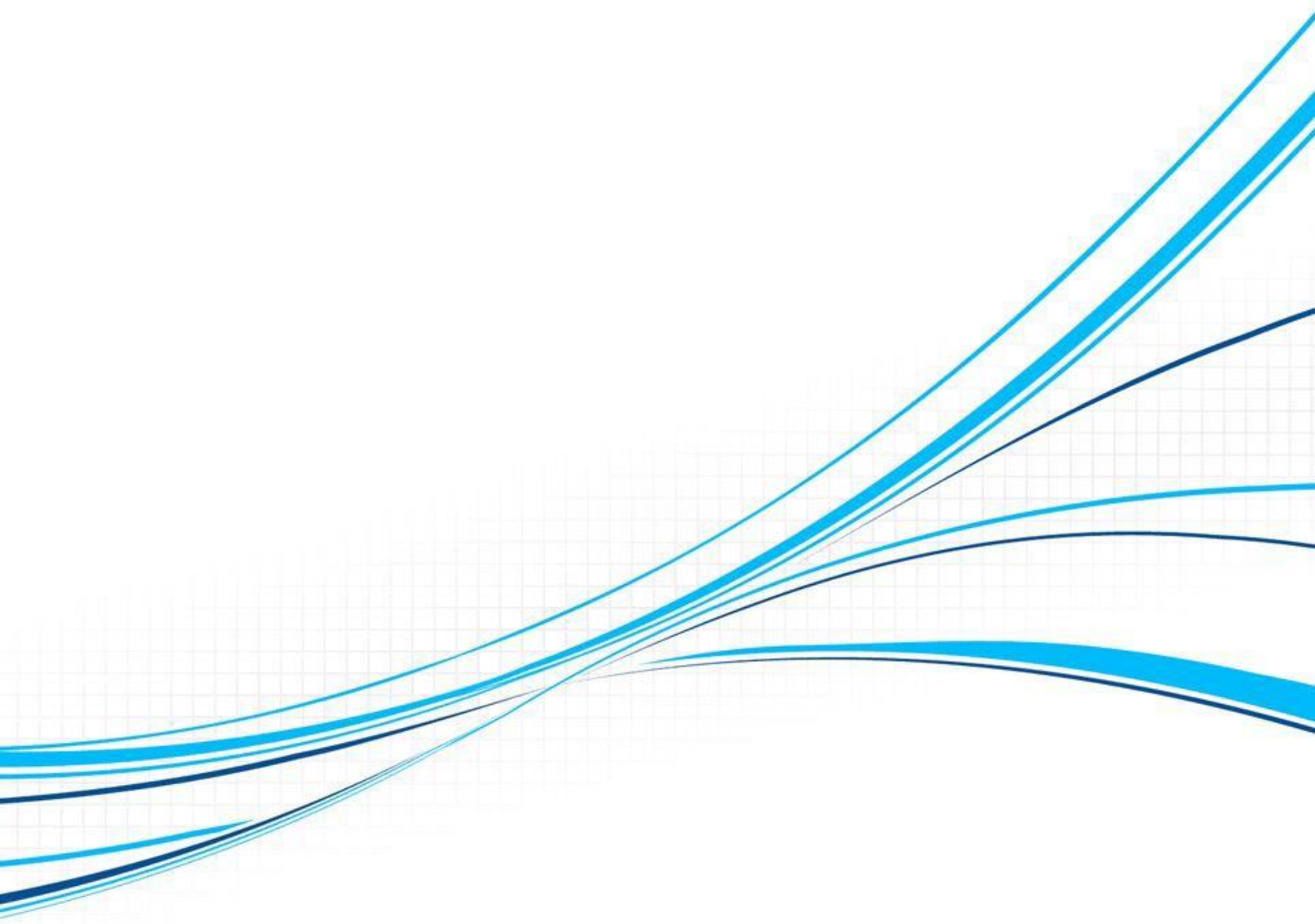
Software Requirements Specification

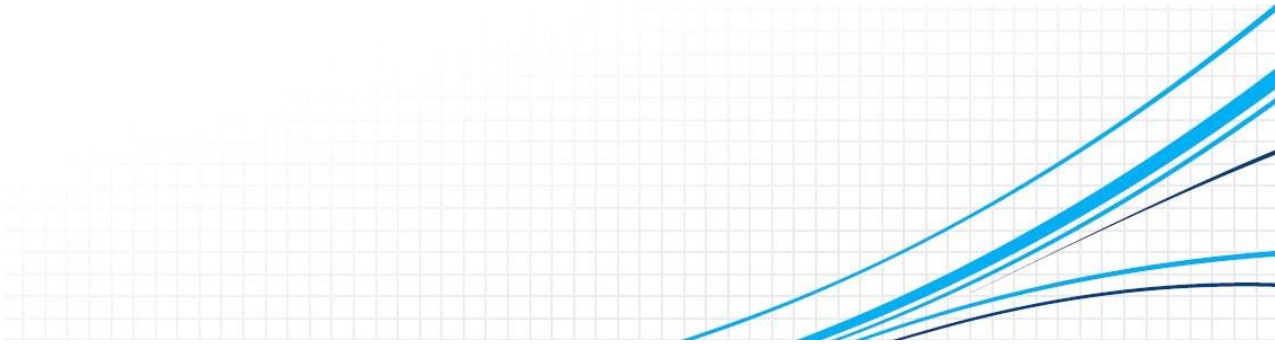
Notezilla Project

Created by Antoine Ratat

Version 1.0 - Issued February 04,2023

This requirements specification is used to record the user requirements.





1. VERSIONS

Ver.	Author(s)	Date	Description
1.0	Antoine RATAT	04/02/2023	Creating Document

2. INTRODUCTION

2.1 Purpose

Notezilla simplifies the note-taking experience by providing a centralized platform accessible from anywhere in the world. Users can create, edit, and organize their notes effortlessly, fostering productivity and efficient information management.

1. **Cross-Device Accessibility**: Access your notes from any device with an internet connection, ensuring flexibility and convenience.

2. **Real-Time Synchronization**: Seamlessly sync your notes across devices in real-time, ensuring that the latest updates are always available.

3. **User-Friendly Interface**: An intuitive and easy-to-use interface allows users to create, edit, and organize notes with minimal effort.

4. **Secure Cloud Storage**: Notes are securely stored in the cloud, providing a reliable and safe repository for your valuable information.

2.2 Project Scope

NoteZilla is a comprehensive online note-taking platform specifically designed for individuals and small businesses venturing into digital note organization and collaboration. The project scope encompasses a set of essential features ensuring a user-friendly experience.

The platform prioritizes accessibility, enabling users to seamlessly access and manage their notes from any device with an internet connection. NoteZilla's responsive design ensures optimal user experience across desktops, tablets, and smartphones.

The project scope for NoteZilla revolves around delivering a versatile, user-friendly, and free note-taking platform that caters to the needs of both individuals and small businesses.

2.3 References

React - https://reactjs.org/
React Router - https://reactrouter.com/web/guides/quick-start
React Bootstrap - https://react-bootstrap.github.io/
React Spinners - http://www.davidhu.io/react-spinners/
React Toastify - https://www.npmjs.com/package/react-toastify/v/1.4.3
Country List JS - https://www.npmjs.com/package/country-list-js
Country Flag Icon - https://www.npmjs.com/package/country-flag-icons

Formik - https://www.npmjs.com/package/formik
Yup - https://www.npmjs.com/package/yup
React Token Auth - https://www.npmjs.com/package/react-token-auth
Flask - https://flask.palletsprojects.com/en/1.1.x/
Flask Mail - https://flask-mail.readthedocs.io/en/latest/
ItsDangerous - https://pypi.org/project/itsdangerous/
Flask SQLalchemy - https://flask-sqlalchemy.palletsprojects.com/en/2.x/
Flask JWT Extended - https://flask-jwt-extended.readthedocs.io/en/stable/
Flask Bcrypt - https://flask-bcrypt.readthedocs.io/en/latest/
Moment - https://www.npmjs.com/package/moment

3. DESCRIPTION

3.1 Product Perspective

The product is a full-stack web application implemented on both the front and back end, it is using API calls to operate the database.

3.2 Features

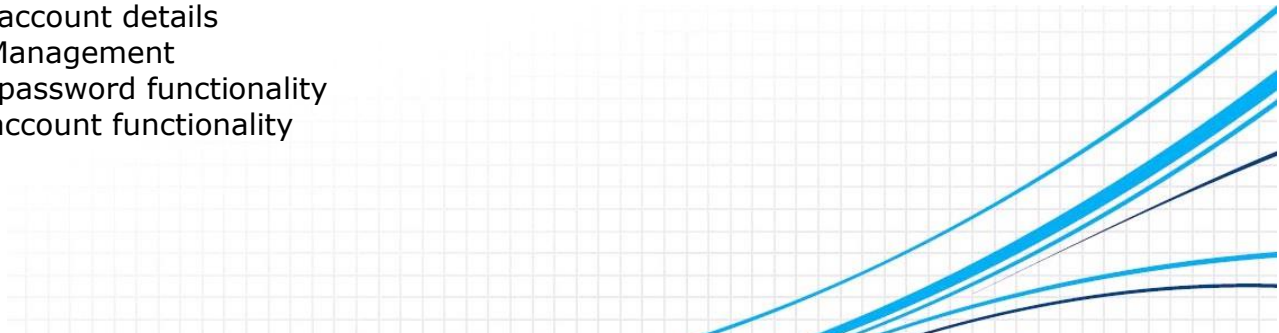
The Shop platform system provides a simple mechanism for users to acquire information.

The following are the main features that are included in the system:

➔ **Notes**
- Notes Display
- Create note
- Toggle note completion status
- Edit existing note
- Delete existing note-taking
- Auto-save note-taking
- Sort by date
- Sort by completion status

➔ **Scratchpad**
- Only displays one scratchpad
- Auto-save scratchpad
- Manually save scratchpad
- Clear scratchpad

➔ **User Management**
- Create account
- Login, Register, Logout
- Update account details
- Profile Management
- Update password functionality
- Delete account functionality



➔ **Performance**
- Async Data Loading
- Each page as a loading state to handle API delay
- Each page as an Error state if no data is returned from the API
- Disable actions while loading

➔ **Security**
- HTTPS use to transmit sensitive data
- JSON Web Token to Authenticate user and transmit information in payload
- Use remote API with Cross-Origin Resource Sharing (CORS) to allow communication origin
- Data validation in Front-end (Formik & Yup)
- Data validation in Back-end
- Verify user email to activate account

➔ **Language**
- Website is completely in English

➔ **Design Chart**
- Use strict graphic charter to create consistency and uniformity on the Website (colors, font, graphics)
- Responsive - Website design will automatically adjust for different screen sizes and viewports. Using CSS to hide, shrink or enlarge Website elements
- Using Bootstrap for element positioning

3.3 User Overview

It is considered that the user does have the basic knowledge of operating the internet and to have access to it.

The administrator is expected to be familiar with HTML, CSS, JavaScript, React library, AJAX (Asynchronous JavaScript and XML), React Router, React Pagination, React Token Authentication, React ChartJS2, React Bootstrap, JWT Decode, Formik, Yup, and Styled components to handle Front-End's side. Regarding the Back-End perspective the administrator is expected to be familiar with Python, Flask, Flask-SQLalchemy, CORS configuration, JWT Tokens, Authentication, and Password hashing mechanisms.

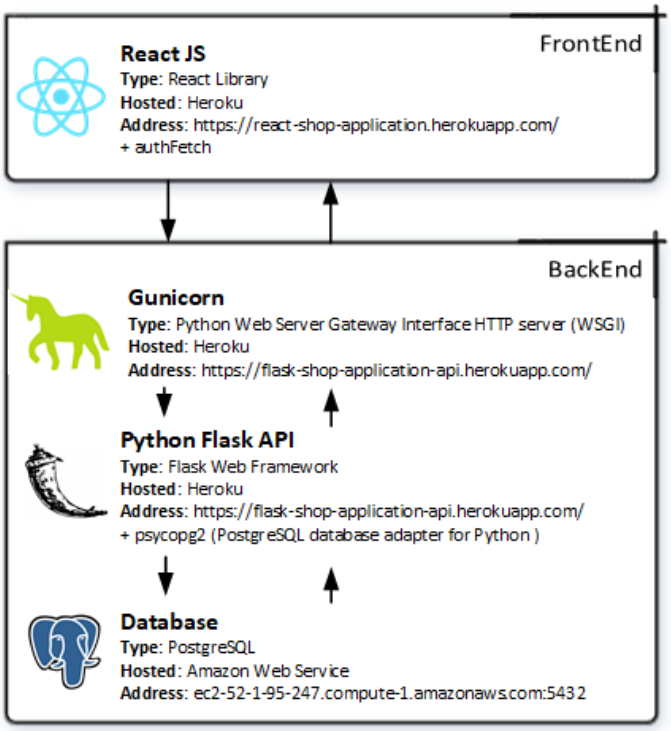
3.4 Operating Environment

This is a web-based system and hence will require the operating environment for a client and server GUI.

Dependencies: This software highly depends on the type and version of the browser being installed in the system i.e. browser version should be used which has HTML5 support.

3.5 Constraints: Implementation / Design

3.5.1 Application Communication Schema



3.5.2 Client-side rendering (frontend)

This system is provisioned to be built in JavaScript using React library which is highly flexible.

The browser will be in charge of rendering this application in its final form, HTML. Some of the logic involved in creating the web page, especially the one in charge of dealing with presentation logic is handled on the client-side.

List of frontend dependencies and version used:

@reduxjs/toolkit v2.0.1,
@testing-library/jest-dom v6.2.0,
@testing-library/react v14.1.2,
@testing-library/user-event v14.5.2,
antd v5.13.1,
formik v2.4.5,
jwt-decode 4.0.0,
lodash v4.17.21,
react v18.2.0,
react-dom v18.2.0,
react-joyride v2.7.2,
react-redux v9.1.0,
react-router-dom v6.21.2,
react-scripts v5.0.1,
react-token-auth v2.3.8,
web-vitals v3.5.1,
yup v1.3.3,
serve v11.3.2

3.5.3 Server-side rendering (backend)

The system is also provisioned to connect and contact a custom-made API. This API handles CRUD request and manage operations on the database.

The API is build using Python with Flask web framework. The application server would be served with GUnicorn (Python WSGI HTTP Server for UNIX)

The API will interface the database using SQLAlchemy. SQLAlchemy is an object-relational mapper (ORM) and provides the data mapper pattern, where classes can be mapped to the database in open-ended, multiple ways - allowing the object model and database schema to develop in a cleanly decoupled way from the beginning.

Authentication will be handled using Flask-JWT-Extended which adds support for using JSON Web Tokens (JWT) to Flask for protecting views.

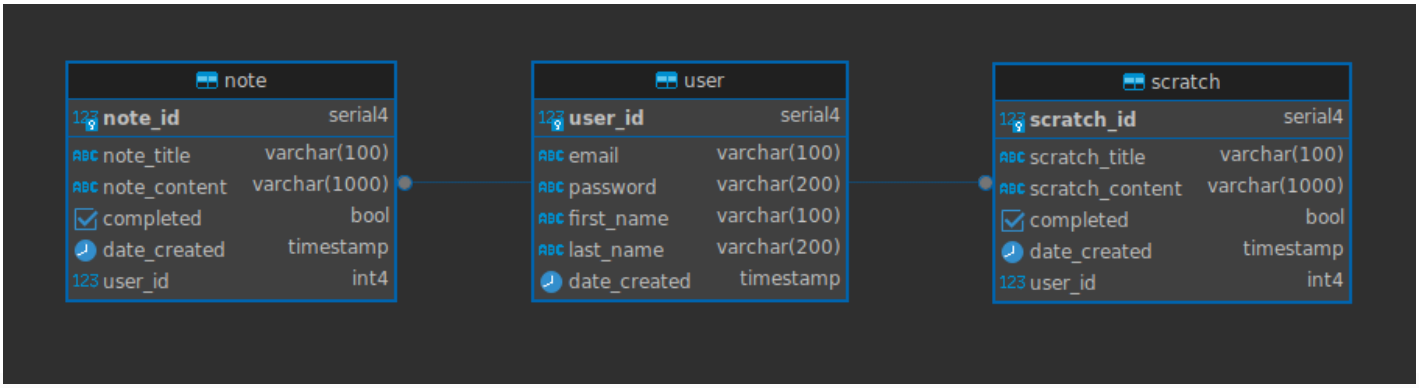
List of used dependencies and version used:

bcrypt v3.2.0
cffi v1.14.6
click v7.1.2
colorama v0.4.4
DateTime v4.3
Flask v1.1.4
Flask-Bcrypt v0.7.1
Flask-Cors v3.0.10
Flask-JWT-Extended v3.25.0
Flask-SQLAlchemy v2.5.1
greenlet v1.1.2
gunicorn v20.1.0
itsdangerous v1.1.0
Jinja2 v2.11.3
MarkupSafe v2.0.1
packaging v23.2
psycopg2 v2.9.1
psycopg2-binary v2.9.9
pycparser v2.20
PyJWT v1.7.1
python-dotenv v1.0.0
pytz v2021.3
six v1.16.0
SQLAlchemy v1.4.25
Werkzeug v1.0.1
zope.interface v5.4.0

The database itself is using PostgreSQL and is accessible for UI management tools like PgAdmin. It would be hosted on AWS (Amazon Web Service) and available on port 5432.

The database would contain the following tables: Note, User, Scratch.

You can find an ER Entity Relationship Diagram to understand its structure:



The user's browser should be HTML5 compatible for a satisfactory user experience.

3.6 System Feature Frontend

3.6.1 Not Authenticated Features

- Create an account
- Login

3.6.2 Authenticated Features

Navigation Experience
- Browse Notes
- Sort notes

User Information
- Edit Profile Information
- Update Password
- Delete Account
- Logout

Notes
- Create, edit, delete notes
- Edit, clear scratchpad

3.7 System Feature Backend

3.7.1 User features

- Allow user to create an account
- Allow user to login
- Allow user to disconnect
- Allow the user to edit profile
- Allow user to delete their account
- Allow user to create, edit, delete note
- Allow user to create, edit, delete scratchpad

User account is only available after the first login with User account authenticated with User JWT.

3. 8 System Feature Database

3.8.1 Note Table

The note table is designed to store information related to individual notes. Each entry includes a unique note_id as the primary key, capturing details such as the note_title, note_content, completed status, date_created, and the associated user_id. The user_id serves as a foreign key, establishing a relationship with the user table.

3.8.2 Scratch Table

The scratch table is dedicated to storing data about scratch notes. Similar to the note table, it includes a unique scratch_id as the primary key. Information such as scratch_title, scratch_content, completed status, date_created, and the associated user_id is recorded. The user_id acts as a foreign key, linking each scratch entry to a specific user in the user table.

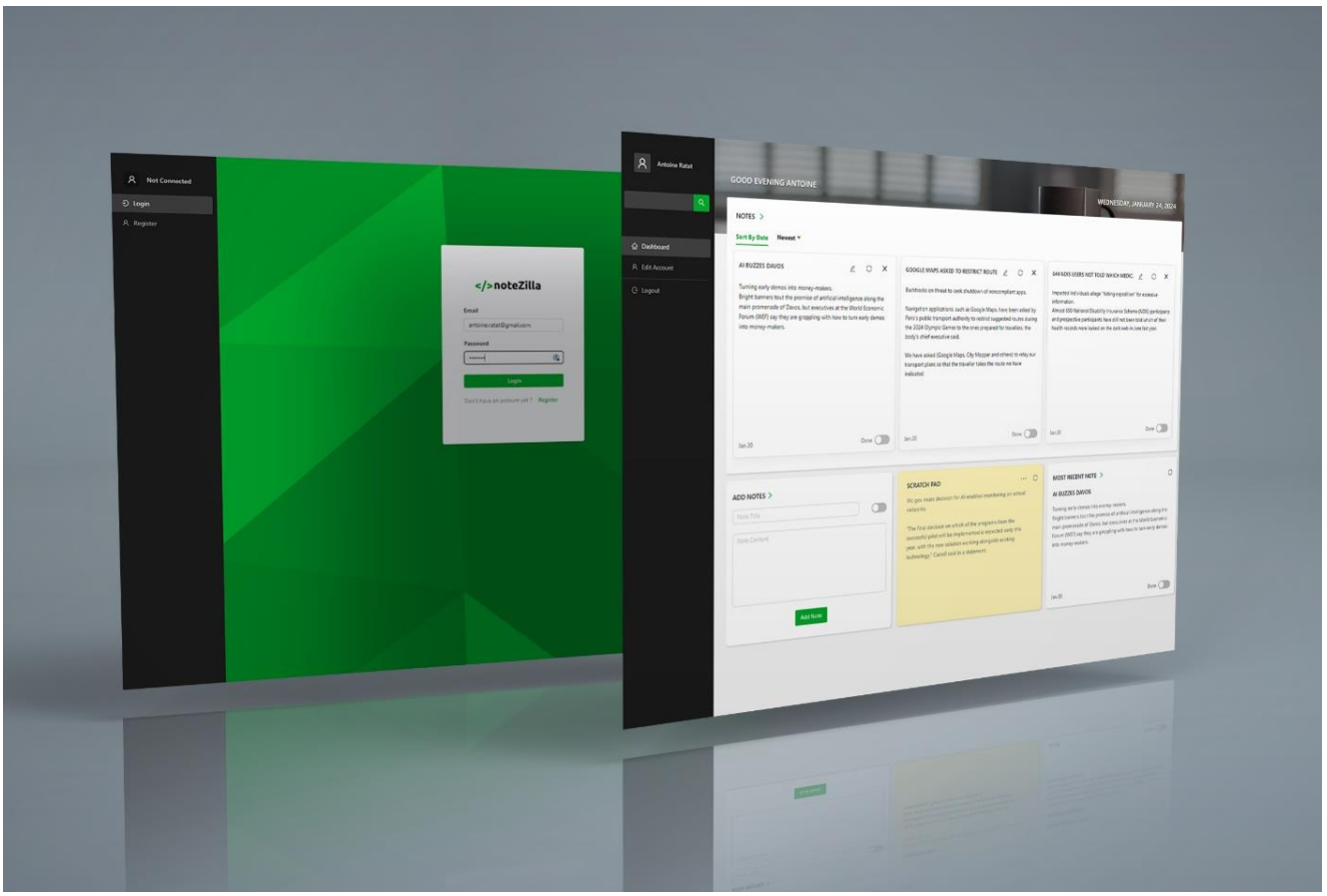
3.8.3 User Table

The user table serves as a repository for user-related details. Each user is uniquely identified by a user_id, which acts as the primary key. The table stores essential information such as email, password (hashed), first_name, last_name, and date_created. The user_id is referenced in both the note and scratch tables, establishing relationships between users and their respective notes.

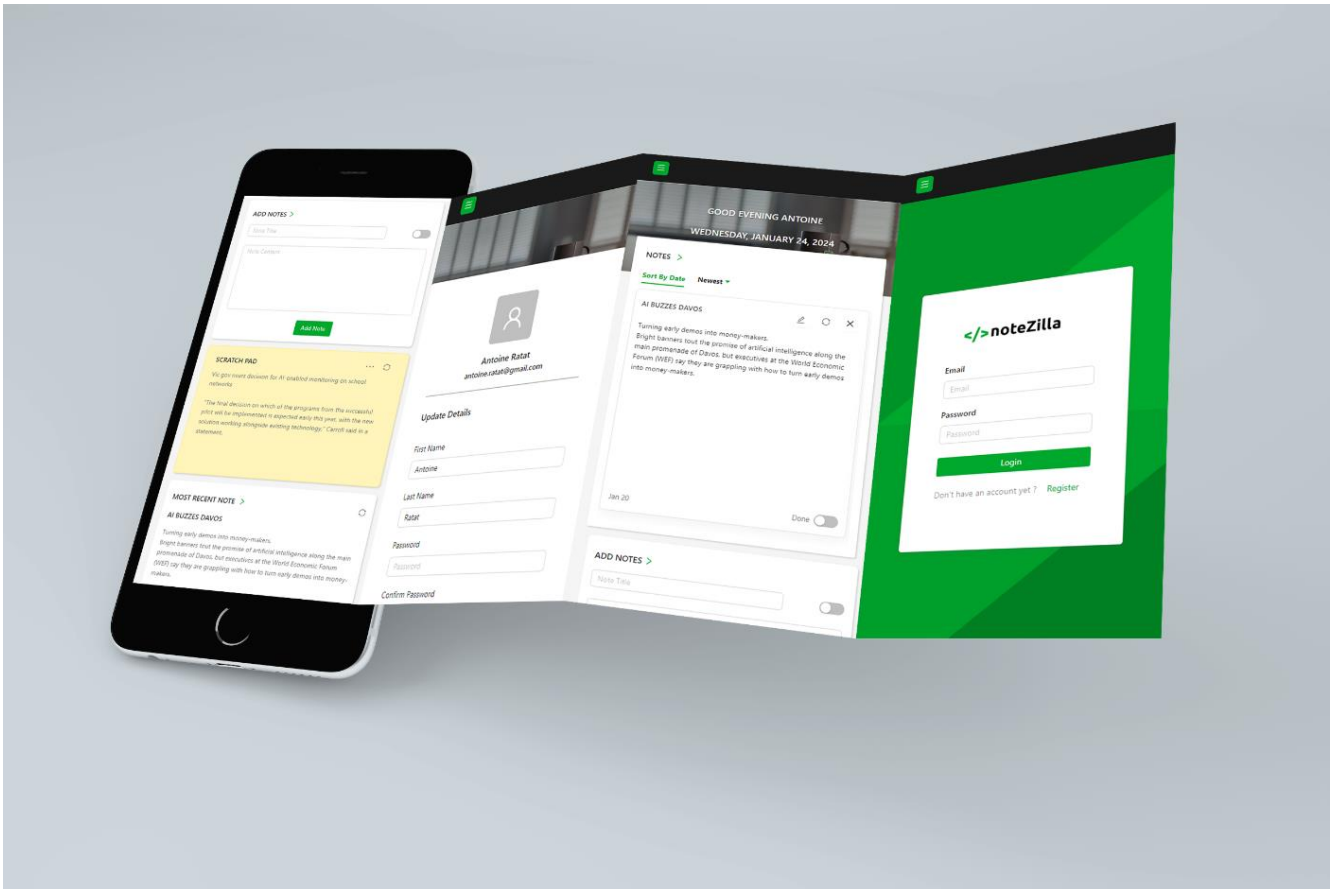
4. REQUIREMENTS OF EXTERNAL INTERFACE

4.1 User Interfaces

Main Pages Full-Screen Desktop Mockup

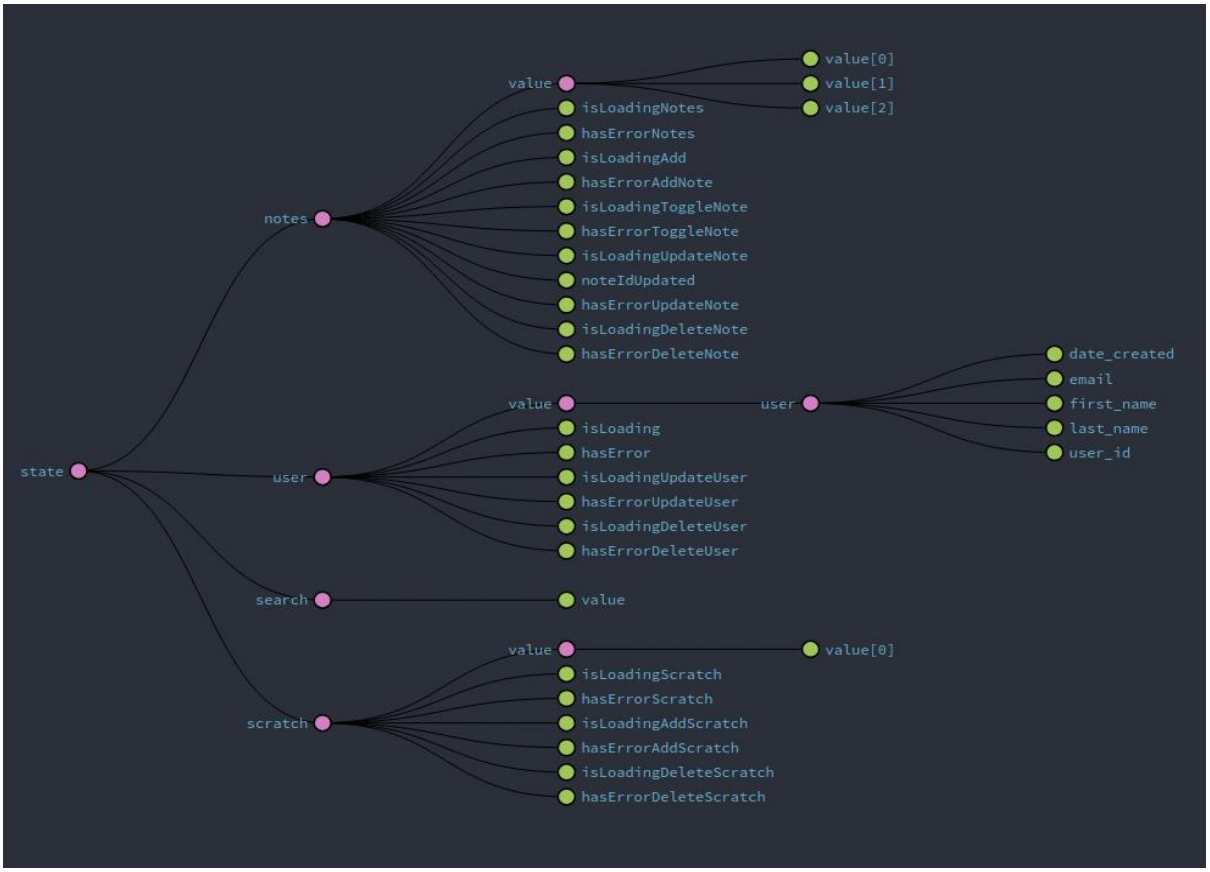


Main Pages Mobile Version Mockup



4.2 Software Interfaces

Front end stores application states in a centralized store. This store is composed of four mains slices: notes, user, search, scratch.



4.3 Communication Interfaces

Communication is assured to external interfaces. The system is connected to several APIs using REST, the payload is defined in the request itself and is formatted in JSON. This API is a CRUD API created with Python and Flask to communicate safely with a PostgreSQL database.

4.3.1 Admin Notes CRUD

```
GET /api/notes Return JSON with all notes.
https://antoineratat.xyz/api_notezilla/api/admin/notes

POST /api/notes Post new note from JSON
https://antoineratat.xyz/api_notezilla/api/admin/notes
{"note_title": "note_title", "note_content": "note_content", "completed": "completed", "user_id":
"user_id"}

GET /api/note/id Return JSON selected note.
https://antoineratat.xyz/api_notezilla/api/admin/note/1

PUT /api/note/id Update selected note from JSON
https://antoineratat.xyz/api_notezilla/api/admin/note/1
{"note_title": "note_title", "note_content": "note_content", "completed": "completed", "user_id":
"user_id"}

DELETE /api/note/id Delete selected note.
https://antoineratat.xyz/api_notezilla/api/admin/note/1

GET /api/note/user/user_id Return JSON with all notes for user.
https://antoineratat.xyz/api_notezilla/api/admin/note/user/1
```

4.3.2 User Notes CRUD

```
GET /api/notes Return JSON with all notes for user.
https://antoineratat.xyz/api_notezilla/api/notes

POST /api/notes Post new note from JSON for user
https://antoineratat.xyz/api_notezilla/api/notes
{"note_title": "note_title", "note_content": "note_content", "completed": "completed"}

GET /api/note/id Return JSON selected note for user.
https://antoineratat.xyz/api_notezilla/api/note/1

PUT /api/note/id Update selected note from JSON
https://antoineratat.xyz/api_notezilla/api/note/6
{"note_title": "note_title", "note_content": "note_content", "completed": "completed"}

DELETE /api/note/id Delete selected note for user.
https://antoineratat.xyz/api_notezilla/api/note/6
```

4.3.3 Admin User CRUD

```
GET /api/admin/users Return JSON with all users.
https://antoineratat.xyz/api_notezilla/api/admin/users

POST /api/admin/users Post new user from JSON
https://antoineratat.xyz/api_notezilla/api/admin/users
{"email": "email", "password": "password", "first_name": "first_name", "last_name": "last_name"}

GET /api/admin/user/id Return JSON selected user.
https://antoineratat.xyz/api_notezilla/api/admin/user/id

PUT /api/admin/user/id Update selected user from JSON
https://antoineratat.xyz/api_notezilla/api/admin/user/id
{"password": "password", "first_name": "first_name", "last_name": "last_name"}
```

```
DELETE /api/admin/user/id Delete selected user.
https://antoineratat.xyz/api_notezilla/api/admin/user/id
```

4.3.4 User CRUD

```
GET /api/user Return JSON with user information.
https://antoineratat.xyz/api_notezilla/api/user

PUT /api/user Update connected user from JSON
https://antoineratat.xyz/api_notezilla/api/user
{"password": "password", "first_name": "first_name", "last_name": "last_name"}

DELETE api/user Delete connected user.
https://antoineratat.xyz/api_notezilla/api/user

POST /api/login Login user and return JWT
https://antoineratat.xyz/api_notezilla/api/login
{"email": "email", "password": "password"}

POST /api/register Post new user and userdetail from JSON
https://antoineratat.xyz/api_notezilla/api/register
{"email": "email", "password": "password", "first_name": "first_name", "last_name": "last_name"}
```

5. NON-FUNCTIONAL REQUIREMENTS

5.1 Performance

The system must be interactive and the delays involved must be fewer. So, in every action-response of the system, there are no immediate delays. In the case of opening new windows, popping error messages, and saving the settings or sessions there is a delay depending on the API response time and availability.

In the case of calling external APIs, the delay is based on editing on the distance of the secondary system and the configuration between them so there is a high probability that there will be or not a successful connection in less than 5 seconds for sake of good communication.

The following also needs to be implemented:

- Async Data Loading: To Load prior fast elements
- Each page has a loading state to handle API delay
- Each page has an Error state if no data is returned from the API
- Disable actions while loading

5.2 Availability

If the internet service gets disrupted while fetching information from the external APIs, the information can be fetched again later.

The system is highly dependent on external APIs, if the communication between the client and the API is disrupted, the information could be requested again.

API contacted: https://antoineratat.xyz/api_notezilla

5.3 Security

- Check input data with Form Validation using Formik and Yup
- Back end verification before any database operation
- Authentication mechanism with JWT
- HTTPS transfer protocol to secure communication over the network
- Hashing sensitive information with Bcrypt before storing it in the database
- Storing an application's sensitive information in an environment variable on the server
- JSON Web Token to Authenticate user and transmit information in the payload
- Use remote API with Cross-Origin Resource Sharing (CORS) to allow communication origin

5.4 Usability

As the system is easy to handle and navigates in the most expected way with no delays. In that case, the system program reacts accordingly and transverses quickly between its states.