**SIP Phone Configuration Management System**

**Report Summary**

**Introduction**

This report describes the development and implementation of a web-based system for managing the configuration of SIP phones and other devices. The system allows users to perform CRUD (create, read, update, and delete) operations on phone and configuration data, as well as generate and download XML files for configuring the devices. The system uses Node.js, TypeScript, MySQL, React, Sequelize and Tailwind CSS as the main technologies. The report also discusses the limitations and challenges of the system, and suggests some possible improvements and future work.

**System Overview**

The system consists of two main components: a server-side API and a client-side website. The server-side API is responsible for handling the requests from the website, interacting with the database, and generating, retrieving and deleting configuration XML files. The client-side website is responsible for providing the user interface, displaying the data, and allowing the user to perform CRUD and configuration operations.

The system follows a modular and layered architecture, which separates the different processes and logic into different folders, files, and modules. The system also uses different layers to separate the business logic, data access, and presentation.

The system uses MySQL as the database management system, and Sequelize as the ORM (object-relational mapping) library. The system uses the base tables provided by the project document to store the configuration data and the user data. The system uses foreign keys, primary keys, and indexes to define the relationships and constraints between the tables.

The system uses React as the library for building the user interface and for connecting to server-side APIs. The system also uses various libraries and tools to enhance the React development, such as React Router, Axios, React-Icons, and more. The system also uses Tailwind CSS for styling and theming the website.

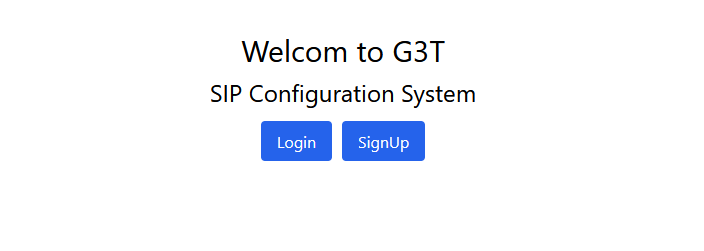
The system uses TypeScript as the superset of JavaScript that adds static typing and other features, to write more robust and maintainable code.

The system uses XML as the markup language that defines the rules for encoding the configuration files. The standard file system package (fs) of node js is used to parse and generate XML files.

**System Features**

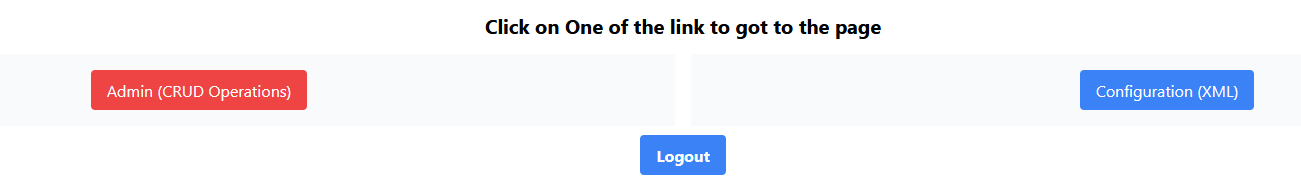
The system provides the following features and functionalities:

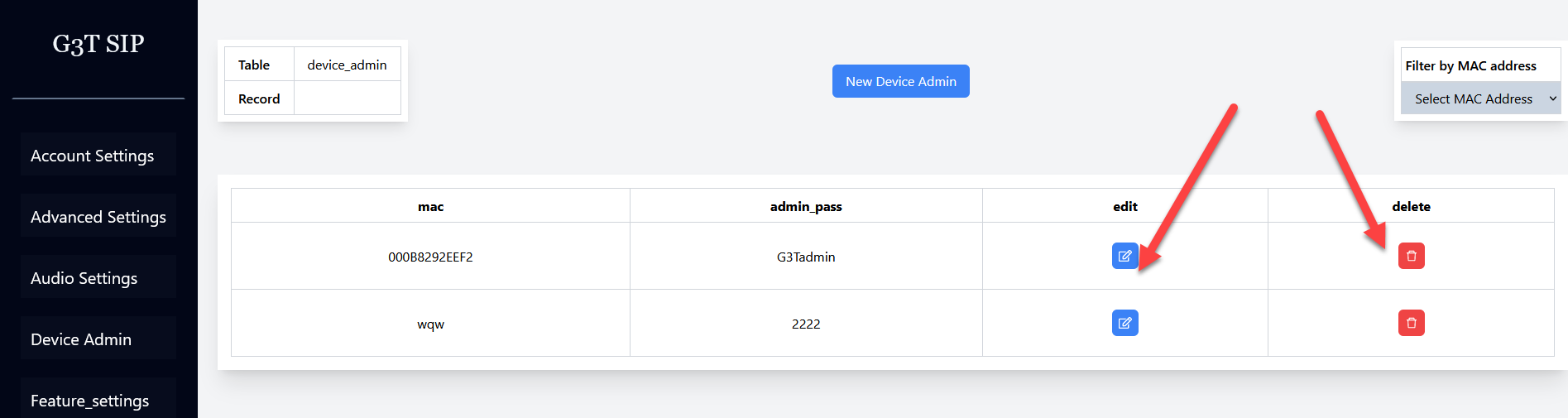
* User authentication: The system requires the user to login before they can access the CRUD and configuration operations. The system uses a login and registration system that validates the user credentials and stores them in the database. The system uses JWT token based authentication that stores tokens in the user’s browser localStorage to maintain the user state and security.



|  |  |
| --- | --- |
|  |  |

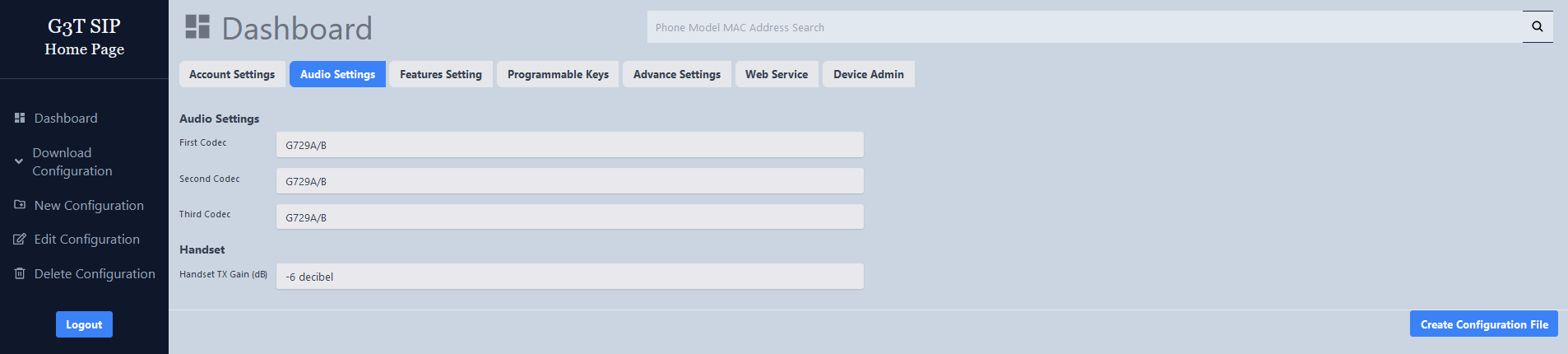
* CRUD operations: The system allows the user to perform CRUD operations on phone and configuration data. The system provides a link to the CRUD interface, where the user can see a list of all the phone and configuration data stored in the database. The system also provides buttons and forms to read, update, create, and delete the data. The system uses Axios to send and receive the data from the server-side API, and React hooks such as UseState and UseEffect to manage the state of the data in the website.

  
*fig4 Home page for login user*



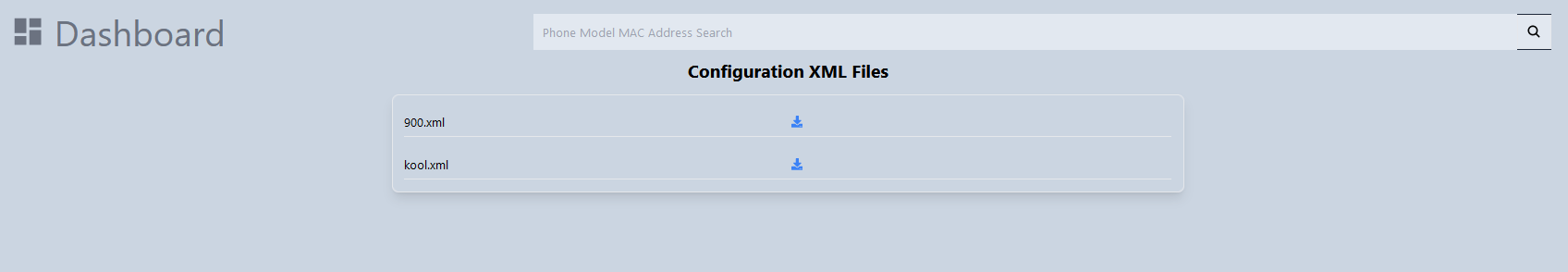
|  |  |
| --- | --- |
|  |  |

* Configuration operations: The system allows the user to create and download XML files for configuring the devices. The system provides a link to the configuration interface, where the user can enter the configuration data and press the “**Create configuration file”** button. The system then generates an XML file and stores it in a location on the server (‘./xmlfiles’). The system also provides a list of all the XML files stored on the server, and allows the user to retrieve, delete, and download them. The system uses node js file system to create and parse the XML files, and fs and path to read and write the files on the server.

  
*Audio settings configuration*

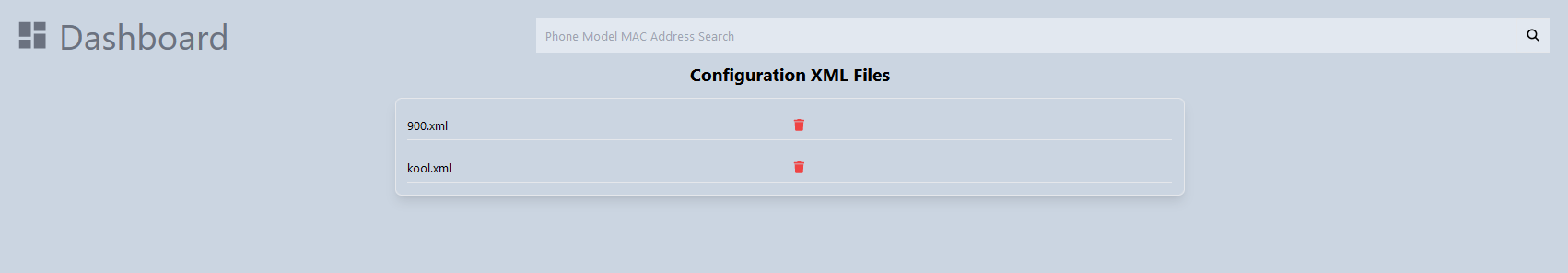
The user completes the form fields in all the tabs and click the “**Create configuration file”** button to create a configuration XML file for the phone/device. The system uses the mac address as the configuration file name so mac address field is required to create an xml file.  
  
The system currently does not provide data validation so the user must make sure to enter valid data.

**Download Xml File**



In the Download configuration page, all created xml files are listed and the user can click the download icon to download the file to their computer/phone.

**Delete Xml File**



From the delete configuration page, the user can click the delete icon to delete a xml configuration file.

**System Limitations and Challenges**

The system has some limitations and challenges that could not be addressed and resolved due to time constraints. The following features have either been partially implemented or not implemented at all:

* Search facility: The system does not provide a search facility that allows the user to search for phone and configuration data using keywords or filters. This feature would improve the usability and efficiency of the system, as the user would be able to find the relevant data more quickly and easily.
* User role management: The system does not provide a feature that allows the admin user to manage the roles and permissions of existing users. This feature would improve the security and functionality of the system, as the admin user would be able to modify the access and actions of different users.
* Display all flag marked configuration in separate section: The system does not provide a feature that allows the user to see all the flag marked configuration in a separate section. This feature would improve the visibility and awareness of the system, as the user would be able to see which configuration files need attention or action.
* Provide an action button to stop download for configuration file: The system does not provide a feature that allows the user to stop the download of a configuration file. This feature would improve the flexibility and control of the system, as the user would be able to cancel the download if they change their mind or encounter an error. However, all modern browsers do offer option to cancel file download.

**Conclusion**

This report has presented the development and implementation of a web-based system for managing the configuration of SIP phones and other devices. The system uses Node.js, TypeScript, MySQL, React, and Tailwind CSS as the main technologies, and provides features such as user authentication, CRUD operations, and configuration operations. The system also has some limitations and challenges that need to be addressed and resolved, such as the lack of search facility, user role management, flag marked configuration display, and download cancellation.

**Appendix**

**Tech Stack**

**Frontend**

## React js

We use React js to create the components for the CRUD and configuration interfaces, where the user can see and manipulate the phone and configuration data, as well as generate and download XML files for configuring the devices. We also use various libraries and tools to enhance our React development, such as React Router, Axios, and more.

We use react hooks for state management.

Sample hook to manage xml file list  
*// Define the state for the file list*const [fileList, setFileList] = *useState*<string[]>([]);

*// Fetch the file list when the component loads  
useEffect*(() => {  
 fetchFileList();  
}, []);

## Vite

Vite is a next-generation frontend tooling for React and other frameworks. Vite provides features such as fast development server, hot module replacement, code splitting, and tree shaking. Vite also supports TypeScript, JSX, CSS modules, and other web standards. Vite helps us to speed up our development process and optimize our production build.

We use Vite to create our development environment and our production build. We also use Vite to provide features such as server-side rendering, static site generation, routing, and API routes. We also use Vite to transpile our TypeScript files to JavaScript and bundle our code and assets.

## Tailwind CSS

Tailwind CSS provides us with a set of low-level utility classes that we can use to style our elements. Tailwind CSS also allows us to customize and extend the default theme, colors, fonts, breakpoints, and more. Tailwind CSS helps us to create responsive, consistent, and elegant designs.

We use Tailwind CSS to style and theme our website.

**Axios**

Another frontend technology that is used in the project is Axios, a library that allows us to make HTTP requests from the frontend to the backend

We use Axios to communicate with the server-side API that is built with Node.js and Express.js. We use Axios to perform CRUD and configuration operations, as well as generate and download XML files for configuring the devices. We use Axios methods such as axios.get(), axios.post(), axios.put(), and axios.delete() to send GET, POST, PUT, and DELETE requests to the API endpoints. We also use Axios to handle the responses and errors from the API, and display them on the website.

**Sample Axios Code to download XML file**

const handleDownload = async (file: string) => {  
  
 const downloadUrl = "http://localhost:3000/api/config/download/"  
 try {  
 *// Send a GET request to the API URL with the file name as a query parameter* const response = await axios.get(`${downloadUrl}?file=${file}`, {  
 responseType: 'blob',  
 });  
  
 *// Check if the response is OK* if (response.status === 200) {  
 *// Get the blob from the response* const blob = response.data;  
  
 *// Create a URL for the blob* const url = URL.createObjectURL(blob);  
  
 *// Create a link element with the URL and the download attribute* const link = document.createElement('a');  
 link.href = url;  
 link.download = file;  
  
 *// Append the link to the document body and click it* document.body.appendChild(link);  
 link.click();  
  
 *// Remove the link from the document body and revoke the URL* document.body.removeChild(link);  
 URL.revokeObjectURL(url);  
 } else {  
 *// Throw an error with the response status* throw new *Error*(response.status.toString());  
 }  
 } catch (error) {  
 *// Handle the error* console.error(error);  
 *alert*('Something went wrong. Please try again.');  
 }  
};

### **Server-Side API**

The server-side API was developed using Node.js, TypeScript, and Express.js. Node.js is a runtime environment that allows running JavaScript code outside the browser. TypeScript is a superset of JavaScript that adds static typing and other features. Express.js is a web framework for Node.js that provides features such as routing, middleware, and error handling.

The server-side API provided various endpoints for performing CRUD operations on phone and configuration data, as well as generating, deleting, fetching and downloading XML configuration files. The API used the REST (Representational State Transfer) architectural style, which defines a set of constraints and principles for designing web services. The API followed the RESTful conventions, such as using HTTP methods (POST, GET, PUT, and DELETE) to indicate the actions, using JSON (JavaScript Object Notation) to format the requests and responses, and using status codes to indicate the results.

The following table summarizes the user routes, methods, and responses of the server-side API:

**Table**

| **Route** | **Method** | **Description** | **Response** |
| --- | --- | --- | --- |
| /user/read | GET | Get all user | An array of user objects |
| /user/create | POST | Add a new user | The added user object |
| /user/update/:id | PUT | Update a user by id | The updated user object |
| /user/delete/:id | DELETE | Delete a user by id | The deleted user object |

Example API endpoint for user in node js app.ts file on the server:  
  
app.use(`/user/read`, user.UserRoute);  
app.use(`/user/create`, user.UserRoute);  
app.use(`/user/update/:id`, user.UserRoute);  
app.use(`/user/delete/:id`, user.UserRoute);  
app.use(`/user/delete/:id`, user.UserRoute);  
app.use(`/user/signin`, user.UserRoute);  
app.use(`/user/signup`, user.UserRoute);

MVC  
  
Example code demonstrating UserRouter functions

UserRouter.post(  
 "/signin",  
 UserValidator.ValidateUserData(),  
 validationMiddleware.handleValidationError,  
 AuthController.signin  
);  
  
UserRouter.post(  
 "/signup",  
 UserValidator.ValidateUserData(),  
 validationMiddleware.handleValidationError,  
 UserController.create  
);  
  
  
UserRouter.get(  
 "/read",  
 AuthMiddleware.authenticate,  
 UserController.read  
);

Example code demonstrating use of controllers in this project

class UserController {  
  
  
 async read(req: Request, res: Response) {  
 try{  
 *//const record = await User.findByPk(req.params.id);  
 //return res.json(record);* const records = await User.*findAll*({ where: {} });  
 return res.json(records);  
  
 }catch (e) {  
 return res.json({  
 msg: "fail to read user",  
 status: 500,  
 route: "/read",  
 });  
 }  
  
 }

async create(req: Request, res: Response) {  
 try {  
  
 const user = await User.*findOne*({ where: { email: req.body.email } });  
 if (user) {  
 return res.json({ message: "Failed! Email already in use" });  
 }  
  
 const record = await User.*create*({ ...req.body });  
 return res.json({  
 record: record,  
 msg: "User created.",  
 });  
 } catch (e: any) {  
 console.log(e.message)  
  
 return res.json({  
 msg: "Fail to create user",  
 status: 500,  
 route: "/create",  
 });  
 }  
}

Routes for XML Config Files  
  
  
ConfigFileRouter.post("/create", ConfigFileController.create);  
ConfigFileRouter.get("/download", ConfigFileController.download);  
ConfigFileRouter.delete("/delete-file", ConfigFileController.delete);  
ConfigFileRouter.get("/fileList", ConfigFileController.fileList);

Contrallers for XML Config Files

class ConfigFileController {  
  
 constructor() {  
 *// Bind the this context to the class instance* this.create = this.create.bind(this);  
 }  
  
 async create(req: Request, res: Response) {  
  
  
 *// Get the form data from the request body* const data = req.body as FormData;  
  
  
 *// Validate the input* if (  
 !data.macAddress  
 *// ... other fields* ) {  
 return res.status(400).send('Invalid input');  
 }  
  
  
 try {  
 this.createXML(data)  
 return res.json('Api Working');  
  
 }catch (e: any) {  
 return res.json({  
 msg: "fail to create ConfigFile",  
 status: 500,  
 route: "/read",  
 error: e.message  
 });  
 }  
  
  
 }  
  
 async delete(req: Request, res: Response) {  
  
 *// Get the file name from the request query* const file = req.query.file;  
  
 *// Check if the file name is a string* if (typeof file === 'string') {  
 *// Define the path to the file* const filePath = `./xmlfiles/${file}`;  
  
 *// Delete the file using fs.unlink* fs.*unlink*(filePath, (err) => {  
 *// Handle any errors* if (err) {  
 console.error(err);  
 res.status(500).send('Something went wrong. Please try again.');  
 } else {  
 *// Send a success response* res.status(200).send('File deleted successfully.');  
 }  
 });  
 } else {  
 *// Send a bad request response* res.status(400).send('Please provide a valid file name.');  
 }  
 }

## Installation

To install and run the system, you need to have Node.js, npm, and MySQL installed on your machine. You also need to clone the repository from GitHub and install the dependencies. Follow these steps:

**Frontend React App**

* Clone the repository from GitHub using the command or extract the zip file:
* Navigate to **sip\_frontend** directory of the project and install the dependencies using the command:

npm install

* Run following command to run app in dev mode

npm run dev

Open your browser and go to <http://localhost:5173/> to see the website.

**Backend Server**

* Next start the node js server by opening a different terminal window
* Navigate to **sip\_backend** directory and install dependencies using the command:

npm install

* Create a .env file in the root directory and add the following variables:

DB\_DIALECT=mysql  
DB\_HOST=localhost  
DB\_NAME=g3t\_voip\_gsphones  
DB\_USER=root  
DB\_PASS=  
DB\_PORT=3306  
  
JWT\_SECRET=mySecretKey1234!@*#$%^*NODE\_ENV=development

* Create a database in MySQL using the above information:
* Import the database schema from the g3t\_voip\_gsphones.sql file in the root directory. Start the server using the command:

npm run dev

* Open your browser and go to http://localhost:3000 to see the website.