Software Engineering Project 3

UUUTorrent

Technical Report

BEST SE TEAM EVER
(UNNAMED UNARMED UNTAMED)

Sambu Aneesh

Vidhi Rathore

Bassam Adnan

Moida Praneeth Jain

Aadi Deshmukh

Contents

1 Requirements and Subsystems

1.1 Requirements

1.1.1 Functional Requirements (FR)

- **FR1: Remote Torrent Download & Transfer:** Utilize an Oracle Cloud compute instance with qBittorrent for downloading torrents. Provide authenticated users a mechanism (via TUI) to initiate downloads and securely transfer completed files (e.g., via user-initiated SFTP/SCP) to local devices.
- **FR2: User and Access Management:** Implement user authentication (login/password) and authorization. Define distinct roles: 'User' (manages personal downloads, feeds, watchlist) and 'Admin' (manages users, system monitoring access).
- **FR3: RSS Feed Management & Automation:** Allow users to add, view, remove, and define filter criteria (keywords, regex) for RSS feeds. The system backend must periodically check feeds, match against filters, and auto-initiate downloads for new items via qBittorrent.
- **FR4: Watchlist Integration & Automation:** Allow users (via TUI) to manage a watchlist (add/remove shows). The backend must integrate with external services (e.g., Anilist API) to fetch watchlist details and identify new episodes/content based on user entries, triggering downloads (potentially correlated with RSS feeds).
- **FR5: Torrent Lifecycle Management:** Enable users (via TUI) to view download status (downloading, seeding, completed, error), pause, resume, and delete their torrents managed by the system.
- **FR6: Admin Monitoring View:** Provide administrators access to a web-based dashboard (Grafana) displaying real-time server resource usage (CPU, RAM, disk, network), qBittorrent statistics, database health, and potentially aggregated user activity/stats.
- **FR7: User Interface (TUI):** Provide a cross-platform Terminal User Interface (Textual) for user interaction: displaying status, adding torrents/feeds, managing the watchlist, and getting information for file transfers. Runs on the user's machine.

1.1.2 Non-functional Requirements (NFR)

- **NF1: Performance (RSS Processing):** Check each RSS feed at least every 10 minutes. Processing time per feed check (filtering, download initiation) < 30 seconds under normal load.
- **NF2: Performance (Concurrency):** Support at least 50 concurrent torrent downloads/seeds without significant API/TUI responsiveness degradation. (Network throughput depends on OCI/peers).
- **NF3: Reliability (Backup & Recovery):** Perform automated daily database backups (PostgreSQL). RTO: 1 hour. RPO: 24 hours.
- **NF4: Security (Authentication):** Secure user authentication (hashed+salted passwords). Protect API endpoints with token-based authentication (e.g., JWT).

- **NF5: Security (Authorization):** Enforce authorization rules: users manage only their own resources; admins have specific elevated privileges (user management, monitoring access).
- **NF6: Usability (TUI):** The TUI should be intuitive for terminal users, providing clear feedback and navigation.
- **NF7: Resource Constraints (OCI):** Operate within OCI Always Free tier limits (Ampere A1 CPU/ RAM, storage, network egress). Monitor usage.
- **NF8: Maintainability:** Ensure modular, documented backend code with tests for easier updates.
- **NF9: Monitoring Availability & Freshness:** The Admin monitoring view (Grafana) should be highly available (best effort on free tier) and display metrics with a latency of <= 1 minute from collection time.

1.1.3 Architecturally Significant Requirements

The following requirements have the most significant impact on the architectural design:

- **NF1 & NF2 (Performance):** Drive choices for async backend, database tuning, OCI instance sizing.
- **NF3 (Reliability):** Dictates backup strategy and operational procedures.
- **NF4 & NF5 (Security):** Fundamental; require secure design of API, auth flows, and data storage.
- **NF6 & FR7 (Usability/TUI):** Drive the choice of Textual and interaction design.
- **NF7 (Resource Constraints):** A primary constraint influencing technology choices and scale.
- **FR6 & NF9 (Monitoring):** Dictates the monitoring stack (Prometheus/Grafana) and its configuration.
- **FR4 (Watchlist Integration):** Requires careful design of backend interaction with external APIs.

1.2 Subsystem Overview

1.2.1 User Interface (TUI - Textual):

1.2.1.1 Role

The primary point of interaction for the end-user. It is a client application designed to run directly on the user’s local machine or accessed via SSH, but operating independently from the backend services.

1.2.1.2 Functionality

- Presents views for displaying current torrent download status (name, size, progress, speed, status like downloading/seeding/paused/error), list of subscribed RSS feeds, and watchlist items.
- Provides input mechanisms for users to:
  - Add new torrents (via magnet link or potentially .torrent file upload handled via the API).
  - Add/remove RSS feeds and configure their associated filters (e.g., keywords, regex).
  - Add/remove shows from their personal watchlist.
  - Trigger actions on existing torrents (pause, resume, delete).

  - Initiate user authentication (login/logout).

1.2.1.3 Interaction

Communicates exclusively with the API Backend over the network via secure HTTPS requests, sending user commands and receiving data to display. It does not directly interact with qBittorrent, the database, or external APIs like Anilist.

1.2.2 API Backend (FastAPI):

1.2.2.1 Role

The central coordinating service running on the OCI instance. It acts as the “brain” of the system, mediating all interactions between the TUI, the torrent client, the database, and external services.

1.2.2.2 Functionality

- Exposes a secure RESTful API (HTTPS/JSON) for the TUI client.
- Handles user authentication (verifying credentials) and authorization (ensuring users only access their own resources) using techniques like JWT.
- Manages user sessions and API tokens.
- Receives requests from the TUI and translates them into actions:
  - Instructing the qBittorrent service (via its Web API) to add, pause, resume, delete torrents, and fetch status updates.
  - Storing and retrieving user data, feed subscriptions, filters, torrent metadata, and watchlist information from the PostgreSQL database.
  - Periodically scheduling and executing RSS feed checks: fetching content, applying filters, identifying new torrents, and triggering downloads.
  - Interacting with external Watchlist APIs (e.g., Anilist) to fetch user watchlist data, search for shows, and potentially identify new episodes.
  - Correlating watchlist information with RSS feeds or other sources to automate downloads for new episodes.
  - May provide administrative endpoints for user management (if required).
- Exposes a `/metrics` endpoint for Prometheus to scrape performance and operational metrics.

1.2.2.3 Interaction

Listens for requests from the TUI; sends commands to qBittorrent Web API; executes queries against the PostgreSQL database; makes outgoing requests to external RSS feed URLs and Watchlist APIs (Anilist); allows Prometheus to scrape its metrics endpoint.

1.2.3 Torrent Client Service (qBittorrent):

1.2.3.1 Role
The core download engine running as a separate process (qBittorrent server) on the OCI instance. It handles the complexities of the BitTorrent protocol.

1.2.3.2 Functionality
- Manages the entire lifecycle of torrent downloads: connecting to trackers and peers, downloading/uploading data pieces, verifying file integrity.
- Handles seeding (uploading) after downloads complete, based on configured rules.
- Parses `.torrent` files and magnet links.
- Maintains the state of active and completed torrents.
- Saves downloaded files to a designated location on the OCI instance's filesystem.
- Provides a Web API that the API Backend uses to control its operations remotely.

1.2.3.3 Interaction
Communicates with external Torrent Trackers and Peers over the internet using the BitTorrent protocol (TCP/UDP). Exposes a Web API (typically HTTP) on a local port for the API Backend to interact with. Reads/writes files to the OCI instance's File Storage.

1.2.4 Database (PostgreSQL)

1.2.4.1 Role
The persistent storage layer for the application's state, running on the OCI instance.

1.2.4.2 Functionality
- Stores user account information (usernames, securely hashed passwords, roles).
- Maintains records of torrents being managed by the system (torrent hash, name, associated user, status flags, potentially download path).
- Stores RSS feed subscriptions for each user, including the feed URL and associated filter rules (keywords, regex patterns).
- Stores user watchlist information (e.g., user ID, show ID from external service like Anilist).
- Potentially stores historical download statistics or logs (though this might be limited to avoid excessive storage use on free tier).

1.2.4.3 Interaction
Responds to SQL queries from the API Backend (CRUD operations - Create, Read, Update, Delete). Provides metrics data to the pgexporter for the Monitoring Service.

1.2.5 Monitoring Service (Prometheus + Grafana + Exporters):

1.2.5.1 Role

Collects, stores, and visualizes system health and performance metrics, running on the OCI instance. Primarily used by the Administrator.

1.2.5.2 Functionality

- Prometheus: Periodically scrapes metrics data from configured targets (node_exporter for OS metrics, pgexporter for PostgreSQL stats, the API Backend's /metrics endpoint). Stores this time-series data.
- Grafana: Queries Prometheus for data and presents it in user-friendly dashboards accessible via a web interface. Visualizes CPU/RAM/Disk/Network usage, database performance (connections, query times), torrent activity (if exposed via API metrics), API request rates/ latency, etc.
- Exporters (node_exporter, pgexporter): Small helper services that gather metrics from the host OS and PostgreSQL, respectively, and expose them in a format Prometheus understands.

1.2.5.3 Interaction

Prometheus makes outgoing HTTP requests to scrape targets. Grafana serves a web UI to the Administrator's browser and queries Prometheus internally. Exporters interact with the components they monitor (OS kernel interfaces, PostgreSQL).

1.2.6 Watchlist API (External - e.g., Anilist):

1.2.6.1 Role

An external, third-party service that provides data about media (like anime or TV shows) and allows users to manage their personal watchlists.

1.2.6.2 Functionality

- Exposes an API (e.g., GraphQL for Anilist) allowing authorized applications (like our API Backend) to:
- Search for media titles.
- Fetch details about specific shows (episode count, release dates, etc.).
- Retrieve the contents of a specific user's watchlist (requires user authentication/authorization with the external service, managed potentially via OAuth flow initiated by the user through the backend or stored API keys).

1.2.6.3 Interaction

Receives API requests (GraphQL/HTTPS) from the API Backend and returns data (JSON). It is external to the OCI instance and the core Torrent Proxy system.