POLITECNICO
MILANO 1863

Design and Implementation of ***WeCraft Map****: a web application to discover Indian artisans and their products by location*

Author:
Giacomo Stefanizzi
907804

Professors:
Sara Comai
Mariagrazia Fugini

2022-2023

Contents

1 Introduction

1.1 Purpose

This document is meant to provide an explanation of the technical details about the implementation of the WeCraft web app.
In particular, the implementation is presented along with the interfaces that compose the web app and the required back-end. Furthermore, the functionalities offered by the web app are demonstrated through the run-time view, highlighting the interactions available to the user.

1.2 Scope

The scope of the application is to create a web app that allows users to discover various items for sale from Indian artisans, organized by location.
Within the web app, users are able to navigate through a map where markers represent artisans and their items for sale. Users have the option to filter the available items using a search filter located in the left sidebar. This sidebar includes item categories and a range bar for setting desired prices. Moreover, users can click on the markers to view all the items for sale by each artisan. This includes item details such as name, description, price, available quantity, image, and contact information for the artisan.

1.3 Definition, acronyms

1.3.1 Definition

- **RESTful:** It's a software architectural style that defines a set of constraints for creating web services.
- **Tier:** Generally, a tier refers to a row or layer within a series of similarly arranged objects. In computer programming, the components of a program can be distributed among several tiers, each located on a different server within a network.
- **Front-end:** This refers to the user-facing part, including design, interface, and interactions that users access through their browsers.
- **Back-end:** The server-side component of the system that provides an API for an application.

1.3.2 Acronyms

- **API:** Application Programming Interface.
- **HTTPS:** Hyper Text Transfer Protocol over SSL.
- **ER:** Entity-Relationship.
- **DBMS:** Database Management System.
- **UI:** User Interface.

2 Architectural Design

2.1 Overview

In Figure 1, the black-box view of the architecture is depicted, defined by three distinct layers representing:

- **Presentation Layer:** This layer is used to present data from the application layer in an accurate, well-defined, and standardized format. In this case, it is represented by the web application (front-end).
- **Application Layer:** This layer manages all the functions that control the business logic of the system (back-end).
- **Data Layer:** This layer controls how data is stored and accessed, often through a database (DB).

Generally, the architecture splits the application into three distinct layers. Users can access the web app, which is connected to the application layer through a RESTful API.
The Application layer serves as the back-end, containing the application logic. The Data layer, on the other hand, consists of the DBMS, which offers functions for data retrieval and storage via DBMS APIs. These functions are utilized by the application server.

2.2 Deployment View

The deployment diagram in Figure 3 illustrates the system's hardware topology and specifies the distribution of components. The Operating System for each device is indicated.

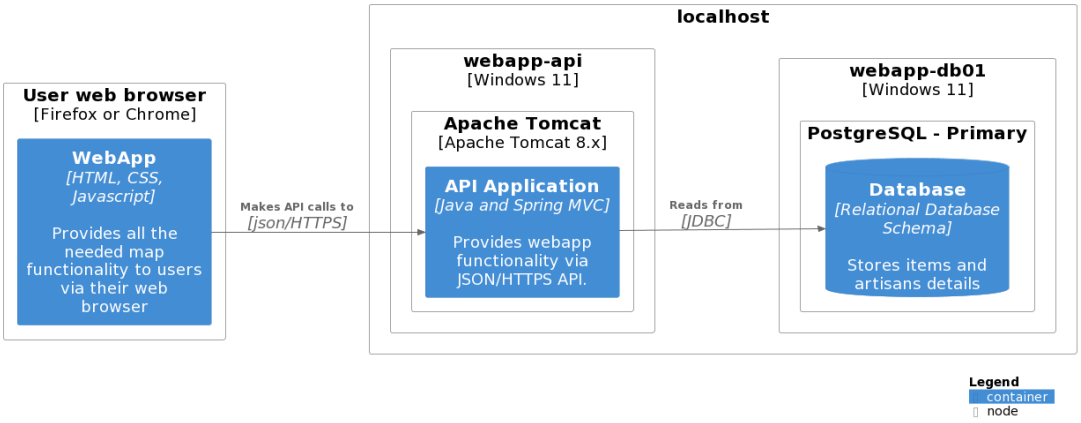

Figure 1: deployment diagram

- **Tier 1: Presentation Layer**
  This is the client machine, typically a PC with a web browser installed.

- **Tier 2: Application Layer**
  This tier comprises the web server, which supplies RESTful APIs. It manages incoming HTTP requests and then forwards them to the application server. This application server houses the business logic, overseeing the requests and establishing connections to the data tier via the DBMS gateway.

- **Tier 3: Data management Layer**
  It consists of the database management system server. Data is stored on this device, and it provides the operations needed to manage data to the application server tier.

2.3 Component View

The following component diagrams illustrate the internal structure of the application server, which cointains both the system's business logic and its data. This high-level diagram illustrates the most significant relationships, which will be further elaborated upon in dedicated diagrams within the upcoming sub-sections.

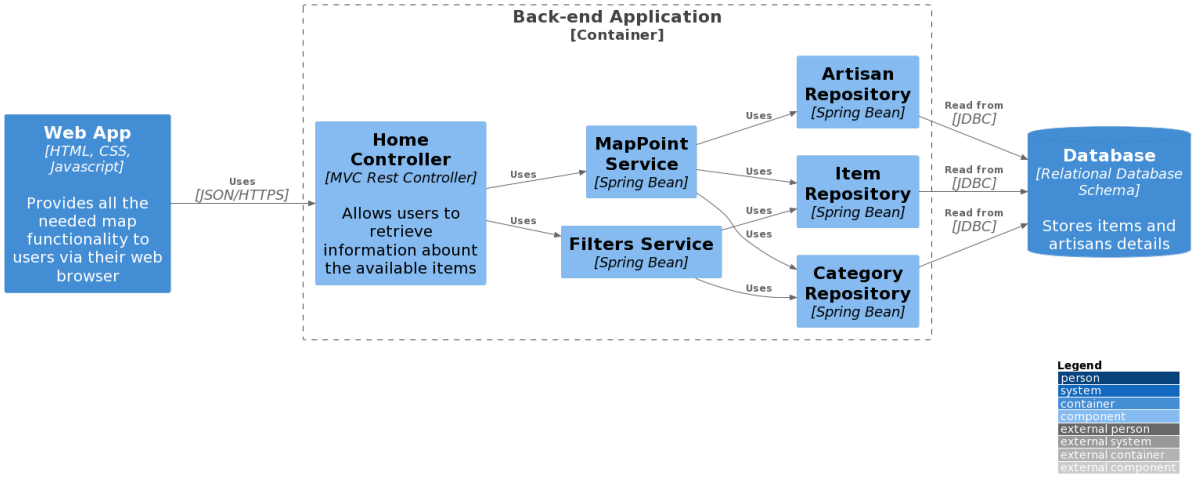

Figure 2: component diagram

- **Web App**
  In the web app ("static" folder), we have an HTML page containing the DOM structure, which includes the sidebar containing checkbox elements for categories, a price range slider, and a button to initiate the search. The styling of these components is located in the CSS stylesheet file, allowing for proper organization of the entire structure and providing a consistent blue, white, and gray color scheme for the page. Finally, the JavaScript script is used to make back-end calls to request filter information, artisan details, and their objects, provide the map, and position artisans on the map.

- **Home Controller**
  In this class, which serves as the actual REST controller, the endpoints that can be called by the front-end are mapped. Depending on the invoked endpoint, a different method

is called, which delegates the operations to be performed to the dedicated service class. After the service class has completed the operations, it takes care of responding to the front-end.

- **MapPoint Service**
  This class is responsible for receiving user-requested filters and constructing the JSON response object containing all the information and related objects that meet the specified criteria. To do so, it utilizes Repository objects that are interfaced with the database, and once the data is retrieved, it restructures it into a custom object for the front-end. Additionally, this class verifies that the filter parameters are valid, in case the user has manipulated the request.

- **Filters Service**
  This service class is used to query the database for available categories and the minimum and maximum prices among the artisans' items in the database. After this, it constructs a JSON response object containing the information, which are provided to the front-end to populate the HTML page.

- **Artisan, Item and Category Repository**
  These classes are used to interface with the database and perform queries using the JPA repository extension provided by Spring Boot. Default methods like findAll() or save() are available, and it's possible to define additional ones that allow querying based on attributes and parameters, even creating custom ones by specifying the SQL query syntax directly.

2.4 Database Design

The Entity-Relationship (ER) schema shown in Figure 4 illustrates the logical organization of database entities, their attributes, and the relationships between them.

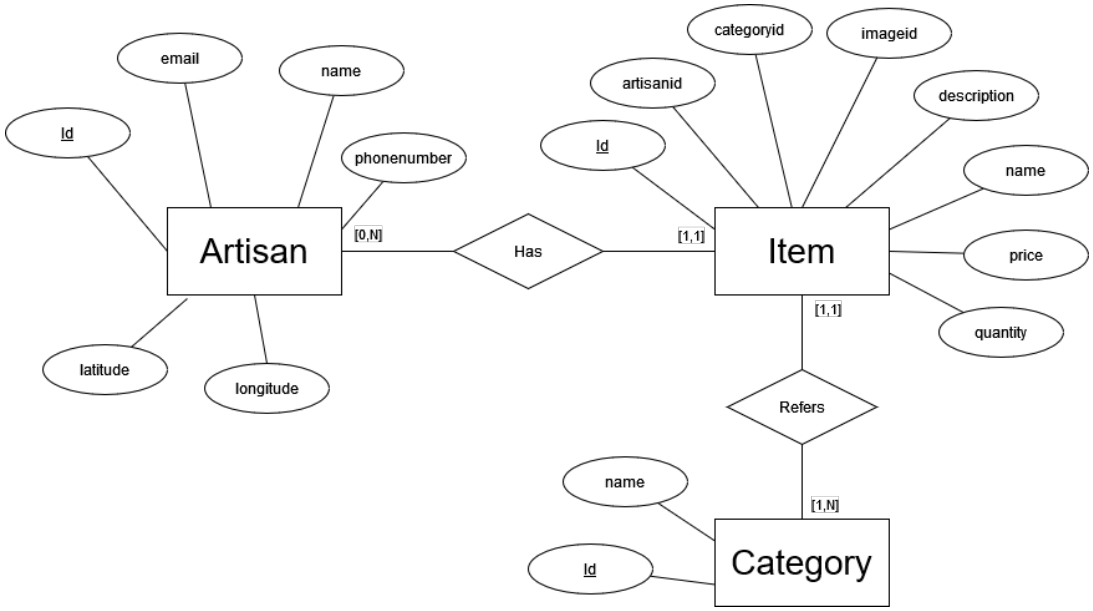

Figure 3: Entity-Relationship schema

2.5 Runtime View

2.5.1 Home Screen

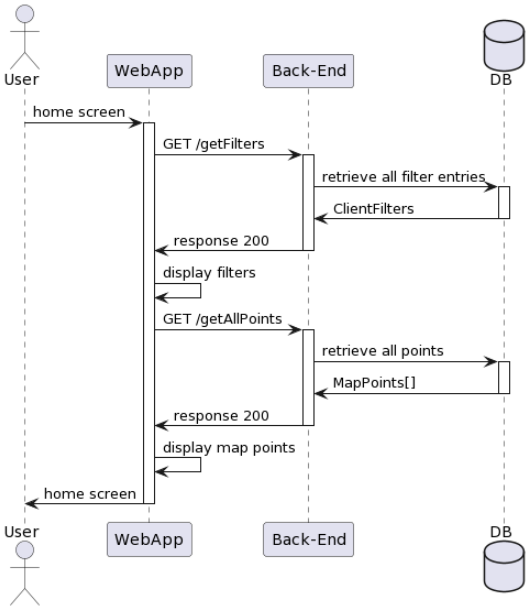

Figure 4: home screen sequence

2.5.2 Search Items

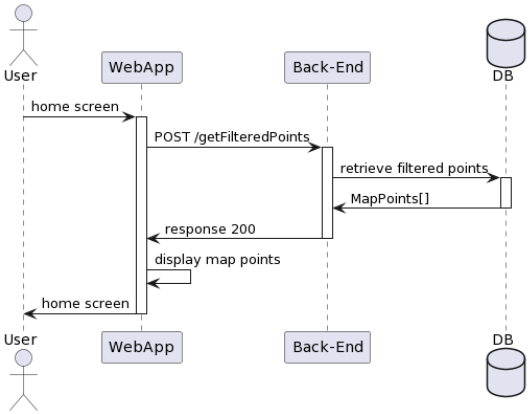

Figure 5: search items sequence

2.6 Selected architectural styles and pattern

2.6.1 Three-tiered architecture

The main reasons why this type of architecture is suitable for the system are as follows:

- **Flexibility:** Once the interfaces are defined, the internal logic remains independent from external factors, allowing each module to be improved without affecting the others.
- **Scalability:** Dividing an application into multiple tiers ensures that the architecture can be scaled selectively for the most critical components. This approach maximizes performance while minimizing costs.
- **Load distribution:** This architecture facilitates the creation of redundant systems, distributing requests among duplicate servers. In contrast, having a single node can lead to an overloading situation, potentially causing the entire system to fail.

2.6.2 Model View Controller (MVC)

Model-view-controller is a software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. These three components are:

- **Model:** The central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data and the logic of the system.
- **View:** The representation of the data provided by the model. In the WeCraft map, the view is static, and the client only needs to render the map points provided by the back-end.
- **Controller:** The intermediate component that accepts input from the user and converts it into commands for the model. This role is played by the Application server which, upon receiving a command, converts it into SQL instructions to be sent to the DBMS.

2.6.3 REST Architecture

All the APIs provided by this web app use the REST architectural style. This style emphasizes the scalability of interactions between components, uniform interfaces, independent deployment of components, and the creation of a layered architecture. This architecture facilitates caching components to reduce user-perceived latency, enforces security measures, and encapsulates legacy systems.

3 User Interface Design

3.1 Application flows

Below, these screenshots showcase all the possible methods for viewing, navigating, and utilizing the web map.

3.1.1 Home Screen

The home screen consists of:
- The map: already zoomed and positioned over India.
- A sidebar: located on the left, offering all the necessary filters to refine the search.

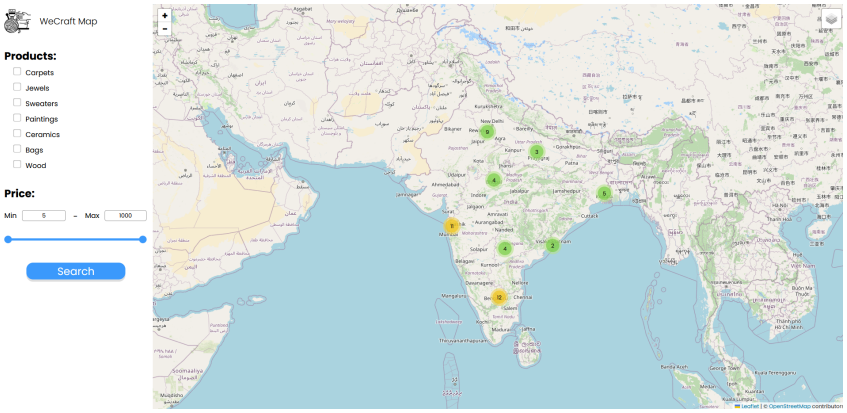

Figure 6: Home Screen UI

3.1.2 Filtering Items

Certain filters have been applied, resulting in fewer markers being displayed on the map.

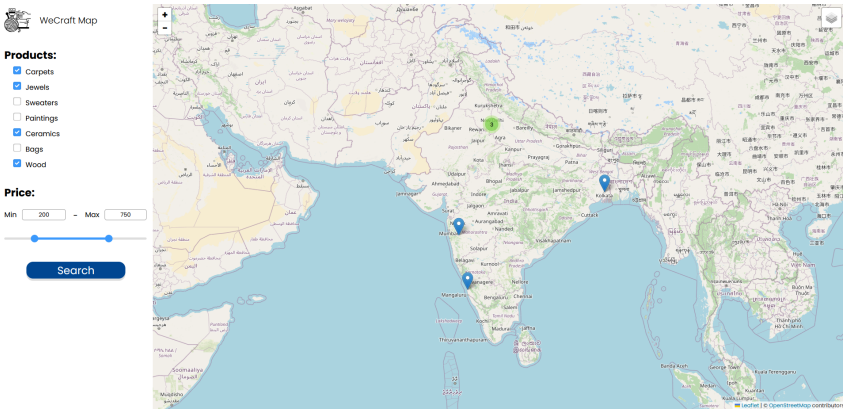

Figure 7: Home Screen UI with filters applied

3.1.3 Zooming In

When zooming in, marker clusters break down into either smaller clusters or individual markers (Figure 9). If a cluster contains points in the same position and is clicked, a "spiderweb" appears to prevent overlap, encompassing all those points (Figure 10).

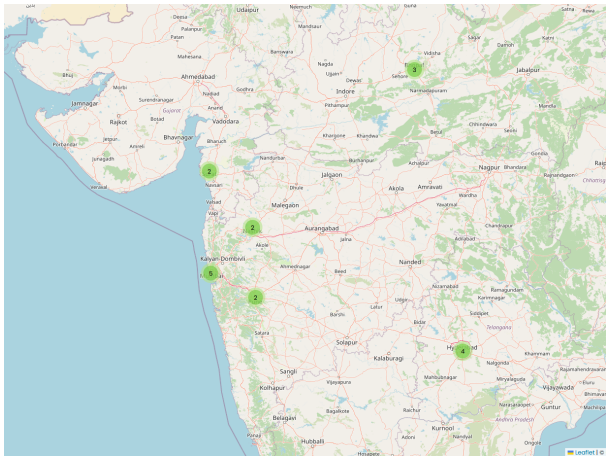

Figure 8: Clusters split up

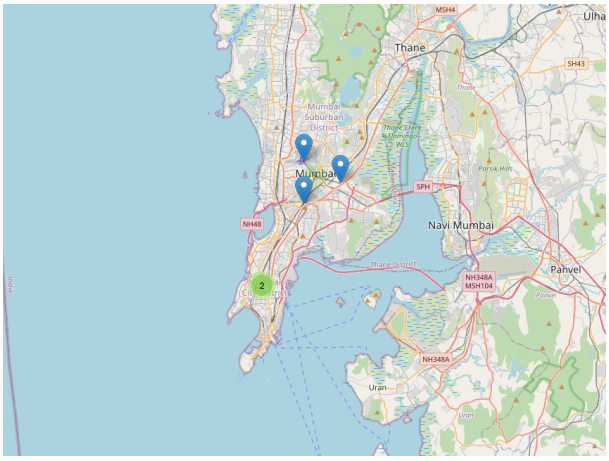

Figure 9: Spiderweb of markers

3.1.4 Item Details

When a marker is clicked, a pop-up appears displaying all the items for sale from that artisan.

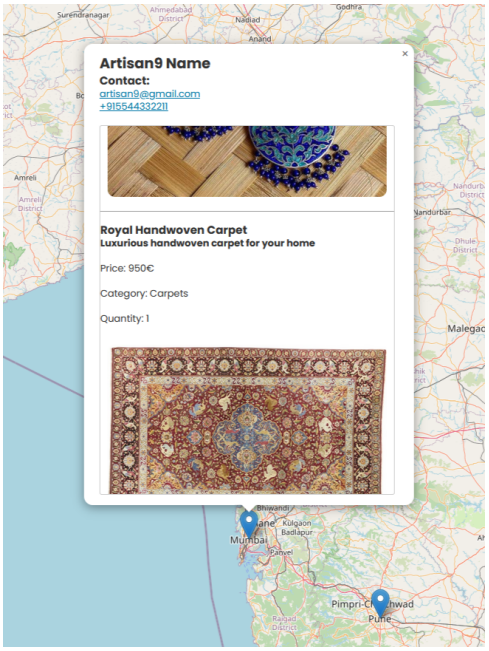

Figure 10: Item Details

4 Implementation, Integration and Testing

4.1 Technologies

This section analyzes all the technologies used to develop each component of the system.

4.1.1 Back-end Server

The back end is implemented in Java using the Spring Boot framework, along with all the necessary dependencies. This approach facilitated the development of REST services, enabling interactions with external services and accessing external databases.

4.1.2 Database

The PostgreSQL server is one of the most widely used databases globally. It is open source, reliable, cost-effective, and easy to manage. Additionally, it offers enhanced capabilities for managing geospatial data.

4.1.3 Web map

The loading of maps was facilitated by employing the Leaflet.js library, which serves as a robust framework. The maps themselves are sourced from OpenStreet Map, a widely acclaimed platform for open and collaborative mapping. The process of point clustering was achieved through the integration of a dedicated plugin.

4.2 Testing methods

During the implementation and testing phases of the web application, the primary focus was on ensuring the robustness and reliability of both the backend and frontend components.

- **Back-end:** To test the backend functionality and REST API calls, it has been used the Postman tool. Postman provides a comprehensive environment to create, manage, and execute various API requests. By crafting test cases and input data, we were able to simulate a wide range of scenarios and validate the backend's responses. This approach allows to identify and rectify potential issues related to data handling, and response codes.
- **Front-end:** For frontend testing, it has been adopted a comprehensive approach to cover all possible edge cases and UI robustness. In cases where errors are detected in the backend or manipulation attempts on the provided parameters are identified, an informative error message is displayed on the UI. This decision aims not only to enhance user-friendliness but also to provide transparency regarding the application's behavior.

5 References

- **Leaflet.js documentation:** https://leafletjs.com/
- **OpenStreetMap:** https://www.openstreetmap.org
- **Sequence diagrams:** All sequence diagrams has been made with PlantUML. https://plantuml.com/
- **Database diagram:** ER diagram has been made with draw.io. https://www.drawio.com/