National College of Ireland

Configuration Manual

MSc Research Project
M.Sc. Cybersecurity

Chinelo Lauren Nwobbi
Student ID: x23333057

School of Computing
National College of Ireland

Supervisor: Khadija Hafeez

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Chinelo Lauren Nwobbi		
Student ID:	x23333057		
Programme:	M.Sc. Cybersecurity	**Year:**	2025
Module:	M.Sc. Research Practicum		
Lecturer:	Khadija Hafeez		
Submission Due Date:	15th September 2025		
Project Title:	Configuration Manual for Practicum: Implementing Adaptive Rate Limiting and Honeytokens for Enhanced REST API Security in JavaScript Web Applications.		
Word Count:	1422	**Page Count:**	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: 

Date: 15th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Chinelo Lauren Nwobbi
Student ID: x23333057

1 Introduction

This study is designed to enhance the security posture of web applications by integrating secure design principles like defense in depth and secure design by default, the proposed model incorporates security mechanisms like adaptive rate limiting, honeytokens and bot detection. This configuration manual defines technical steps that outlines system requirements, software's, frameworks and tools required to replicate the experimental set up of the model.

2 System Configuration

This section outlines the hardware specifications and development environment used to build the Adaptive Rate Limiting (ARL) model, along with tools and libraries used to build API

2.1 Hardware and Operating System

Component	Details
Operating System	macOS Sonoma 14.6.1 (23G93)
Processor	1.4 GHz Quad-Core Intel Core i5
Memory	8 GB 2133 MHz LPDDR3
Development tools	Visual Studio Code, Terminal, Postman, pgAdmin, RedisInsight

Table 1: Details on components used.

2.2 Prerequisites

To adequately replicate this project the following is required:

1. Node.js
2. npm
3. PostgreSQL
4. Postman
5. OWASP ZAP
6. Visual Studio Code

2.3 Software Used

This section gives an overview on software used in this project along with their versions and brief descriptions.

Software	Description
Node.js (22.9.0)	This is a runtime environment used for server-side scripting; it allows developers create servers that can make calls to a database.

Npm (11.4.1)	Package manager for installing and managing packages
PostgreSQL (15.2)	Database for storing user data, request logs, and denylist entries
RedisInsight	GUI for monitoring and managing Redis Database.
Postman (11.56.4)	Used for testing API endpoints with performance and functional tests
Visual Studio Code (1.102.3)	Integrated development environment for development of the project
React (^19.1.0)/Vite (^6.3.5)	This is an opensource library for user interface (frontend) development that allows the development of the user interface using components, the frontend application was broken down into multiple components including Login.jsx, CreateUser.jsx, Home.jsx and App.jsx

Table 2: Details on Software used

2.4 Libraries and Tools

Library/Tool	Description
axios: "^1.11.0",	Axios is a tool used to create http request to external resources, it automatically transforms data to JSON for easier processing. Axios was used to make requests directly to the backend using the backends https URL.
cors: "^2.8.5",	For authorized resource sharing with external third parties. Used for allowing requests between frontend and backend.
dotenv: "^16.5.0",	Loads environment variables from an .env file
express: "^5.1.0",	This is a Node.js framework designed for building web applications and APIs
express-rate-limit: "^7.5.1",	This is a rate limiting tool offered by Express.js to limit requests to a server per hour, this tool was used in Baseline model 2 and in beginning implementation stages of the ARL model.
express-validator: "^7.2.1"	Middleware for validating and sanitizing incoming request data.
helmet: "^8.1.0",	Adds security-related HTTP headers to protect the app from well-known vulnerabilities.
jsonwebtoken: "^9.0.2",	This is a standard for transmitting data securely between parties using JSON ref here, in this application it was used primarily on the login API endpoints for authentication and authorisation.
react: "^19.1.0",	Main library for building user interfaces in the frontend.
react-dom: "^19.1.0",	used to render components in the browser
react-router-dom: "^7.6.2",	This tool was used in react to enable navigation across different components. Its Route function was used to define routes to each component in App.jsx and it Link function was used to call those routes in

	other components. That is the link function is used in Home.jsx to link to the Login.jsx by calling on its predefined route.
bcrypt	This tool was used to hash passwords into the database and compare hashed passwords on creation of users and logging in respectively.
ioredis	This was used to connect express application to a Redis instance to enhance adaptive rate limiter.

Table 3: Details on Libraries and tools used.

3 Steps required to configure and run project

This section establishes steps required to successfully set up and run the project developed for this study. Users can either clone codebase from GitHub or download directly from Moodle with both methods leading to the same results.

3.1 Environment set up

1. Download Postgres via: https://www.postgresql.org/download/
2. Install node:

```
Mac:
Download and install Homebrew
curl -o- https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh
 | bash
Download and install Node.js:
brew install node@22
 Verify the Node.js version:
node -v # Should print "v22.18.0".
 Verify npm version:
npm -v # Should print "10.9.3".

Windows:
# Download and install Chocolatey:
powershell -c "irm https://community.chocolatey.org/install.ps1|iex"
# Download and install Node.js:
choco install nodejs --version="22.18.0"
# Verify the Node.js version:
node -v # Should print "v22.18.0".
# Verify npm version:
npm -v # Should print "10.9.3".
```

3.2 Code set up

1. Open Visual Studio Code
2. Navigate to Integrated Terminal
3. Clone and navigate into project (If cloning from GitHub)

```
git clone https://github.com/chinelon/Practicum.git
cd Practicum
```

4. Download and extract zip files (If downloaded from Moodle)
5. Navigate to backend and install dependencies

```
cd backend
npm install
```

6. Create and configure .env file n backend with the following:

```
DB_USER=postgres
```

```
DB_HOST=localhost
DB_NAME=practicum
DB_PASSWORD=chinelo
DB_PORT=5432
PORT=10000
JWT_SECRET=b292398ded794e65efbb474e7f6b6be7449a516069b9f7098661f932121f8d796aa8e4373d218498d523eaea7d2727effc3fdbcc2e35cf5ec348f37624669663

DB_URL=postgresql://neondb_owner:npg_P3lomrjShCs1@ep-little-forest-a273lczg-pooler.eu-central-1.aws.neon.tech/practicum?sslmode=require&channel_binding=require

REDIS_USERNAME=default
REDIS_PASSWORD=8Luau8iJycqpnw3thBZeAxP1mAMZfQrT
REDIS_HOST=redis-17508.c11.us-east-1-3.ec2.redns.redis-cloud.com
REDIS_PORT=17508
```

7. Start backend with:

```
node index3.js
```

8. Open a new terminal and navigate into the frontend folder, install dependencies and run.

```
cd Practicum
npm install
npm run dev
```

9. If dependencies are missing or installation fails, manually run:

```
Backend:
npm install  cors dotenv express express-rate-limit express-validator helmet jsonwebtoken bcrypt ioredis winston

frontend:
npm install axios react react-dom react-router-dom
```

4 Source code walkthrough

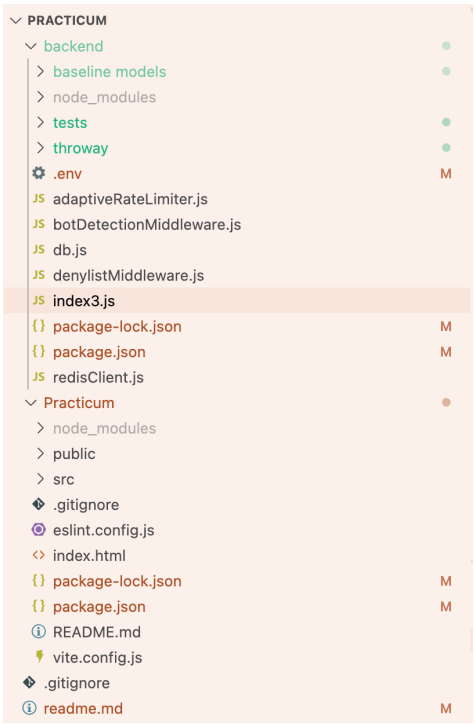

Figure 1. Screenshot of project structure

4.1 Frontend

The frontend section of this code is comprised of what the user can see i.e. home, login, create users and admin pages.

Home.jsx renders the home page/landing page that links to login page. This page also has the bot honeytoken.

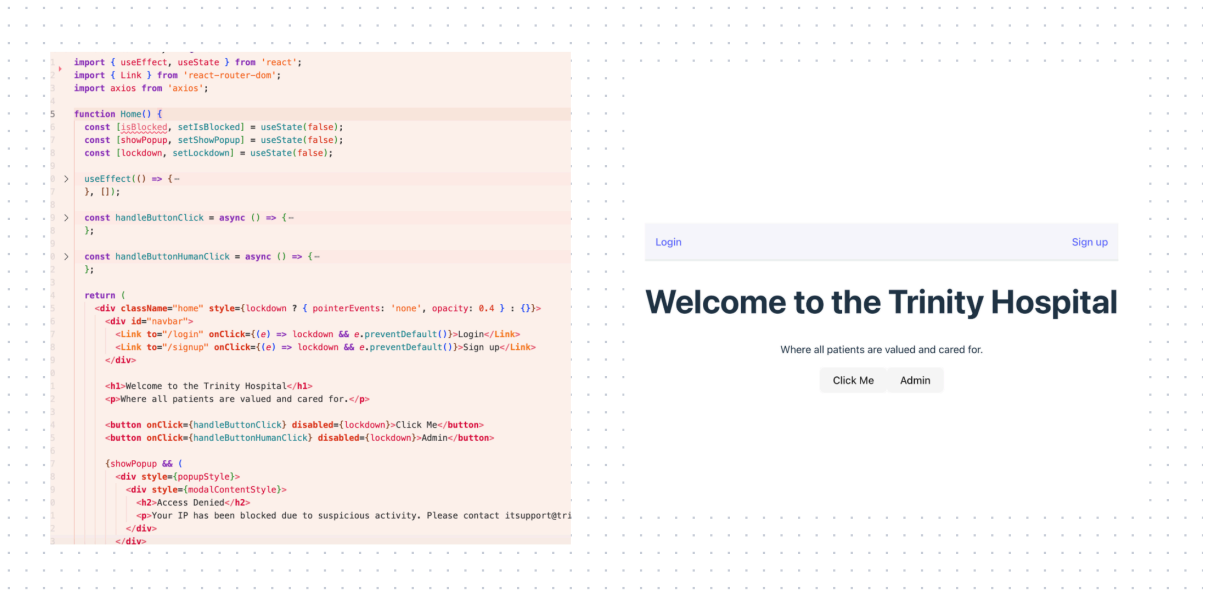

Figure 2: Home.jsx

App.jsx contains links to all components in the project to allow for navigation using react-router-dom

```jsx
import './App.css'
import Home from './Home'
import { BrowserRouter as Router, Route, Routes, Navigate } from 'react-router-dom';
import Login from './Login';
import Signup from './Signup';
import AllUsers from './AllUsers';
import PageNotFound from './PageNotFound';

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/signup" element={<Signup />} />
        <Route path="/allusers" element={<AllUsers />} />
        <Route path="/404" element={<PageNotFound />} />

      </Routes>
    </Router>
  );
}

export default App;
```

Figure 3: App.jsx

Login.jsx, this component is the main component for admin login.

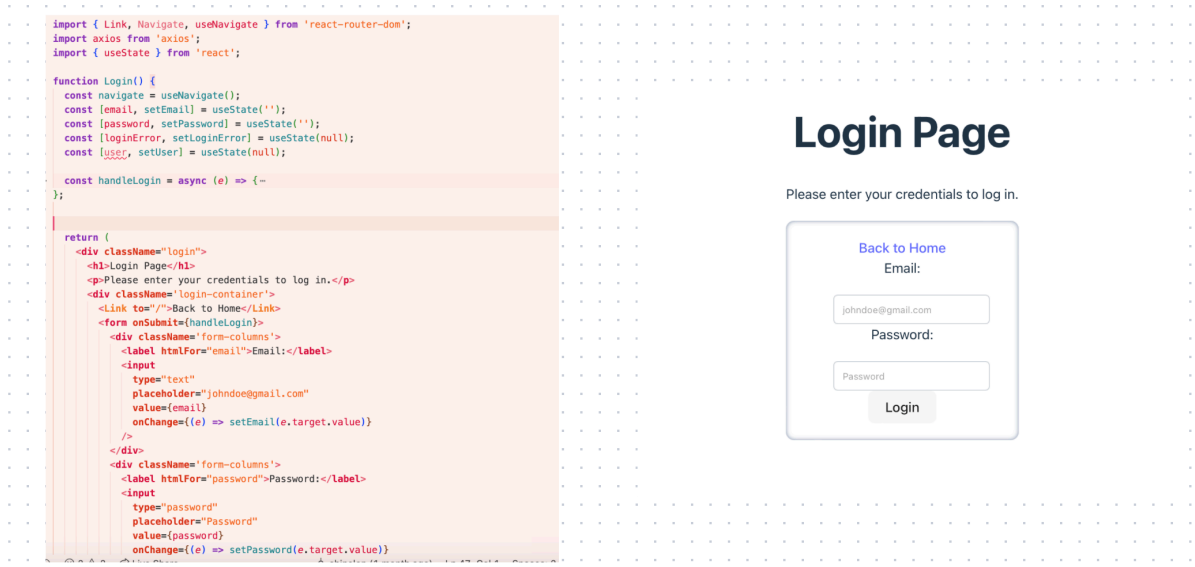

Figure 4: Login.jsx

Alluser.jsx is the component for viewing all users in the database which should only be viewable to admins.

Signup.jsx is used to create new users, only available to admins

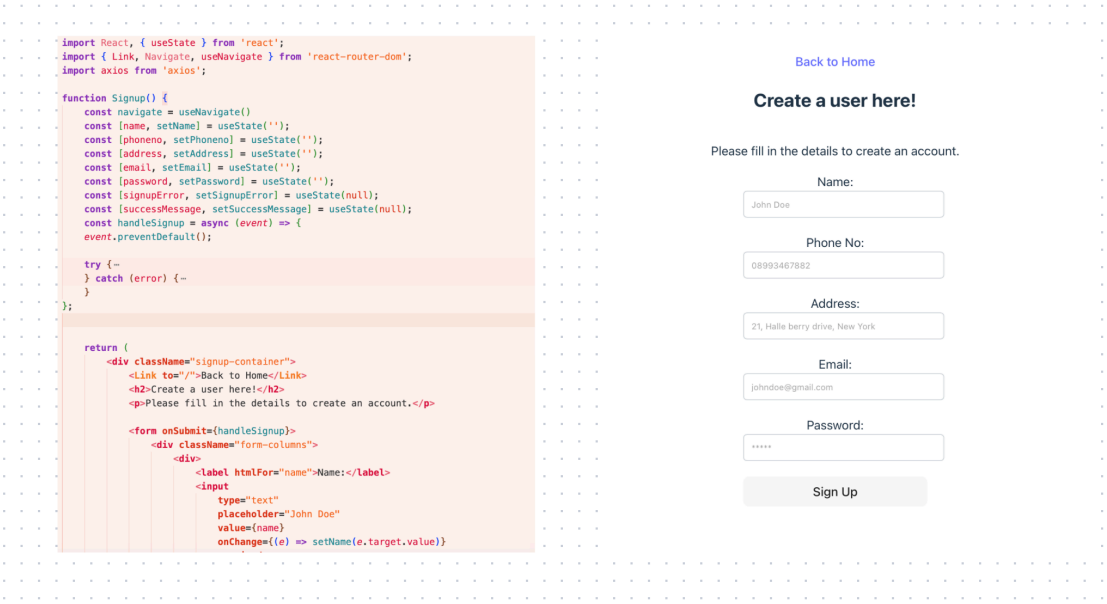

Figure 5: Signup.jsx (To create users)

Admin.jsx is a fake form used to lure threat actors, when submit button is clicked the human honeytoken API endpoint is triggered

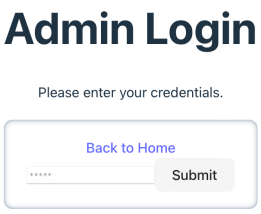

Figure 6: Admin.jsx

4.2 Backend

Adaptiveratelimiter.js applies rate limits based on denylist and bot detection middleware process with the use of Redis with the use of (Adel, 2025)

Figure 7: adaptiveRateLimiter.js

Botdetectionmiddleware.js detects and logs potential bots based on user agent analysis and request patterns

Figure 8: botDetectionMiddleware.js

Index3.js is the backend entry point, it initialises express server and loads libraries and middleware for CORS, helmet, adaptiveratelimting.js, botDetectionMiddleware.js and denylistmiddleware.js. this is where all API endpoints are defined.

```
require('dotenv').config();
const pool = require('./db');
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const { body, validationResult, param } = require('express-validator');
const botDetectionMiddleware = require('./botDetectionMiddleware');
//const  = require('./adaptiveRateLimiter').fetchRateLimitMax;
//const {adaptiveRateLimiter, fetchRateLimitMax } = require('./adaptiveRateLimiter');
const adaptiveRateLimiter = require('./adaptiveRateLimiter');
const denylistMiddleware = require('./denylistMiddleware');

const app = express();
app.use(denylistMiddleware);
app.set('trust proxy', true);

app.use(helmet());
//app.use(fetchRateLimitMax);
app.use(adaptiveRateLimiter);
app.use(botDetectionMiddleware);

app.use(cors({
    origin: ['http://localhost:5173', 'https://practicum-eta.vercel.app'],
    methods: ['GET', 'POST', 'PUT', 'DELETE'],
    credentials: true
}));

app.use(express.json());

function generateToken(user) {…
}
```

```
app.get('/', denylistMiddleware, (req, res) => {
    res.status(200).json({ message: 'Welcome to the API' });
    res.set({
        'X-RateLimit-Limit': maxRequests,
        'X-RateLimit-Remaining': Math.max(maxRequests - current, 0),
    });

});

app.post('/signup',…
    });

// Login
app.post('/login',…
    });

// Authentication middleware
function authenticateToken(req, res, next) {…
}

app.get('/allusers', authenticateToken, async (req, res) => {…
});

app.get('/users/:id',…
    });

app.put('/users/:id',…
    });

app.delete('/users/:id',…
    });


//honeytoken endpoint human detection
```

Figure 9: Index3.js

References

Adel, A. (2025) *Rate Limiting with Redis and Node.js: Under the Hood*, *Webdock*. Available at: https://webdock.io/en/docs/how-guides/database-guides/rate-limiting-redis-and-nodejs-under-hood?srsltid=AfmBOooKnR2_SeKi9di123WAZvxjCtYyeD9r69oYqfm_wU4bQJ2aZeoL (Accessed: 10 August 2025).