tim@sentinel:~/backend $

Sentinel

Интеллектуальный тайм-менеджер

Серверная часть

Максимов Тимофей Степанович
БПИ-236

Руководитель: Гурин С. Б.

$ # 02 — Предметная область

III — AI-ассистент

> умное планирование · диалог · ребалансировка

> Motion, Reclaim.ai — платно · закрыто · нет русского

II — Мотивация и вовлечение

> геймификация · система достижений · выработка привычек

> Habitica, Streaks — изолированно, вне планировщика

I — Базовый календарь

> хранение событий · напоминания · синхронизация

> Google Calendar · Apple Calendar · Яндекс

Sentinel = I + II + III

$ # 03 — Глоссарий

Tool-use / Function calling

Механизм OpenAI API: LLM вызывает функции бэкенда вместо генерации свободного текста

Propose-Before-Apply

Наш паттерн: LLM не мутирует данные напрямую, а создаёт предложения со статусом `proposed`, применяемые только после явного подтверждения пользователем

Rebalance

Алгоритм перераспределения событий по временным слотам через LLM с учётом нагрузки по дням

Event / Reminder

Два типа записей в расписании: событие имеет начало и конец; напоминание — только начало

Chat

Сессия диалога пользователя с AI-ассистентом; хранит историю сообщений и предложенные действия

System prompt

Инструкция, которая передаётся модели с каждым запросом: описывает роль ассистента, текущее время и часовой пояс пользователя

$ # 04 — Цели и задачи

Цель

Разработать серверную часть мобильного приложения Sentinel — масштабируемый асинхронный `REST API` с интеграцией LLM

Задачи

[1] Анализ аналогов и формирование требований

[2] Выбор стека технологий и архитектурных решений

[3] Проектирование схемы базы данных

[4] Разработка REST API (аутентификация, события, синхронизация)

[5] Интеграция LLM с паттерном Propose-Before-Apply

[6] Алгоритмы: ребалансировка расписания и система достижений

[7] Обеспечение надёжности: тестирование, rate limiting, CI/CD

$ # 05 — Анализ аналогов

Критерий	Todoist	Google Cal	Motion	Reclaim.ai	Sunsama	Things 3	Notion Cal	Sentinel
Отображение событий	+	+	+	+	+	+	+	+
Генерация через ИИ	-	-	-	+	-	-	-	+
Нативная интеграция с Apple Calendar	-	+	-	-	-	+	-	+
Ребалансировка расписания	-	-	+	+	-	-	-	+
Модель энергии дня	-	-	-	-	-	-	-	+
Геймификация	-	-	-	-	+	-	-	+
Кроссплатформенность	+	+	+	+	+	-	+	-
Уведомления	+	+	+	+	+	+	+	+

$ # 06 — Стек технологий

Python 3.11 + FastAPI
async-first веб-фреймворк, OpenAPI 3.x из коробки

SQLAlchemy 2.0 async + asyncpg
нативная асинхронность драйвера и ORM

Alembic
версионируемые миграции схемы БД

PostgreSQL 15
реляционная модель с FK-каскадами и JSONB

OpenAI SDK
tool-use из коробки, structured outputs

Docker Compose
воспроизводимое окружение dev и prod

JWT (HS256)
stateless-аутентификация, TTL 7 дней

pytest + httpx
unit и integration тестирование async-эндпоинтов

S3
хранилище изображений в чатах

Сервер: async I/O на всех уровнях — LLM-запрос занимает 2–5 с, без блокировки event-loop'а

$ # 07 — Архитектура бэкенда

LAYER · 01

API Layer

принимает HTTP-запросы · валидирует через Pydantic · проверяет JWT · не содержит бизнес-логики

`auth.py` `events.py` `chats.py` `achievements.py` `rebalance.py`

↓ DEPENDS ON

LAYER · 02

Business Layer

сложная логика · оркестрация LLM · ребалансировка · достижения · email · не знает про HTTP

`LLMService` `RebalanceService` `AchievementService` `EmailService` `StorageService`

↓ DEPENDS ON

LAYER · 03

Infrastructure Layer

SQL-запросы к PostgreSQL · вызовы OpenAI API · операции с S3 · не знает про бизнес-логику

`UserRepository` `EventRepository` `ChatRepository` `AchievementRepository`

`CounterRepository` `OpenAIClient` `S3Client`

Async-first
весь I/O неблокирующий

Repository pattern
инфраструктура за интерфейсами

Dependency Injection
через FastAPI `Depends`

$ # 08 — Схема базы данных

9 таблиц

- users
- events
- chats
- chat_messages
- achievements
- user_achievements
- user_counters
- idempotency_keys
- auth_tokens

Архитектурные решения

TIMESTAMPTZ + IANA timezone
UTC в БД, IANA-timezone у пользователя — конвертация на клиенте

JSONB для content_structured
EventAction-предложения AI хранятся без изменения схемы БД

is_fixed
флаг у события — не трогать при ребалансировке

user_counters
достижения — данные в БД, не захардкоженная логика

$ # 09 — REST API

Auth	9
Events	6
Chats	7
Rebalance	2
Achievements	1
System	1

X-Idempotency-Key

iOS повторяет запрос при обрыве сети — сервер находит ключ в БД (TTL 24ч) и возвращает ранее созданный объект вместо дубликата

POST /events/sync

Батч до 500 upsert + 500 delete в одной транзакции — для офлайн-режима iOS

Cursor pagination

`?before=<uuid>&limit=50` — при новых сообщениях страницы не смещаются в отличие от offset-based

Документация OpenAPI 3.x генерируется автоматически · sentinel-ai.tech/docs

$ # 10 — LLM · Propose-Before-Apply

Проблема: LLM не должен изменять расписание пользователя без его ведома

1
Сообщение
Клиент отправляет текст в чат; FastAPI принимает запрос

системный промпт: роль ассистента · время · ai_instructions

→

2
Tool-calls
LLM вызывает `search_events`, `create_event`, `update_event`, `delete_event`

→

3
Proposed
Создаются `EventAction` со `status = proposed` — БД не меняется

→

4
Confirm
Клиент отображает предложения; пользователь выбирает подмножество

→

5
Apply
`POST /apply` → одна транзакция, атомарное применение в БД

✓ **Гарантия:** LLM никогда не мутирует данные напрямую — только через подтверждённый apply

$ # 11 — Алгоритмы

Rebalance

массовый перенос событий через propose-before-apply

1 Сбор контекста
события за период · границы `is_fixed` · пожелания пользователя

2 LLM перестраивает расписание
получает JSON событий + workload summary · возвращает новые слоты

3 Валидация + retry
fixed не сдвинуты · нет перекрытий · retry при ошибке (до 2 попыток)

4 Подтверждение клиентом
пользователь выбирает подмножество переносов

5 Применение одной транзакцией
атомарно, без частично применённых изменений

Тот же паттерн, что в чате — пользователь видит и выбирает

Achievements

data-driven выдача через счётчики активности

1 Создание события
триггер — каждое новое событие от пользователя

2 Инкремент затронутых счётчиков
`total_events` · `ai_events` если source=ai · `total_reminders` если reminder

3 Проверка порогов
только по изменившимся счётчикам · `target ≤ current`

4 Выдача атомарно
`INSERT user_achievements` · достижения не отзываются при удалении

5 Без изменения кода
новое достижение — одна строка в таблице

Логика в данных, не в коде — миграция не нужна

$ # 12 — Надёжность и тестирование

JWT + аутентификация

Подпись `HS256` · `TTL 7 дней` для мобильного · проверка без обращения к БД · одноразовые токены для email и сброса пароля

CI/CD

push → тесты (unit + integration) → деплой на VPS только если тесты прошли

Docker Compose

3 сервиса: `PostgreSQL` · `S3` · `FastAPI app`

Rate limiting

Защита дорогих эндпоинтов: LLM 10 req/min, upload 30 req/min (`slowapi`)

80%+

Кодовая база покрыта unit и integration тестами

31 тестовый файл · pytest + httpx · async-фикстуры

$ # 13 — Демонстрация

[здесь будет видео]

регистрация → создание события → чат с AI → предложение → подтверждение → ребалансировка

$ # 14 — Результаты и выводы

[1] ✓ **Анализ аналогов и формирование требований**
8 конкурентов · выявлены ключевые отличия Sentinel

[2] ✓ **Выбор стека технологий и архитектурных решений**
FastAPI · SQLAlchemy 2.0 async · PostgreSQL 15 · слоистая архитектура

[3] ✓ **Проектирование схемы базы данных**
9 таблиц · JSONB · TIMESTAMPTZ · idempotency_keys

[4] ✓ **Разработка REST API**
25 эндпоинтов · аутентификация · офлайн-sync · cursor pagination

[5] ✓ **Интеграция LLM с паттерном Propose-Before-Apply**
4 tool-функции · системный промпт · ai_instructions · атомарное apply

[6] ✓ **Алгоритмы: ребалансировка расписания и система достижений**
JSON + workload summary → LLM · data-driven счётчики без изменения кода

[7] ✓ **Надёжность: тестирование, rate limiting, CI/CD**
80%+ покрытие · Docker Compose · GitHub Actions · sentinel-ai.tech

$ # 15 — Направления развития

> Prometheus + Grafana мониторинг

> Распознавание голоса через AI для голосового управления

> Монетизация: реклама в AI-чатах и premium-подписка

> Redis для distributed rate limiting

> WebSocket для real-time push-уведомлений

Список использованных источников:

1. FastAPI Documentation — fastapi.tiangolo.com
2. SQLAlchemy 2.0 Documentation — docs.sqlalchemy.org
3. OpenAI API Reference — platform.openai.com/docs
4. PostgreSQL 15 Documentation — postgresql.org/docs/15
5. Docker Documentation — docs.docker.com
6. pytest Documentation — docs.pytest.org

GitHub репозиторий

tim@sentinel:~/backend $

Спасибо за внимание

Готов ответить на вопросы