

Forest Gate

Автор: Георги Веселинов Атанасов

ФН: 2501322014

Използвани GenAI технологии: OpenRouter (достъп до модели като ChatGPT, DeepSeek, Claude и други)

- 1) Въведение и идея
- 2) Функционалности/Дейности/Процеси
- 3) Графики и Диаграми
- 4) Разработка, срещнати проблеми и предизвикателства
- 5) Заключение

Въведение и идея

Когато използваме дадено приложение или онлайн услуга, едно от първите неща, с които се сблъскваме, е входът. Зад тази на пръв поглед проста стъпка стои цяла система, която има за цел да гарантира, че достъп до информацията получават само правилните хора. Именно тук се появяват понятията автентикация и авторизация – основа за сигурността на всяко софтуерно решение.

Днес темата за сигурността е по-актуална от всякога. С нарастването на броя на онлайн услугите, информацията и интеграциите, свързани с изкуствения интелект, се увеличават и опитите за злоупотреби – от кражба на пароли до сложни атаки срещу потребителски акаунти. Това прави изграждането на надеждна система за удостоверяване и контрол на достъпа ключово условие за успеха на всяко приложение.

В този ред на мисли, проектът “Forest Gate” е пример за начална фаза за изграждане на по-сигурна среда за управление на потребителски достъп. Основната му цел е да комбинира добре познатите механизми за автентикация със съвременни методи за наблюдение и анализ на поведенчески модели. Така системата не просто проверява дали дадено име и парола са правилни, а следи и как се използва акаунтът – откъде се осъществява достъпът, колко често се влиза, има ли необичайни действия и др. Проектът цели да бъде стартова точка за хора, които искат да изградят система за автентикация.

Системата е разработена на **Rust**, който е известен със своята висока производителност и надеждност при работа с паметта (механизъм за ownership & borrowing). За съхранение на данните се използва **Postgres**. За кеширане на заявки, сесии и други дейности, свързани с поведението на потребителя, се използва **Redis**. За по-добро емулиране на реална среда са добавени **Docker**

контейнери и **Nginx** като reverse proxy – подход, който е широко използван в истински проекти. За тестови цели е използван **Postman**.

Особеното при “Forest Gate” е включването на инструменти с изкуствен интелект и по-специално **GenAI**. Те подпомагат процеса на разработка и надграждане на системата, като дават възможност за автоматично обобщаване на потребителско поведение, по-лесно откриване на аномалии и генериране на текстове за известия и доклади. Допълнително е предвидена **базова оценка на риск** чрез алгоритъма *Isolation Forest*, използван за сигнализиране при нетипични модели на достъп, без да се превръща в централна част от решението.

За да се осигури още по-добра сигурност, “Forest Gate” интегрира и **GeoIP обогатяване**, което позволява да се проследява от коя държава, град или доставчик идва достъпът. Така се откриват сценарии като „невъзможно пътуване“ или подозрителна смяна на мрежов доставчик. При установяване на риск или промяна в конфигурацията системата може автоматично да изпраща **имейл известия**, които информират потребителите или администраторите за случващото се в реално време. Логиката за изпращане на имейли е зависима от предпочитанията на конкретната система и наличните данни за обучение, затова в настоящата фаза е реализирано известяване при промени по конфигурацията.

Всичко това прави “Forest Gate” не просто прототип на система за достъп, а пример за това как GenAI и съвременни инженерни практики могат да се използват за изграждане на по-сигурни и по-гъвкави софтуерни услуги.

Функционалности/Дейности/Процеси

Още при първото зареждане на страницата приложението създава pre-auth cookie и извлича отпечатък на устройството (fingerprint). За всеки посетител се генерира и уникален подпис (visitor signature). Тази информация се записва в базата и позволява да се свързват следващи действия с конкретно устройство/посещение, дори преди потребителят да се е идентифицирал. Това представлява структурата, която клиентът е длъжен да изпрати:

```
pub struct CreateDeviceDto {  
    pub os_name: Option<String>,  
    pub os_version: Option<String>,  
    pub locale: Option<String>,  
    pub device_type: DeviceType,  
    pub app_version: Option<String>,  
    pub fingerprint: Option<String>,  
    pub extra_data: StableFingerprintData,  
}
```

Дали fingerprint-а ще се генерира на ниво клиент или сървър е въпрос на предпочитание.

Докато посетителят взаимодейства със сайта (scroll, навигация, клик по ключови елементи, заявки към API), **събитията се записват в Redis** и се свързват с **pre-auth идентификатора/подписа**. Поставя се **таймер ключ с TTL** за неактивност. Когато ключът изтече, **Redis Pub/Sub** изпраща „expired“ събитие. Задължително е да се изпълни следната команда в redis-cli:

```
> CONFIG SET notify-keyspace-events Ex
```

Background процес („**flusher**“) слуша тези събития. При сигнал за изтичане, той прочита натрупаните действия, изчиства ги от кеша и създава Markdown обобщение с помощта на моделите, осигурени от **OpenRouter**, в **/interactions/{interaction_id}.md**. Така всяко „посещение“ или период на активност образува самостоятелен запис. За да може моделът от **OpenRouter**

да направи обобщение на сесията на потребителя, трябва да се зададе конкретна инструкция, за да извлечем максимума:

```
role: "system".into(),

    content: r#"You write short, fluent summaries of user interactions for
internal audit logs.

- Use simple, clear English (B1-B2 level).

- Past tense; 3-6 sentences total.

- Group similar actions; avoid duplicates and noise.

- Infer the user's goal when clear, but do not invent facts.

- If the user asked questions, include them as: The user asked: "...".

- Do NOT include IDs or internal metadata in the prose."#.into(),

    };

let user_content = format!(

    "Create a brief summary for interaction_id: {interaction_id}
{prefix}\nEvents:\n- {events_block}"

    );
```

При вход в системата се обогатява контекста с **държава, град** (ако е налично) и **ASN/ISP** чрез **MaxMind**. Това помага да се засекат сценарии като „невъзможно пътуване“ или внезапна смяна на интернет доставчик.

Системата поддържа **имейл известия** чрез **SendGrid**. В текущата фаза фокусът е върху **известия при промени в конфигурацията**. По избор може да се добавят тригери и за определени пред-логин събития (напр. твърде чест трафик, подозрителни модели).

Дори без отделен UI, **Markdown обобщенията** и логовете дават добра видимост: списък посещения, детайл за конкретно посещение (събития + обобщение), филтри по време/локация/сигнал. Ако има фронтенд, тези изгледи се визуализират в панели за по-бърз анализ.

В текущия прототип събитията живеят в **Redis**, а след неактивност се извеждат в **Markdown** файлове в **/interactions**. В хранилището има и **примерен PDF** за база данни. При нужда логиката може да се прехвърли към **Postgres** без промяна на основната идея.

Проектът използва **Docker Compose** за по-лесно тестване и разработка.

Nginx работи като **reverse proxy**, което се доближава до поведението на продукционна среда. Има скрипт **./scripts/app.bash** за стартиране на приложението и базата заедно, ако се избере такава конфигурация.

Forest Gate е **прототип**: не е готов за реална среда, не записва всичко, не е завършен, но предразполага да бъде завършен спрямо съответни изисквания. За да работят ключовите процеси, трябва да се активират **Redis keyspace notifications** (CONFIG SET notify-keyspace-events Ex), да се настроят ключовете за подписване (EC), API ключове (**OpenRouter**, **SendGrid**), и **MaxMind** база:

Email (SendGrid)

SENDGRID_API_KEY=Your_SendGrid_ApiKey

FROM_EMAIL=your_email@example.com

FROM_NAME="Your Name"

REPLY_TO_EMAIL=reply_to_email@example.com

NOTIFY_EMAIL=notify_email@example.com

Visitor HMAC

VISITOR_HMAC_KEY=32 bit HMAC key

Auth token signing (EC keys)

AUTH_EC_PRIVATE_PEM_PATH=/path/to/ec_private.pem

AUTH_EC_PUBLIC_PEM_PATH=/path/to/ec_public.pem

AUTH_ISSUER=issuer_name

AUTH_AUDIENCE=issuer_audience

LLM summaries (OpenRouter)

OPENROUTER_API_KEY=Your_Open_Router_ApiKey

OPENROUTER_MODEL=Some_Open_Router_Model

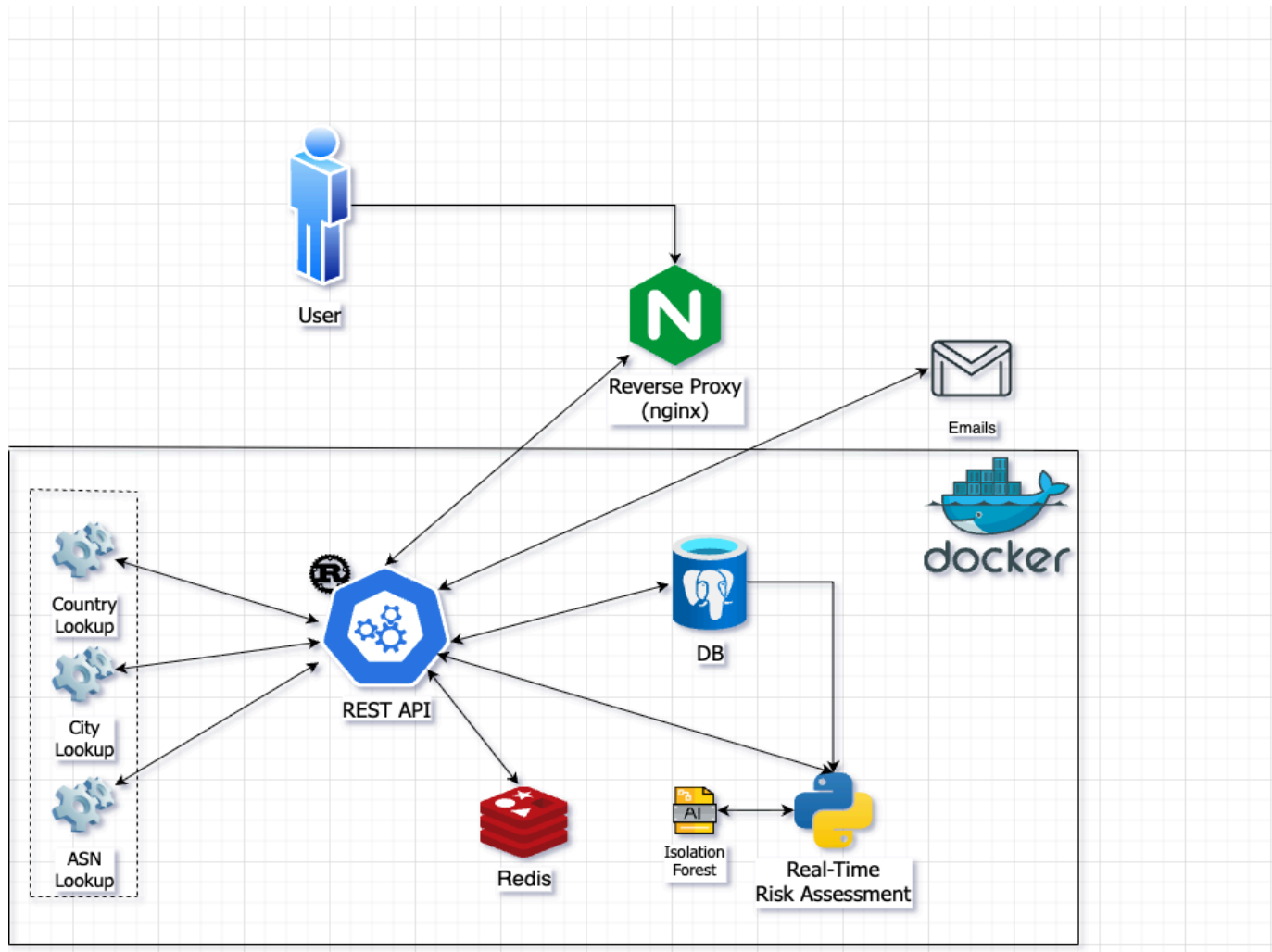
OPENROUTER_APP_NAME=Your_App_Name

(Optional) MaxMind local DB path if you use GeoLite2 locally

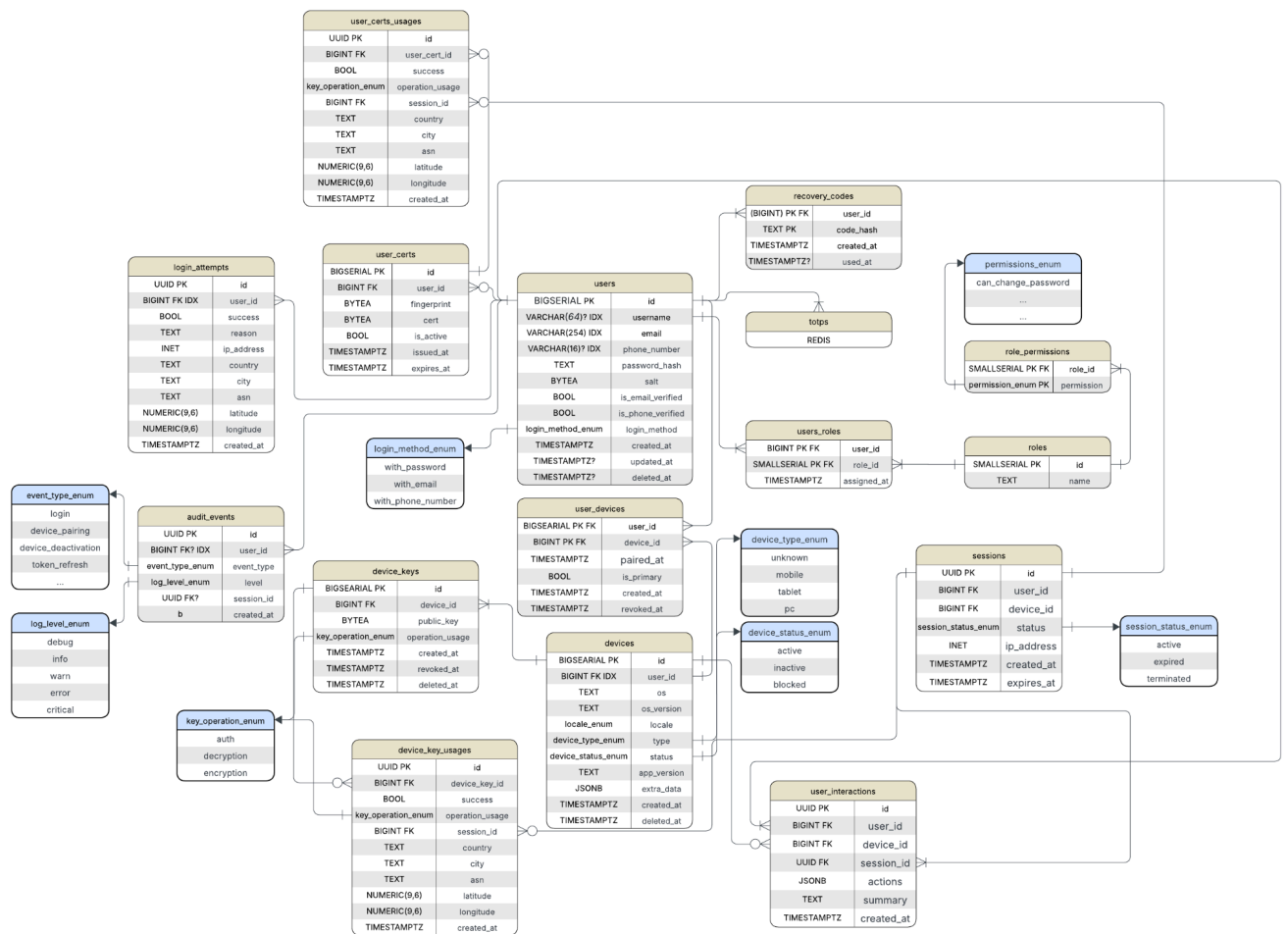
MAXMIND_DB_PATH=/path/to/GeoLite2-City.mmdb

Графики и Диаграми

Това би следвало да представлява системата:

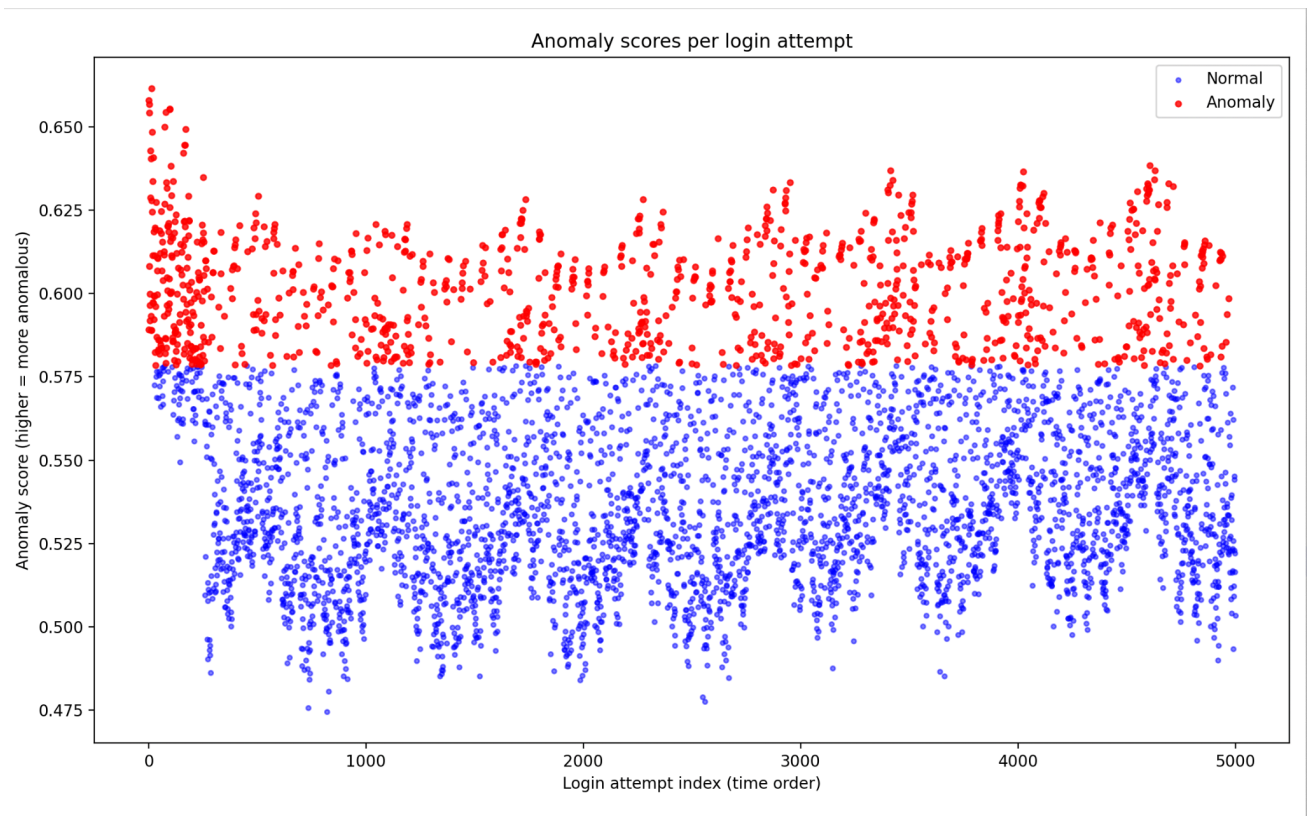


Това е примерна база данни:



За да мога да тествам **Isolation Forest**, с помощта на **GenAI** генерирах скрипт, който ми вкара данни в таблица **users** и таблица **login_attempts**. Изрично в командата, зададена на **GenAI** съм казал че скрипта искам да ми генерира данни, като **15%** от тях да са аномалии/съмнителни опити за вход в системата.

В диаграмата долу е показано как на база тези данни, **Isolation Forest** може да бъде трениран и да засече съмнителните опити:



Пример за това какво би представлявал админ панела, написан с React.js:

[← Back](#)

User Management

Manage and view all registered users

EMAIL STATUS

PHONE STATUS

LOGIN METHOD

PER PAGE

All

All

All Methods

50

Refresh

Total: 1500 users


Showing 1-50 of 1500

| ID | USERNAME | EMAIL | PHONE | EMAIL STATUS | PHONE STATUS | LOGIN METHOD | CREATED | ACTIONS |
|------|-------------------|----------------------------|--------------|--------------|--------------|----------------|------------------------|---------|
| 6914 | teodora.moore.9 | teodoramore9@outlook.com | — | | | Magic Link | Sep 25, 2025, 05:49 AM | Info |
| 6915 | ivan.ivanov.1034 | ivanivanov1034@outlook.com | +359899246 | | | Magic Link | Sep 25, 2025, 05:49 AM | Info |
| 6916 | lena.johnson.1444 | lenajohnson1444@gmail.com | — | | | Password | Sep 25, 2025, 05:49 AM | Info |
| 6917 | john.taylor.3972 | johnntaylor3972@proton.me | +49711561842 | | | OAuth Facebook | Sep 25, 2025, 05:49 AM | Info |
| 6918 | mark.miller.3936 | markmiller3936@gmail.com | +359871477 | | | OAuth Google | Sep 25, 2025, 05:49 AM | Info |

Пример за това как би изглеждало форма за регистрация на база “Forest Gate”


← Back

1 2 3
EMAIL VERIFY CREATE



Create Account
Enter your email to get started

EMAIL ADDRESS

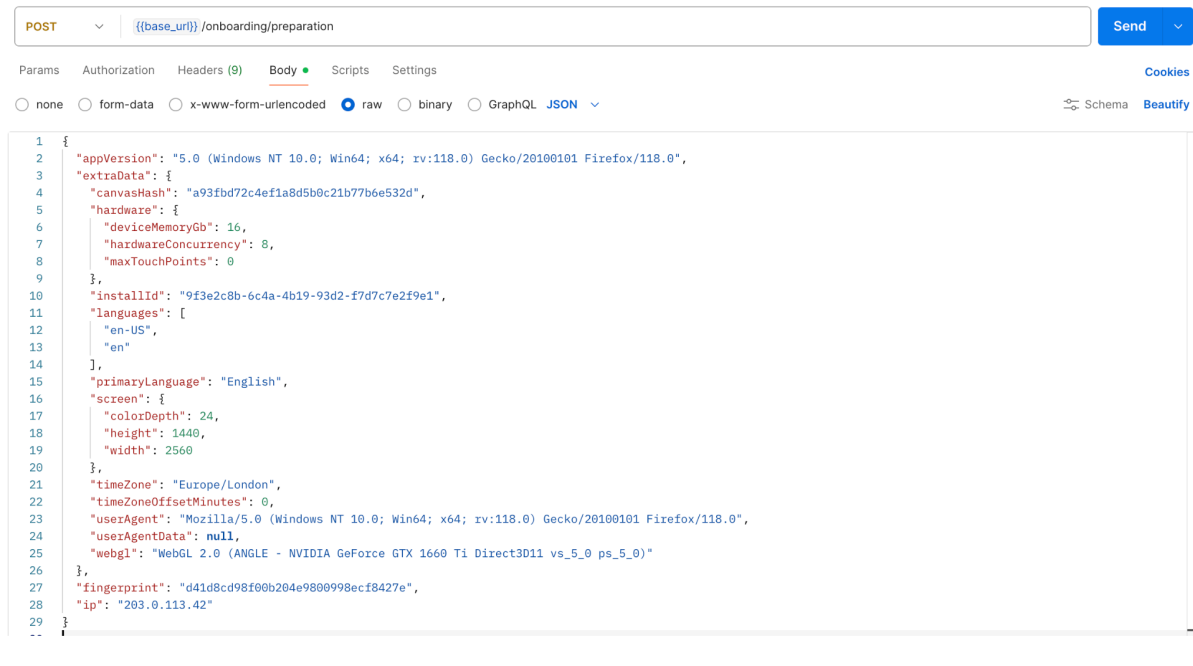


Send Verification Code →

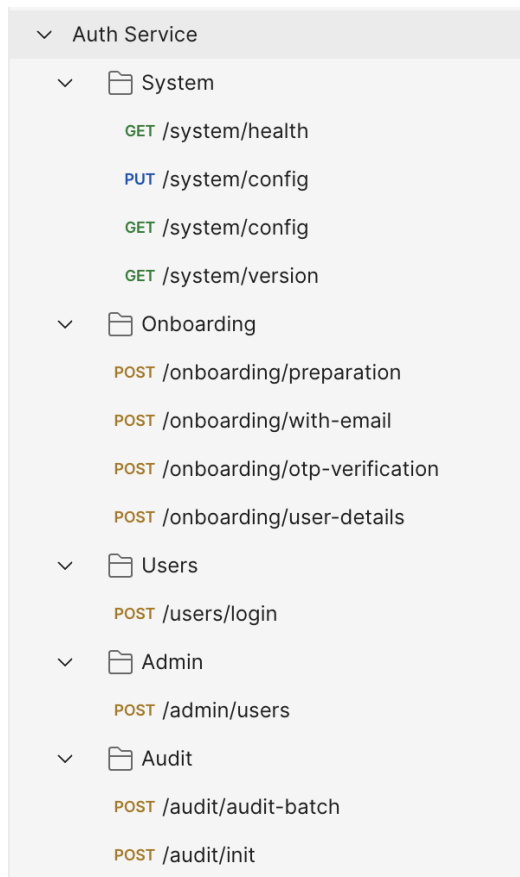
Swagger - средство, което позволява по-лесно тестване на съответните endpoint-и:

| | | |
|-------------------|------------------------------|-----|
| Auth API | RUST Actix-web and sqlx API | ^ |
| admin | | |
| POST | /admin/app-users | v |
| audit | | |
| POST | /audit/batch | 📄 v |
| POST | /audit/init | v |
| onboarding | | |
| POST | /onboarding/otp-verification | v |
| GET | /onboarding/preparation | v |
| POST | /onboarding/user-details | v |
| POST | /onboarding/with-email | v |
| system | | |
| GET | /system/config | v |
| PUT | /system/config | v |
| GET | /system/health | v |
| GET | /system/version | v |
| users | | |
| POST | /users/Login | v |

Тестване на endpoint-и с помощта на Postman:



Postman тестове (качени са в GitHub):



Разработка, срещнати проблеми и предизвикателства

Едно от най-сложните неща при създаването на каквото и да било е това да имаме ясна представа за това, което искаме. Трябва предварително да сме си го представили, да сме наясно с детайлите относно имплементацията, това колко време тя ще отнеме и да си направим изводи относно това дали инвестираното време ще бъде от полза както за нас, така и за другите. Ако нямаме ясна представа за крайната ни цел и това какво искаме да направим, то ние губим мотивация и стремеж към приключване на дадения проект.

Първо, трябва кратко, точно и ясно да се дефинира това, което се желае. Второ, трябва да се направят прототипи - било то на бази данни или код, един разработчик трябва да има кристална представа относно това как потокът от данни ще минава през приложението му. Колкото повече се мисли за това преди разработка, толкова по-малко време ще бъде загубено - толкова по-приятно ще бъде за разработчиците да завършат съответния проект. Постоянната промяна в изискванията предразполага към по-ниско качество на код.

Затова, преди изобщо да започна да пиша код, първо използвах **LucidChart**, за да създам диаграма на базата данни така, както си я представях. Още на етап проектиране промених основния модел четири пъти, защото исках да изградя структура, която да дава добър старт на всеки, който започне да разработва услуга за автентикация. Чрез **DevTools** в браузъра наблюдавах как OpenAI изпращат заявки и оттам видях идеята за **pre-auth cookie**, както и че те използват механизъм за **двуфакторна автентикация**, базиран на cookies, съхранявани в браузъра. Вдъхновен от този подход, стигнах до крайната идея за базата – всяко устройство да се съхранява още преди потребителят да е влязъл в акаунта си.

След като създадох модела на базата данни, започнах да пиша кода. Винаги е препоръчително преди да се започне, да се конфигурират основните компоненти на системата - API ключове, променливите за средата, библиотеките, които ще използваме. Един от проблемите, с които се срещнах, беше конфигурацията на **Redis** - за да може да се използва **Pub-Sub**

механизма, с който “зачиствам” събитията на потребителя от кеша, трябва или на всяко рестартиране на машината да се изпълнява тази команда:

```
redis-cli CONFIG SET notify-keyspace-events Ex
```

или в конфигурационния файл да се `redis.conf` да се сложи съответния текст:

```
notify-keyspace-events Ex
```

След като изпълних горната команда, **Pub-Sub** механизма работеше успешно.

Друг проблем беше цялостната контейнеризация на приложението. Тъй като при компилиране на програмата, библиотеката **sqlx** се опитва да се свърже с базата данни, когато базата данни и приложението са в един **docker-compose**, въпреки че изрично беше описано във файла, че приложението зависи от базата данни, възникнаха проблеми - приложението не можеше да се свърже с базата и даваше грешки. Решението беше **docker-compose** файла да се раздели на две части - едната `docker-compose.app.yml`, в който се състои **приложението** и **nginx**, а в другата част `docker-compose.db.yml` - **базата данни** и **redis**.

Цялостно, един от най-големите проблеми по време на разработката е правилната конфигурация на основните компоненти. Едва когато сме напълно сигурни, че всичко е настроено коректно, можем да имаме увереност, че вървим в правилната посока и правим нещата както трябва. **GenAI** ми помогна при създаването на кода, като аз ясно му задавах съответните команди относно това, което искам. Помогна ми при създаването на **Pub-Sub** механизма с **Redis** и конфигурацията към него, **OpenRouter** клиента. Преди да започна да използвам **GenAI**, трябваше да се запозная с основните концепции на езика Rust - концепции като **ownership**, **borrowing** - обикновено в езици от по-ниско ниво, ние не мислим за тези неща, защото имаме **Garbage Collector** - при **Rust** това е уникалното - без механизъм за почистване на паметта и без това ние

ръчно да я зачистваме (както при C free()), на база дефинирани правила на езика, той ни помага да напишем кода си без memory leaks.

Заклучение

В крайна сметка *Forest Gate* е повече от код — това е навик да мислим и да пробваме. Започваме малко, правим прототип, конфигурираме, тестваме, променяме посоката, ако трябва. Това е нормалният път. Важното е да не бързаме да „слагаме капак“ върху нещо, което още узрява. Прототипът не обещава съвършенство; той обещава честност: показва какво работи днес и какво има смисъл да развием утре.

GenAI помага, но не решава вместо нас. Той е като добър колега: дава идея, подсказва формулировка, ускорява рутинното. Понякога и греши. Нашата работа е да проверим, да четем документации, да сравняваме алтернативи и да избираме най-опростеното работещо решение. Защото отговорността е наша — ако нещо се счупи, никой модел няма да поеме вината. Затова тестовите, журналите, границите и ясните правила са повече от техника; те са отношение.

Има и още нещо, по-човешко: **времето**. Всеки трябва да знае колко време е готов да отдели — за проучване, за проби и грешки, за доизкусуряване на детайлите. Ако си дадем реалистичен период от време, работим по-спокойно и по-умно. По-добре е да направим малка, надеждна стъпка днес, отколкото голямо, крехко обещание за „утре“.

А парите? За жалост или не, повечето хора оценяват не как е свършена работата, а **дали крайният продукт работи**. Това донякъде убива красотата на занаята, но поставя реален въпрос: **как се обезпечават финансите**. Тук идват няколко практични принципа:

- **MVP преди всичко**: първо най-малкото работещо ядро, което носи стойност; красотата и удобствата могат да дойдат после.
- **Време–обхват–разход**: ако увеличим едното, другите две страдат. Ясно казваме кое фиксираме (напр. срок и бюджет) и кое е гъвкаво (обхват).
- **Технически дълг с мярка**: понякога е ОК да оставим идеалното решение за по-късно, но го **описваме** и планираме кога ще го върнем.

- **Поддръжката струва:** евтиното днес често е скъпо утре. Включваме в сметката логове, наблюдение, бекъпи, ъпдейти и сигурност.
- **Прозрачност към заинтересованите:** комуникираме докъде сме, какво струва следващата стъпка и какви са рисковете. Това пази доверието.

Парите не са враг на качеството — те са **рамка**, която ни напомня да подреждаме приоритети. Да изберем ясно **кое е важно сега**, кое може да почака, и кое изобщо да отпадне. Така продуктът се движи напред, а ние запазваме и занаятчийската чест, и финансовия реализъм.

Оттук нататък посоката е ясна: да запазим свободата да експериментираме, но и дисциплината да проверяваме. Да ползваме GenAI като ускорител, а не да се опрем сляпо. Да пазим простотата в решенията и да приемаме, че добрата система е жива — тя се учи, променя и расте заедно с нас.

GitHub “Forest Gate” - https://github.com/georgi2005atanasov/forest_gate