← View All Projects

Description

Build Backend for Online-Store

GitHub: https://github.com/mattjinks/online-store-backend

This project builds a backend API that powers an online store application.

Key experiences gained from this project include:

• Database schema design/implementation

• API endpoint design/implementation

• Password hashing

• Token authentication

• Unit testing

Technologies Used:

• **PostgreSQL** for database

• **Express** (Node framework) for implementing RESTful APIs

• **dotenv** from npm (node package manager) for managing environment variables

• **db-migrate** from npm for database migrations

• **jsonwebtoken** from npm for working with JWTs (Authentication, Authorization, Verification)

• **jasmine** from npm testing

Demo

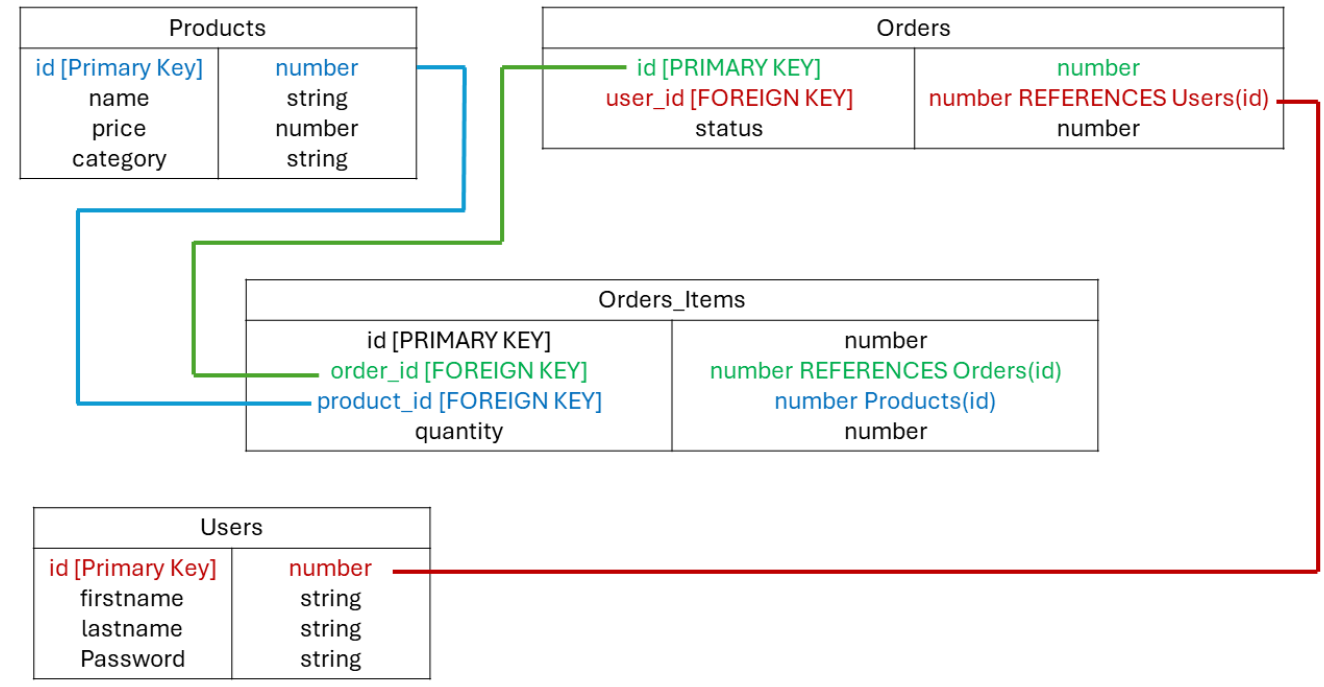
Overview

Online Store API Project O...

Demo

Online-store API Demo

Database Schema



API Endpoints

Products

• **INDEX: '/products' [GET]** Retrieves a list of all products available in the storefront

• **INDEX: '/products/?category=categoryName' [GET]** Retrieves a list of products filtered by a specific category

• **SHOW: '/products/:id' [GET]** Retrieves the details of a single product, indentified by its ID

• **CREATE [token required]: '/products' [POST]** Adds a new product to the storefront. Requires an authentication token to ensure only authorized users can create products

Users

• **Index [token required]: '/users' [GET]** Retrieves a list of all users. Access requires auth token to ensure only authorized users can retrieve users

• **Show [token required]: '/users/:id' [GET]** Retrieves the details of a single user, identified by its ID. Access requires auth token to ensure only authorized users can retrieve user details

• **Create [token required]: '/users' [POST]** Adds a new user to the storefront

Orders

• **SHOW [token required]: '/users/:userId/orders/current' [GET]** Current Order by user (args: user id)

• **INDEX [token required]: '/users/:userID/orders/completed' [GET]** Completed Orders by user (args: user id)

Password Security

To ensure the security of user passwords, this project utilizes the bcrypt library. Bcrypt is a widely-used password hashing algorithm that employs a technique called salting.

What is Hashing and Salting?

• **Hashing:** Hashing transforms a password (or any data) into a fixed-length string of characters that appears random and meaningless. It's designed to be a one-way function, meaning it's virtually impossible to reverse the hash back to the original password.

• **Salting:** Salting adds an extra layer of security by combining each password with a unique, random string (the salt) before hashing. This prevents attackers from using precomputed tables to crack common passwords.

How bcrypt Protects User Passwords

1. Salt Generation: A unique salt is generated for each user.

2. Combined Input: The salt is appended to the user's plain text password.

3. Hashing: The combined salt and password string are processed through the bcrypt algorithm, producing a secure hash.

4. Storage: The resulting hash (not the original password) is stored in the database along with the salt.

Web Tokens

Security with JSON Web Tokens (JWTs) Salting adds an extra layer of security by combining each password with a unique, random string (the salt) before hashing. This prevents attackers from using precomputed tables to crack common passwords.

JWTs are compact, URL-safe tokens that securely transmit information between parties as a JSON object. They are digitally signed using a secret key known only to the server, ensuring the authenticity and integrity of the transmitted data.

How JWTs Are Used in This Project

1. Authentication: After a user successfully creates an account or logs in, the server issues a JWT. This token is then sent to the client and stored locally.

2. Authorization: Subsequent requests to protected endpoints, such as those involving creating products, retrieving user details, or managing orders, require the client to include the JWT in the request headers.

3. Verification: Upon receiving a request, the server validates the JWT's signature using its secret key. If the token is valid and hasn't expired, the server extracts the user's information from the token's payload, allowing it to verify the user's identity and authorize their actions.

Unit Testing

This project includes unit testing for all CRUD (create, read, update, delete) operations implemented in the project for storage. Jasmine was used to write the unit test because it is a framework designed specifically for testing JavaScript code.

← View All Projects