Project Clacks

Technical Design Document

Version 1.2.0

Lionel Morrison - 7/1/2023

Project Overview

Project Clacks gets its name from the world of Terry Pratchett. The Clacks was a system by which letters could be transcribed and communicated via light to another Clacks station.

In this project we will be developing a universal RESTful API that allows for Rich Communication Services (RCS) and Real-time messaging that can be integrated into any platform.

System Overview

The API will be written in Nest.js for the API; Google Firestore Database for the messages and other data and Google Storage to host all the files.

Components

- Authentication - As this is an API, we will only need only need need a basic key / value pair which returns a JWT for that the client uses for all subscuient API calls. If the JWT expires, re-authentication will be required.
- RTM - Real-time Messaging, is the core of the system. The place where people communicate.
  - Delivery receipt
  - Read receipt
  - Typing notification
  - Last seen
  - Group Messaging
  - Quote-in-reply
  - Hyperlink support
  - Image support - WebP, JPG, PNG, GIF
  - Audio playback support - Lyra, MP3, MP4, AAC
  - Video playback support - WebM, MP4
  - File Transfer support - Zip, PDF

- File Storage - All files need to be stored somewhere so the conversation can be picked up on another device
  - Features should include, Upload, Download, Delete
  - Security needed to scan the files for malware to ensure the safety of user-uploaded content

Preferences - Global preferences for your communication

- Profile
  - Away status
  - Avatar
  - Display Name

Architecture

The overall style will be that of Client-Server with the appropriate hooks or the use of WebSockets necessary for Real Time communication between the Clients. The demo site will be written in React and Angular using a Mobile First approach. The API of the site will be written in Nest.js. This allows us to use TypeScript natively and use Google Firebase as for the hosting, database store and file storage.

Functional Requirements

Auth API endpoint - the user will need to auth once per session

Registration API endpoint - will have a user registration endpoint. This will take their name and email address. The credentials to access the system will be generated by the system as a Key Value pair.

Profile API endpoint - will allow the user to set some basic settings. Status, Avatar and Display Name

RTM API endpoint - manages the message storage and delivery.

Storage API endpoint - manages the access and storage of files uploaded

Non-functional Requirements

Performance and scalability requirements

It's essential to conduct performance testing and ensure that the API meets the desired performance benchmarks. Monitor and optimize the system as needed.

Security measures (authentication, authorization, encryption)

Authentication will be handled by Google Authentication

To access a RTM chat stream you will need to be Authenticated and have Authorization to the streams. All files messages and files within that stream will be available to all that have access to the stream.

All messages and files will be encrypted to the message stream. Allowing for end-to-end encryption

API documentation and versioning

Setting up a developer page for API documentation and versioning is a recommended practice. It helps developers understand how to use the API effectively and provides a way to manage changes and backward compatibility.

System Components

Client application

For the demonstration of this project a React Client website will be set up.

API server

The API will be written in Nest.js and hosted on Google Functions. Each API endpoint will run as a separate function.

Database

We will utilize a NoSQL approach with Google Firestore Database for all the database needs.

External services and APIs

Apart from Google Cloud, CircleCI is the only external service which will be a development dependency.

Data Storage and Persistence

Data models and schema design

Object model will be mapped out using TypeScript interfaces

API Design

API endpoint design and naming conventions

- Endpoints will use nouns
- Will be in lowercase letters
- No endpoint shall have special characters other that a hyphen
- Avoid deep nesting
- No file extensions
- Consistent naming
- Use of camelCase for parameters
- Use API Versioning

Request/response formats

Payload data going to the API in the form of a Request should be sent to the API in the BODY as a JSON string. Using GET, POST, UPDATE, PUT and DELETE verbs.

The API will response with the appropriate HTTP status code and any Payload from will be in the form of a JSON string

Error handling and response codes

All responses will use standard HTTP Status Codes https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/418

200 OK
201 Created
400 Bad Request
401 Unauthorized
403 Forbidden
404 Not Found
405 Method Not Allowed
408 Request Timeout
409 Conflict
410 Gone
415 Unsupported Media Type
500 Internal Server Error
501 Not Implemented
503 Service Unavailable

Test-Driven Development (TDD)

Explanation of TDD methodology and benefits

TDD helps cut the hours needed to develop the API as we will scope out the behavior of the API first, write test cases for the happy and sad paths before we write our code. This was as we're developing the API we will be testing before we have written a single line of code and which will help us avoid some of the pitfalls of writing tests only after code is written and ensure thorough test coverage for all critical functionalities.

Unit testing frameworks and tools for Nest.js

Jest and Supertest are popular and capable testing frameworks for Nest.js and utilizing them effectively, will help create unit tests and ensure the reliability of the API.

Security Considerations

Input validation and sanitization

In order to prevent SQL Injection and cross-site scripting (XSS), all input should be treated as though it is contaminated and will be sanitized and validated.

Encryption for secure communication (HTTPS)

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) v1.3 or greater will be used.

Deployment

Deployment architecture

The API will be deployed and hosted on Google Hosting which is a cloud based infrastructure

Infrastructure requirements (servers, network, storage)

For this demo, requirements will be kept to a minimum

CI/CD pipelines for automated deployment

CircleCI will be utilized for automated deployment, code validation and test runners.