

# MINERVA

Децентрализирана peer-to-peer платформа  
за споделяне и стрийминг на музика

Проектна документация

2025/2026 г.

## **1. ТЕМА**

Minerva – Децентрализирана peer-to-peer (P2P) платформа за споделяне и стрийминг на музика.

Проектът представлява настолно приложение, което комбинира BitTorrent технология за децентрализирано разпределение на музикални файлове с модерен потребителски интерфейс за управление на музикална библиотека, търсене в мрежата от равноправни възли (peers), стрийминг на аудио и управление на плейлисти.

## **2. АВТОРИ**

(Информацията за авторите е пропусната по съображения за поверителност. Моля попълнете: три имени, ЕГН, адрес, телефон, имайл, училище, клас.)

## **3. РЪКОВОДИТЕЛ**

(Моля попълнете: три имени, телефон, имайл, длъжност.)

## **4. РЕЗЮМЕ**

### **4.1. Цели**

#### **Предназначение:**

Minerva е създадена с цел да предостави на потребителите средство за споделяне, откриване и слушане на музика чрез децентрализирана мрежа, без нужда от централен сървър. Всеки потребител е едновременно клиент и сървър – споделя музика от своята библиотека и може да открива и изтегля музика от други участници в мрежата.

#### **Анализ на потребностите:**

Съвременните платформи за музикален стрийминг (Spotify, Apple Music, YouTube Music) са централизирани – зависят от корпоративни сървъри, изискват абонамент и налагат ограничения върху наличното съдържание. Потребителите нямат контрол върху своите данни и библиотеки. Необходимо е решение, което дава пълен

контрол на потребителя и позволява свободно споделяне на музика в рамките на самоорганизираща се мрежа.

#### **Анализ на съществуващите решения:**

- BitTorrent клиенти (qBittorrent, Transmission) – мощни за файлов трансфер, но нямат музикален интерфейс, търсене по ключови думи или стрийминг.
- Децентрализирани музикални проекти (Audius, Funkwhale) – или са базирани на блокчейн (бавни, скъпи), или изискват централен сървър за федерация.
- Класически P2P (Napster, LimeWire, Soulseek) – липса на модерен интерфейс, слабо структуриране на метаданни, изоставени проекти.

Minerva запълва тази ниша, като комбинира изпитания BitTorrent протокол с DHT-базирано търсене по ключови думи, директен TCP протокол между Minerva възли и модерен Electron интерфейс за пълноценно музикално изживяване.

### **4.2. Основни етапи в реализирането на проекта**

#### **Етап 1: Проектиране на архитектурата**

Дефиниране на модулната структура: Java бекенд (REST API + BitTorrent двигател + DHT мрежа), Electron фронтенд (потребителски интерфейс), комуникационен протокол между възлите (MINERVA1).

#### **Етап 2: Реализация на ядрото**

Имплементиране на торент мениджъра (JLibTorrentManager) за сидване, изтегляне и управление на торенти чрез библиотеката jlibtorrent. Създаване на LibraryManager за управление на локалната библиотека, метаданни и файла структура.

#### **Етап 3: DHT и мрежово търсене**

Разработка на DHTKeywordManager за обявяване и търсене на ключови думи чрез DHT. Имплементиране на протокола MINERVA1 – TCP сървър (KeywordSearchServer) и клиент (KeywordSearchClient) за директна комуникация между Minerva възли.

#### **Етап 4: REST API и бекенд интеграция**

Създаване на BackendServer с Javalin framework – 25+ REST ендпойнта за стрийминг, търсене, качване, изтегляне, управление

на плейлисти и мониторинг на трансфери.

### **Етап 5: Потребителски интерфейс**

Разработка на Electron приложение с модерен дизайн – начална страница, библиотека с грид изглед, детайлен изглед на албуми, търсене в мрежата (Discover), мониторинг на изтеглянията (Downloads), upload wizard, плейлисти и аудио плейър.

### **Етап 6: Тестване и оптимизация**

Тестване с множество инстанции чрез Docker Compose и ръчно тестване с изолирани директории. Отстраняване на проблеми с peer discovery, торент сийдване, конкурентен достъп и native сегментации.

## **4.3. Ниво на сложност на проекта**

Проектът е с високо ниво на сложност поради комбинацията от множество технологии и необходимостта от коректна синхронизация между тях:

### **Проблем 1: DHT-базирано търсене по ключови думи**

BitTorrent DHT е проектиран за намиране на peers по infohash, не по ключови думи. Създаден е оригинален подход: всяка ключова дума се хешира (SHA-1) с наставка ".minerva" до детерминиран infohash, който се обявява в DHT. Така DHT се превръща в разпределена хеш-таблица за keyword-peers mapping.

### **Проблем 2: Разпознаване на Minerva възли от стандартни BitTorrent peers**

Тъй като Minerva използва стандартен DHT, обикновени BitTorrent клиенти могат да се свържат. Създаден е TCP хендшейк протокол "MINERVA1", който позволява разпознаване на Minerva възли – само peers, отговарящи с правилния хендшейк, се заявяват за ключови думи.

### **Проблем 3: Конкурентен достъп до native библиотека**

jlibtorrent използва native C++ код чрез SWIG bindings. Едновременен достъп от множество нишки води до сегментационни грешки (SIGSEGV). Решението включва ReentrantReadWriteLock за всички SessionManager операции и отложено пресийдване на импортирани торенти до рестартиране.

### **Проблем 4: Създаване на валидни .torrent файлове**

Генерирането на .torrent файлове изисква правилно bencode

кодиране с валидни piece hashes. Използвана е комбинация от torrent за създаване при качване и TorrentInfo.bencode() от jlibtorrent за запис при изтегляне от мрежата.

#### **Проблем 5: Peer свързване между локални инстанции**

TorrentHandle.connectPeer() не е наличен в Java обвивката на jlibtorrent 2.0.11.0. Намерено е решение чрез директно извикване на SWIG слоя: th.swig().connect\_peer(new tcp\_endpoint(address.from\_string(host), port)).

#### **Проблем 6: Дедупликация на резултати от търсенето**

Търсенето в мрежата връща резултати от множество peers. Дедупликацията само по torrentHash губи песни от различни албуми. Имплементиран е композитен ключ torrentHash|title за прецизна дедупликация, запазваща разнообразието на резултатите.

### **4.4. Логическо и функционално описание на решението**

#### **Обща архитектура:**

Minerva следва двуслойна архитектура: Java бекенд сървър (REST API + P2P двигател) и Electron фронтенд (настолно приложение). Комуникацията между тях е чрез HTTP REST API на localhost. P2P комуникацията с други Minerva възли е чрез BitTorrent протокол (UDP/TCP) и MINERVA1 TCP протокол.

#### **Модул 1: JLibTorrentManager (Торент двигател)**

Отговорност: Управление на libtorrent сесията – сийдане, изтегляне, DHT комуникация.

- Singleton инстанция с ReentrantReadWriteLock за нишкова безопасност
- Конфигурация: настройваем порт, DHT bootstrap възли, 250 конекции, 100 активни seeds
- seedTorrent() – добавя торент за сийдане с forceRecheck за верификация
- seedMagnet() – резолва magnet URI, изтегля метаданни, стартира изтегляне
- Alert handler – следи TORRENT\_FINISHED, PEER\_CONNECT, PEER\_DISCONNECTED
- saveTorrentFile() – записва .torrent файл чрез bencode сериализация

## **Модул 2: LibraryManager (Управление на библиотеката)**

Отговорност: CRUD операции върху музикалната библиотека, метаданни, файлова структура.

- Файловая структура:  
library/{Артист}/{Албум}/{DD\_TT\_Заглавие.ext}
- Метаданни: JSON файлове в torrents/ директория (TorrentMetadata)
- loadLibraryFromTorrents() – зарежда всички песни от метаданни файловете
- importCompletedDownload() – премества файлове от downloads/ в library/, създава .json
- uploadSingleFile() / uploadAlbum() – обработка на качени файлове
- announceAllKeywords() – извлича ключови думи и ги обявява в DHT
- search() / searchLocal() – локално търсене по заглавие/артист/жанр

## **Модул 3: DHTKeywordManager (DHT търсене по ключови думи)**

Отговорност: Обявяване и търсене на ключови думи чрез BitTorrent DHT мрежата.

- keywordToInfohash() – SHA-1(keyword.toLowerCase() + ".minerva") → infohash
- announceKeyword() – dhtAnnounce() за регистриране на възела за дадена ключова дума
- searchKeyword() – двустранно търсене: (1) директно в известни peers, (2) чрез DHT peers
- Дедупликация с композитен ключ, паралелно изпълнение с 4s timeout

## **Модул 4: KeywordSearchServer / KeywordSearchClient (MINERVA1 протокол)**

Отговорност: Директна TCP комуникация между Minerva възли за търсене по ключови думи.

- Протокол: клиентът изпраща "MINERVA1\n" → сървърът отговаря "MINERVA1\n" → клиентът изпраща "keyword.minerva\n" → сървърът връща JSON масив с резултати

- ServerSocket с CachedThreadPool, 2s connect timeout, 3s read timeout
- Отхвърля заявки без .minerva наставка (фильтрира стандартни BitTorrent peers)

## **Модул 5: BackendServer (REST API)**

Отговорност: HTTP API за комуникация с фронтенда, оркестрация на всички подсистеми.

- Javalin framework на настройваем порт (по подразбиране 4567)
- 25+ REST ендпойнта: tracks, albums, playlists, stream, download, upload, search, dht-search, fetch-torrent, downloads management, cover art
- CORS поддръжка за cross-origin заявки
- Download completion callback – свързва TORRENT\_FINISHED alert c library import
- pendingDownloads ConcurrentHashMap за проследяване на метаданни за текущи изтегляния

## **Модул 6: TorrentCreator (Създаване на торенти)**

Отговорност: Създаване на .torrent файлове от качени аудио файлове.

- createSingleTorrent() / createAlbumTorrent() – използва ttorrent библиотека
- Копира файлове в библиотеката със стандартизиирани имена
- 10 публични трекера за по-добра peer discovery
- Автоматично изчисляване на SHA-1 infohash и преименуване на .torrent файла

## **Модул 7: PlaylistManager (Плейлисти)**

Отговорност: Управление на потребителски плейлисти.

- JSON-базирано съхранение с атомични записи (temp file → rename)
- ReentrantReadWriteLock за нишкова безопасност
- Системен плейлист "Liked Songs" (не може да се изтрие)
- CRUD: създаване, редактиране, изтриване, добавяне/премахване на песни, пренареждане

## **Модул 8: MusicMetadataExtractor (Извличане на метаданни)**

Отговорност: Извличане на ID3 тагове от аудио файлове.

- Използва jaudiotagger за четене на тагове: заглавие, артист, албум, жанр, номер на песен и др.
- Извличане на аудио параметри: битрейт, продължителност, формат, sample rate
- Извличане на вградено album art → base64 кодиране + запис на диск
- Fallback парсиране от име на файл (формати: "Артист - Заглавие", "Артист\_Заглавие")

## **Модул 9: Electron Frontend (Потребителски интерфейс)**

Отговорност: Настолно приложение с модерен музикален интерфейс.

- main.js – Electron main process, BrowserWindow, IPC handlers
- preload.js – Context bridge, expose window.minerva API
- Модулна JavaScript архитектура: 17 специализирани модула
- CSS архитектура: base/ (reset, variables, typography), layout/ (app, sidebar, player-bar), components/ (buttons, discover, downloads, forms, home, library, modals, playlist, track-list), utils/ (scrollbar, utilities), responsive.css

### **Взаимодействия между модулите:**

1. Потребителят взаимодейства с Electron интерфейса.
2. Electron изпраща IPC заявки → main.js → HTTP към BackendServer.
3. BackendServer делегира към LibraryManager (библиотека), JLibTorrentManager (торенти), PlaylistManager (плейлисти), DHTKeywordManager (мрежово търсене).
4. JLibTorrentManager комуникира P2P с други Minerva възли чрез BitTorrent + DHT.
5. KeywordSearchServer/Client осъществява директна TCP комуникация за търсене.
6. При изтегляне: TORRENT\_FINISHED alert → callback → importCompletedDownload() → файлове се преместват в библиотеката.

## **4.5. Реализация**

### **Програмни езици и платформи:**

- Java 17 – бекенд (REST API, торент двигател, DHT, мрежови протоколи)

- JavaScript (ES6+) – фронтенд (Electron, модулна архитектура)
- HTML5/CSS3 – потребителски интерфейс (CSS Grid, Flexbox, CSS Custom Properties)

#### **Ключови библиотеки и фреймуъркове:**

- Javalin 4.6.8 – лек REST API фреймуърк за Java
- jlibtorrent 2.0.11.0 (frostwire) – native BitTorrent и DHT имплементация с SWIG bindings към libtorrent-rasterbar C++
- ttorrent-core 1.5 – създаване на .torrent файлове
- jauditotagger 3.0.1 – извличане на аудио метаданни (ID3v1, ID3v2, Vorbis)
- Jackson 2.15.2 – JSON сериализация/десериализация
- Electron 28 – настолно приложение с Chromium рендер
- Feather Icons – SVG иконна библиотека
- SLF4J 2.0.12 + Logback 1.4.14 – логиране

#### **Инструменти за разработка:**

- Apache Maven – build management, dependency resolution, shade plugin за fat JAR
- npm / electron-builder – управление на Electron зависимости
- Docker + Docker Compose – контейнеризация и тестване с множество възли
- Git – версионен контрол

#### **Ключови алгоритми:**

- SHA-1 хеширане на ключови думи за DHT mapping (keyword → deterministic infohash)
- Паралелно търсене с ExecutorService и CompletionService за мрежови заявки
- Релевантност при търсене: праг  $\geq 50\%$  съвпадение на ключови думи, ранкиране по брой съвпадения
- Композитна дедупликация (torrentHash|title) за резултати от множество peers
- Атомични файлови записи (temp file → rename) за JSON персистенция
- forceRecheck алгоритъм за верификация на piece hashes след добавяне на торент

#### **4.6. Описание на приложението**

#### **Стартиране и инсталлиране:**

Предварителни изисквания:

- Java 17 или по-нова версия (JRE или JDK)
- Node.js 18+ и npm (за Electron фронтенда)
- Native libtorrent библиотека за съответната платформа (включена за Linux x86\_64)

Стартиране на бекенда:

```
# Компилиране  
mvn package -DskipTests
```

```
# Стартiranе
```

```
java -Djava.library.path=natives/lib/x86_64 -jar target/minerva-1.0.0.jar
```

Конфигуриране чрез променливи на средата:

- API\_PORT (подразбиране: 4567) – HTTP порт за REST API
- SEARCH\_PORT (подразбиране: 4568) – DHT и TCP порт за мрежово търсене
- LIBRARY\_DIR (подразбиране: library) – директория за музикалната библиотека
- TORRENT\_DIR (подразбиране: torrent\_files) – директория за .torrent файлове
- DOWNLOADS\_DIR (подразбиране: downloads) – директория за изтегляния
- ALBUM\_ART\_DIR (подразбиране: album\_art) – директория за album art
- DHT\_BOOTSTRAP\_NODES – адреси на известни Minerva възли (формат: host:port,host:port)

Стартиране на фронтенда:

```
cd minerva-electron  
npm install  
npm start
```

Стартиране с Docker Compose (две инстанции за тестване):

```
docker-compose up --build
```

## **Как се използва:**

### **Начална страница (Home)**

При стартиране потребителят вижда поздравително съобщение

(различно според часа на деня) и три секции с по 6 карти на албуми от библиотеката, произволно разбръкани. Щракването върху карта показва детайлния изглед на албума.

### **Библиотека (Library)**

Показва всички албуми от локалната библиотека в CSS Grid изглед с обложка, заглавие, артист, година и брой песни. Албумите се зареждат от /api/albums ендпойнта.

### **Търсене (Search)**

Полето за търсене в горната лента извършва локално търсене в библиотеката по заглавие и артист.

### **Мрежово търсене (Discover)**

Позволява търсене в P2P мрежата. Потребителят въвежда заявка, системата я разделя на ключови думи и изпращай заявки към всички известни Minerva peers и DHT peers. Резултатите се групират по торент хеш (албуми срещу единични песни) и се показват като карти с бутон за изтегляне. Филтри: All / Albums / Singles.

### **Изтегляния (Downloads)**

Мониторинг на активните торент трансфери с актуализация на всеки 2 секунди. Показва общ брой торенти, скорост на качване/сваляне, статус (downloading/seeding/paused), прогрес бар и контроли за пауза/подновяване/премахване.

### **Качване (Upload)**

Тристъпков wizard: (1) Избор на файлове – автоматично разпознаване на единична песен или албум. (2) Попълване на метаданни – автоматично извлечени от ID3 тагове, с възможност за ръчна редакция (заглавие, артист, албум, година, жанр, обложка). (3) Потвърждение и качване.

### **Плейлисти**

Потребителят може да създава, редактира и изтрива плейлисти от страничната лента. Системният плейлист "Liked Songs" се попълва чрез бутона ❤ на всяка песен. Контекстното меню на всяка песен позволява добавяне към плейлист или към опашката.

### **Аудио плейър**

Долната лента съдържа информация за текущата песен (обложка, заглавие, артист), контроли за възпроизвеждане (предишна/play-pause/следваща), прогрес бар с възможност за seek,

и регулатор на силата на звука. Аудиото се стриймва от бекенда чрез HTML5 `<audio>` елемент. При край на песента автоматично се пуска следващата от опашката.

### **Опашка (Queue)**

Линейна опашка от песни. При пускане на албум/плейлист/резултат от търсене, цялата опашка се заменя. Отделни песни могат да се добавят чрез контекстното меню. Модален прозорец показва текущата опашка с възможност за премахване и изчистване.

### **Как се поддържа:**

Приложението не изиска специална поддръжка. Данните се съхраняват като файлове на диска (музикални файлове, .torrent файлове, JSON метаданни, JSON плейлисти). За добавяне на нов възел е достатъчно да се стартира нова инстанция с DHT\_BOOTSTRAP\_NODES, сочещ към съществуващ Minerva възел. Обновяването става чрез повторно компилиране и рестартиране.

## **4.7. Заключение**

### **Основен резултат:**

Създадена е напълно функционална децентрализирана P2P платформа за споделяне и стрийминг на музика. Системата позволява качване на музика, автоматично споделяне чрез BitTorrent, търсене по ключови думи в мрежата от Minerva възли, изтегляне и импортиране в локалната библиотека, стрийминг на аудио и управление на плейлисти – всичко това без централен сървър.

### **Текущо състояние:**

Проектът е тестван успешно с множество инстанции (чрез Docker Compose и ръчно с изолирани директории). Торент трансферите работят коректно между възлите, DHT търсенето връща релевантни резултати, а изтеглените албуми се импортират в библиотеката и се зареждат при следващото стартиране.

### **Възможности за развитие:**

- Автоматично пресийдане след импорт – елиминиране на необходимостта от рестартиране чрез решаване на native threading проблемите с jlibtorrent
- Криптирана комуникация – TLS за MINERVA1 протокола и

криптирани торент конекции

- Потребителски профили – децентрализирана идентификация с публичен/частен ключ
- Автоматично откриване на peers – mDNS/Bonjour за локални мрежи без ръчна конфигурация
- Мобилно приложение – React Native или Flutter фронтенд със същия REST API бекенд
- Препоръчителна система – базирана на колаборативно филтриране между peers
- Поддръжка на повече формати – FLAC, OGG, WAV, AAC освен MP3
- Last.fm интеграция – вече частично имплементирана (OAuth flow), за пълно scrobbing
- Подобрена UI анимация и достъпност – keyboard navigation, screen reader поддръжка