Middlesex University London

Report for Nike Football Boots Price Comparison website- “SoleMate Finder”

By Maaz Chowdhry

M00910300

SoleMate Finder

"SoleMate Finder" is a web-based application designed for facilitating the search and comparison of different Nike Football Boots. This project utilizes technologies like Vue.js for front-end development, Express.js and Node.js for the backend, and MySQL for the database.

Web Scraping and Data Storage

The Java Maven project employs Selenium and Chrome Web Driver to scrape data from 5 different websites simultaneously in 5 threads, focusing primarily on details related to Nike Football Boots. The scraped data is then stored into the MySQL database using Hibernate, ensuring data integrity and ease of access.

The RESTful web service

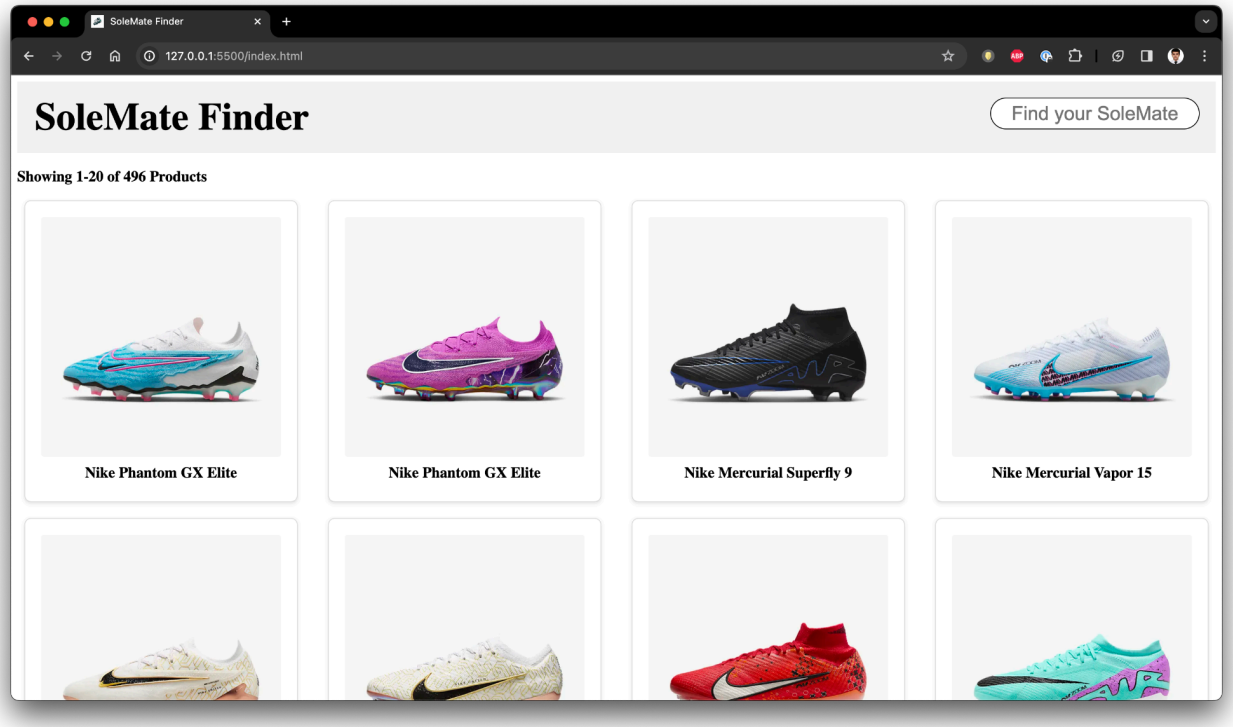
The RESTful web service in this project provides several endpoints:
GET /shoes/:id?: Returns a list of shoes. If an ID is provided, it returns specific shoe details.
Example: GET /shoes/dj5625 might return details of a specific shoe model with the SKU 'dj5625'.
GET /search/: Used for searching shoes with query parameters like query, offset, and limit.
Example: GET /search/?query=Nike&limit=10 will return the first 10 Nike shoe models.
GET /search/count/: Returns the total count of shoes that match a given search query.
Example: GET /search/count/?query=Adidas would return the count of Adidas shoes in the database.

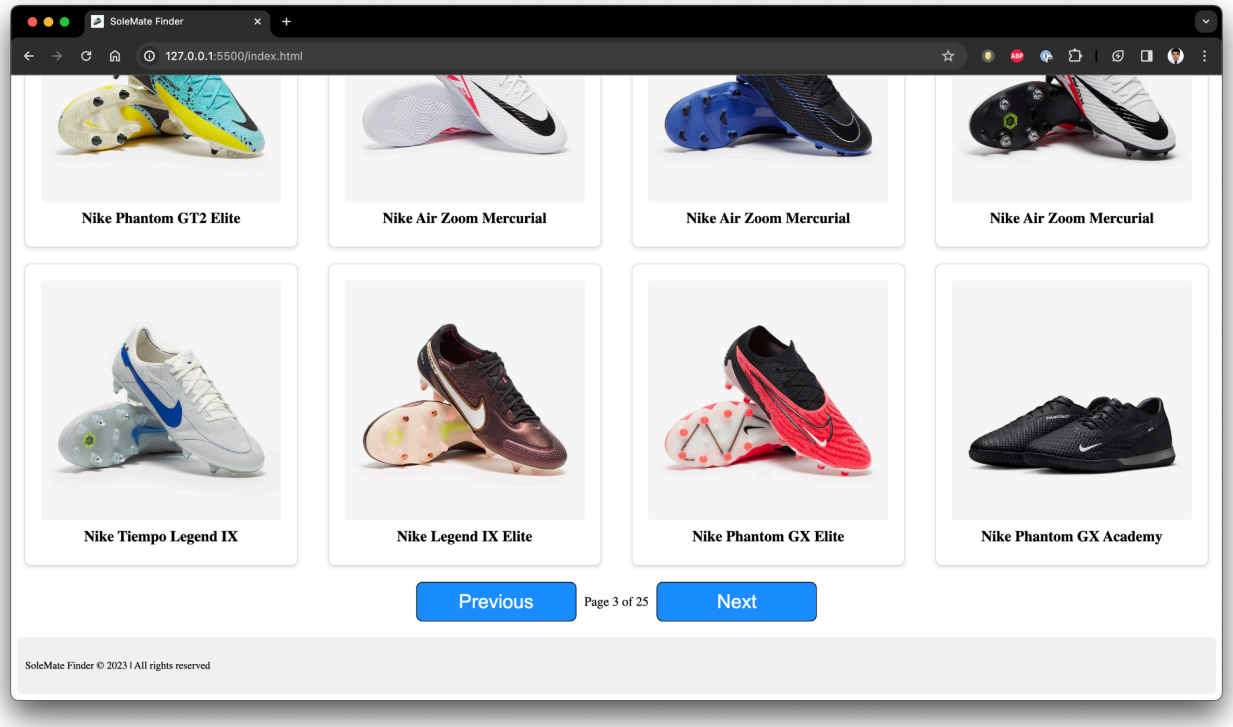
The front end

The front end of the project is developed using Vue.js and is structured with a user-friendly interface.
The main page displays a search bar and a list of shoe models. Users can search for specific shoe models using the search bar, and the results are dynamically updated. Each shoe model is displayed with an image and name, and users can click on a model to view more details. Pagination is implemented for navigating through the list of shoes. Additionally, the front end handles error states gracefully, displaying appropriate messages when no results are found or if there's a problem with the fetch operation.

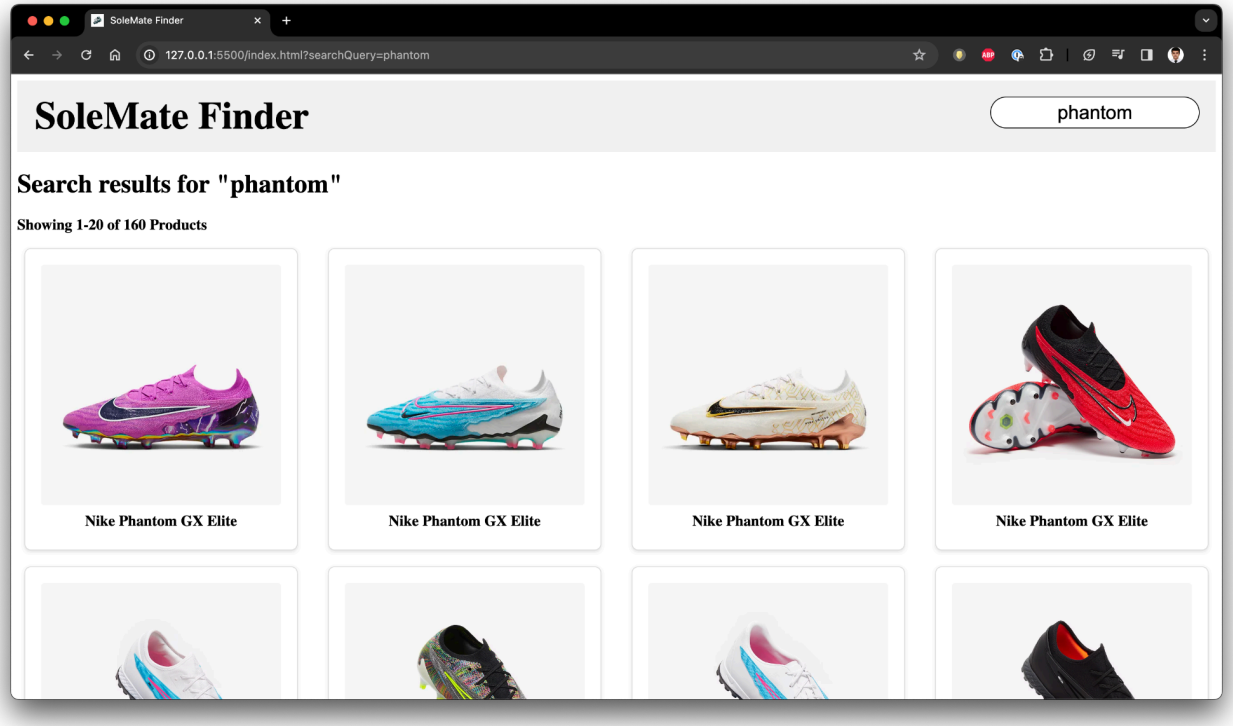
Screenshots

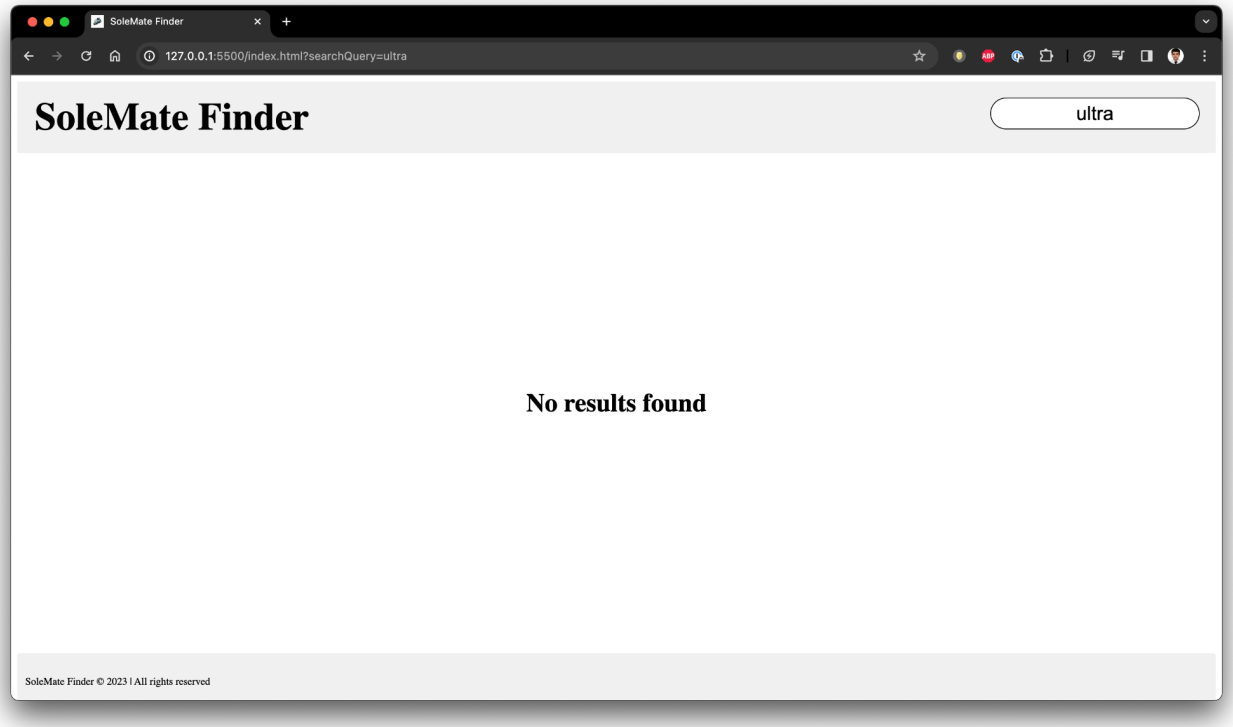
Home Screen



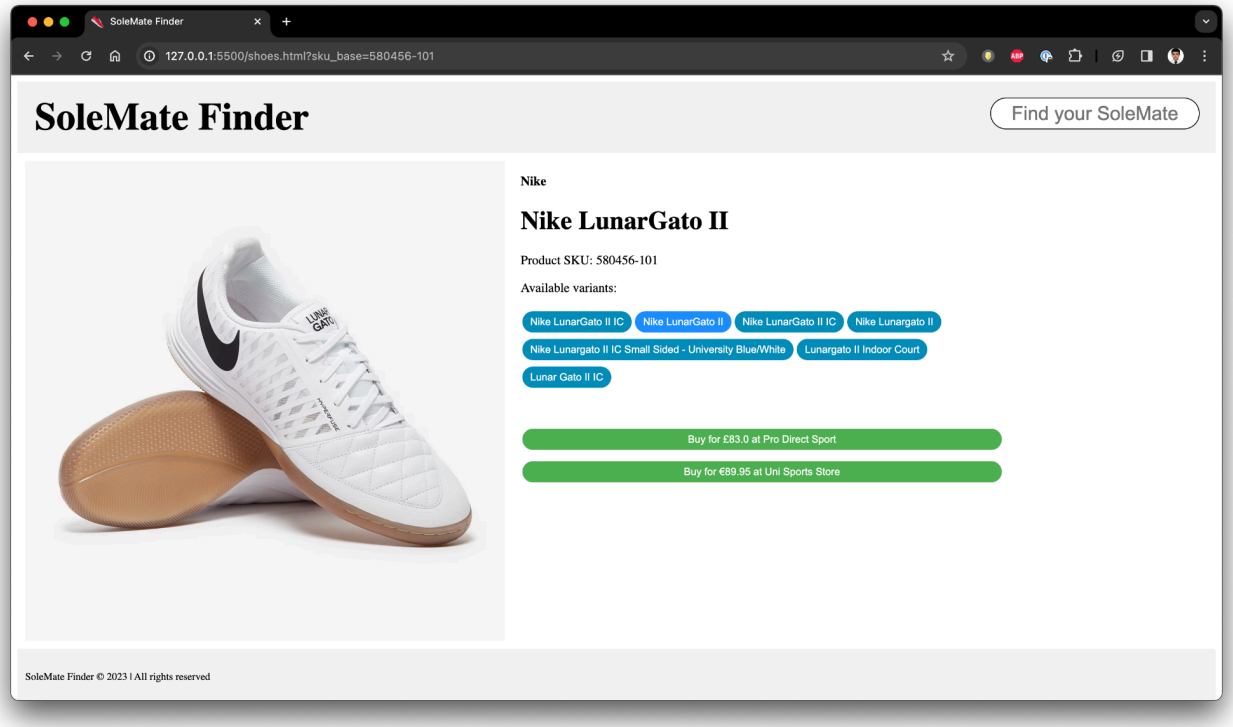


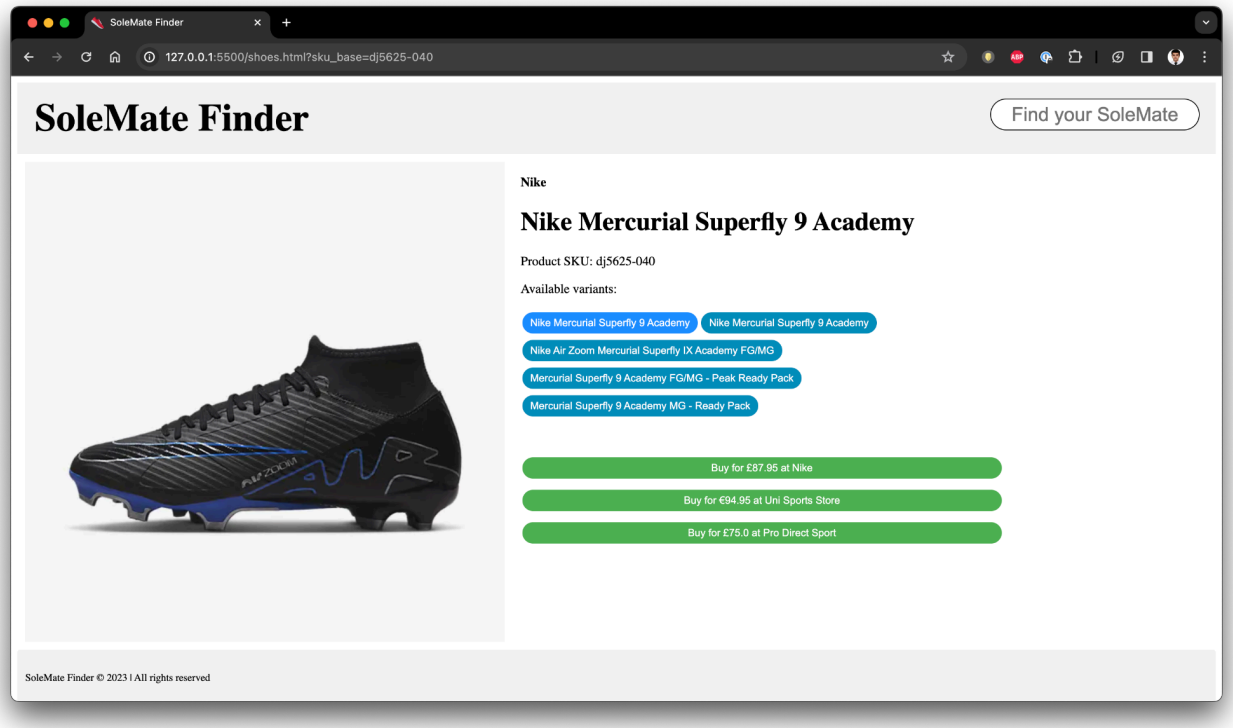
Search


SoleMate Finder
127.0.0.1:5500/index.html?searchQuery=phantom
SoleMate Finder
phantom
Search results for "phantom"
Showing 1-20 of 160 Products
Nike Phantom GX Elite
Nike Phantom GX Elite
Nike Phantom GX Elite
Nike Phantom GX Elite


SoleMate Finder
127.0.0.1:5500/index.html?searchQuery=ultra
SoleMate Finder
ultra
No results found
SoleMate Finder © 2023 | All rights reserved

Product Page



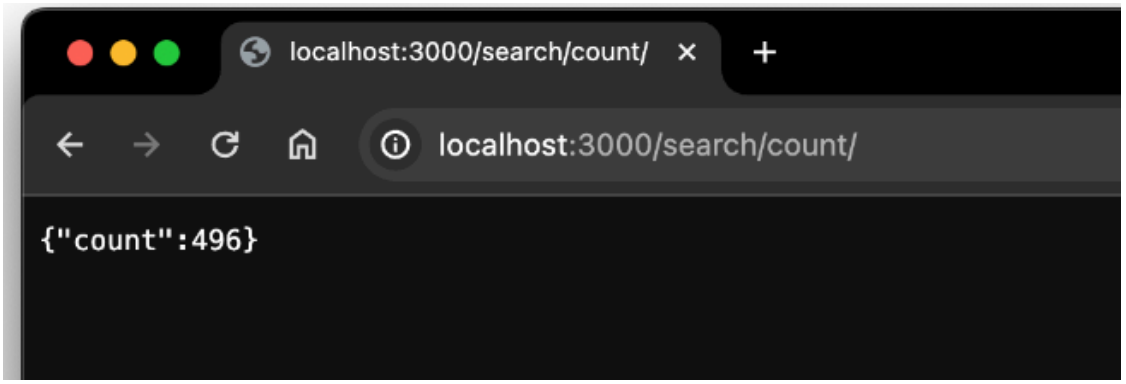


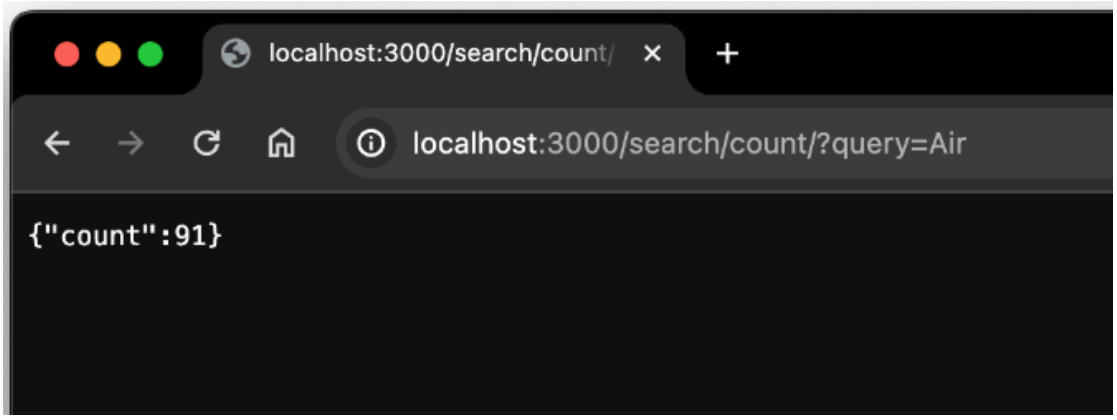
Rest API Screenshots

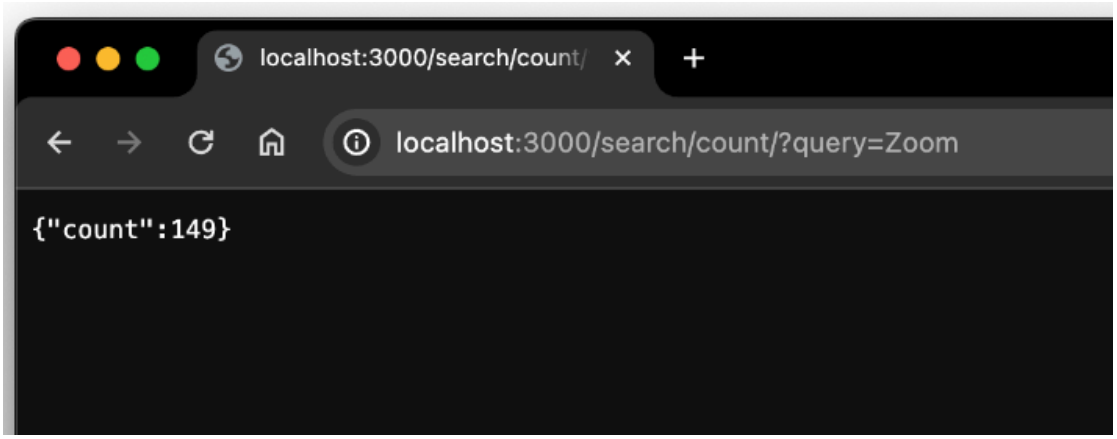
The /search/count endpoint in the API is designed to provide the count of total results based on the given search criteria. Its primary functions are:

Total Count Display: It returns the total number of products or search results. This is useful for showing users how many total items are available or how many items match their search query.

Aid in Pagination: By knowing the total count of items, this endpoint assists in managing pagination. It helps to determine how many pages of results there are, based on the specified limit per page.







The /search/ endpoint in the API is designed for querying shoes based on their full_name. It functions as follows:

Search Functionality: When a search query is provided, the endpoint retrieves shoes where the query matches or is contained in the full_name of the shoes. This allows for targeted searches based on shoe names.

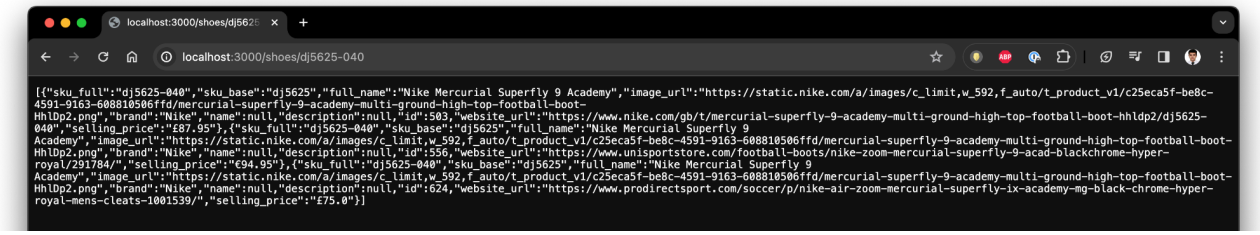
Default Behavior: By default, if no search query is specified, the endpoint returns all available shoes. This default behavior is particularly useful for displaying a broad range of shoes on the homepage.

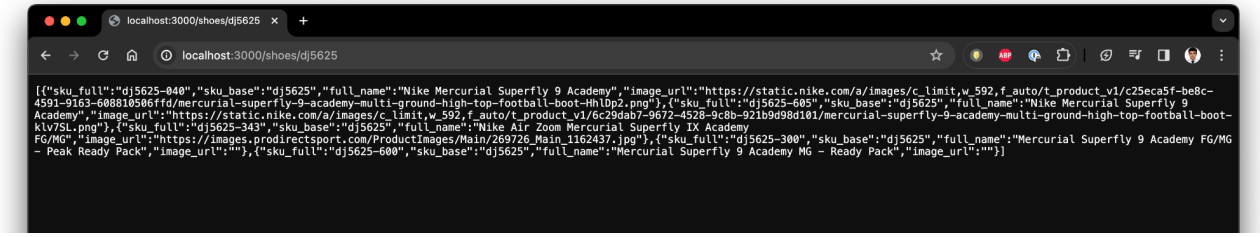
Pagination: The endpoint supports pagination through limit and offset parameters. The limit controls the number of shoes returned in a single response (defaulting to 20 if not specified), and the offset determines the starting point for the query results, facilitating browsing through a large number of shoes in an organized manner.

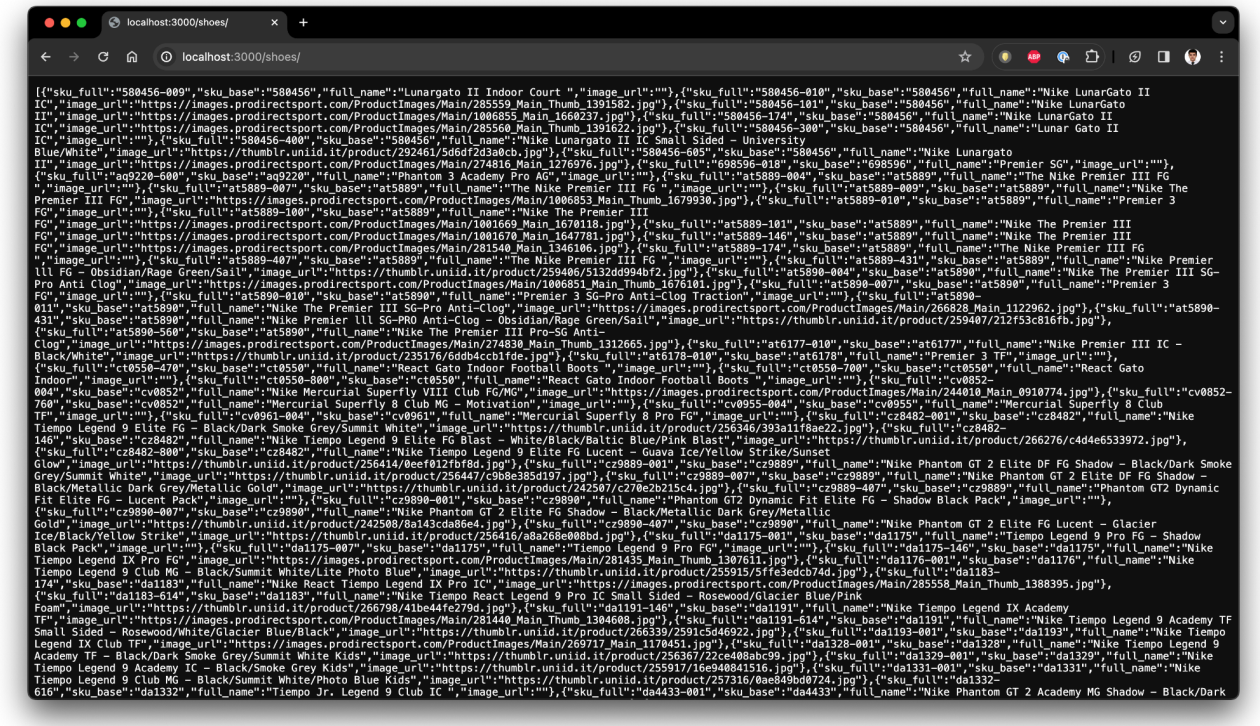
This /shoes/ endpoint in the API serves two purposes based on the provided SKU (Stock Keeping Unit):

With sku_base: When provided with a sku_base, the endpoint retrieves all variants of a shoe model. The sku_base is a general identifier for a shoe model, encompassing various versions or colorways of that model.

With sku_full: When the sku_full is provided, the endpoint returns all the buying options for that specific shoe. The sku_full is a more specific identifier, including details such as full name of the variant, image url and price and urls of the websites from where you can buy the show. This identifier contains a hyphen (-), distinguishing it from the sku_base.







Database Diagram

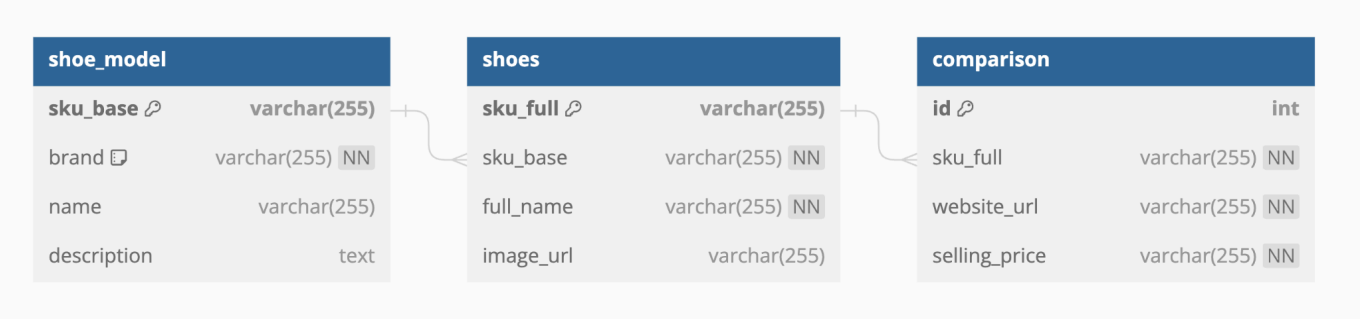


Table: shoe_model

Purpose: This table stores basic information about different shoe models.

- ***sku_base:*** The primary key, a unique identifier for each shoe model.
- ***brand:*** The brand of the shoe, with a default value of 'Nike'.
- ***name:*** The name of the shoe model.
- ***description:*** A textual description of the shoe model.

Table: shoes

Purpose: This table is designed to store specific instances of shoe models, including details like full name and image URLs.

- ***sku_full:*** The primary key, representing a unique identifier for each specific shoe instance.
- ***sku_base:*** A foreign key that links to the sku_base in the shoe_model table, establishing a relationship between a specific shoe and its model.
- ***full_name:*** The full name of the shoe instance.
- ***image_url:*** The URL of the shoe's image.

Table: comparison

Purpose: This table is used to store price comparisons across 5 websites.

- ***id:*** The primary key, an auto-incrementing identifier for each comparison entry.
- ***sku_full:*** A foreign key that connects to the sku_full in the shoes table, linking the comparison data directly to a specific shoe instance.
- ***website_url:*** The URL of the website where the shoe is being sold.
- ***selling_price:*** The selling price of the shoe on the respective website.

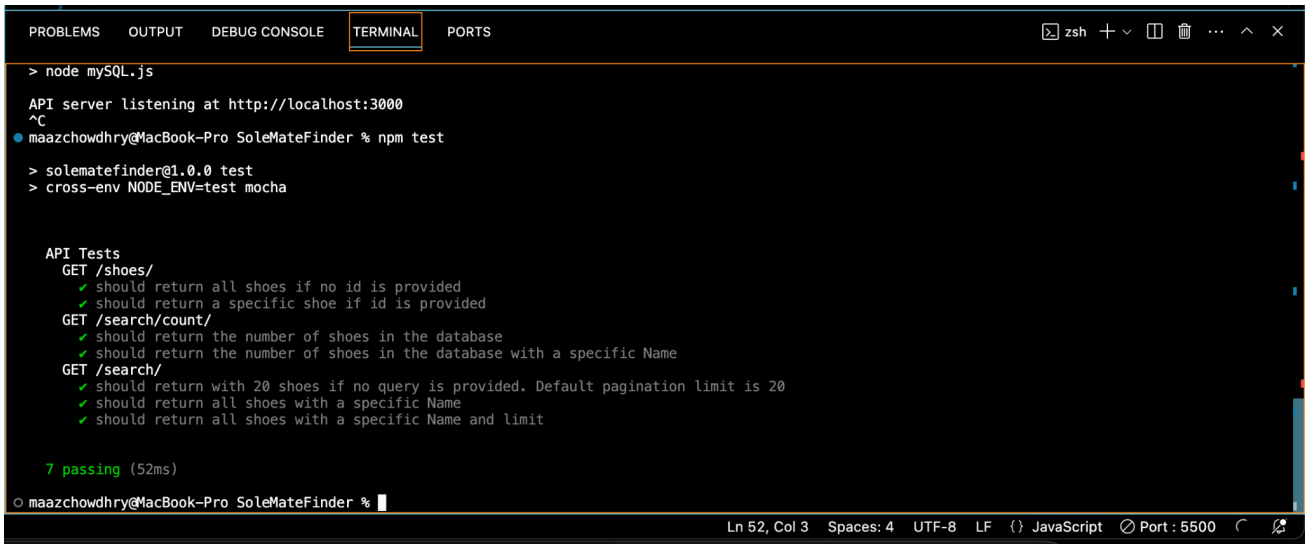
Test Results

Unit test for the REST API uses Mocha for structuring tests and Chai with chai-http for making HTTP requests and assertions. It includes:

Mocha Hooks: before and after hooks set up the test environment and close database connections.

Chai-HTTP for Simulating Requests: Tests API endpoints like /shoes/, /search/count/, and /search/.

Different Scenarios: Checks if the API correctly returns all shoe models, specific models by ID, the total count of shoes, shoes with a specific name, and supports query parameters for detailed searches.



Unit tests for the DatabaseUtility class are written using the Mockito framework.

These tests are designed to ensure that the 'DatabaseUtility' class functions correctly, focusing on its interaction with Hibernate to perform various database operations such as querying and persisting data related to shoes, shoe models, and comparisons.

The tests use mock objects to simulate the behaviour of Hibernate's SessionFactory and Session, allowing for isolated testing without needing an actual database connection.