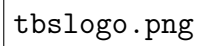


UNIVERSITY OF TUNIS
TUNIS BUSINESS SCHOOL

Technical Report

GreenerTn API

Nawres Ayadi
ID:12363237

A rectangular box containing the text 'tbslogo.png' in a monospaced font, indicating a missing image file.

tbslogo.png

January 21, 2024

Contents

1	Introduction	2
1.1	Motivation	2
1.2	Aims and Objectives	2
2	Information System Building	3
2.1	System Requirements Specification	3
2.2	Class Diagram	4
3	Project Structure	5
3.1	Security	5
3.2	Endpoints	6
3.2.1	POST Endpoints	6
3.2.2	Get Request Endpoints	7
3.2.3	PUT Endpoints	7
3.2.4	DELETE Endpoints	8
4	Technologies and Libraries	9
4.1	Technologies	9
4.1.1	Node.JS Express Framework Contribution:	9
4.1.2	MongoDB, Mongoose Contribution:	9
4.1.3	Insomnia Contribution:	9
4.1.4	Git Contribution:	9
4.1.5	Swagger Contribution:	9
4.2	Libraries:	10
4.3	Github and Swagger:	10

1 Introduction

GreenerTn is a RESTful API based on a non-relational database, main functionalities are: carbon tracking and notifications system. The main idea of this API is to track carbon emissions from cars using user's input about daily miles and information about the car, this API assesses the cars conditions and use the miles input to calculate average carbon consumption using a specific formula and returns notifications about average consumption and alternative transportation options. (version 1 depends solely on user's willingness to enter miles ran in a day, next version will read directly from odometer)

1.1 Motivation

The motivation behind this API is the climate change issue. People don't have an incentive to do anything about this issue, so cooperation from governments are required. I believe that raising awareness by tracking carbon emission and using this technique by governments to enforce a ***carbon tax*** as suggested by Sir.William Nordhaus, a Nobel prize winner for his work on the economics of climate change and carbon tax advocacy, is one of the few techniques left to save the world we live in. In his Nobel panel discussions he underlined the fact that we should work on tax on the CO2 emissions that by consequence will lead to higher dirty fuel prices and will turn people into consumers of renewable energy.

1.2 Aims and Objectives

- track user's average carbon emissions to raise awareness about global warming and the environment.
- Allow users to receive notifications about daily consumption and alternative transportation choices.

2 Information System Building

2.1 System Requirements Specification

- Purpose:
GreenerTn API helps raise environmental awareness around carbon emissions for people who commute daily by calculating carbon emission and providing alternatives.
- Scope:
The purpose of this system is to create an easy-to-use API for users. The system is based on a relational database with vehicles management and user operational process. Therefore, the API will allow the following functionalities:
Users' functionalities:
Securely create/log in into a new account, browse your vehicles and their average carbon emission, receive notifications.
car owners functionalities:
Securely account registration and login to the platform, manage and publish their cars data by:
 - Inserting relevant information.
 - Updating specific data such as model, unique ID and years in usage, etc.
 - Deleting their car's data from GreenerTn's platform's database.Manage and provide real-time information about daily commutes in miles:
 - Inserting miles ran today information.
 - get your suggestion of the day along with the average emissions of CO₂ for that specific car across all days.

General Description:

- Product Perspective:
"GreenerTn API" is an online system. The system interfaces with an automated confirmation system, and the browsers used by clients to access it. The system provides a secure environment for all transactions and for the storing and managing all users information. As well as securing long-term customer loyalty.
- Product Function:
This API allows users to enter their cars information and journey, and receive advice in regard to average carbon emissions and alternative transportation options.

- User Characteristics:

The main users of this system are car owners and admin of system. The User can register online by recording their relevant attributes on this system. This does not require training, only needs the basic skill to use a browser and website.

The skills demanded from the admin is being able to use the website, so with no knowledge of his/her educational level, he/she needs little to no training.

2.2 Class Diagram

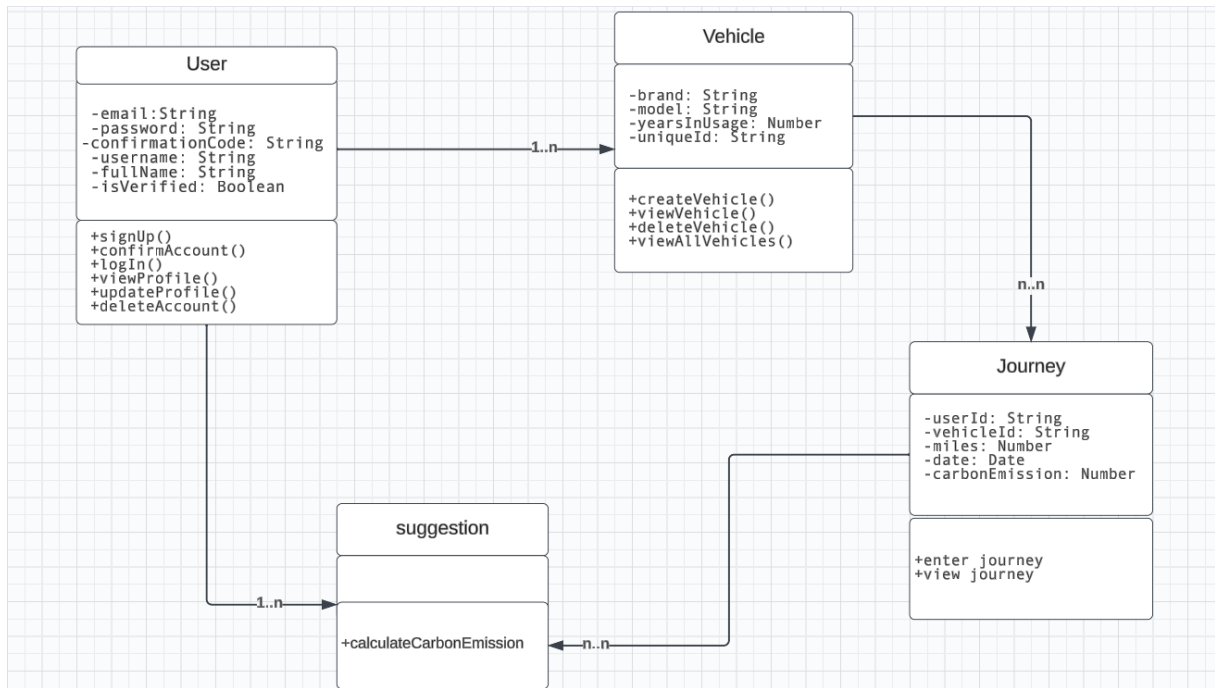


Figure 1: Architecture of GreenerTn API

3 Project Structure

3.1 Security

- **Identification:**

First of all, and before accessing GreenerTn API, the user needs to create an account. Each account has a full name, a unique username, a password, a unique email address and is of a verified email address structure, This process is done through the `"/Signup"` endpoint. the password entered by our user is saved into MongoDB database, along with other provided information, hashed using Bcrypt package in Node.js.

- **Authentication:**

After Identification, the user is authenticated. Right after the creation of the account with valid credentials and a specific token, the API informs the user that an email has been sent to the address provided with a verification code for login, this is done using Google API and OAUTH2, the user then proceeds to copy confirmation code into `"/confirm"` endpoint once the confirmation code is checked to be true and account validated, now The login is possible through the `"/login"` endpoint where the user has to type his username and password. When all these attributes are valid, the user is authenticated.

- **Authorization** right after signup an access a token is generated and granted to the user but can only be used to access endpoints after confirmed login. Example of an access token:

```
"token":  
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFnZW50IjoiIiwiaWF0IjoxNzA1NDMzMzI4LCJleHAiOiE3MDU1MTk3Mjh9.1P7-
```

JWT is imported from JsonWebToken package in order to implement the generation of tokens using a secret key to check if a specific token is really true. Each user has a unique access token. This access token is a JWT token, signed by our API app, where stored the username and has a lifespan of 3600 seconds. This token permits to the user access to specific protected endpoints of that specific user.

3.2 Endpoints

In this API used 12 methods to control users requests and tested them using Insomnia.

3.2.1 POST Endpoints

Send users information to the server to create users accounts and provide information about his car/s .

Create journey where daily miles are provided.

```
// Signup and issue JWT token
app.post('/signup', async (req, res) => {
  try {
    const { email, password, username, fullName } = req.body;

    // Check if the email or username is already registered
    const existingUser = await User.findOne({ $or: [{ email }, { username }] });

    if (existingUser) {
      return res.status(400).json({ error: 'Email or username already in use.' });
    }

    // Hash the password
    const hashedPassword = await bcrypt.hash(password, 10);

    // Generate confirmation code
    const confirmationCode = jwt.sign({ email }, SECRET_KEY, { expiresIn: '1d' });

    // Save user with hashed password and confirmation code
    const newUser = new User({ email, password: hashedPassword, confirmationCode, username, fullName });

    try {
      await newUser.save();
    } catch (saveError) {
      console.error('Error saving user:', saveError);
      return res.status(500).json({ error: 'Error creating user' });
    }

    // Send confirmation email
    await sendConfirmationEmail(newUser.email, confirmationCode);

    // Issue JWT token
    const token = jwt.sign({ email: newUser.email }, SECRET_KEY, { expiresIn: '1d' });

    // Customize the response with additional information
    res.status(201).json({
      message: 'User created successfully. Check your email for confirmation.',
      userId: newUser._id,
      email: newUser.email,
      username: newUser.username,
      fullName: newUser.fullName,
      token,
    });
  } catch (error) {
    console.error('User creation error:', error);
    res.status(400).json({ error: error.message });
  }
});
```

Figure 2: POST ”/signup” endpoint

3.2.2 Get Request Endpoints

Return the users list stored in the database.

Return the list of vehicles by user.

Return the journeys of each user for a specific car

```
app.get('/vehicle/:vehicleId', async (req, res) => {  
  try {  
    const vehicle = await Vehicle.findById(req.params.vehicleId);  
    res.status(200).json(vehicle);  
  } catch (error) {  
    res.status(404).json({ error: 'Vehicle not found' });  
  }  
});
```

Figure 3: GET `"/vehicle/vehicle:id"` endpoint

3.2.3 PUT Endpoints

Update user's information.

Update a user's vehicle information.

```
// Update user by ID  
app.put('/user/:userId', async (req, res) => {  
  try {  
    const updatedUser = await User.findByIdAndUpdate(req.params.userId, req.body, { new: true });  
    res.status(200).json(updatedUser);  
  } catch (error) {  
    res.status(404).json({ error: 'User not found' });  
  }  
});
```

Figure 4: PUT `"/user/user:id"` endpoint

3.2.4 DELETE Endpoints

Delete a specific car based on it's ID.

Delete a user completely from database.

```
// Delete vehicle by ID
app.delete('/vehicle/:vehicleId', async (req, res) => {
  try {
    await Vehicle.findByIdAndDelete(req.params.vehicleId);
    res.status(204).send();
  } catch (error) {
    res.status(404).json({ error: 'Vehicle not found' });
  }
});

// Get all vehicles
```

Figure 5: DELETE `"/vehicle/vehicle:id`

4 Technologies and Libraries

4.1 Technologies

4.1.1 Node.JS Express Framework Contribution:

Node.js is a JavaScript runtime environment that allows to execute JavaScript code server-side. It provides an event-driven, non-blocking I/O model that makes it efficient and lightweight, making it well-suited for building scalable and high-performance network applications.

Express, on the other hand, is a web application framework for Node.js. It is designed to simplify the process of building robust and scalable web applications and APIs. Express provides a set of features for web and mobile application development, including routing, middleware support, and templating. It is often referred to as a "micro" framework because it provides a minimal set of tools and features, allowing developers to choose and integrate additional libraries as needed.

Together, Node.js and Express form a powerful combination for building server-side applications and APIs using JavaScript.

4.1.2 MongoDB, Mongoose Contribution:

MongoDB is a document-oriented NoSQL database, while Mongoose is an Object Data Modeling (ODM) library for Node.js that provides a higher-level abstraction layer on top of MongoDB, allowing developers to define data models using a schema-based approach.

4.1.3 Insomnia Contribution:

Insomnia is actually an API client tool that helps developers design, test, and debug APIs

4.1.4 Git Contribution:

able to add my project to github and be able to add changes, commit and push them

4.1.5 Swagger Contribution:

full documentation of all endpoints with details.

4.2 Libraries:

- * Express.js: Web application framework for Node.js.
 - * Body-parser: Middleware to parse incoming request bodies.
 - * jsonwebtoken (jwt): Library for generating and verifying JSON Web Tokens.
 - * Nodemailer: Module for sending emails.
 - * Bcrypt: Library for hashing passwords.
 - * Googleapis (OAuth2): Library for interacting with Google services and OAuth2 authentication.
 - * Swagger-jsdoc and Swagger-ui-express: Libraries for generating and serving Swagger documentation.
- * dotenv: Library for loading environment variables from a .env file.

4.3 Github and Swagger:

Link Github: <https://github.com/NAwresAyadi25/GreenerTn-RESTfulAPI/tree/master>
Link Swagger: <http://localhost:3000/api-docs/>