OCTOBER 23, 2023

EXCHANGE RATE RECOMMENDATION SYSTEM

FEASIBILITY OF REST AND MVC ARCHITECTURE

Author: Hira Arif

Table of Contents

1 Introduction:

The Exchange Rate Recomendation System is a comprehensive software project designed to provide users with a user-friendly web interface to analyze historical currency exchange rate data, make predictions for future exchange rates, and offer recommendations for buying and selling currencies. This system combines data extraction, preprocessing, model training, prediction, recommendation, data visualization, caching, and logging modules to offer a holistic solution for currency traders and analysts.

2 General Answers:

2.1 What will be the structure of a software built using MVC and REST?

Software built using MVC and REST typically has the following structure:

2.1.1 Model:

This component represents the application's data and business logic. It is responsible for retrieving, processing, and storing data. In RESTful applications, the model often corresponds to the database or data storage.

2.1.2 View:

The view is responsible for presenting the data to the user in a user-friendly format. It can be a web page, mobile app interface, or any other client-side presentation layer.

2.1.3 Controller:

The controller handles user requests and acts as an intermediary between the model and the view. It processes incoming requests, interacts with the model to retrieve, or update data, and then determines how the data should be presented in the view.

2.1.4 RESTful Web Services:

In RESTful architecture, the software exposes APIs (web services) that follow REST principles. These APIs are used to perform CRUD (Create, Read, Update, Delete) operations on resources. Resources are represented by URLs, and operations are performed using standard HTTP methods (GET, POST, PUT, DELETE).

2.2 How are CRUD operations performed using MVC and REST?

CRUD operations are typically performed in a RESTful MVC application as follows:

2.2.1 Create (POST):

To create a new resource, you send a POST request to the appropriate URL, with data in the request body. The controller receives the request, processes the data, and updates the model accordingly.

2.2.2 Read (GET):

To retrieve data, you send a GET request to the URL representing the resource you want to read. The controller fetches data from the model and sends it to the view component for presentation.

2.2.3 Update (PUT or PATCH):

To update an existing resource, you send a PUT or PATCH request to the URL of that resource with the updated data. The controller updates the model with the new information.

2.2.4 Delete (DELETE):

To delete a resource, you send a DELETE request to the resource's URL. The controller removes the resource from the model.

2.3 How is request dispatcher created in MVC and REST?

In traditional web development using MVC (e.g., in Java Servlets), a request dispatcher is used to forward or include a request to another resource (such as a servlet or JSP). However, in RESTful architecture, the concept of a request dispatcher is not commonly used. RESTful APIs are stateless, and each API endpoint is responsible for handling its requests independently. The request routing and dispatching are often handled by the web server or framework, and the controller (resource) that matches the URL path is responsible for processing the request.

2.4 How are APIs used for client-server communication in MVC and REST?

In MVC and REST architecture, APIs are used for client-server communication as follows:

- Clients (e.g., web applications, mobile apps) make HTTP requests to specific API endpoints (URLs) on the server.
- The server's controller component processes the requests and interacts with the model to retrieve or update data.
- The server sends an HTTP response, often in JSON or XML format, containing the requested data or confirmation of the action taken.
- Clients can then parse the response and use the data to update their views or perform further actions.

In REST, APIs are designed to be stateless and follow the principles of using standard HTTP methods and resources. This makes client-server communication more standardized and allows for scalability and flexibility in web applications.

3 Answers specific to the project:

3.1 What will be the structure of a software built using MVC and REST?

The "Exchange Rate Recommendation System" can be structured using the Model-View-Controller (MVC) architectural pattern in conjunction with RESTful (Representational State Transfer) principles.

3.1.1 Model:

In this system, the Model represents the data and logic behind currency exchange rate prediction. It includes components for data extraction, preprocessing, model training, and prediction generation. This model layer handles data manipulation, transformation, and the prediction algorithm.

3.1.2 View:

The View component is responsible for the presentation of the system. In this context, it involves the user interface (UI) of the web application, which allows users to interact with the system. Users can select base and comparison currencies, historical analysis times, and prediction horizons through the web interface.

3.1.3 Controller:

The Controller acts as an intermediary between the Model and the View. It processes user requests from the web interface, communicates with the Model to retrieve data and predictions, and returns the results to the View for presentation. It also handles user interactions like currency selections and time-period choices.

3.1.4 REST:

In addition to the MVC structure, the system uses RESTful principles to enable client-server communication. RESTful APIs are employed to expose various functionalities, such as data retrieval, prediction, and recommendation generation. The API endpoints enable clients (e.g., the web interface) to make HTTP requests to interact with the system. This approach simplifies the communication between the client and server, making it stateless, scalable, and efficient.

3.2 How are CRUD operations performed using MVC and REST?

In the context of the "Exchange Rate Recommendation System," CRUD (Create, Read, Update, Delete) operations can be performed as follows:

3.2.1 Create:

The system allows users to create new predictions and recommendations by selecting specific base and comparison currencies, analysis time periods, and prediction horizons. The web interface sends a

POST request to the RESTful API endpoint responsible for prediction creation. The Controller processes this request and communicates with the Model to create a new prediction.

3.2.2 Read:

Users can read (retrieve) historical exchange rate data, predictions, and recommendations through the web interface. The web interface sends GET requests to specific RESTful API endpoints, triggering the Controller to retrieve the relevant data from the Model and present it to the user through the View.

3.2.3 Update:

If the system needs to update a prediction or any other information, users can trigger an update by sending a PUT request to the corresponding RESTful API endpoint. The Controller processes the request and communicates with the Model to make the necessary updates.

3.2.4 Delete:

Users can request the deletion of specific predictions or recommendations by sending a DELETE request to the corresponding RESTful API endpoint. The Controller handles the request and coordinates with the Model to remove the specified data.

3.3 How is request dispatcher created in MVC and REST?

In the context of a web-based application following the MVC architecture and RESTful principles, a request dispatcher is typically not explicitly created as in traditional server-side web applications. Instead, the request is handled through routing mechanisms provided by web frameworks or libraries.

For example, in a Java-based web application, we might use a framework like Spring MVC or a similar framework for request handling. In RESTful APIs, requests are mapped to specific controller methods based on the URL paths and HTTP methods. The controller methods process the requests, interact with the model, and return the appropriate responses.

So, in the "Exchange Rate Recommendation System," request dispatching is handled implicitly by the web framework and the routing configuration. The system doesn't create a request dispatcher in the traditional sense but relies on the framework's routing mechanisms.

3.4 How are APIs used for client-server communication in MVC and REST?

In the "Exchange Rate Recommendation System," APIs (Application Programming Interfaces) are used for client-server communication, following RESTful principles. Here's how APIs are utilized:

3.4.1 API Endpoints:

The system defines specific RESTful API endpoints to expose its functionalities. These endpoints are URLs that clients (e.g., the web interface) can use to interact with the system. For example, there might be endpoints for data retrieval, prediction generation, or recommendation creation.

3.4.2 HTTP Methods:

RESTful APIs use standard HTTP methods to perform operations. For example, GET requests are used for data retrieval, POST for creating predictions, PUT for updates, and DELETE for deletion. These HTTP methods are mapped to corresponding actions in the system.

3.4.3 Data Format:

The system communicates data in a structured format, typically JSON or XML, which is easy to parse and understand for both the server and client.

3.4.4 Stateless Communication:

RESTful communication is stateless, meaning each request from the client to the server must contain all the information needed to understand and fulfill the request. The server does not store client state between requests.

3.4.5 Response Codes:

The server responds with appropriate HTTP status codes to indicate the result of the request (e.g., 200 for success, 404 for not found, 500 for server errors).

3.4.6 Communications:

Clients (e.g., web browsers) make HTTP requests to these API endpoints to perform various actions, such as retrieving historical data, creating predictions, or obtaining recommendations. The server, following REST principles, processes these requests, communicates with the Model and Controller, and returns responses in the agreed data format.

This implies that APIs in the "Exchange Rate Recommendation System" serve as the gateway for client-server communication and enable seamless interaction between the user interface (View) and the underlying system (Model and Controller).
