ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

Факултет по Приложна математика и информатика

ДИПЛОМНА РАБОТА

Тема: **„Платформа за популяризиране на независими музикални изпълнители и изграждане на общност около алтернативната музикална сцена“**

Дипломант: Михаил Илидонов Щерев

Факултетен номер: 471221045

Научен ръководител: доц. д-р Златко Захариев

Специалност: Информатика и софтуерни науки

ОКС: Бакалавър

СОФИЯ 2025

СЪДЪРЖАНИЕ

Въведение

В днешния дигитален музикален пейзаж сме изправени пред фундаментален парадокс: макар че имаме достъп до повече музика от всякога, откриването на наистина значими нови изпълнители става все по-трудно. Съвременната музикална индустрия е доминирана от алгоритмични системи за препоръки, които дават приоритет на показателите за популярност, броя на стрийминг слушанията и търговската жизнеспособност пред артистичната автентичност и иновациите. Музикалните ентусиасти, които търсят автентични, неоткрити таланти, се сблъскват със значителни бариери. Традиционните методи за откриване на музика - от разглеждане на магазини за плочи до препоръки от уста на уста - до голяма степен са заменени от алгоритмични системи, които дават приоритет на търговския успех пред артистичните качества. Това създава разминаване между страстните любители на музиката, търсещи автентични преживявания, и ъндърграунд артистите, създаващи иновативно съдържание. Нововъзникващите и ъндърграунд артисти са изправени пред безпрецедентни предизвикателства при достигането до идеалната си аудитория. Макар социалните медии да предоставят някои възможности за популяризиране, тези платформи не са специално създадени за откриване на музика и често изискват от артистите да станат създатели на съдържание, вместо да се фокусират върху занаята си. Независимите артисти се нуждаят от специални пространства, където музиката им може да бъде оценявана по артистични качества, а не по показатели за ангажираност в социалните медии.

Настоящият проект на име **TuneSpace** се цели да адресира тези критични предизвикателства, като създава цялостна платформа, специално разработена за откриване на ъндърграунд музика и изграждане на общност около нея. Смисленото откриване на музика изисква повече от сложни алгоритми - то изисква подход, ръководен от общността, който цени артистичната автентичност, истинската музикална страст и културното значение на ъндърграунд музикалните сцени.

Приложението не е просто поредната музикална платформа - то е всеобхватен социален център, който подкрепя ъндърграунд музикалната култура. Мисията на проекта е да свързва любителите на музиката с изгряващи изпълнители, преди те да станат популярни, насърчавайки общност, където автентичните таланти могат да процъфтяват, а страстните слушатели могат да открият следващата си любима група.

Платформата предоставя на потребителите:

- **Интелигентно откриване на музика**: Усъвършенствана система за препоръки с изкуствен интелект, която анализира историята на на слушане на потребителя в Spotify и комбинира данни от множество източници, включително Last.fm и MusicBrainz за да предложи изпълнители, които отговарят на вкусът на потребителя, като същевременно въвеждат смислено музикално изследване.

- **Фокус върху ъндърграунд изпълнители**: За разлика от масовите платформи, TuneSpace специално подкрепя ъндърграунд и нововъзникващи изпълнители. Алгоритмите са проектирани да откриват таланти в началото на тяхното пътуване, предоставяйки на потребителите ранен достъп до музика, преди тя да стане масова.

- **Жива общност**: Чрез интегрирана система от форуми, възможности за чат в реално време и социални функции, TuneSpace създава пространства за задълбочени музикални дискусии, връзки между изпълнители и фенове и изграждане на автентична общност около споделени музикални страсти.

- **Интеграция със събития на живо**: Системата за събития помага за свързването на изпълнители с местни фенове и популяризира сцената на живо с музика, която е от съществено значение за ъндърграунд културата.

- **Инструменти за поддръжка на изпълнители**: Освен откриването, TuneSpace предоставя на изпълнителите цялостни инструменти за изграждане на тяхната фен база, включително управление на профили, добавяне на стоки, директни канали за комуникация с фенове и промоционални възможности.

Глава 1: Цел на проекта

1.1 Към какъв тип потребители е насочен проекта?

Платформата е създадена с фокус върху две основни потребителски групи: **музикални ентусиасти** и **независими (ъндърграунд) артисти**. Взаимодействието между тях формира ядрото на цялостното потребителско изживяване.

- **1.1.1 Музикални ентусиасти**: Страстни слушатели, които копнеят за автентични, неоткрити таланти и искат да бъдат част от пътешествието на един артист от самото начало. Тези потребители участват активно в откриването на музика, споделят своите открития с общността и помагат за оформянето на кариерата на изгряващи артисти.

- **1.1.2 Независими (Ъндърграунд) артисти**: Музиканти и групи, които създават автентично изкуство и се нуждаят от платформа, за да достигнат до оценяваща публика. TuneSpace предоставя на тези артисти промоционални инструменти, директно ангажиране на феновете и общност, която цени артистичната почтеност пред търговския успех.

1.2 Отличителни характеристики на TuneSpace в сравнение с други платформи

В настоящата дигитална среда съществуват множество платформи за разпространение на музика, които доминират пазара и привличат милиони потребители. Въпреки това, много от тях често приоритизират популярни изпълнители и разчитат на алгоритмични препоръки, които често затварят слушателите в ограничен кръг от добре познати имена и жанрове. В този контекст TuneSpace се отличава, като поставя в центъра на вниманието независимите артисти и музикалните ентусиасти, които търсят свежи, оригинални и неоткрити звуци.

Платформата съчетава социални и аналитични инструменти, които не само улесняват откриването на нова музика, но и създават възможности за по-директна и смислена комуникация между артистите и техните фенове. Интеграцията на иновативни

препоръчителни системи, базирани на напреднали алгоритми и изкуствен интелект, позволява персонализирано изживяване, което е адаптирано към индивидуалните музикални предпочитания и активност на потребителите.

Този подход прави TuneSpace предпочитана среда както за музикалните ентусиасти, които искат да бъдат активни участници в процеса на откриване и подкрепа на нови таланти, така и за независимите артистите, които търсят реална подкрепа, видимост и възможности за развитие извън традиционните канали и ограничения на големите музикални индустрии.

Глава 2: Технологии и инструменти за разработка на приложението

Приложението е изградено с помощта на модерен технологичен стек и архитектура, които осигуряват мащабируемост, производителност и поддръжка.

2.1 Използвани технологии – Фронтенд

2.1.1 Next.js - Готова за производство React програмна рамка, която предоставя цялостно решение за изграждане на модерни уеб приложения. Тя предлага рендериране от страна на сървъра (SSR), генериране на статични сайтове (SSG), API маршрути, автоматично разделяне на код, вградена CSS поддръжка и маршрутизиране на база на файлове. Next.js оптимизира производителността чрез функции като оптимизация на изображения, автоматично разделяне на пакети и предварително извличане [4].

2.1.2 React - Декларативна, базирана на компоненти JavaScript библиотека за изграждане на потребителски интерфейси. React използва виртуален DOM за ефективни актуализации, поддържа на повторна употреба на компоненти и следва еднопосочен модел на поток от данни. Тя позволява на разработчиците да създават сложни потребителски интерфейси от малки, изолирани части код, наречени компоненти. Екосистемата на React включва богата общност от библиотеки на трети страни [5].

2.1.3 TypeScript - Строго типизиран език, който разширява и се компилира до обикновен JavaScript код. Той въвежда статична типизация, която позволява откриване на грешки още на етапа на компилация, а не чак по време на изпълнение. TypeScript е особено полезен при разработката на сложни приложения, където сигурността и яснотата на типовете са от ключово значение за поддръжката и стабилността на кода [9].

2.1.4 Zustand - Малка, бърза и мащабируема state-management библиотека, която предоставя минималистичен API за управление на глобално състояние без boilerplate код. Поддържа middleware, persist и devtools интеграция [16].

2.1.5 TanStack Query (React Query) - Мощна библиотека за извличане на данни с вградени фонови актуализации, автоматично кеширане, оптимистични актуализации и синхронизация между разделите [17].

2.1.6 Axios - популярен HTTP клиент, базиран на Promises, който улеснява извършването на заявки към отдалечени сървъри от браузъра или Node.js среда. Той предоставя широк набор от функционалности, включително прехващачи на заявки и отговори (interceptors), автоматично преобразуване на отговорите във формат JSON, поддръжка на таймаути, сериализация на данни, както и механизми за защита срещу CSRF атаки чрез автоматично изпращане на токени. В сравнение с вградения fetch API, Axios предлага по-богата функционалност, по-добра поддръжка на стари браузъри и по-удобен синтаксис за обработка на грешки и конфигурация на заявки [18].

2.1.7 React Hook Form - Ефективна, гъвкава React форма с лесна валидация, минимални повторни изобразявания, вградена валидация, интеграция с UI библиотеки и отлична поддръжка на TypeScript [19].

2.1.8 Hookform Resolvers - Помощни модули за React Hook Form, които осигуряват безпроблемна интеграция с външни библиотеки за валидация като Yup, Joi, Zod и други. Те служат като междинен слой между React Hook Form и съответната библиотека, позволявайки използването на съществуващи схеми за валидация в рамките на формата. Това улеснява валидирането на данни и допринася за по-ясна и поддържаема архитектура на формулярите [20].

2.1.9 Zod – TypeScript-базирана библиотека за валидиране и дефиниране на схеми, която осигурява валидация в реално време (по време на изпълнение), съчетана с автоматичен извод на типове. За разлика от други решения, Zod е проектиран с пълна съвместимост с TypeScript, което позволява дефиниране на схемата веднъж, като от нея директно се извеждат съответните типове. Това значително намалява дублирането на код и повишава типовата безопасност. Подходящ е за валидиране на входни данни от формуляри, API отговори, конфигурационни файлове и други случаи, при които е необходима надеждна и строго типизирана валидация [21].

2.1.10 Leaflet - JavaScript библиотека с отворен код за интерактивни карти [23].

2.1.11 React Leaflet - React компоненти за Leaflet карти с декларативен подход за създаване на карти в React приложения. Предоставя инструменти за управление на жизнения цикъл и интеграция с React състояние [24].

2.1.12 Tailwind CSS - CSS рамка, която предоставя богата колекция от малки, преизползваеми класове за стилове като разстояния, цветове, типография, подредба и други. Вместо да предлага предварително дефинирани компоненти, Tailwind позволява създаване на персонализиран дизайн директно в HTML структурата, чрез комбиниране на тези класове. Това улеснява бързото прототипиране, насърчава създаването на последователни дизайнерски системи и премахва нуждата от писане на големи количества ръчен CSS [11].

2.1.13 PostCSS - Инструмент за трансформиране на CSS с помощта на JavaScript плъгини [41].

Допълнителни помощни пакети - Embla Carousel [33], React Day Picker [34], Sonner [35], Date-fns [36]

2.2 Използвани технологии – Бакенд

2.2.1 C# - Език за програмиране на високо ниво с общо предназначение, разработен от Microsoft, който работи в екосистемата .NET и поддържа множество парадигми. Той обхваща статично типизиране, силно типизиране, лексикално ограничено, императивно, декларативно, функционално, обобщено, обектно-ориентирано и компонентно-ориентирано програмиране [42].

2.2.2 ASP.NET Core - Модерна, междуплатформена, високопроизводителна рамка на Microsoft за изграждане на уеб приложения и API. Поддържа множество среди за внедряване (Windows, Linux, macOS), осигурява отлични характеристики на производителност, включва вградени функции за сигурност и предлага богата колекция от инструменти [6].

2.2.3 Entity Framework Core - Разширяема и междуплатформена рамка за обектно-релационно картографиране (ORM) на Microsoft. EF Core предоставя възможности за LINQ заявки, проследяване на промени, миграции на бази данни, обединяване на връзки и поддръжка на множество бази данни. Абстрахира операциите с базата данни в силно типизирани C# обекти, обработва генерирането на SQL, управлява връзките с базата данни и предоставя механизми за кеширане. EF Core поддържа подходите Code First и Database First, което позволява на разработчиците да работят с бази данни, използвайки .NET обекти, а не суров SQL [7].

2.2.4 PostgreSQL - Усъвършенствана релационна система за бази данни с отворен код, известна със своята надеждност, устойчивост на функции и производителност. PostgreSQL поддържа разширени типове данни, търсене на пълен текст, поддръжка на JSON/JSONB, персонализирани функции и обширни опции за индексиране. Npgsql е доставчикът на .NET данни за PostgreSQL, предлагащ високопроизводителна свързаност, асинхронни операции, групови операции и пълна съвместимост с Entity Framework Core. Разширяемостта на PostgreSQL го прави идеален за приложения, изискващи сложни заявки и взаимовръзки между данни [8].

2.2.5 Pgvector.EntityFrameworkCore - Доставчик на Entity Framework Core за разширението pgvector, което добавя възможности за търсене на векторно сходство към PostgreSQL. Това позволява съхраняване и заявки за високоразмерни вектори за AI/ML приложения, включително вграждания за обработка на естествен език, търсене на сходство на изображения и системи за препоръки. Pgvector поддържа различни функции за разстояние (L2, косинус, вътрешно произведение), индексиране за бързо търсене на сходство и интеграция с работни процеси за машинно обучение [30] [31].

2.2.6 Microsoft SignalR - Библиотека за .NET, която улеснява внедряването на функционалности в реално време за уеб приложения. Тя позволява двупосочна комуникация между сървъра и клиента, използвайки WebSockets, Server-Sent Events или дълго запитване (long polling), като автоматично избира най-подходящата технология в зависимост от поддържаните възможности [22].

2.2.7 ASP.NET Core Identity - гъвкава и разширяема система за управление на потребителска идентификация и достъп в приложения, изградени с ASP.NET Core. Тя предоставя вградена поддръжка за регистрация, вход, управление на роли и потребителски заявки (claims), както и интеграция с външни доставчици за удостоверяване като Google, Facebook, Microsoft и други. Системата включва важни механизми за сигурност, като хеширане и съхранение на пароли, заключване на акаунти след неуспешни опити за вход, потвърждение на имейл и поддръжка на двуфакторно удостоверяване (2FA). Благодарение на високата си степен на конфигурируемост и вградените шаблони, ASP.NET Core Identity е подходящ избор както за малки, така и за мащабни приложения, изискващи надеждна автентикация и авторизация [25].

2.2.8 JWT Bearer Authentication - Механизъм за удостоверяване в ASP.NET Core, базиран на JSON Web Tokens (JWT), който позволява безсъстояние (stateless) управление

на достъп до уеб приложения и API услуги. Вместо да съхранява на сървъра на сесии, удостоверяването се извършва чрез подписани токени, които клиентът изпраща с всяка заявка. Тази схема поддържа валидност на токени, използвайки потребителски заявки (персонализирани искове), конфигурация на правила за достъп и лесна интеграция с външни доставчици за идентичност, като OAuth2 и други [26].

2.2.9 ONNX Runtime - Високопроизводителен мултиплатформен инструмент за изпълнение на машиннообучителни модели, базиран на отворения формат ONNX (Open Neural Network Exchange). Той позволява използване на модели, създадени с различни ML рамки. ONNX Runtime е оптимизиран за бързо и ефективно извеждане (inference) както на CPU, така и на GPU, с минимално потребление на ресурси и ниска латентност [28].

2.2.10 Microsoft ML Tokenizers - Библиотека за токенизиране, предназначена за обработка на текстови данни в машиннообучителни приложения. Тя предоставя предварително обучени токенизатори, които поддържат различни езици и модели, улеснявайки разделянето на текст на по-малки единици (токени) за последваща обработка [29].

2.2.11 FluentEmail - .NET библиотека за изпращане на имейл-и с Fluent API за създаване на имейли и поддръжка на шаблони, обработка на прикачени файлове и различни доставчици (SMTP, SendGrid и др.) [32].

2.2.12 HTTP Resilience - Библиотеки за устойчива HTTP комуникация с правила за повторен опит, прекъсвачи, обработка на изчакване и ограничаване на скоростта за HTTP заявки [43].

2.3 Среди за разработка

2.3.1 Visual Studio - интегрирана среда за разработка (IDE), разработена от Microsoft. Използва се за разработване на компютърни програми, включително уебсайтове, уеб приложения, уеб услуги и мобилни приложения [1].

2.3.2 Visual Studio Code - редактор на изходен код, разработен от Microsoft, известен със своята скорост, гъвкавост и широк набор от функции. Той работи на Windows, macOS и Linux, което го прави междуплатформен инструмент, подходящ за разработчици в

различни среди. Проектиран да бъде лек, но мощен, VS Code включва функции като персонализируем интерфейс, интегриран терминал, възможности за отдалечена разработка и поддръжка за работни процеси [2].

2.3.3 pgAdmin - PostgreSQL Tools - графичен инструмент (GUI) с отворен код за управление на бази данни PostgreSQL. Това е официалното приложение, препоръчано от PostgreSQL и е създадено да улесни работата с бази данни чрез визуален интерфейс. Приложението предлага удобен SQL редактор с функции като автоматично довършване на код, оцветяване на синтаксиса и визуализиране на планове за изпълнение на заявки. pgAdmin е подходящ както за разработчици, които работят с PostgreSQL, така и за администратори на бази данни, които се нуждаят от инструменти за мониторинг, управление на права за достъп и резервно копиране [3].

Глава 3: Анализ на изискванията към проекта

3.1 Функционални изисквания към проекта

3.1.1 Публична част (за крайни потребители)

Чрез публичната част, клиентите имат възможност за:

3.1.1.1 Управление на потребителски профил:

- Регистрация с верификация по имейл
- Вход чрез имейл/парола или Spotify OAuth
- Редакция на профилна информация и качване на профилна снимка
- Проследяване на последователи и последвани

3.1.1.2 Откриване и препоръки за музика:

- Получаване на AI-базирани препоръки чрез векторни технологии за вграждане и специални алгоритми за филтрация и оценяване
- Интеграция със Spotify за разглеждане и споделяне на музика
- Персонализирани препоръки на база история на слушане
- Разглеждане по жанрове и откриване на нови изпълнители и групи

3.1.1.3 Управление на банди и артисти:

- Създаване и редактиране на профил на банда
- Управление на членове и присъединяване
- Създаване и промотиране на музикални събития
- Управление на мърчандайз
- Комуникация с фенове чрез специални съобщения

3.1.1.4 Управление на събития:

- Създаване на събития с локация и подробности
- Филтриране и търсене на събития по място и дата
- Промотиране на събития чрез вградени инструменти
- Визуализация на локации чрез вградена карта

3.1.1.5 Комуникационна система:

- Чат между потребители в реално време
- Канали за комуникация между банди и фенове
- Известия за важни събития и съобщения

3.1.1.6 Социални функции и общност:

- Създаване и участие във форумни дискусии
- Харесване и отговаряне на публикации
- Дискусии, фокусирани върху конкретни банди

3.1.1.7 Потребителски интерфейс и достъпност:

- Разбираем и достъпен интерфейс (интуитивен UI/UX дизайн)
- Поддръжка на мобилни устройства (mobile responsive)

3.2 Нефункционални изисквания към проекта

3.2.1 Сигурност

Системата трябва да осигурява сигурни механизми за удостоверяване и да внедрява контрол на достъпа. Трябва да използва сигурно управление и хеширане на пароли и да защитава поверителността на потребителите. Личната информация трябва да бъде обработена сигурно и да поддържа подходящи методи за съхранение на данни.

3.2.2 Производителност

Системата трябва да осигурява бързо време за реакция при потребителски взаимодействия, да оптимизира операциите с базата данни и да управлява системните ресурси ефективно. Платформата трябва да поддържа мащабируема архитектура. Системата трябва да внедрява ефективни стратегии за кеширане, да оптимизира доставянето на съдържание и да гарантира адаптивно потребителско изживяване във всички функции.

3.2.3 Поддръжка

Системата трябва да предоставя изчерпателна API документация и документация на ниво код за да се улесни бъдещата поддръжка и нужните промени или подобрения от страна на разработчиците.

3.2.4 Употреба и достъп

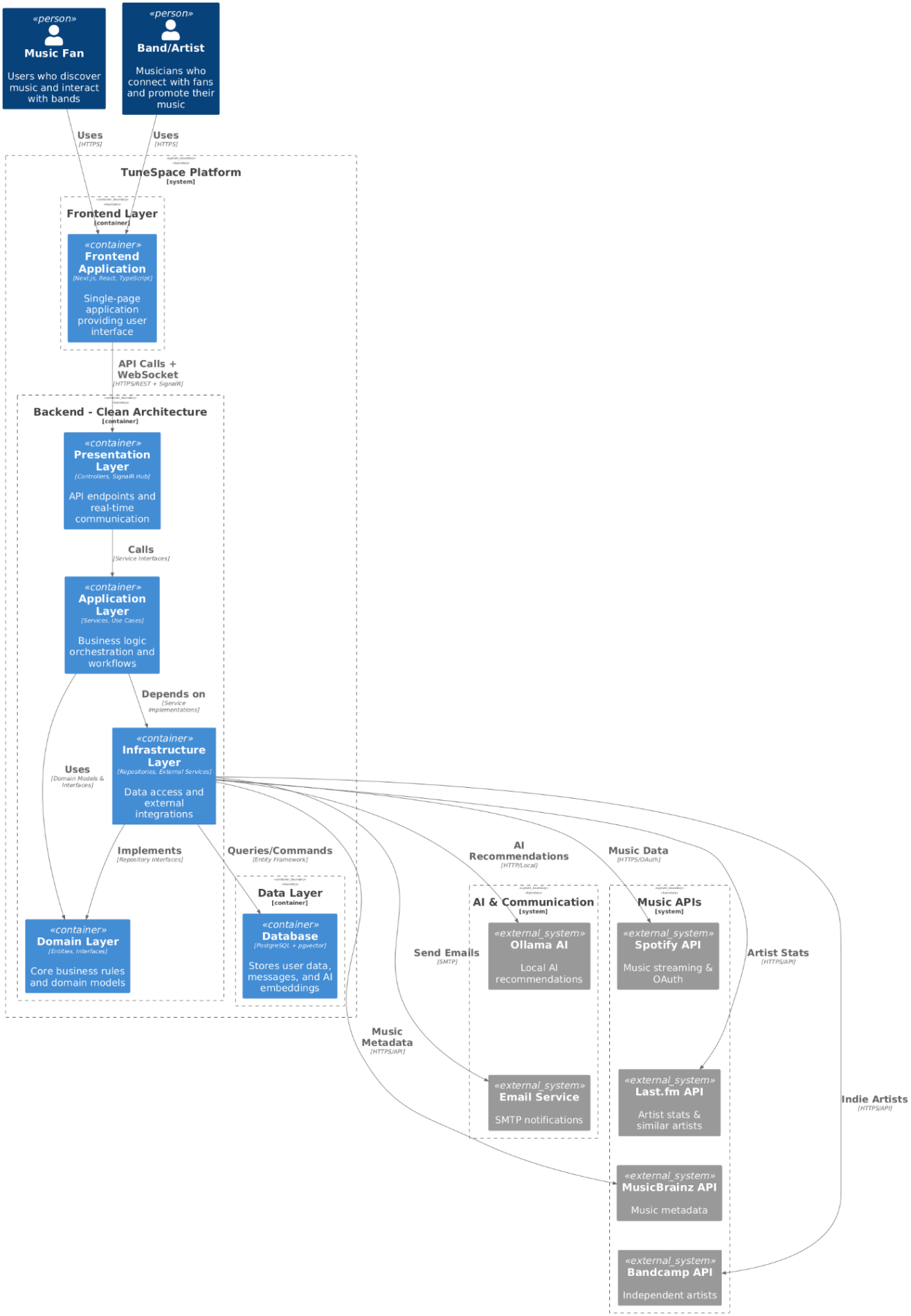
Системата трябва да отговаря на стандартите за уеб достъпност, да поддържа алтернативни методи за навигация, да е съвместима с помощни технологии и да се адаптира към различни размери на екраните и устройства. Платформата трябва да предоставя интуитивни потребителски интерфейси, да поддържа последователен дизайн на всички устройства, да осигурява бързо зареждане на страниците и безпроблемно взаимодействие с потребителите.

3.2.5 Съвместимост

Системата трябва да поддържа съвременни уеб браузъри, да функционира на мобилни платформи, да внедрява съвременни уеб стандарти и да работи на различни операционни системи. Платформата трябва да поддържа различни типове устройства. Системата трябва да следва стандартните за индустрията API практики, да използва стандартни формати на данни, да спазва протоколи за удостоверяване и да се интегрира надеждно с външни услуги.

3.2.6 Архитектура на проекта

Проекта следва модела на чистата архитектура (**Clean Architecture**) с ясно разделяне на отговорностите и принципи за инверсия на зависимости. Чистата архитектура е модел на софтуерен дизайн, който насърчава разделянето на отговорностите чрез организиране на кода в отделни слоеве с ясни зависимости, насочени навътре към основната бизнес логика. Архитектурата се състои от концентрични кръгове, където най-вътрешният слой съдържа бизнес правила на предприятието (обекти), заобиколен от бизнес правила на приложенията (случаи на употреба), след това интерфейсни адаптери (контролери, презентатори) и накрая рамки на най-външния слой. Ключовият принцип е правилото за зависимости: зависимостите на изходния код трябва да сочат навътре, което означава, че вътрешните слоеве не трябва да зависят от външните слоеве, което позволява по-голяма гъвкавост, тестваемост и поддръжка. Следвайки чистата архитектура, може да създават приложения, които са независими от рамки, бази данни и силно модулни, което ги прави по-лесни за модифициране, разширяване и поддръжка с течение на времето [37].



Фигура 3.1. Преглед на системната архитектура

Отговорности на всеки слой

1. Слой на представяне (**Presentation Layer)** / **TuneSpace.Api + Frontend**

- API контролери: Обработват HTTP заявки и отговори
- SignalR хъбове: Крайни точки за комуникация в реално време
- Middleware: Удостоверяване, регистриране, обработка на грешки
- Frontend компоненти: React компоненти и страници
- Управление на състоянието: Състояние от страна на клиента и кеширане

2. Слой на приложението (**Application Layer)** / **TuneSpace.Application**

- Услуги: Имплементация на бизнес логика
- Случаи на употреба: Специфични за приложението бизнес правила
- Фонови услуги: Дълго изпълняващи се задачи и планирани задачи

3. Слой на домейна (**Domain Layer)** / **TuneSpace.Core**

- Обекти: Основни бизнес обекти
- Интерфейси: Договори за услуги и хранилища
- Изключения: Персонализирани бизнес изключения
- Модели: Обекти на стойности и модели на домейни

4. Слой на инфраструктурата (**Infrastructure Layer) / TuneSpace.Infrastructure**

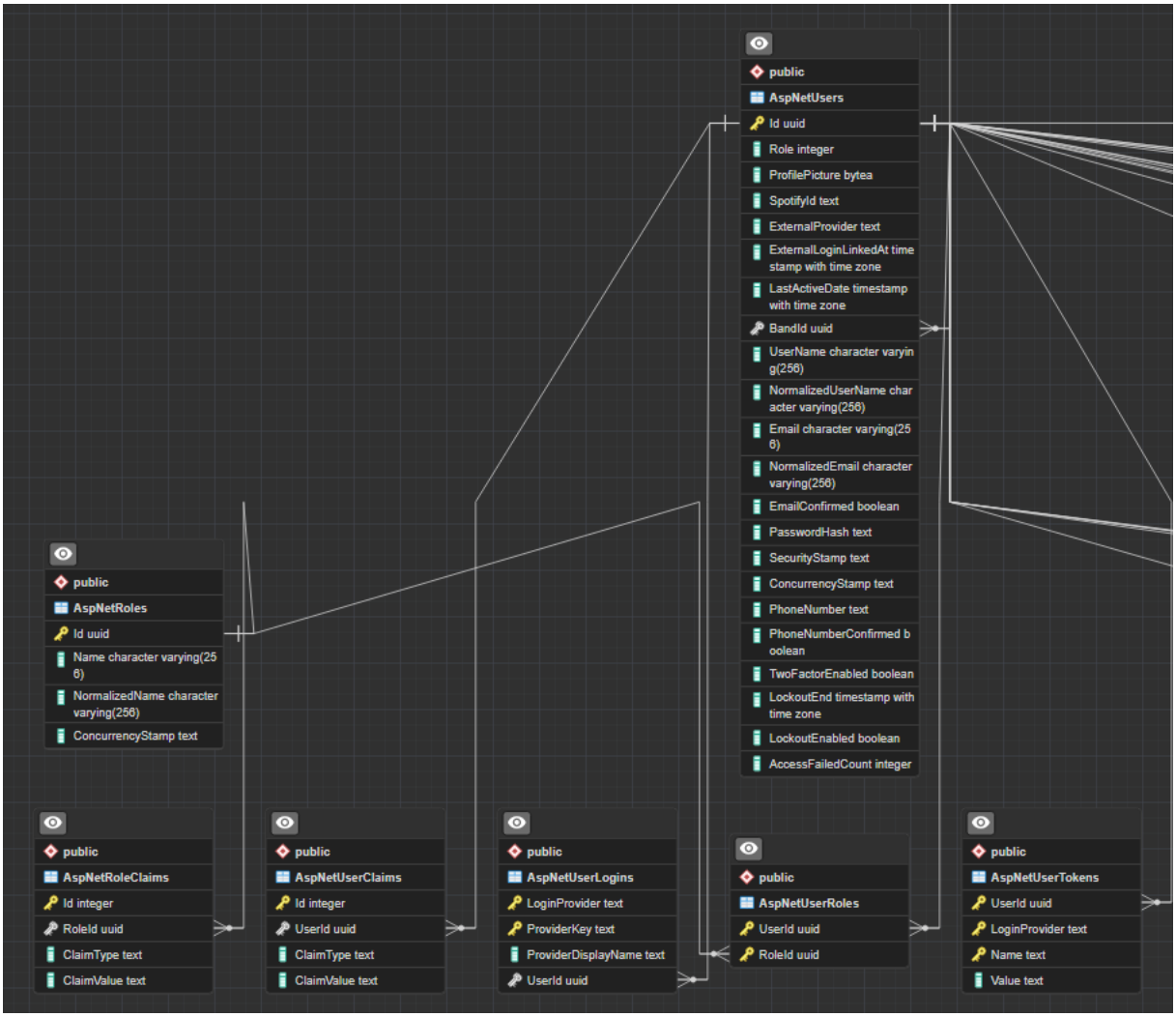
- Хранилища: Имплементации на достъп до данни
- Контекст и връзка с базата данни: Конфигурация на **Entity Framework**
- Външни услуги: API интеграции на трети страни
- Услуги за идентичност: удостоверяване и оторизация

3.2.7 Структура на базата данни

PostgreSQL е базата данни на приложението поради изключителната си комбинация от разширени функции, които директно отговарят на сложните изисквания на платформата. Като надеждна релационна база данни с отворен код, PostgreSQL осигурява ACID (Atomicity, Consistency, Isolation, Durability) съответствие и силна цялост на данните, от съществено значение за управление на данните в приложението. Най-важното е, че

разширението pgvector на PostgreSQL предоставя вградени възможности за търсене на векторно сходство, което го прави идеален избор за системата за музикални препоръки, задвижвана от изкуствен интелект, която разчита на векторни вграждания, за да съпоставя потребителите с подходящи изпълнители и групи. Базата данни се състои от 28 основни таблици, които поддържат цялостните функции на платформата.

Основно управление на потребителите



Фигура 3.2. Таблици на основното управление на потребителите

Таблицата „Потребители“ (AspNetUsers) служи като основа, изградена върху ASP.NET Identity, съдържаща данни за удостоверяване на потребителите, информация за профила, подробности за интеграцията със Spotify, роли и снимки на профила. Всеки потребител може да бъде свързан с група и има различни социални връзки в платформата.

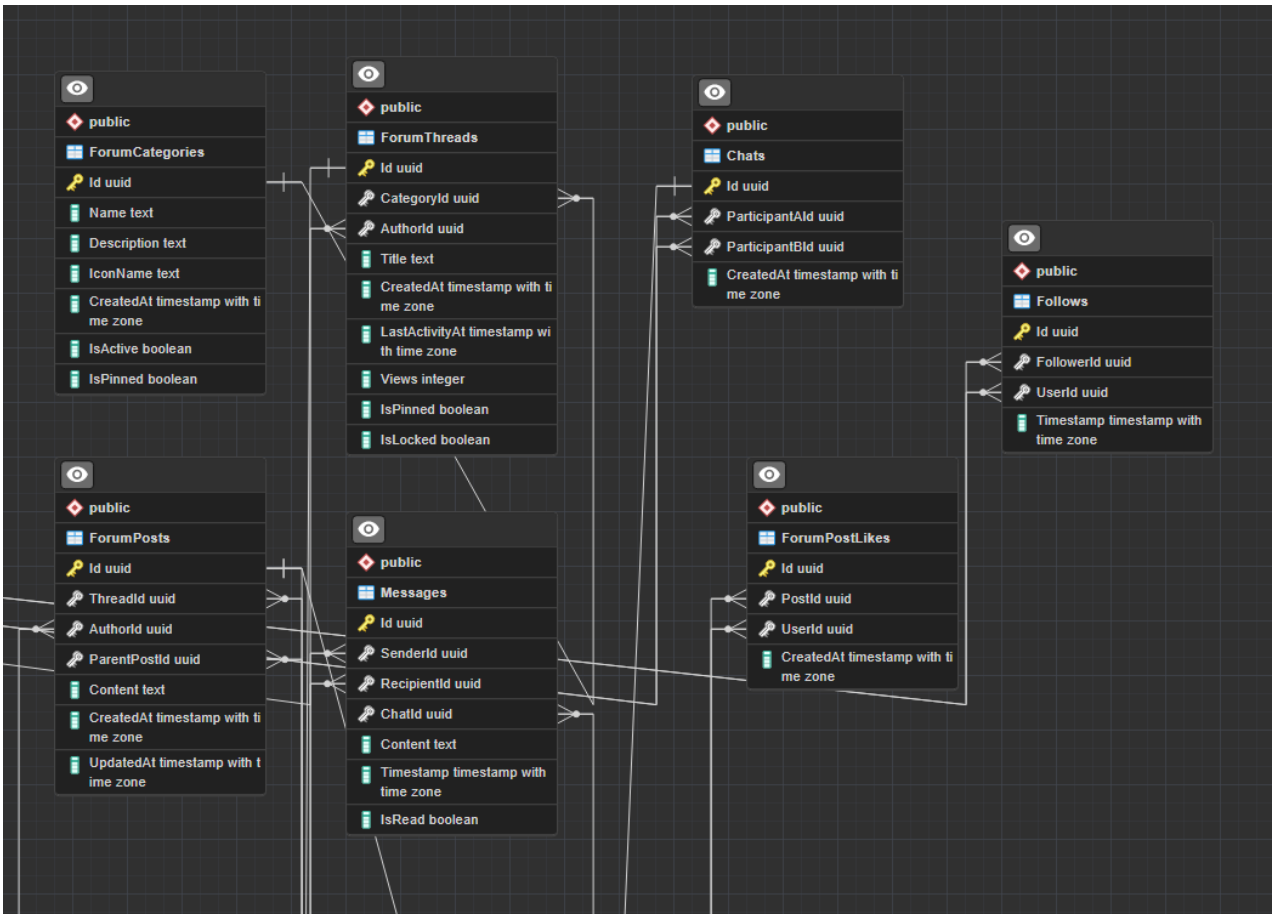
Екосистема за музика и групи

Фигура 3.3. Таблици на екосистема за музика и групи

Таблицата „Bands“ съхранява регистрирани музикални групи с техните метаданни, включително име, описание, жанр, местоположение, изображение за корица и интеграции с външни платформи (Spotify, YouTube). Групите могат да имат множество членове чрез връзка с външен ключ, събития, стоки и последователи. MusicEvents са свързани с групите и съдържат информация за концерти/представления с подробности за

мястото на провеждане, дати, цени на билети и координати за местоположение. Артикулите на стоки са свързани с групите и съхраняват информация за продукти.

Социални функции и комуникация



Фигура 3.4. Таблици на социалните функции и комуникация

Платформата поддържа множество комуникационни канали:

- Чатовете улесняват директния обмен на съобщения между потребители.
- Съобщенията съхраняват действителното съдържание на разговора между потребителите в специфични чатове.
- BandChats позволяват комуникация между потребители и групи.
- BandMessages съдържат съобщенията в рамките на разговорите в групите, поддържайки комуникация както между потребители, така и между групи.

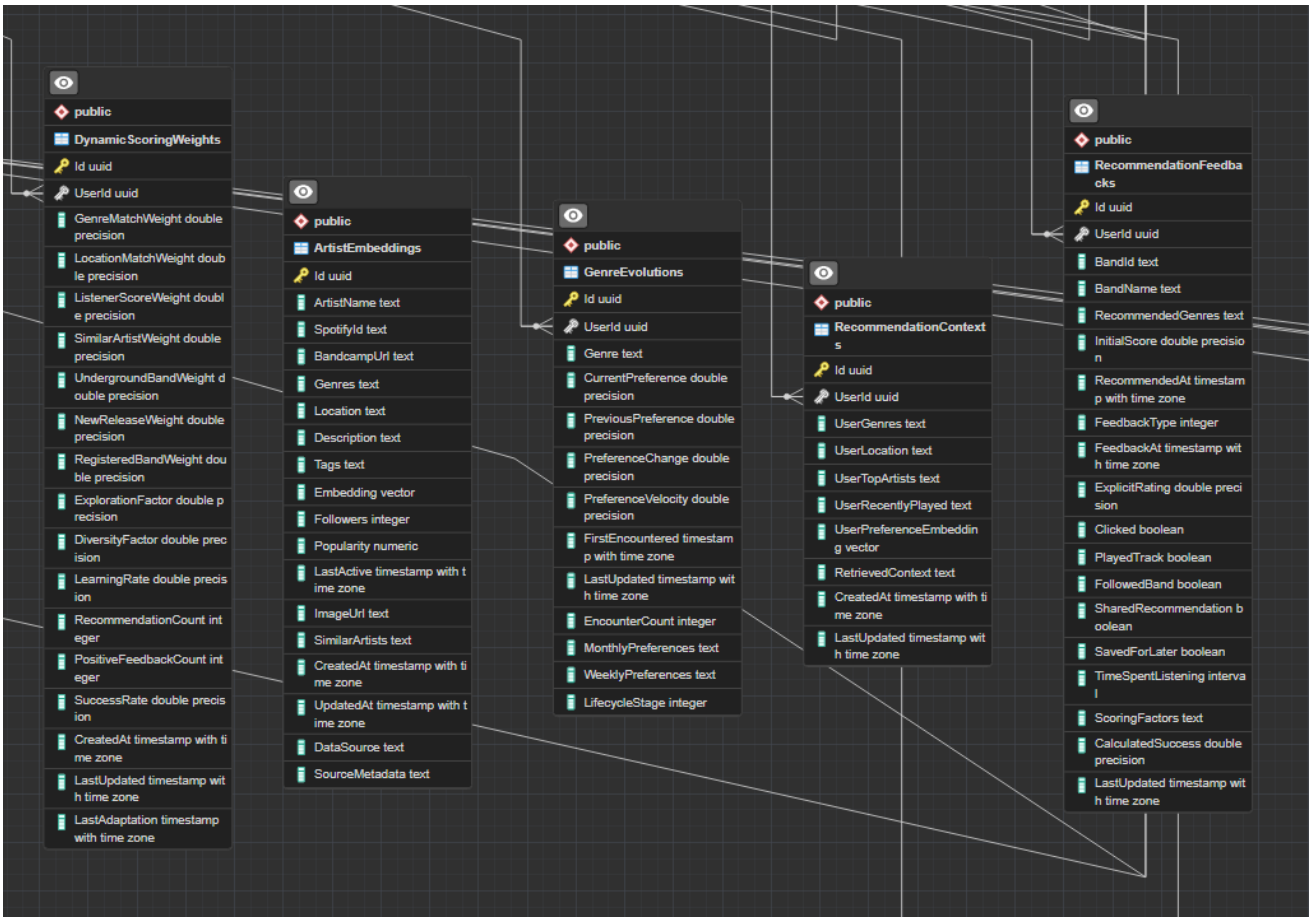
„Follows“ съдържа връзките „следване“ между потребители, а „BandFollows“ между потребители и групи, като и двете имат времеви отметки за проследяване на активността.

Форум система

Структурата на форумите се състои от йерархични дискусии:

- ForumCategories организира темите с имена, описания, икони и административни флагове
- ForumThreads принадлежат към категории и съдържат теми за дискусии с брой преглеждания
- ForumPosts поддържа вложени разговори с отношения родител-дете за отговори
- ForumPostLikes проследява ангажираността на потребителите с отделни публикации

Откриване на музика



Фигура 3.5. Таблици за откриването на музика

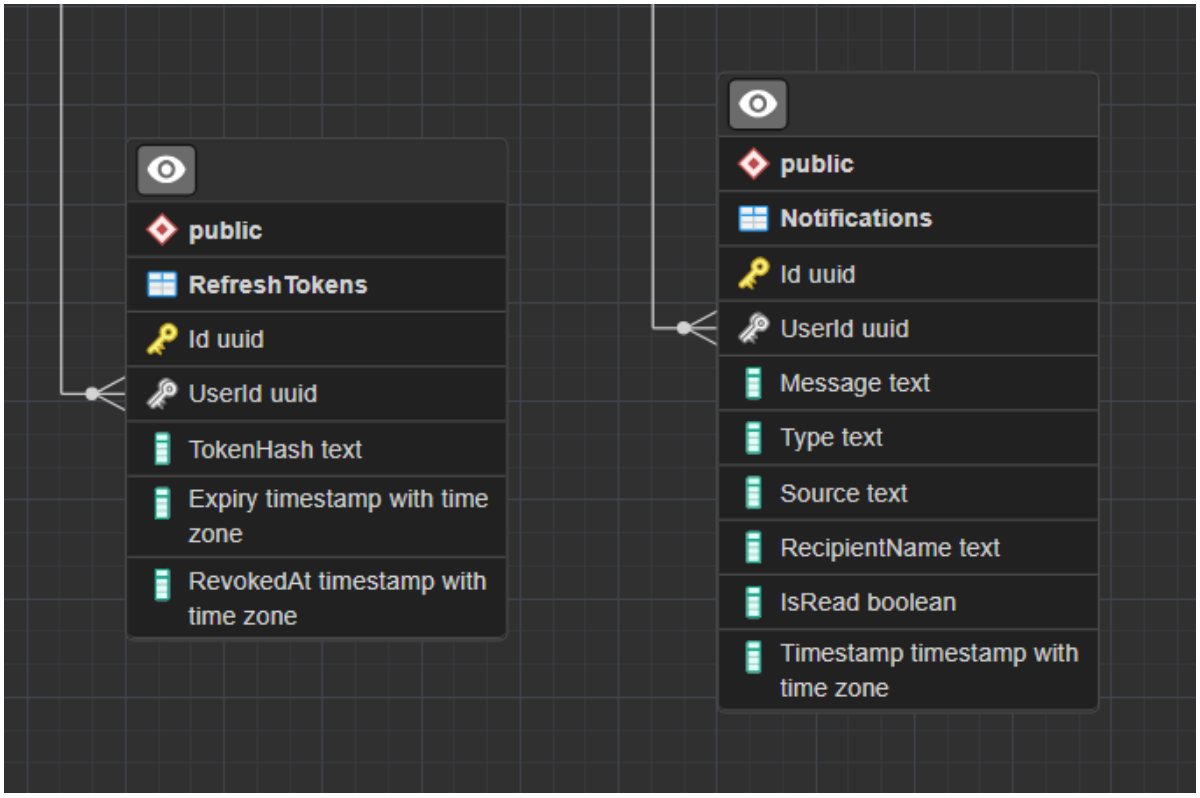
Платформата разполага с усъвършенствани възможности за препоръчване на музика:

- ArtistEmbeddings съхранява векторни представяния на изпълнители, използвайки разширението pgvector на PostgreSQL, съдържащо 384-измерни вграждания за

съпоставяне на сходства, заедно с метаданни за изпълнителите, жанрове и информация за външни платформи

- RecommendationContext съдържа данни за предпочитанията на потребителите за жанрове, най-добри изпълнители, наскоро слушани песни и персонализирани вграждания на предпочитания
- DynamicScoringWeights адаптира алгоритмите за препоръки въз основа на обратната връзка от потребителите с регулируеми параметри за различни фактори на препоръки
- GenreEvolution проследява как предпочитанията на потребителите се променят с течение на времето
- RecommendationFeedback улавя взаимодействията на потребителите с препоръки

Системна инфраструктура



Фигура 3.6. Таблици на системната инфраструктура

RefreshTokens таблицата обработва сигурно удостоверяване с управление на токени, проследяване на изтичане и възможности за отмяна за подобрена сигурност, а Notifications таблицата осигурява системна комуникация с потребителите относно важни събития, актуализации и социални взаимодействия.

Глава 4: Техническа реализация

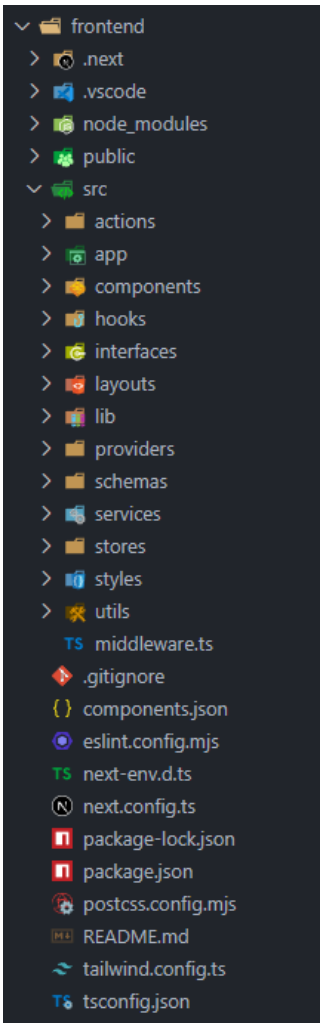
4.1 Структура на проекта

Проектът следва добре организирана структура, която разделя отговорностите и насърчава поддръжката. Състои се от няколко главни директории:

- **frontend**
- **backend**
    - **TuneSpace.Api**
    - **TuneSpace.Application**
    - **TuneSpace.Core**
    - **TuneSpace.Infrastructure**
    - **TuneSpace.Tests**

4.1.1 Frontend

Директорията съдържа папки и файлове, които се отнасят за клиентската част на приложението.



Фигура 4.1: Frontend директория

.next - Папка с резултатите от билд процеса на Next.js и кеш файлове. Съдържа компилирани файлове на приложението, билд манифести и кеш за development сървъра. Тази папка се генерира автоматично по време на разработка и билд и не трябва да се променя ръчно.

node_modules - Папка с всички инсталирани npm пакети и техните зависимости, дефинирани в package.json. Генерира автоматично при изпълнение на npm install или yarn install.

.vscode - Настройки и конфигурации за Visual Studio Code.

.public - Статични ресурси, които се обслужват директно от уеб сървъра

src - Основна директория със сорс код:

- **actions -** Сървърни действия и логика за обработка на формуляри в Next.js сървърни компоненти.

- **app –** Маршрути, лейаути за Next.js App Router и отделните страници на приложението
- **components -** Самостоятелни React компоненти, изградени с TypeScript, които използват JSX (TSX) синтаксис и могат да бъдат лесно използвани повторно в различни части на потребителския интерфейс на приложението.
- **hooks -** Папката съдържа потребителски (custom) React hooks – това са функции, които улесняват използването на React функционалности като състояние (state) и ефекти. Основната им цел в проекта е да извличат данни от сървъра и да управляват тази информация в компонентите.
- **interfaces -** Типове и интерфейси на TypeScript
- **layouts -** Лейаути, които обгръщат страниците с общи UI елементи (например странична лента, хедър)
- **lib –** Папка с помощни библиотеки
- **providers -** React context providers
- **schemas –** Zod схеми за валидация
- **services –** Файлове с API функции за комуникация с бекенда
- **stores –** zustand stores/съхранители
- **styles -** Глобални CSS стилове с Tailwind CSS.
- **utils -** Помощни функции и константи, като API endpoints и дефиниции на маршрути.

Основни конфигурационни файлове:

.env – Файл с променливи на средата за приложението

.gitignore – Файлове и папки, изключени от версия контрола на Git

components.json – Конфигурация за компоненти от shadcn/ui

eslint.config.mjs – Конфигурация на ESLint за проверка на кода

next.config.ts – Конфигурационен файл за Next.js

package.json – Зависимости и скриптове за проекта

postcss.config.mjs – Конфигурация за PostCSS (обработка на CSS)

tailwind.config.ts – Конфигурация на Tailwind CSS

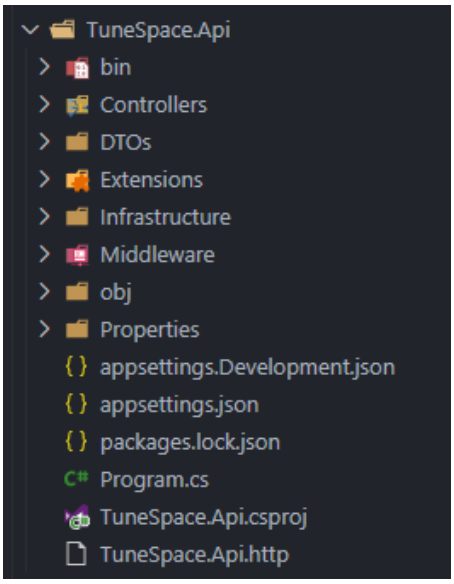
tsconfig.json – Конфигурация за TypeScript

4.1.2 Backend

Директорията съдържа папки и файлове, които се отнасят за

сървърната част на приложението.

4.1.2.1 TuneSpace.Api



Фигура 4.2: TuneSpace.Api директория

- **Controllers -** Тази папка служи като основна входна точка за всички HTTP заявки, постъпващи в приложението.
  Тя съдържа специализирани контролери, които обработват различни аспекти на платформата. Всеки контролер представлява специфичен домейн или функция на приложението,
  от удостоверяване на потребителите
  и управление на профили до усъвършенствани алгоритми за откриване на музика. Контролерите действат като интерфейс между фронтенд приложението React и бекенд бизнес логиката, трансформирайки HTTP заявките в извиквания на услуги и връщайки подходящи отговори.

- **DTOs -** Папката „Обекти за трансфер на данни“ съдържа специализирани модели на заявки, които определят структурата на данните, изпращани от фронтенда към крайните API точки на бекенда. Тези DTO служат като договори между клиента и сървъра, осигурявайки безопасност на типа и валидиране на входящите данни.

- **Extensions -** Тази папка съдържа методи за разширение, които подобряват съществуващите .NET типове с допълнителна функционалност, специфична за нуждите на приложението.

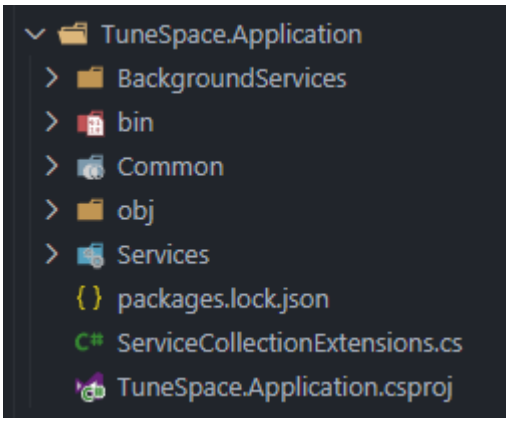
- **Infrastrucute** - Тази инфраструктурна папка в API слоя съдържа междусекторни задачи и помощни програми, които поддържат цялостната работа на уеб API.

- **Middleware** - Персонализирани middleware компоненти, които са част от канала за обработка на заявки в ASP.NET Core. Те имат възможността да проверяват, променят или прекъсват HTTP заявки и отговори, докато те преминават през приложението.

- **Program.cs** - Файлът Program.cs служи като централен център за оркестрация и входна точка за цялото backend приложение. Този файл въплъщава съвременния .NET минималистичен модел за хостинг и представлява готова за производство конфигурация на уеб приложение, която обединява всички части от архитектурата в една сплотена, работеща система.

4.1.2.2 TuneSpace.Application

- **BackgroundServices -** Тази папка съдържа дългосрочни фонови задачи, които работят независимо от потребителските заявки, обработвайки важни задачи за поддръжка и оптимизация.



Фигура 4.3: TuneSpace.Application директория

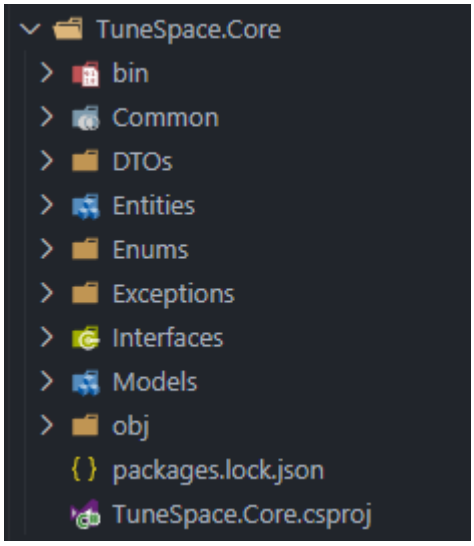
- **Common** - Папката служи като хранилище за споделени помощни програми в приложния слой, съдържащо функции и инструменти, използвани в множество услуги и компоненти.

- **Services** - Това е сърцето на приложния слой, съдържащо бизнес услуги, които реализират основната функционалност на приложението. Всяка услуга се фокусира върху специфична област на домейна, от управление на потребители и удостоверяване до сложно откриване на музика и управление на интеграцията с изкуствен интелект. Услугите в тази папка капсулират бизнес логиката, координират между различни хранилища и реализират случаите на употреба, които управляват функционалността на приложението. Те служат като мост между API контролерите и слоя за достъп до данни, като гарантират, че бизнес правилата се прилагат и сложните операции са правилно оркестрирани.

- **ServiceCollectionExtensions.cs -** Този файл служи като център за оркестрация на бизнес логиката, управляващ регистрацията на всички услуги на приложно ниво, които реализират основната бизнес функционалност и изисквания за жизнен цикъл.

4.1.2.3 TuneSpace.Core

- **Common** – Тук се съдържат споделени константи, изброявания и класове помощни програми.



Фигура 4.4: TuneSpace.Core директория

- **DTOs** - Основната папка DTOs предоставя по-изчерпателен набор от обекти за трансфер на данни в сравнение с API слоя, включително модели на заявки и отговори. Подпапките Requests и Responses съдържат подробни модели за сложни операции във всички области на домейна. Тези DTOs служат като договор между различните слоеве на приложението и външните услуги, осигурявайки безопасност на типа и ясни дефиниции на структурата на данните в цялата система.

- **Entities** – Тук стоят основните домейн обекти, които представляват бизнес обектите на приложението. Всеки обект представлява реална концепция и капсулира както данни, така и поведение, свързани с тази концепция.

- **Enums** – Папката съдържа типове изброявания, които дефинират фиксирани набори от стойности.

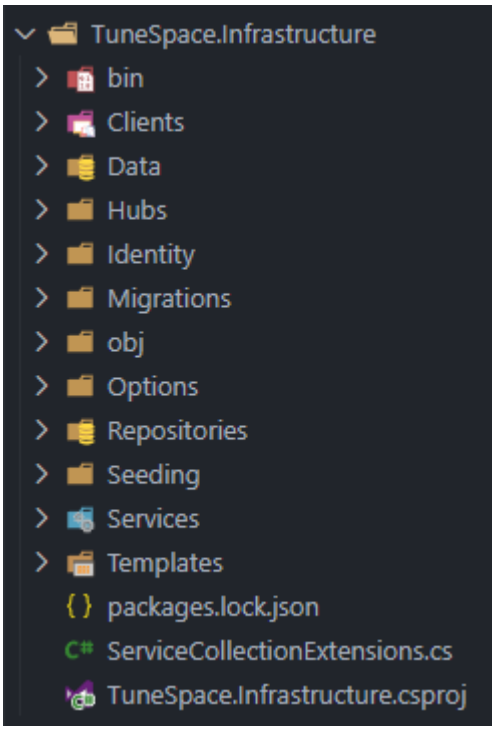
- **Exceptions** - Тази папка съдържа персонализирани класове изключения, които представляват специфични условия за грешки в приложението. Персонализираните изключения позволяват по-прецизна обработка на грешки и по-добро потребителско изживяване, като предоставят специфични съобщения за грешки и стратегии за обработка на различни видове грешки.

- **Interfaces** - Тази папка съдържа всички дефиниции на интерфейси, които установяват договори за хранилища, услуги и външни клиенти. Организирана в четири подпапки (IClients, IInfrastructure, IRepositories, IServices), тя определя слоя на абстракция, който позволява на основния домейн да остане независим от детайлите на имплементацията. Тези интерфейси позволяват инверсия на зависимостите, което прави кода по-тестируем, поддържаем и гъвкав. Интерфейсите определят кои операции са налични, без да се уточнява как са имплементирани, което позволява замяната на различни имплементации, без да се засяга основната бизнес логика.

- **Models** - Тази папка съдържа допълнителни класове модели, използвани в приложението за оформяне на някои модели от данни.

4.1.2.4 TuneSpace.Infrastructure

- **Clients** - Тази папка съдържа HTTP клиентски имплементации за интегриране с външни услуги и API. Всеки клиент обработва специфичните API изисквания, удостоверяване, ограничаване на скоростта и трансформация на данни, необходими за съответната му услуга.



Фигура 4.5: TuneSpace.Infrastructure директория

- **Data** - Тази папка съдържа **Entity Framework DbContext**, който служи като основен интерфейс към базата данни. Класът TuneSpaceDbContext определя как всички обекти на домейна се съпоставят с таблици в базата данни, обработва връзките между приложението и базата данни и управлява операциите за запазване на данни. Този един файл представлява централната точка за достъп до данни за цялото приложение.

- **Hubs** - Папката съдържа имплементации на SignalR хъбове, които позволяват комуникация в реално време.

- **Identity -** Тази папка управлява инфраструктурата за удостоверяване и оторизация, съдържаща класа ApplicationRole, който разширява рамката Identity, за да поддържа специфичните изисквания за роли и разрешения.

- **Migrations** - Миграциите на базата данни на Entity Framework се съхраняват в тази папка, осигурявайки контрол на версиите за промени в схемата на базата данни. Миграциите позволяват структурата на базата данни да се развива с течение на времето, като същевременно се запазват съществуващите данни, което позволява безпроблемно внедряване и актуализации на базата данни в различни среди - от разработка до производство.

- **Options** - Тази папка съдържа силно типизирани конфигурационни класове, които се съпоставят с настройките на приложението в конфигурационни файлове. Моделът с опции (**Options pattern**) гарантира, че конфигурацията е валидирана при стартиране и предоставя поддръжка на IntelliSense при достъп до конфигурационни стойности.

- **Repositories** - Тази папка съдържа имплементации на хранилища, които обработват достъпа до данни за всички основни обекти в системата. Всяко хранилище се фокусира върху специфичен тип обект и имплементира съответния интерфейс от основния слой. Хранилищата капсулират заявки към базата данни, обработват сложни взаимоотношения между обекти и осигуряват чиста абстракция върху операциите на Entity Framework.

- **Seeding** - Папката за засяване съдържа класове, отговорни за инициализирането на базата данни с важни данни. Класът RoleSeeder гарантира, че необходимите потребителски роли се създават при първоначалното настройване на базата данни, осигурявайки последователна основа за модела на сигурност на приложението.

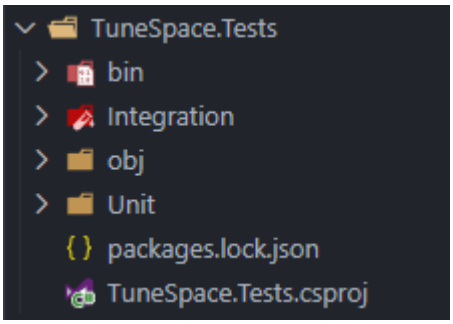
- **Services** - Тази папка с инфраструктурни услуги съдържа реализации на междусекторни логики и външни интеграции. Тези услуги предоставят функционалност на ниво инфраструктура, която поддържа бизнес логиката, без да е част от основния домейн.

- **Templates** - Папката съдържа HTML шаблони за различни видове имейл комуникации.

- **ServiceCollectionExtensions.cs** - Този файл управлява инфраструктурата и външните зависимости, които поддържат целия стек на приложението.

4.1.2.5 TuneSpace.Tests

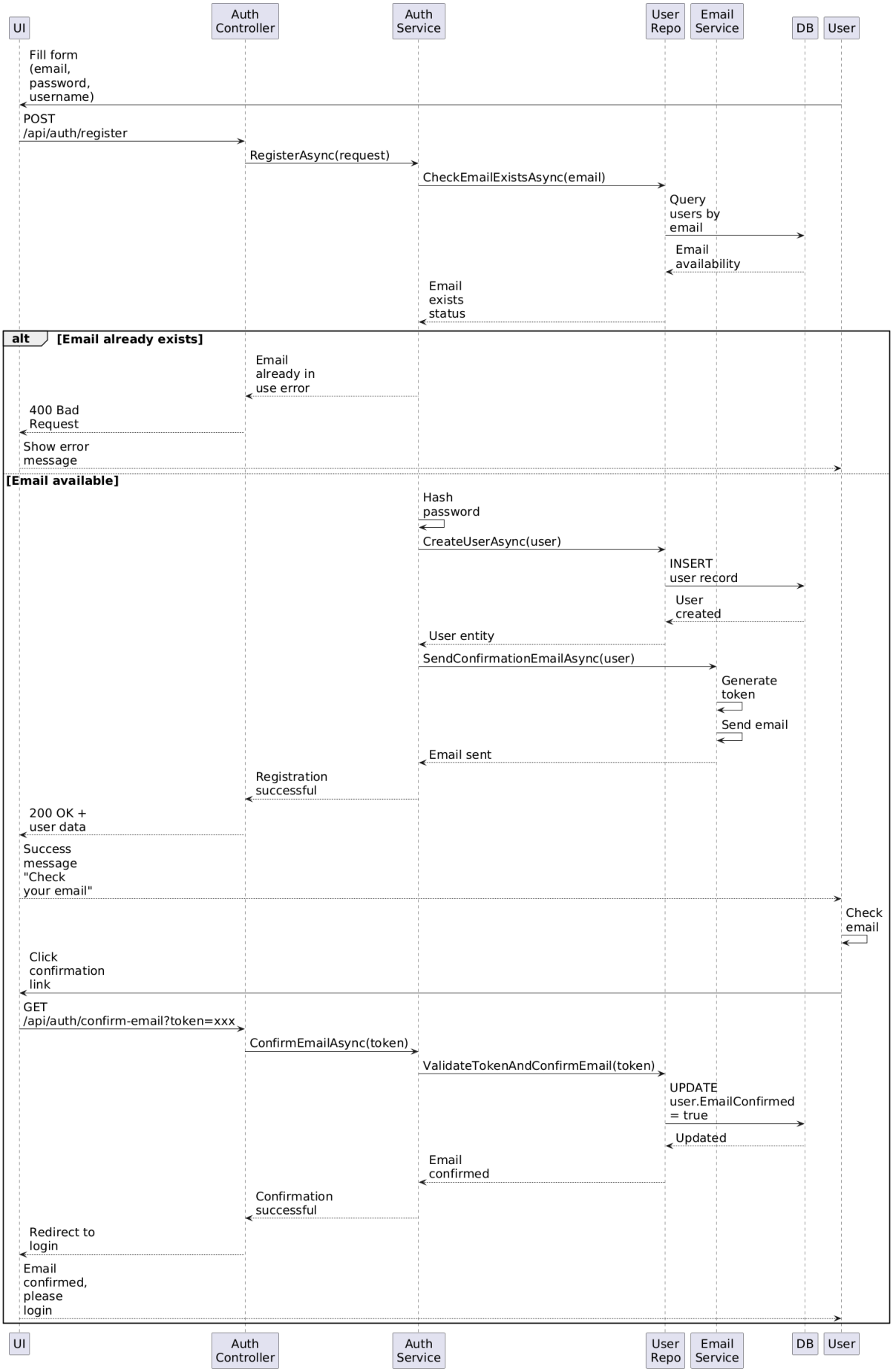
Проектът TuneSpace.Tests предоставя автоматизирано тестване за бекенд логиката на приложението, като гарантира надеждност и функционалност чрез тестово покритие.



Фигура 4.6: TuneSpace.Tests директория

4.2 Процес на регистрация на потребител и потвърждение по имейл

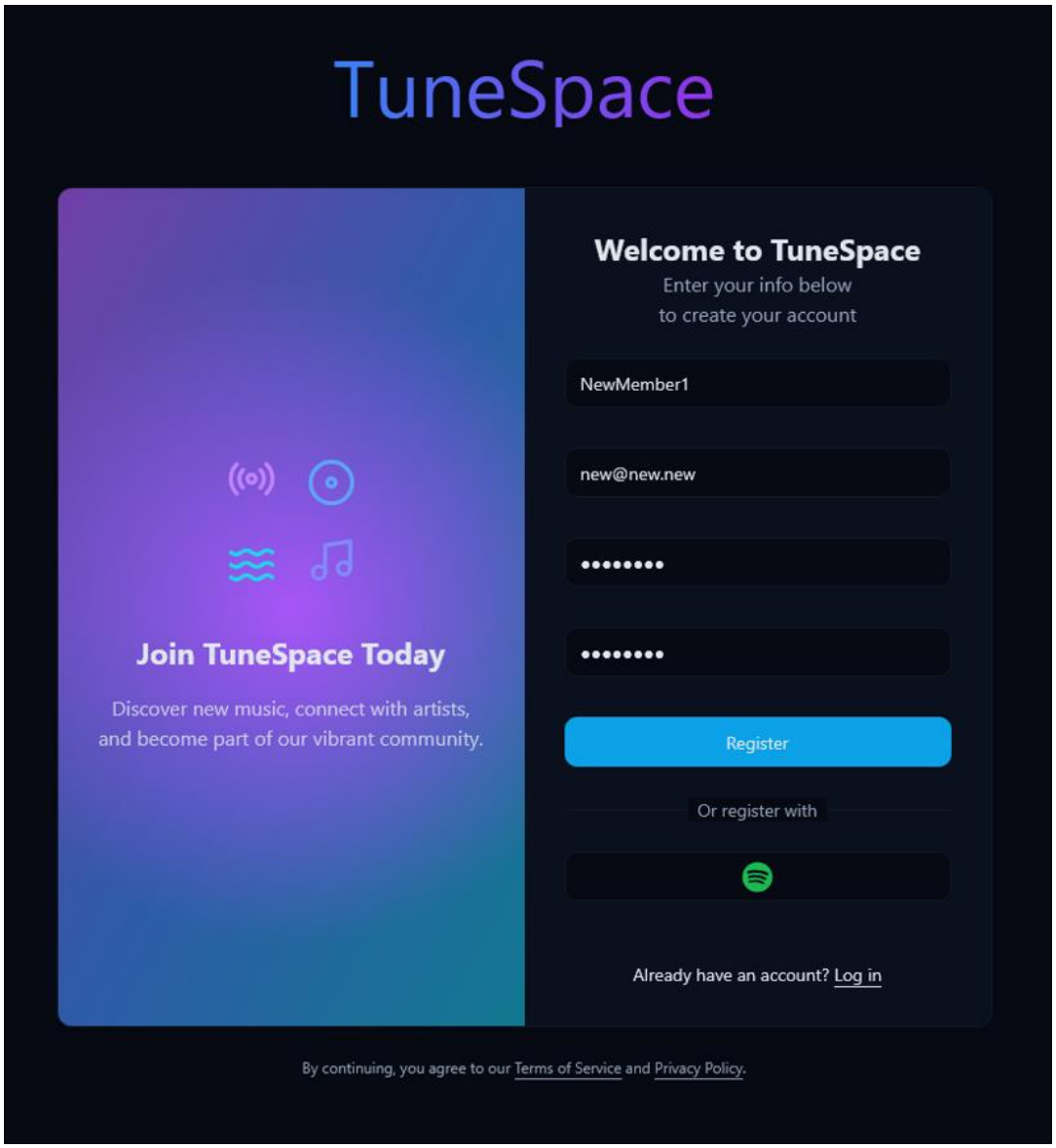
Фигура 4.7 илюстрира пълния процес на регистрация на потребители в системата, включително проверка на имейл адреса за сигурност на акаунта.



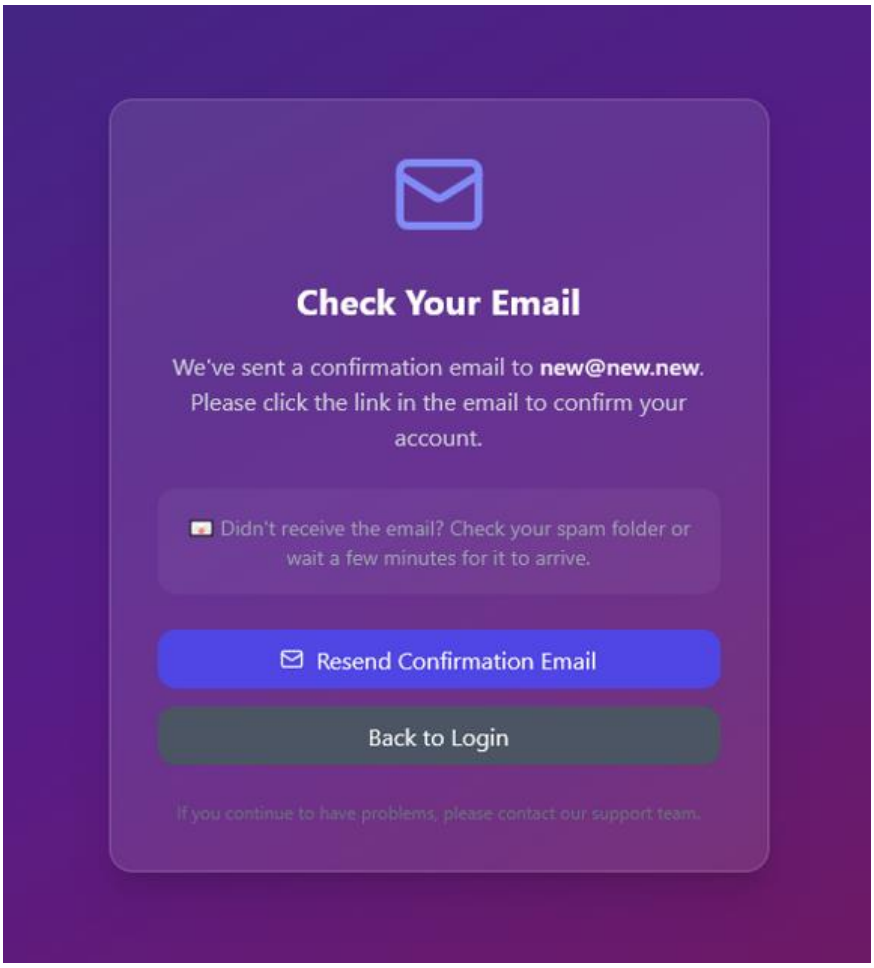
Фигура 4.7. Преглед на процеса на регистрация на потребител и потвърждение по имейл

Когато потребителите за първи път открият TuneSpace, те създават акаунт чрез лесен формуляр с имейл, потребителско име и парола. Системата веднага проверява имейла и дава обратна връзка. Регистрацията се обработва сигурно с хеширане на пароли.

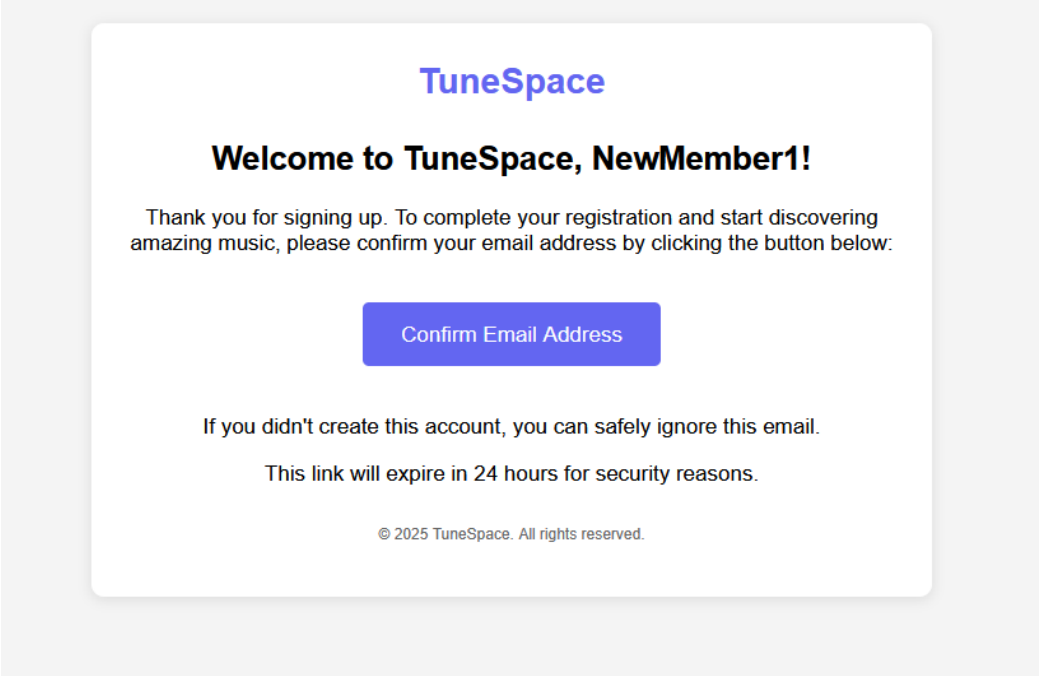
След това потребителите получават имейл с временен линк за потвърждение, който активира акаунта им. Процесът гарантира сигурност и предотвратява злоупотреби. През цялото време системата предоставя ясни съобщения и насоки, осигурявайки лесна и защитена регистрация.



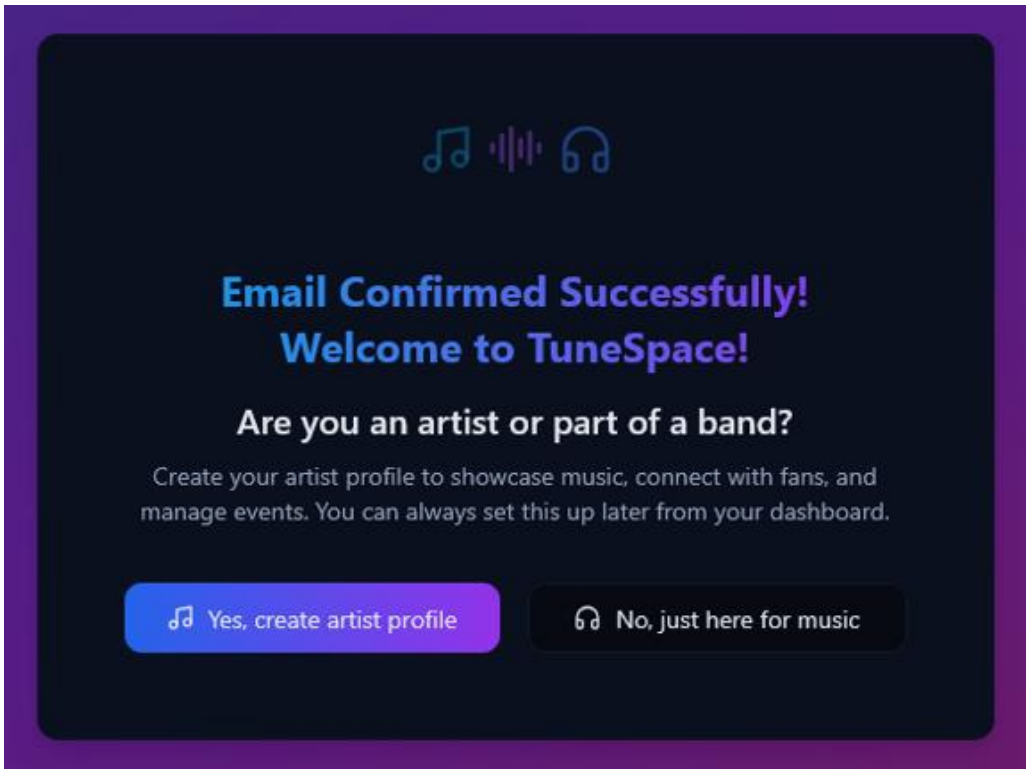
Фигура 4.8. Екран за регистрация



Фигура 4.9. Екран за потвърждаване на имейл



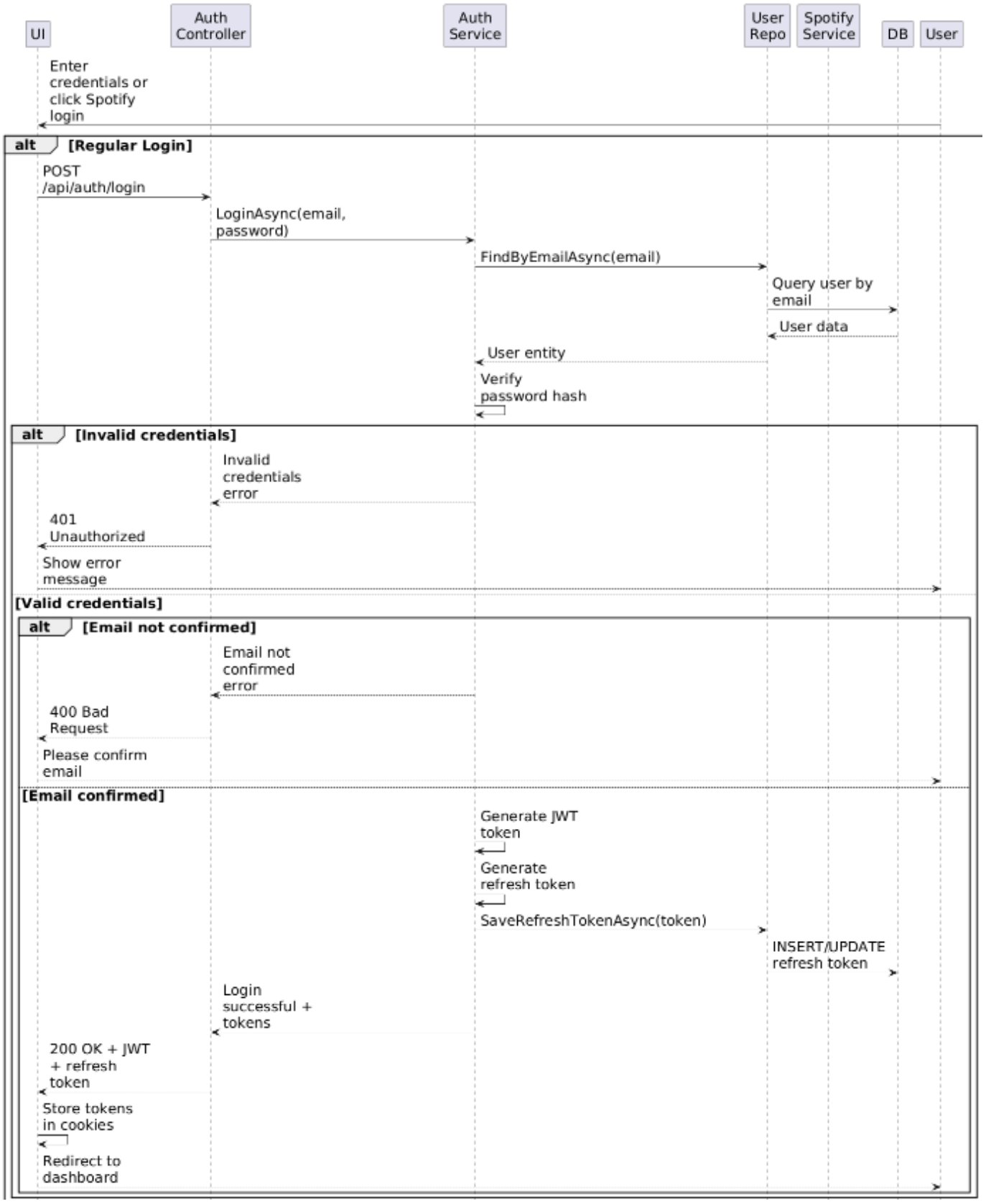
Фигура 4.10. Имейл за потвърждение

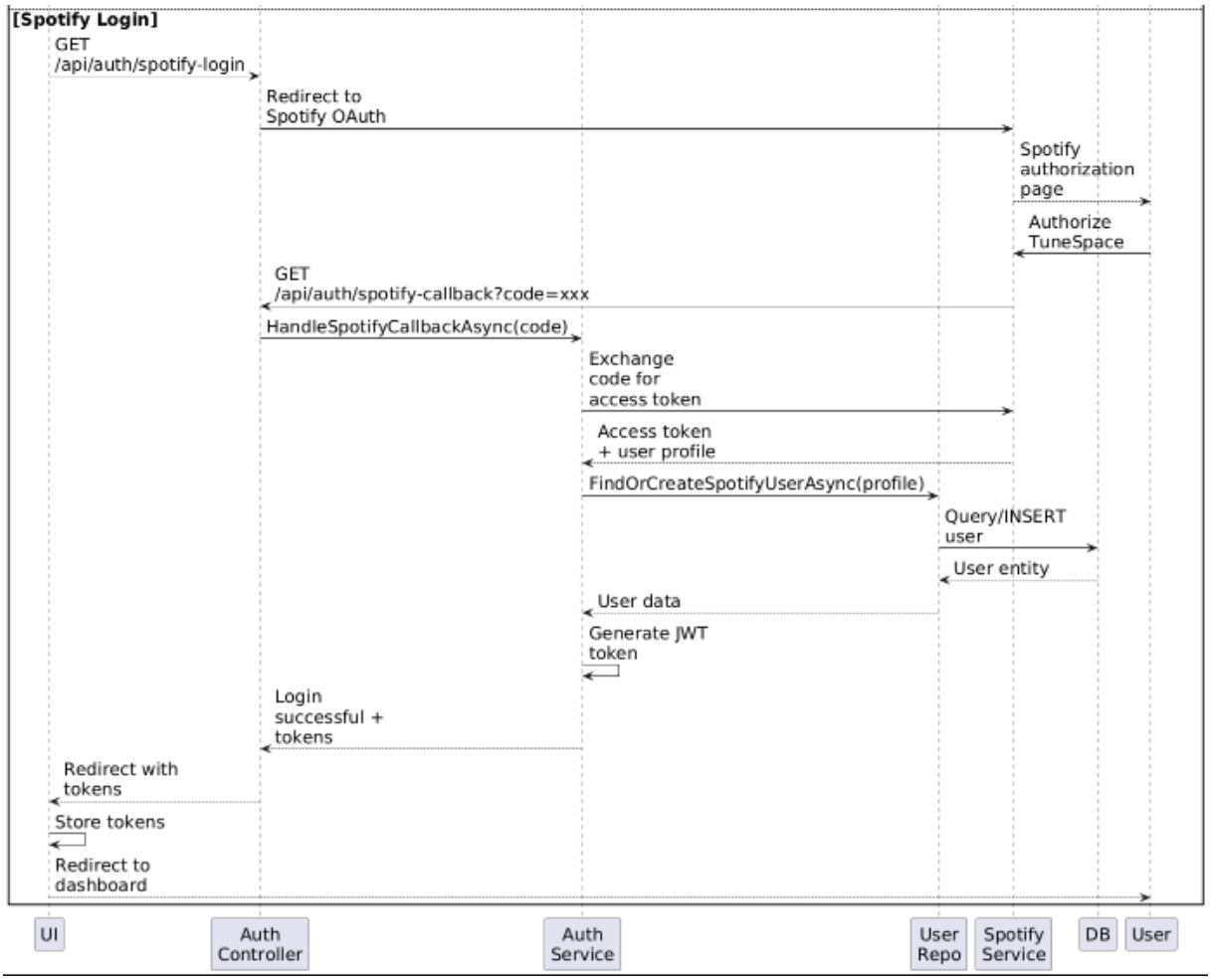


Фигура 4.11. Диалог за избор на създаване на „artist" профил

4.3 Процес на влизане на потребителя (с интеграция със Spotify)

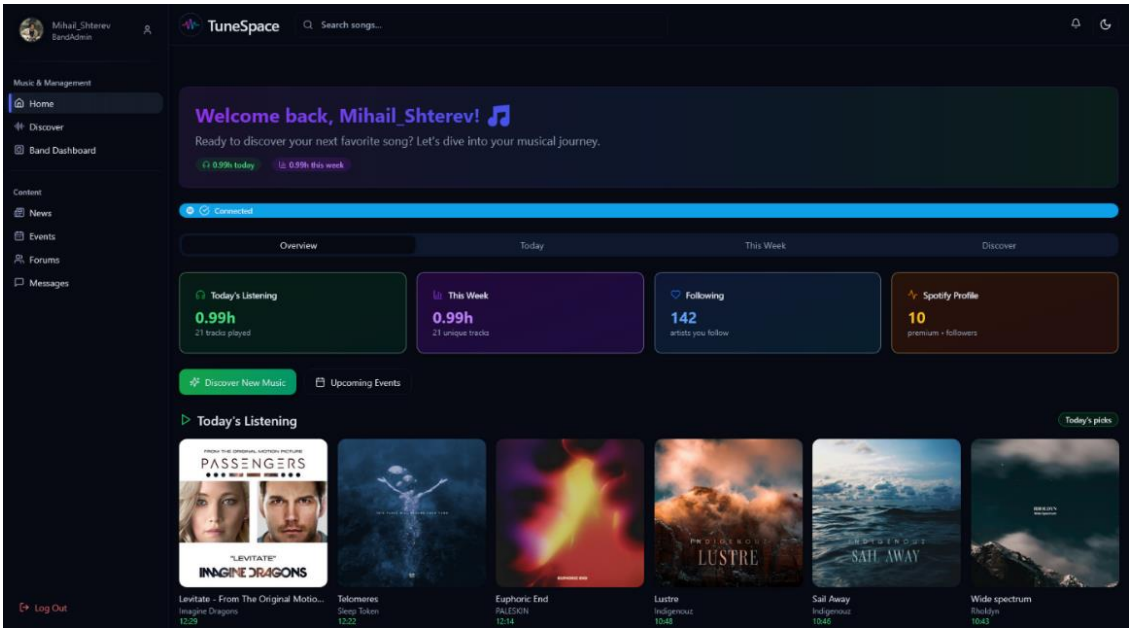
Фигура 4.12 показва системата за двойно удостоверяване на TuneSpace, която поддържа както традиционно влизане с имейл/парола, така и интеграция със Spotify OAuth.


UI
Auth Controller
Auth Service
User Repo
Spotify Service
DB
User
Enter credentials or click Spotify login
alt [Regular Login]
POST /api/auth/login
LoginAsync(email, password)
FindByEmailAsync(email)
Query user by email
User data
User entity
Verify password hash
alt [Invalid credentials]
Invalid credentials error
401 Unauthorized
Show error message
[Valid credentials]
alt [Email not confirmed]
Email not confirmed error
400 Bad Request
Please confirm email
[Email confirmed]
Generate JWT token
Generate refresh token
SaveRefreshTokenAsync(token)
INSERT/UPDATE refresh token
Login successful + tokens
200 OK + JWT + refresh token
Store tokens in cookies
Redirect to dashboard

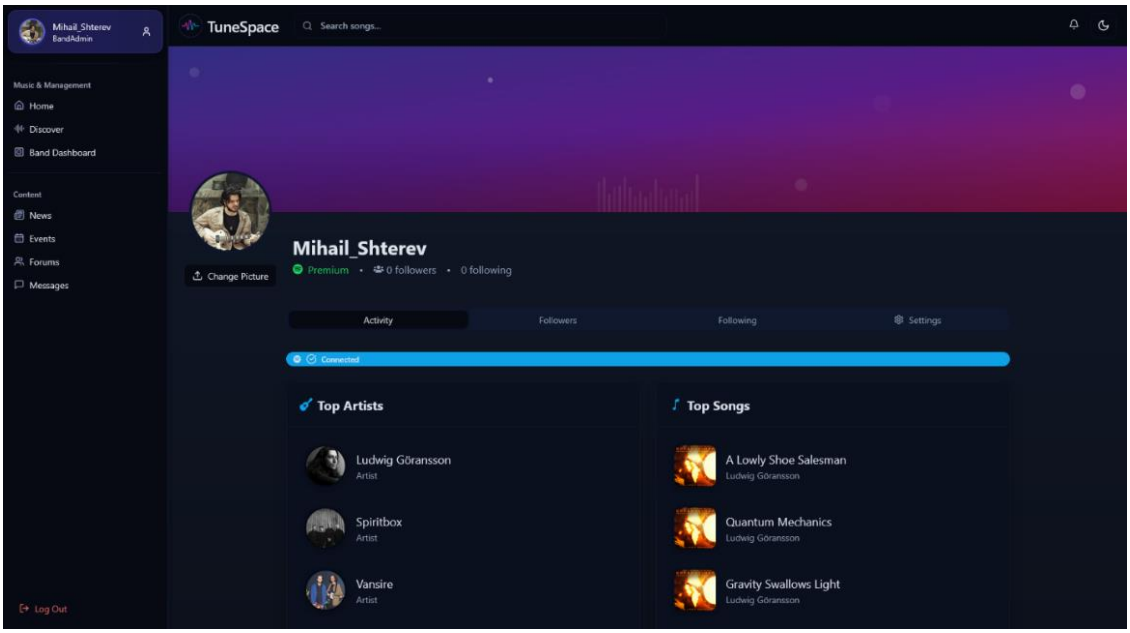


Фигура 4.12. Процес на влизане на потребителя (с интеграция със Spotify)

При избор на вход чрез Spotify, потребителите се пренасочват към страница за оторизация. Всички потребители получават сигурни JWT токени с възможност за автоматично опресняване, осигуряващи стабилна сесия на различни устройства. При всякакви затруднения системата дава ясна обратна връзка.



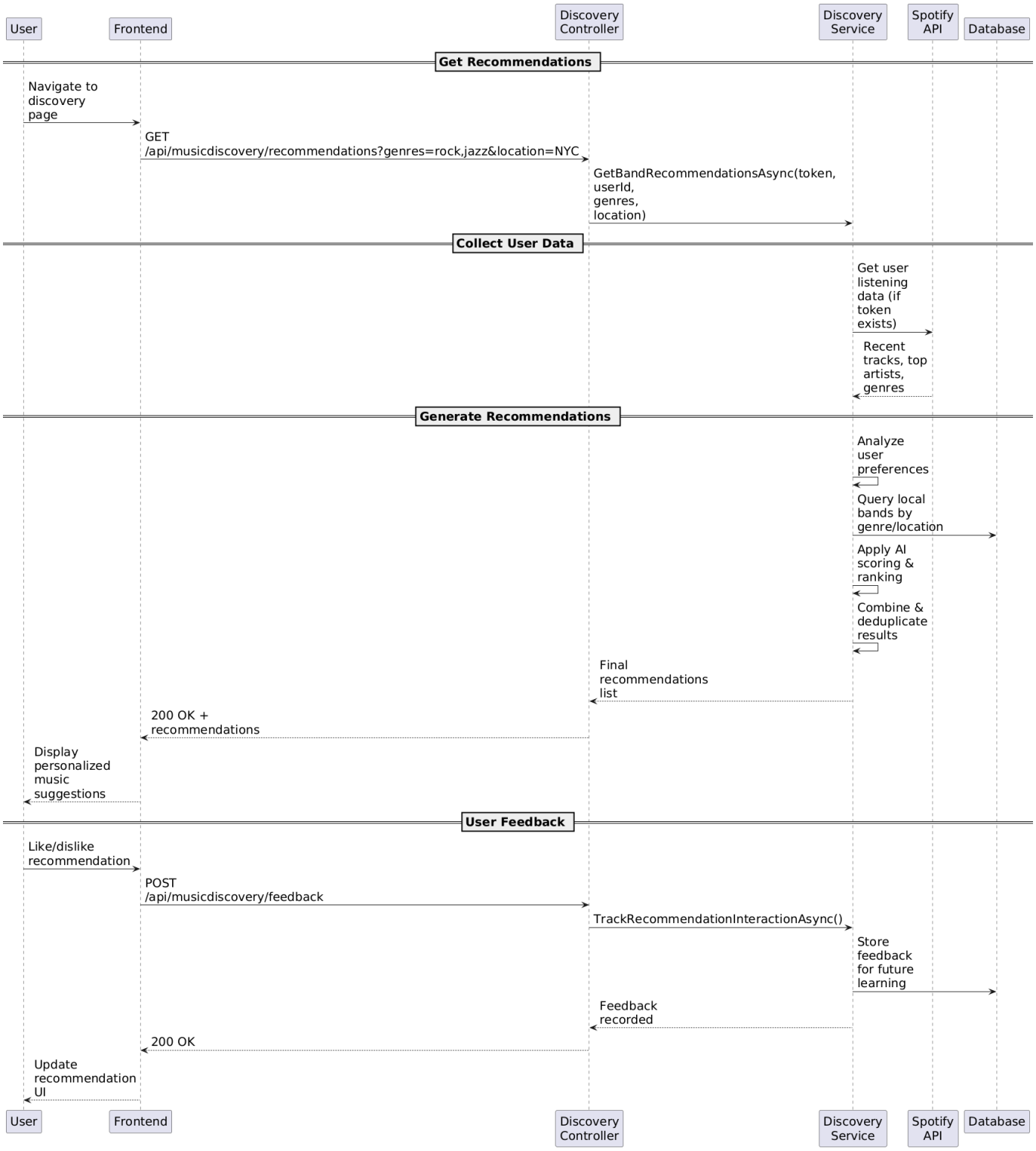
Фигура 4.13. Начална страница



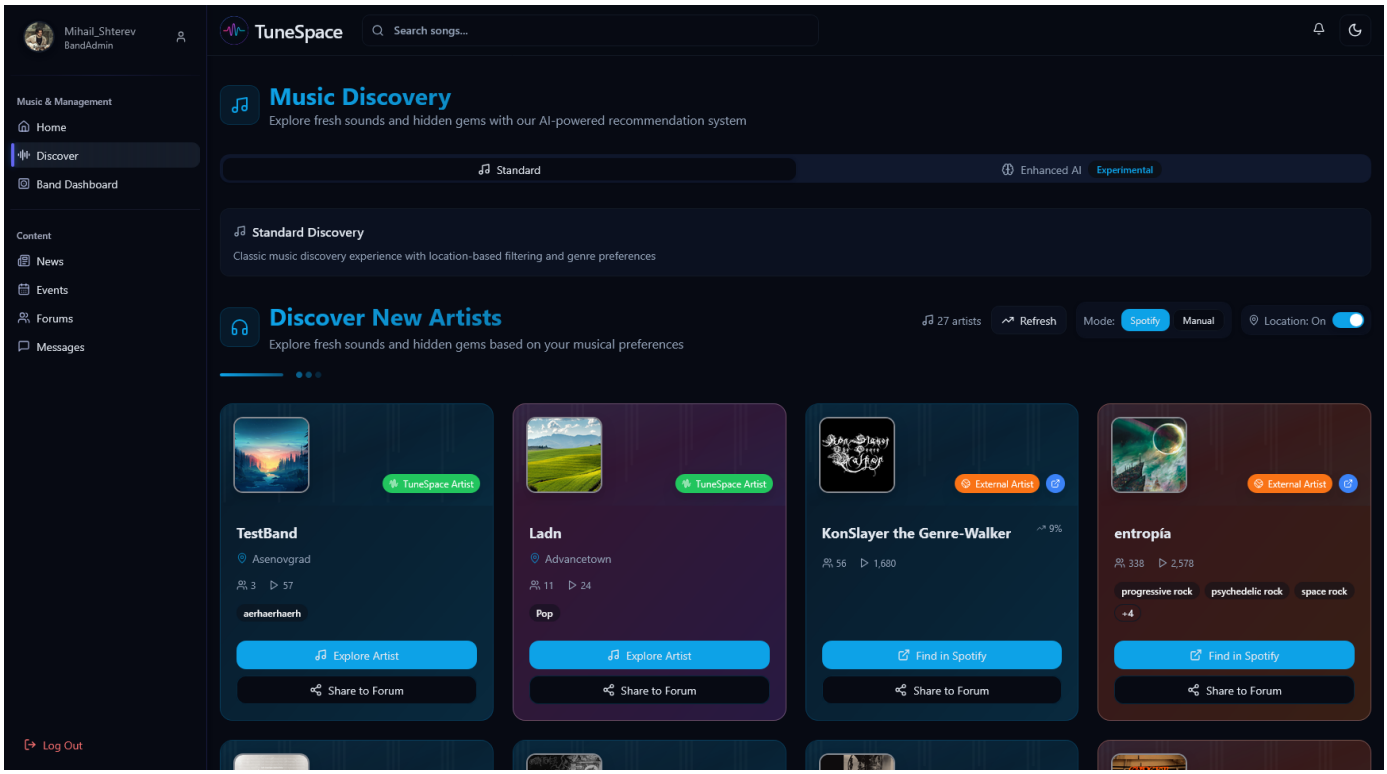
Фигура 4.14. Страница на профила на потребителя

4.4 Процес на откриване и препоръчване на музика

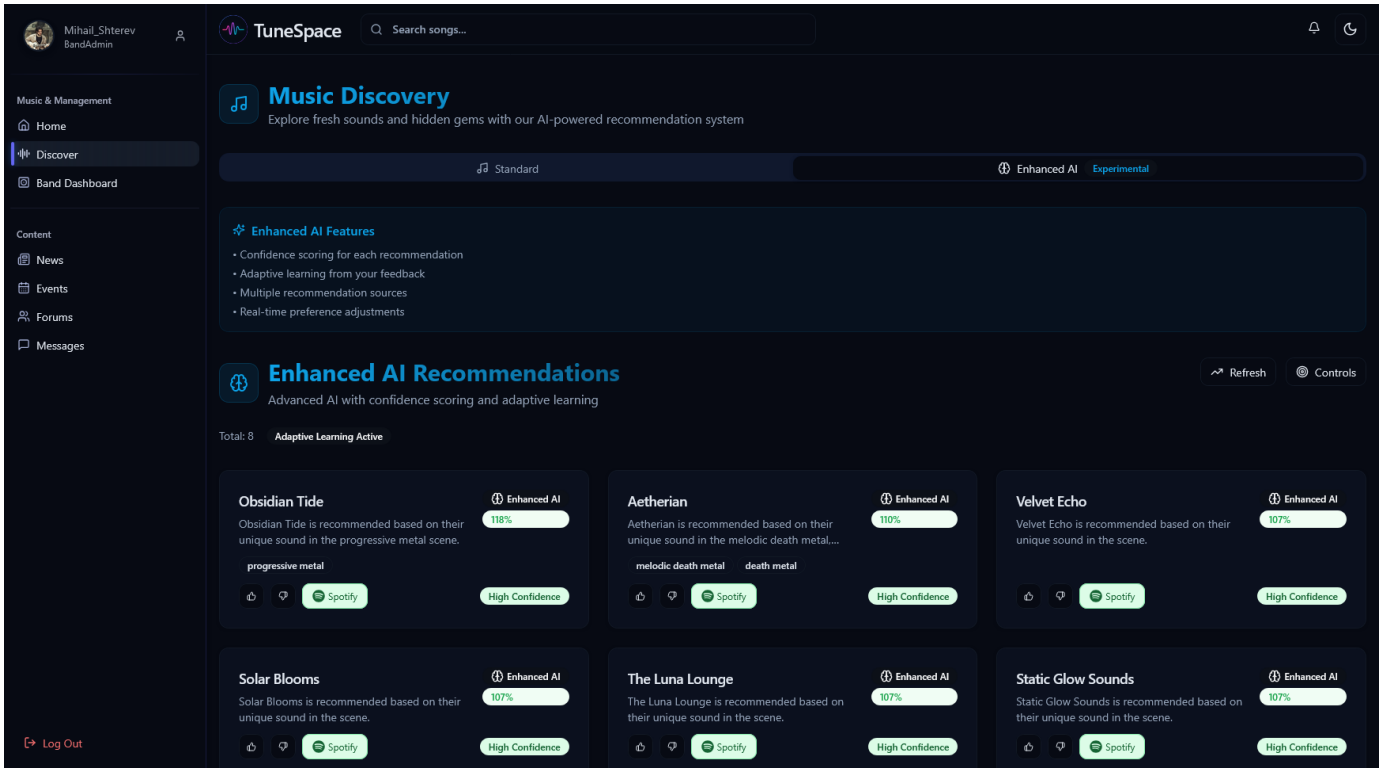
Фигура 4.15 показва усъвършенстваната система за музикални препоръки на TuneSpace, включваща изкуствен интелект, която комбинира множество източници на данни. Системата предоставя персонализирани музикални предложения въз основа на потребителските предпочитания, историята на слушане и усъвършенстваните алгоритми.



Фигура 4.15. Процес на откриване и препоръчване на музика



Фигура 4.16. Страница за откриване на музика



Фигура 4.17. Страница за препоръки от изкуствен интелект

Основната иновация на TuneSpace се крие в усъвършенстван хибриден механизъм за препоръки за музика, който комбинира множество допълващи се подходи, за да осигури персонализирано откриване на музика. Вместо да разчита единствено на изкуствен интелект, платформата стратегически интегрира изкуствен интелект със съвместно филтриране, адаптивни алгоритми за оценяване и традиционни техники за препоръки, за да създаде надеждна система за препоръки, базирана на множество сигнали.

Архитектура на препоръките от множество източници:

Механизмът за препоръки работи чрез внимателно организиран процес, който черпи от няколко източника за препоръки:

1. **Откриване на местни групи** - Използва данни от MusicBrainz и Bandcamp, за да идентифицира нововъзникващи местни изпълнители въз основа на географски и жанрови предпочитания

2. **Промотиране на регистрирани групи** - Приоритизира групи, които са регистрирани в платформата, създавайки стойност за общността на изпълнителите

3. **Откриване на ъндърграунд изпълнители** - Използва възможностите за търсене на Spotify, за да открива по-малко известни изпълнители с ниски оценки за популярност, но висока релевантност към жанра

4. **Откриване на подобни изпълнители**: Използва алгоритмите за сходство на Last.fm, за да разшири мрежите от препоръки въз основа на акустични и стилистични връзки

5. **Препоръки, подобрени с изкуствен интелект** - Включва големи езикови модели чрез интеграция с Ollama, използвайки Retrieval-Augmented Generation (RAG)

6. **Съвместно филтриране** - Анализира моделите на сходство между потребителите, за да препоръчва изпълнители, харесвани от потребители с подобно поведение и предпочитания за слушане

Интеграция с изкуствен интелект в хибридната система:

Изкуственият интелект служи като един от ключовите компоненти в този многостранен подход. Подсистемата с изкуствен интелект използва:

1. **Вградяване на вектори**: 384-измерни представяния, които улавят семантични връзки между изпълнители, жанрове и потребителски предпочитания чрез разширението pgvector на PostgreSQL

2. **Динамични тегла за оценяване**: Адаптиране на оценяването на препоръките въз основа на обратна връзка от потребителското взаимодействие

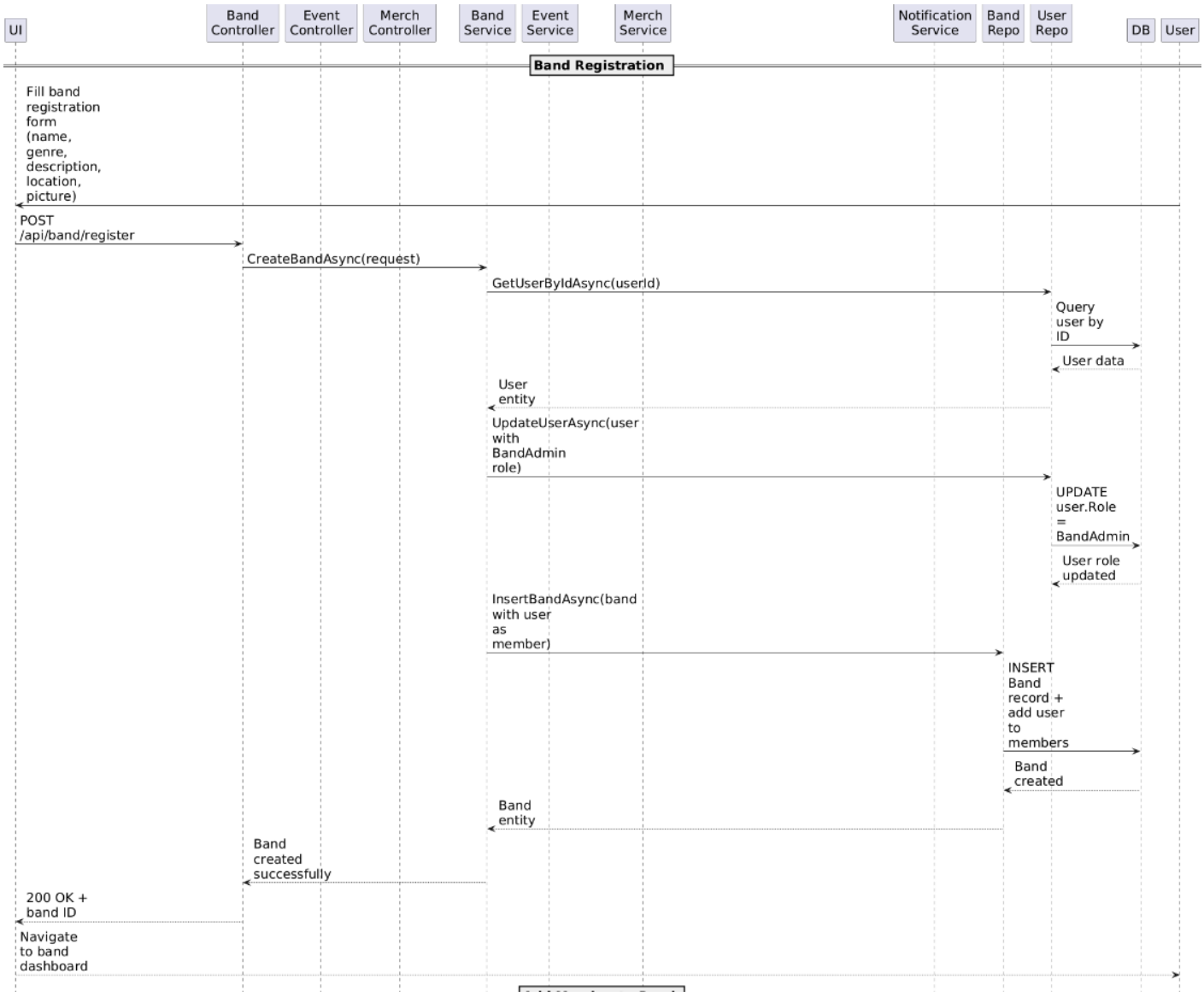
3. **Оценка на доверието и обясним изкуствен интелект**: Многоизмерна оценка, адаптивно калибриране на доверието и автоматично генериране на обяснения за всички видове препоръки

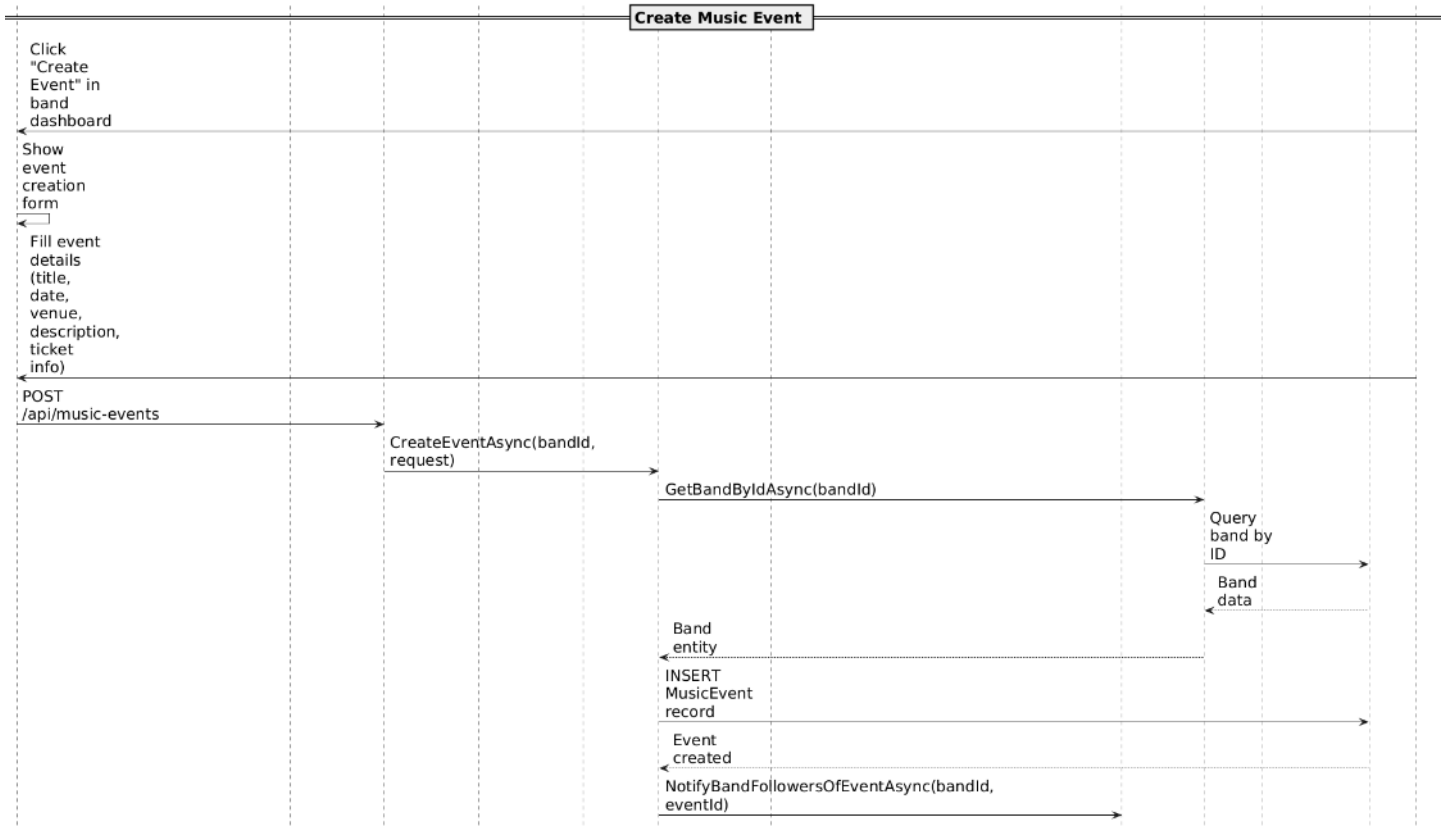
4. **Обработка на обратна връзка от потребителите**: Проследяване на потребителското взаимодействие с интеграция за обратна връзка, незабавни актуализации на параметрите за оценяване и анализ на процента на успех

Хибридната система за препоръки интелигентно обработва и съпоставя данни от потребителски профили на Spotify, каталози на изпълнители на Bandcamp, бази данни на MusicBrainz, изчерпателната база от музикални знания на Last.fm, модели за сътрудничество между потребителите и взаимодействия в реално време. Този многоизточников подход с усъвършенствани алгоритми осигурява цялостно покритие на пейзажа на откриването на музика, балансирайки иновациите в областта на изкуствения интелект с доказани техники за препоръки, усъвършенствани алгоритми за персонализиране, за да осигури превъзходно изживяване при откриване на музика.

4.5 Процес на регистрация и управление на групи

Фигура 4.18 илюстрира как музикантите използват TuneSpace, за да създават и управляват групи, поддържайки професионално присъствие на платформата. Процесът започва с регистрация на група – индивидуални артисти могат да създадат солови проекти, съществуващи групи могат да се регистрират, или членове да бъдат поканени да изградят профила съвместно.





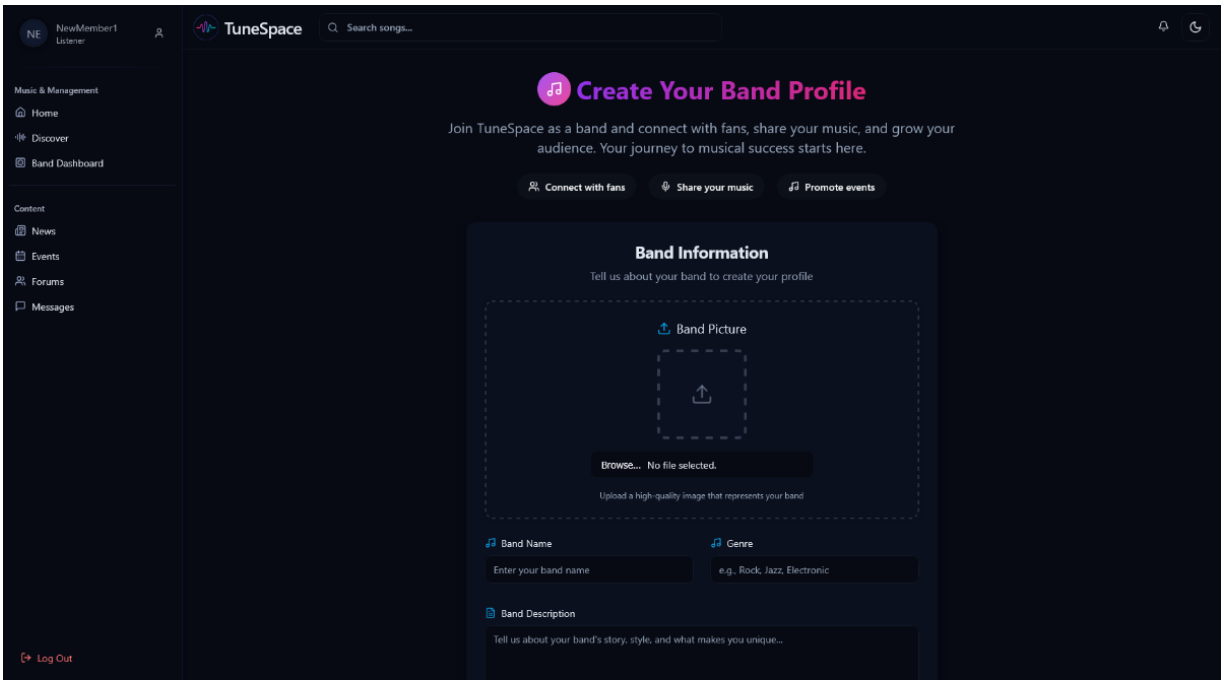
Фигура 4.18. Процес на регистрация и управление на групи

След създаването на групата основателят автоматично получава ролята на администратор с пълни права. Администраторите имат възможност да добавят нови членове, да разпределят роли като вокалист, китарист и други, както и да определят необходимия достъп и възможностите за редакция на всеки член. Груповото табло предоставя набор от инструменти за персонализация на профила, включително качване на снимки, добавяне на връзки, редакция на биография и други.

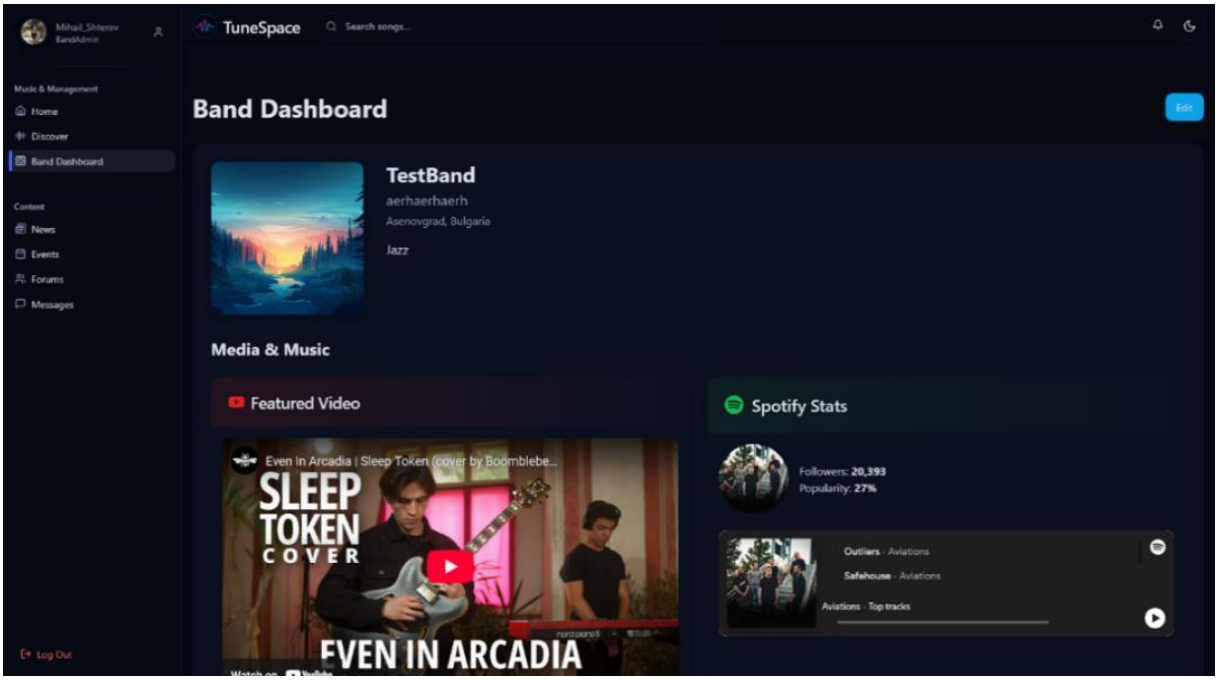
Управлението на събитията е също централен елемент от системата. Администраторите могат да създават музикални събития с подробна информация – описания, локации, дати и данни за билети. Платформата включва инструменти за промоция чрез вградени социални функции, възможност за феновете да утвърдят присъствието (RSVP), както и автоматични известия за предстоящи концерти.

Платформата предлага и цялостна система за управление на мърчандайз. Администраторите могат да качват снимки и описания на продуктите и да настройват цените.

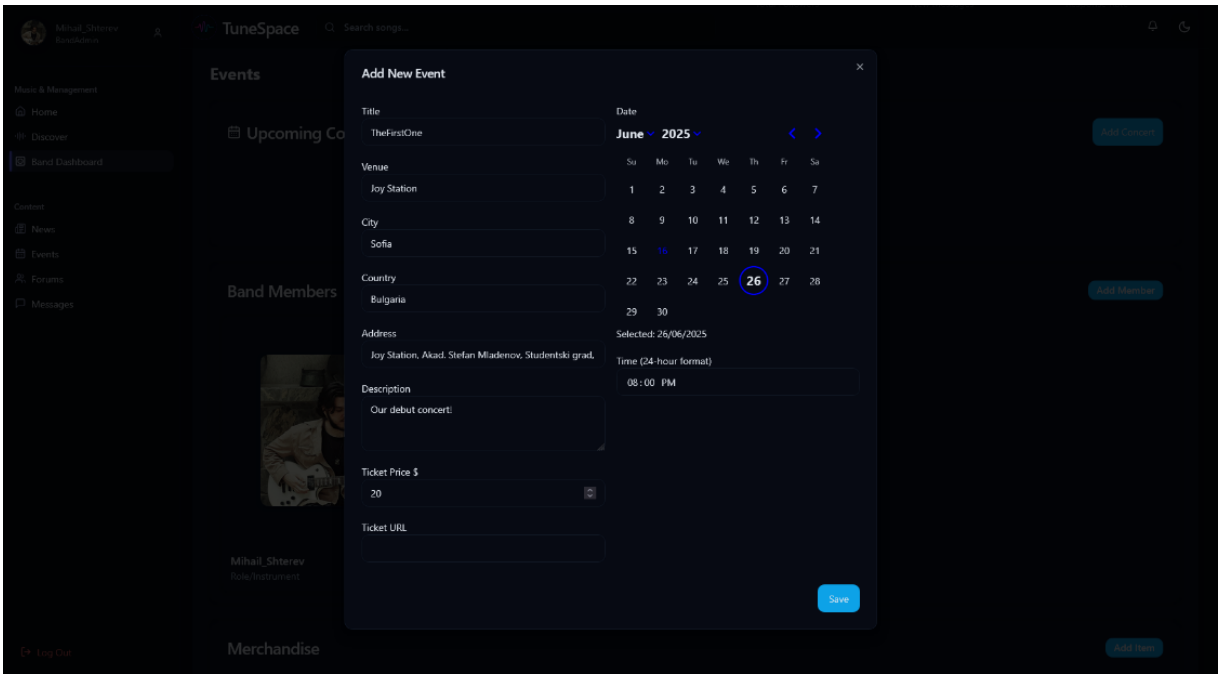
Цялостната структура на системата осигурява гъвкаво и ефективно управление и ясно разпределение на ролите – подходяща е както за солови изпълнители, така и за големи музикални формации.



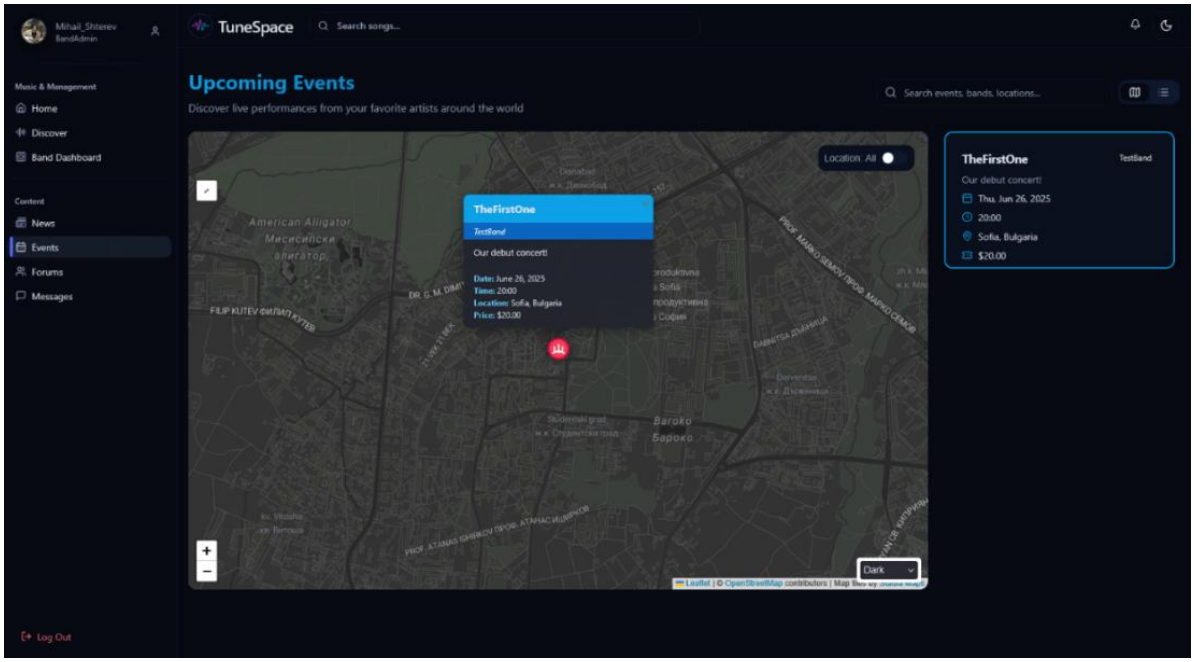
Фигура 4.19. Страница за регистриране на група



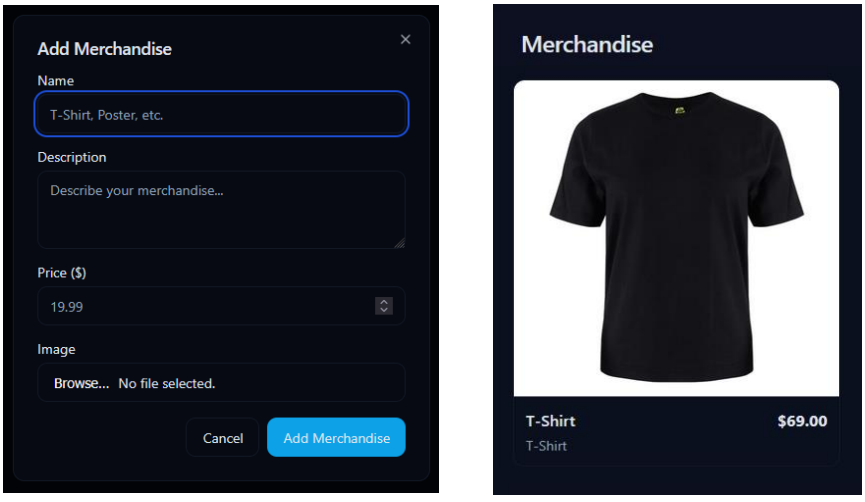
Фигура 4.20. Дашборд на група



Фигура 4.21. Добавяне на събитие



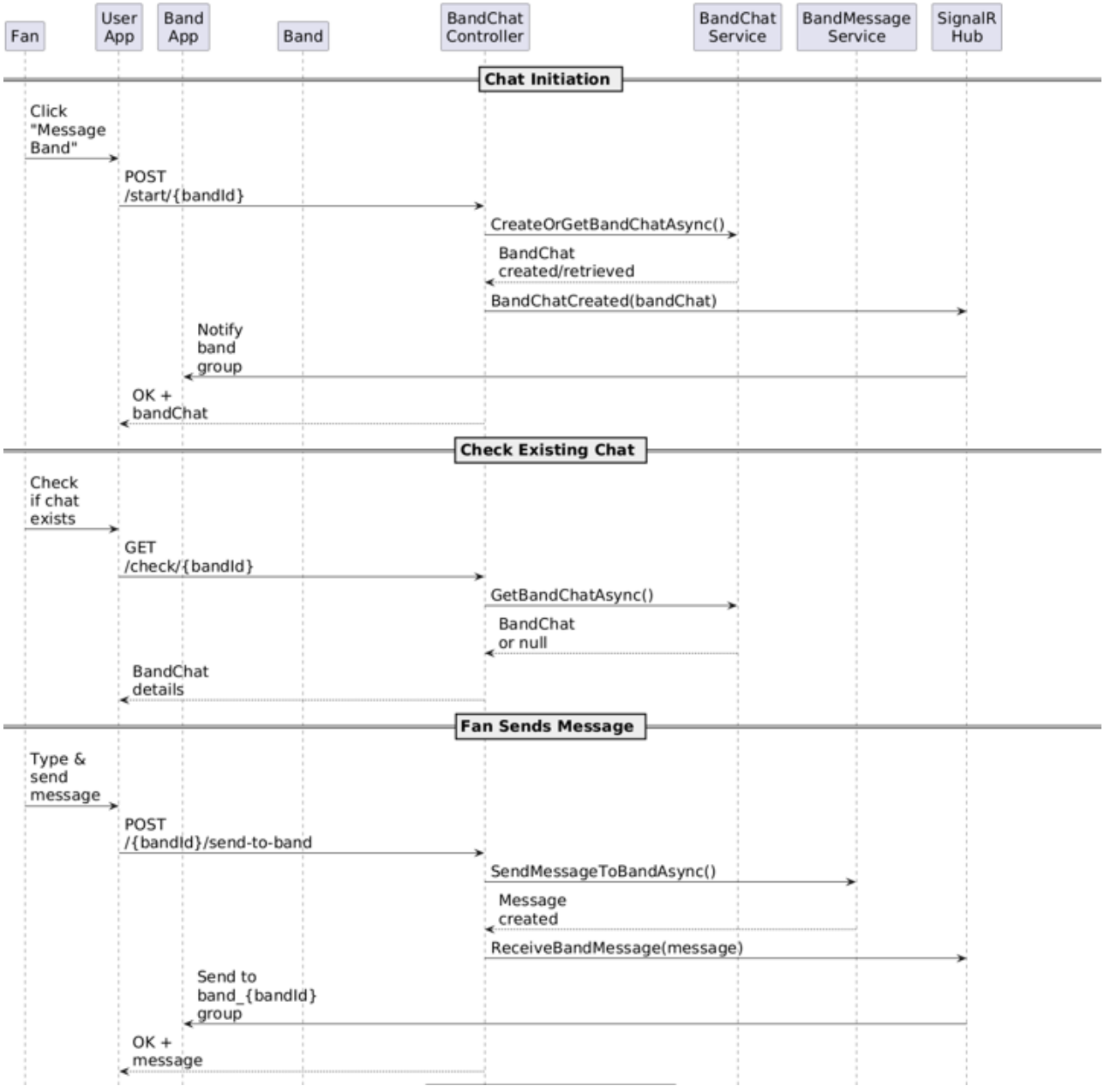
Фигура 4.22. Страница за събития

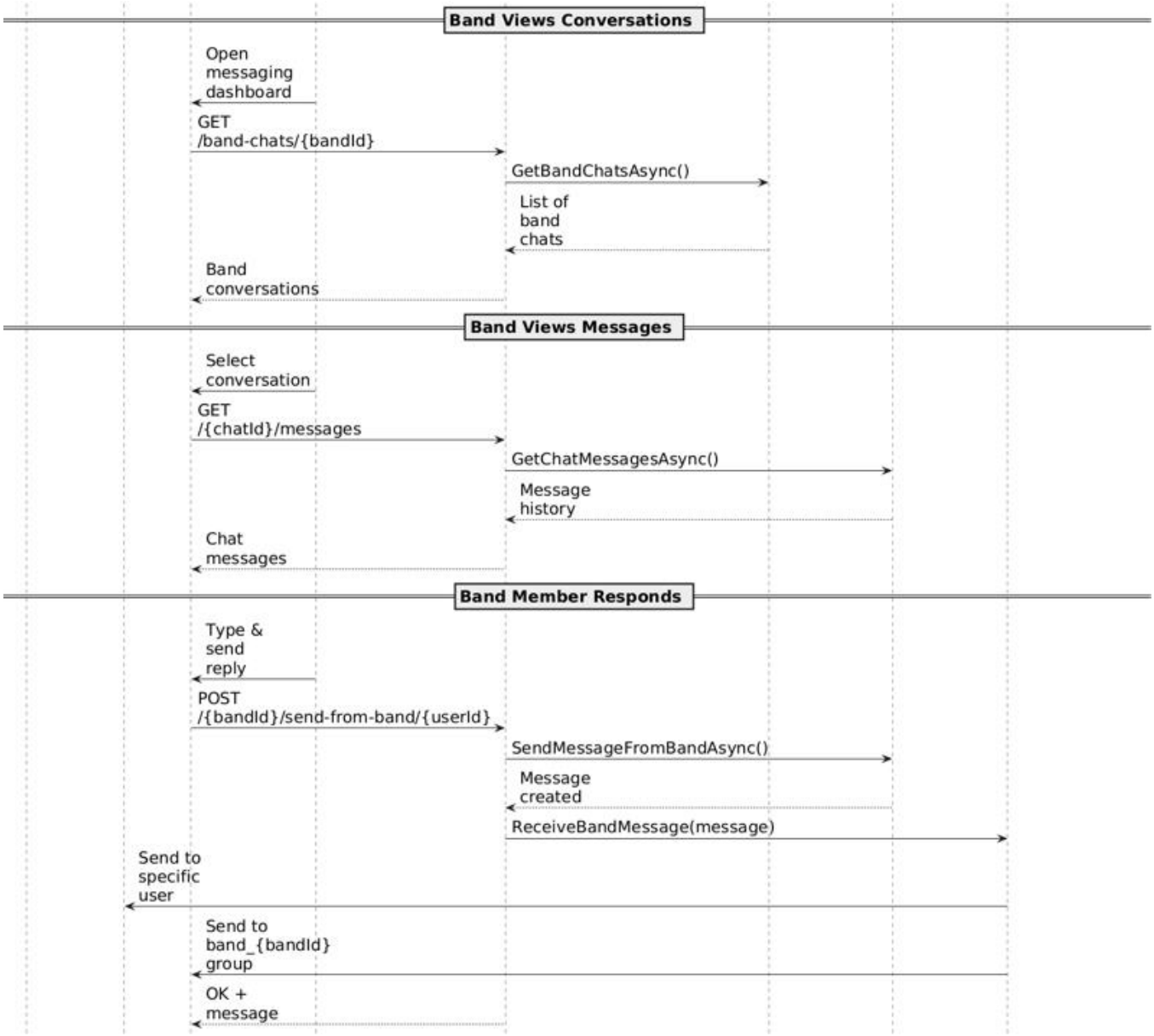


Фигура 4.23. Добавяне на стоки

4.6 Процес на чат с група

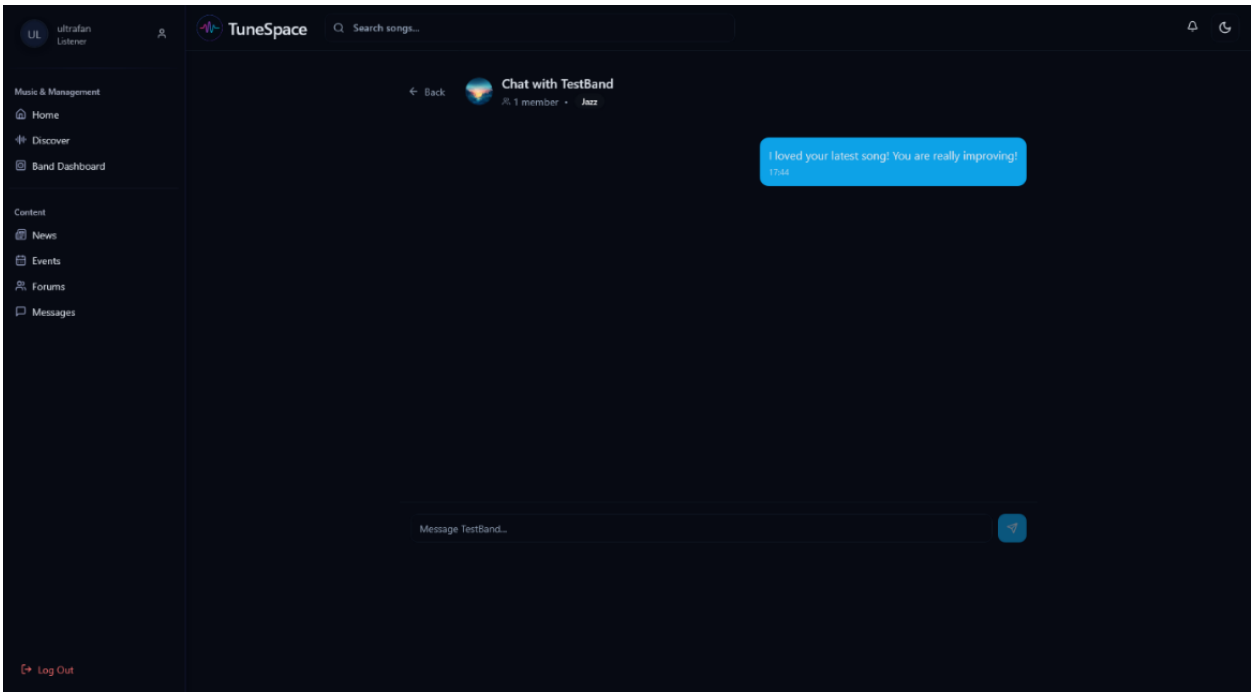
Фигура 4.24 илюстрира системата за съобщения в реално време на TuneSpace, която позволява директна комуникация между музикалните фенове и любимите им групи. Процесът демонстрира двупосочно изпращане на съобщения с незабавна доставка, push известия и постоянна история на чата за текущи разговори.



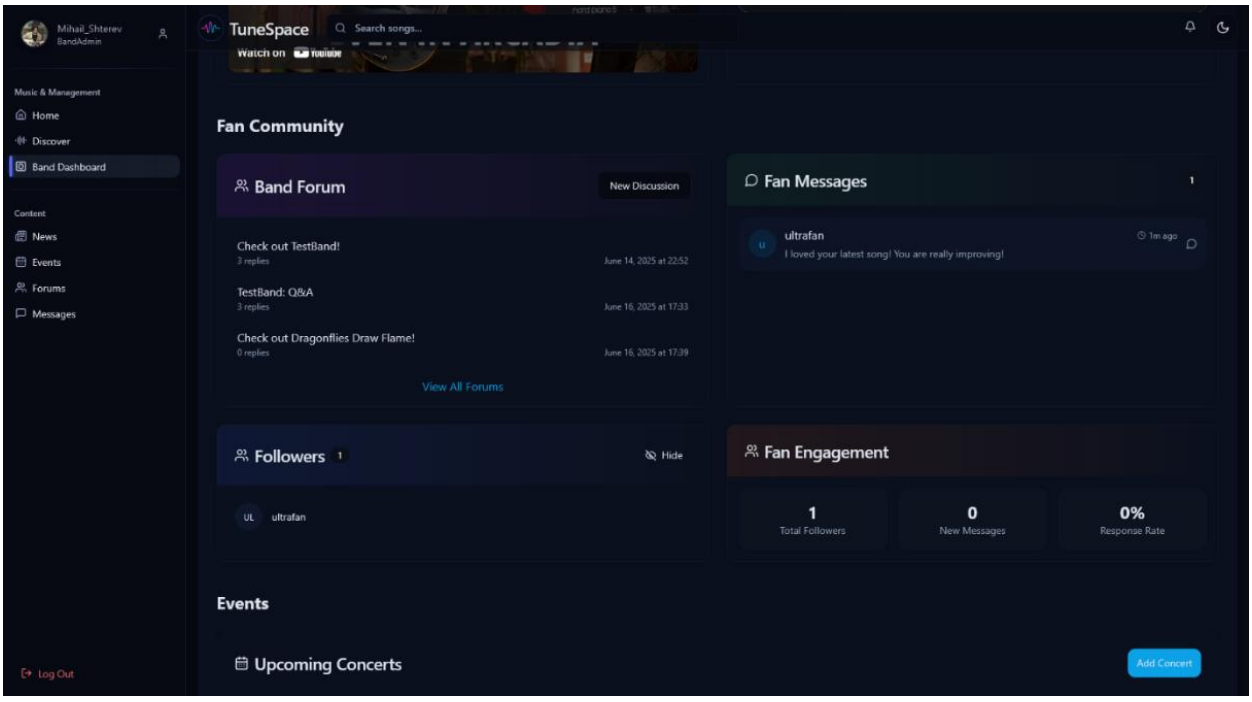


Фигура 4.24. Процес на чат с група

Потребителите могат лесно да започнат личен разговор с дадена група. Съобщенията се изпращат в реално време чрез SignalR WebSocket. Феновете могат да споделят впечатления, а групите да отговарят лично, изграждайки автентична връзка. Групите разполагат с удобно табло за управление на разговорите, получават известия за нови съобщения и могат ефективно да комуникират с феновете си.



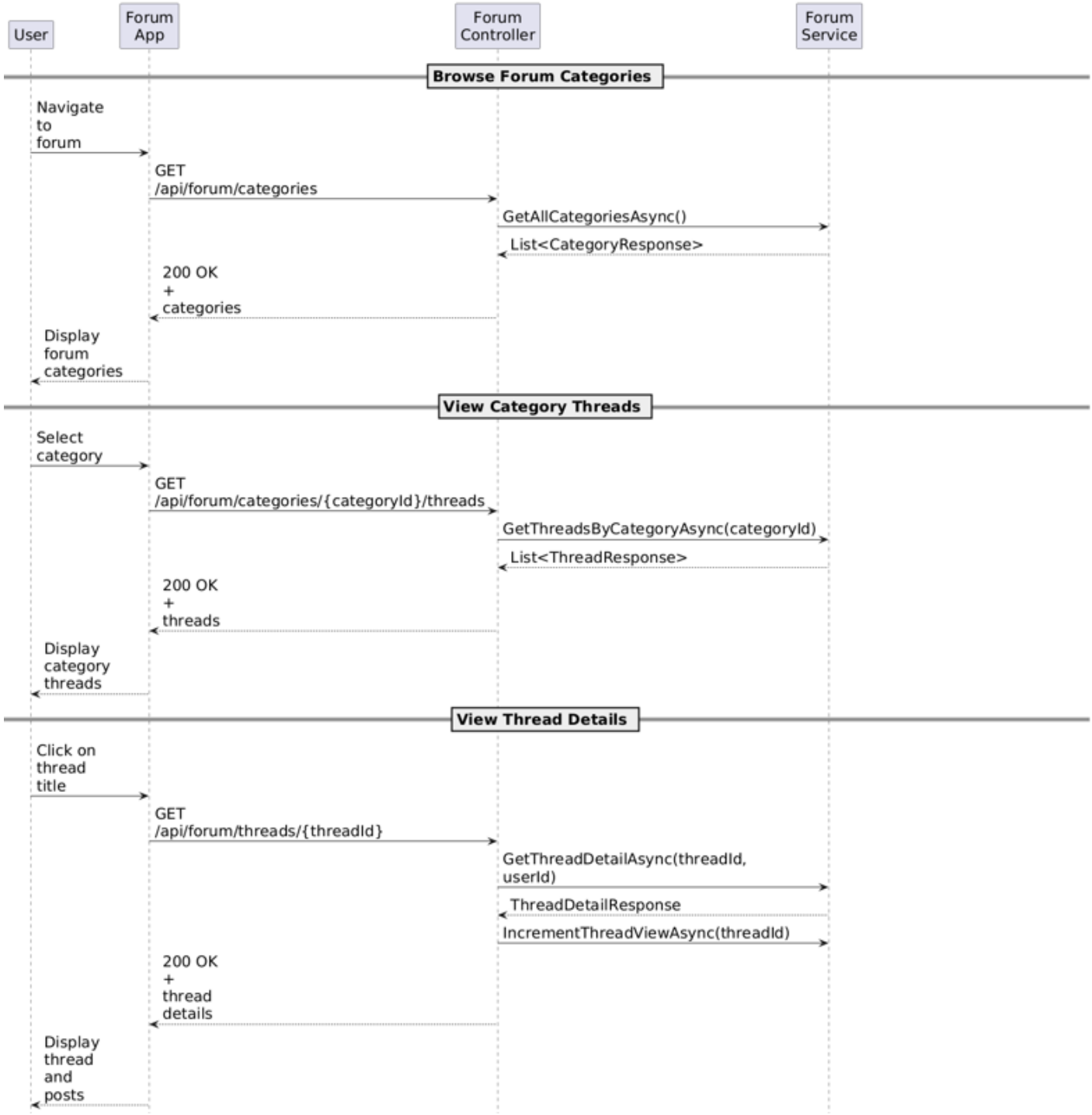
Фигура 4.25. Чат с група

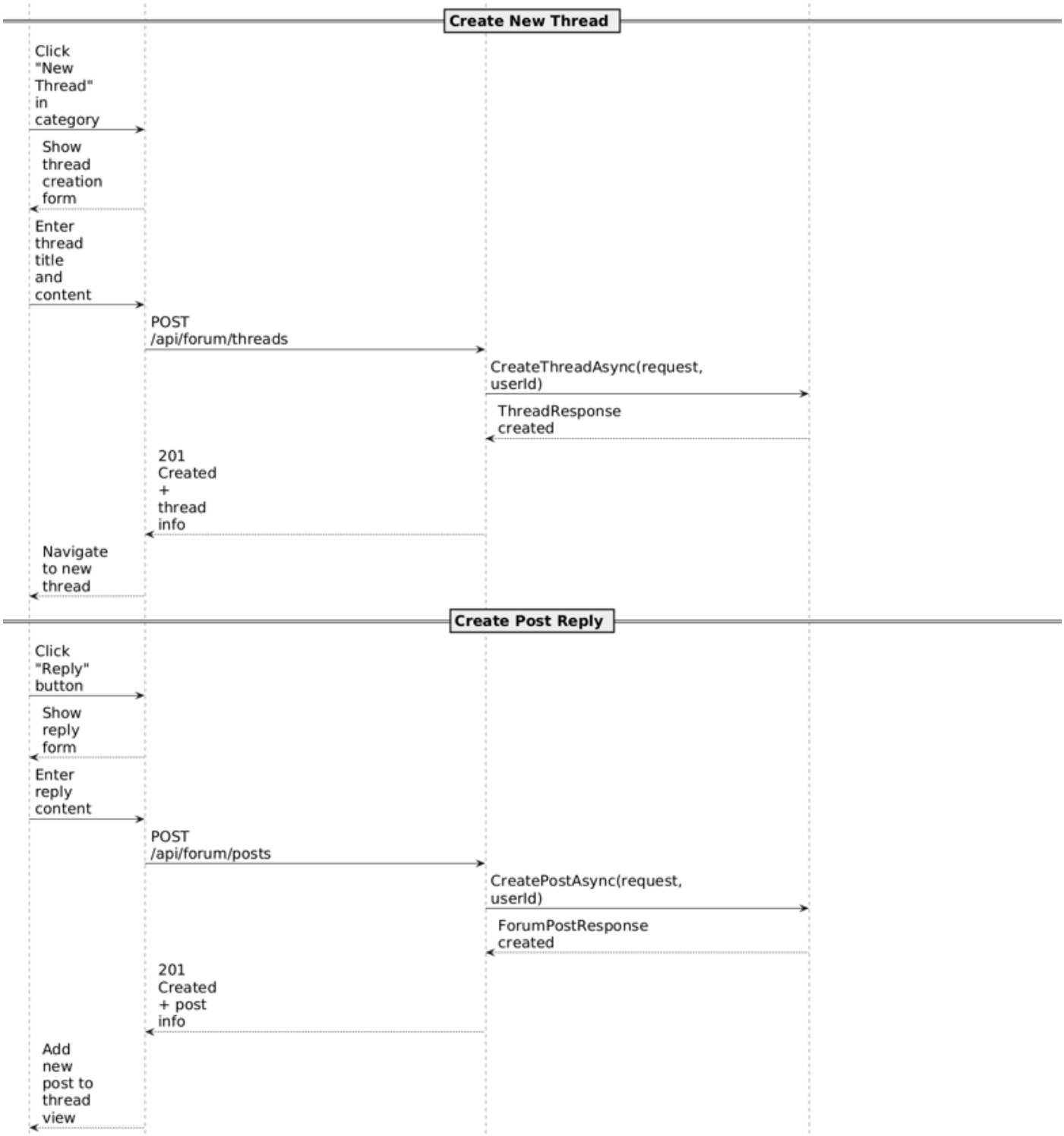


Фигура 4.26. Фен чатове и статистики в дашборда на групата

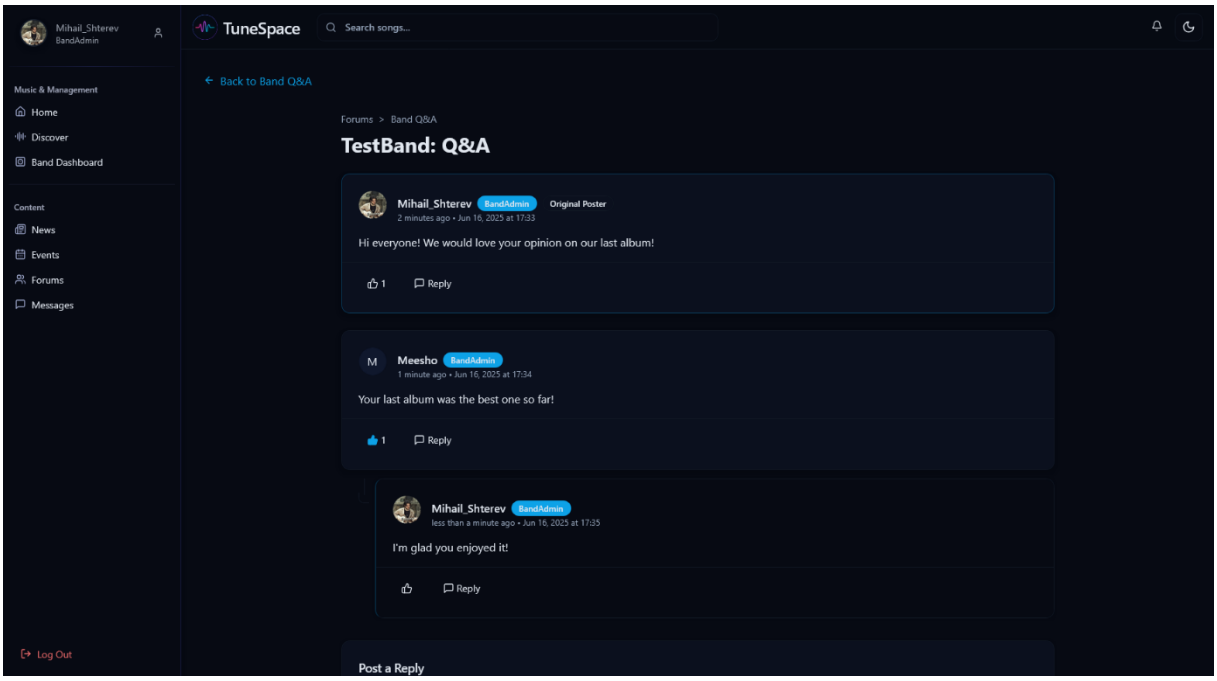
4.7 Поток на публикации във форума

Фигура 4.27 показва системата от форуми на общността на TuneSpace, където потребителите могат да създават дискусионни теми, да отговарят на публикации и да взаимодействат със съдържание чрез харесвания и известия.

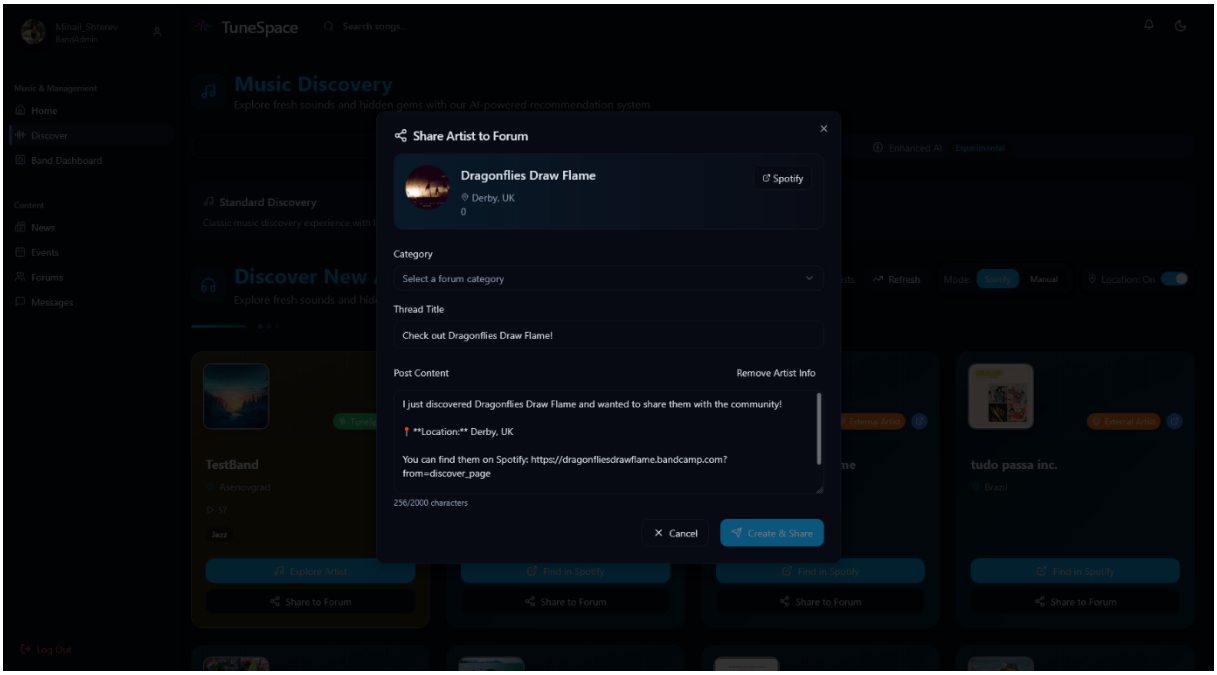




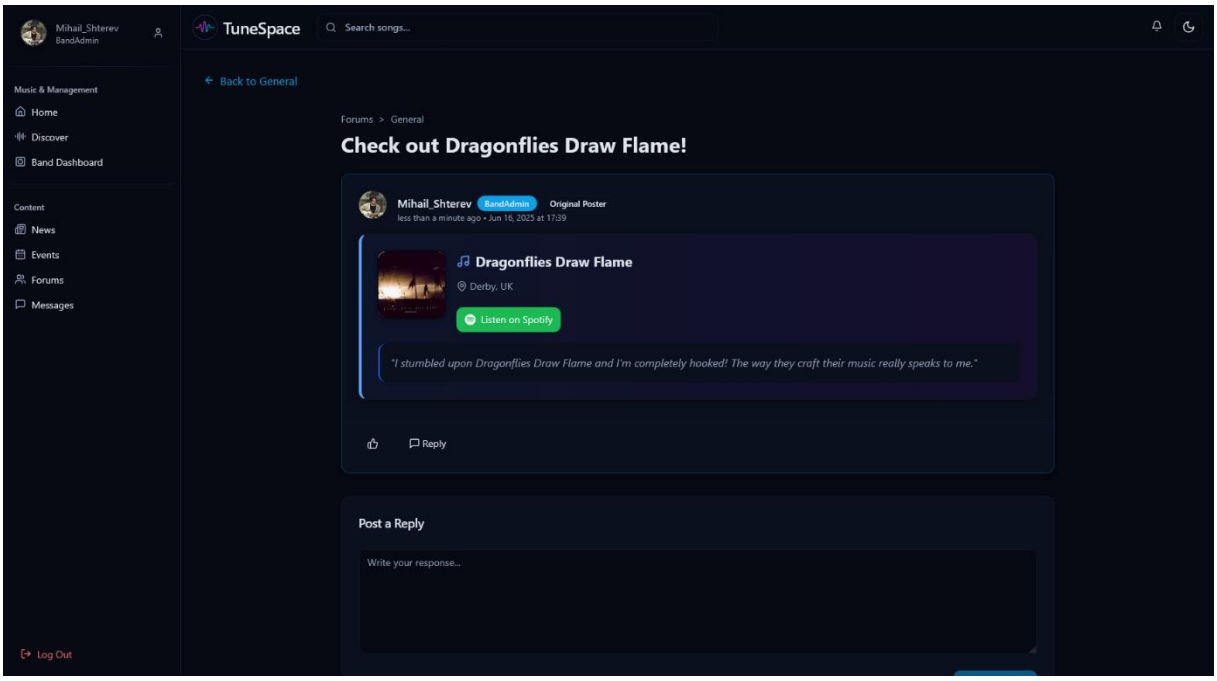
Фигура 4.27. Поток на публикации във форума



Фигура 4.28. Страница на форум



Фигура 4.29. Споделяне на изпълнител във форум



Фигура 4.30. Споделен изпълнител във форум

Форумната система на TuneSpace създава оживено пространство за музикални фенове, където те могат да обсъждат любими изпълнители, жанрове, концерти и преживявания. Темите са организирани по категории, което улеснява потребителите да намират и създават дискусии по интересуващи ги теми. Разговорите са структурирани в нишки, където всеки може да отговаря и да взаимодейства с други участници. Публикациите могат да бъдат харесвани, което насърчава стойностен принос и откроява полезни коментари. Системата изпраща известия при нови отговори или харесвания, поддържайки ангажираността и интереса към активните теми. Така се стимулира активност и се създава динамична общност, където музикалните ентусиасти могат да обменят идеи и да се свързват с хора със сходни интереси.

Заключение

В заключение, разработката на платформата TuneSpace демонстрира завършен, иновативен и технологично съвременен подход към създаването на музикална платформа, обогатена с възможности за изкуствен интелект. Проектът успешно интегрира многослойна архитектура, включваща Next.js за фронтенд, .NET API за бекенд логика и PostgreSQL с pgvector за съхранение и обработка на данни. Това осигурява стабилна и мащабируема основа за бъдещо развитие.

TuneSpace съчетава персонализирани AI препоръки, откриване на ъндърграунд изпълнители и интелигентно съпоставяне на музикални вкусове, като използва векторно търсене и разширени алгоритми. Реалновременната комуникация чрез SignalR, съчетана с адаптивен уеб интерфейс, превръща платформата в динамична и интерактивна среда за фенове и артисти.

Проектът е разработен в съответствие с добрите практики за структуриране на код, модулност и чиста архитектура. Благодарение на множеството интеграции с API на Spotify, Bandcamp, Last.fm и други, TuneSpace вече предлага богат набор от функции и силен потенциал за комерсиализация на международния пазар.

Планира се допълнително разширяване на функционалността чрез интеграция със стрийминг услуги и допълнителни музикални платформи като Apple Music и Tidal, системи за билети и търговия с музикални стоки, както и внедряване на разширена AI функционалност и аналитика, базирана на потребителското поведение. Всичко това ще допринесе за още по-добро потребителско изживяване и ще превърне TuneSpace в пълноценна платформа в сферата на музикалните технологии.

Чрез реализацията на този проект са придобити задълбочени знания в областта на full-stack разработката, интеграцията на изкуствен интелект, уеб технологиите в реално време и съвременните архитектурни решения. TuneSpace не само решава реални нужди на музикалната общност, но и демонстрира как технологиите могат да обогатят и трансформират начина, по който откриваме и преживяваме музиката.

ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

Факултет по Приложна математика и информатика

ДЕКЛАРАЦИЯ ЗА АВТОРСТВО НА ДИПЛОМНА РАБОТА

Долуподписаният: Михаил Илидонов Щерев

Специалност : Информатика и софтуерни науки Фак. № 471221045

ДЕКЛАРИРАМ:

Представената от мен дипломна работа на тема: **Платформа за популяризиране на независими музикални изпълнители и изграждане на общност около алтернативната музикална сцена** е лична моя авторска разработка, резултат от собствени изследвания.

Потвърждавам, че тя в нейната цялост и отделни части не е била използвана за придобиване на образователна и/или научна степен в ТУ - София или в други университети.

Формулировки, идеи и текстове, взети от други източници, са цитирани точно и с коректно посочване на техните автори. Дипломната работа не е публикувана на друго място.

Декларирам, че предоставям правото на Факултет Приложна

математика и информатика при ТУ - София, съгласно процедурите и правилниците на университета, да архивира и съхранява тази дипломна работа с цел доказване във времето на моето авторство.

Дата: Дипломант:

София

ИЗПОЛЗВАНА ЛИТЕРАТУРА

[1]. Visual Studio – IDE and Code Editor for Software Developers.
https://visualstudio.microsoft.com/

[2]. Visual Studio Code – Code Editing. Redefined.
https://code.visualstudio.com/

[3]. pgAdmin – PostgreSQL Tools.
https://www.pgadmin.org/

[4]. Next.js.
https://nextjs.org/docs

[5]. React – A JavaScript library for building user interfaces.
https://react.dev/

[6]. ASP.NET Core.
https://docs.microsoft.com/en-us/aspnet/core/

[7]. Entity Framework Core.
https://docs.microsoft.com/en-us/ef/core/

[8]. PostgreSQL.
https://www.postgresql.org/docs/

[9]. TypeScript.
https://www.typescriptlang.org/docs/

[10]. Radix UI Primitives.
https://www.radix-ui.com/primitives/docs/overview/introduction

[11]. Tailwind CSS.
https://tailwindcss.com/docs

[12]. Lucide React – Icon Library.
https://lucide.dev/guide/packages/lucide-react

[13]. React Icons.
https://react-icons.github.io/react-icons/

[14]. Styled-Components.
https://styled-components.com/docs

[15]. next-themes.
https://github.com/pacocoursey/next-themes

[16]. Zustand.
https://docs.pmnd.rs/zustand/getting-started/introduction

[17]. TanStack Query for React.
https://tanstack.com/query/latest/docs/framework/react/overview

[18]. Axios.
https://axios-http.com/docs/intro

[19]. React Hook Form.
https://react-hook-form.com/

[20]. React Hook Form Resolvers.
https://github.com/react-hook-form/resolvers

[21]. Zod.
https://zod.dev/

[22]. .NET Core SignalR.
https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction

[23]. Leaflet Reference.
https://leafletjs.com/reference.html
[24]. React-Leaflet.
https://react-leaflet.js.org/
[25]. ASP.NET Core Identity.
https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity
[26]. JWT Authentication in ASP.NET Core.
https://docs.microsoft.com/en-us/aspnet/core/security/authentication/jwt-authn
[27]. Microsoft.IdentityModel.Tokens API Reference.
https://docs.microsoft.com/en-us/dotnet/api/microsoft.identitymodel.tokens
[28]. ONNX Runtime.
https://onnxruntime.ai/docs/
[29]. Microsoft.ML.Tokenizers API Reference.
https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.tokenizers
[30]. pgvector.
https://github.com/pgvector/pgvector
[31]. pgvector-dotnet.
https://github.com/pgvector/pgvector-dotnet
[32]. FluentEmail.
https://github.com/lukencode/FluentEmail
[33]. Embla Carousel.
https://www.embla-carousel.com/
[34]. React Day Picker.
https://react-day-picker.js.org/
[35]. Sonner Notification Library.
https://sonner.emilkowal.ski/
[36]. date-fns.
https://date-fns.org/
[37]. The Clean Architecture by Robert C. Martin.
https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html
[38]. Microservice DDD and CQRS Patterns – Microsoft Docs.
https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design
[39]. Dependency Injection in .NET.
https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection
[40]. HTTP Resilience in .NET.
https://docs.microsoft.com/en-us/dotnet/core/resilience/http-resilience
[41]. PostCSS.
https://postcss.org/docs/
[42]. A Tour of C# – Microsoft Learn.
https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
[43]. HTTP Resilience with .NET CLI – Microsoft Learn.
https://learn.microsoft.com/en-us/dotnet/core/resilience/http-resilience?tabs=dotnet-cli

Приложение – изходен код

Пълният изходен код е наличен като приложение с отворен код в GitHub хранилище (repository) на този линк: https://github.com/Shredrox/TuneSpace