Наивный метод и PERT показали близкие результаты в районе 2900 человеко-часов, что объясняется схожестью подходов - оба основаны на декомпозиции работ и экспертных оценках, при этом PERT добавляет вероятностную модель с оптимистичными и пессимистичными сценариями, давая более взвешенную оценку. Эти методы просты в применении и понятны команде, но их главный недостаток - субъективность оценок и зависимость от опыта экспертов, а также недоучёт технической сложности системы.

UCP выдал самую низкую оценку в 1383 человеко-часа, поскольку этот метод фокусируется исключительно на функциональных требованиях через анализ вариантов использования и акторов системы. Метод учитывает техническую и организационную сложность через коэффициенты TCF и ECF, что делает его более объективным для оценки чисто разработки, однако он систематически недооценивает реальные трудозатраты, так как не включает инфраструктурные работы, DevOps, тестирование, документацию и управление проектом - всё то, что составляет значительную часть реальной работы.

COCOMO II показал максимальную оценку в 6992 человеко-часа, что связано с его подходом оценки через объём исходного кода и множество корректирующих факторов стоимости. Этот метод особенно чувствителен к размеру кодовой базы и учитывает зрелость процессов разработки, опыт команды, сложность продукта и требования к надёжности, что делает его наиболее консервативным. Высокая оценка объясняется тем, что COCOMO изначально разрабатывался для крупных enterprise-проектов и склонен завышать трудозатраты для современных проектов с использованием готовых фреймворков и микросервисной архитектуры.

Наиболее реалистичной оценкой следует считать среднее между наивным методом и COCOMO II, то есть около 4500-5000 человеко-часов, что учитывает как функциональную разработку, так и все сопутствующие работы по инфраструктуре, тестированию и документации, при этом не перегибая с консервативностью COCOMO II для современного стека технологий.

Зарплата

Позиция	Зарплата в месяц (рублей)
Backend Lead	200 000
DevOps Engineer	150 000
Backend Developer	150 000
Frontend Developer	150 000

Дано:
Трудозатраты = 5400 часов
Команда = 4 человека
Часов в месяц на человека = 100 часов
Отпуск = 1 месяц на человека

Эффективные часы в месяц = 100 * 4 = 400 часов

Длительность = 5400 / 400 = 14 месяцев

Если 1 человек в отпуске: 400 - 100 = 300 часов/мес
Усредненная производительность = 350 часов/мес

Реальная длительность = 5400 / 350 = 18 месяцев

Прямые затраты на оплату труда = 650.000тр * 18 месяцев = 11.700.000

Взносы (30%) = 11.700.000 + 30% = 15.210.000

Офис (моя квартира) = 90 000 * 18 = 1 620 000

Резерв = 15.210.000 + 10% = 16.731.000

Себестоимость: 16.731.000 + 1 620 000 = 18 351 000
Цена вместе с прибылью = 18 351 000 + 30% = **23 856 300**