M7024E Laboratory 4: Programming Cloud Services - RESTful APIs

submitted by

Md Asif Mahmod Tusher Siddique, Muiz Olalekan Raheem

January 10, 2023

submitted to

Dr. Karan Mitra

1 Objectives

The objective of this lab is to:

- Understand methodologies for developing Cloud application and services.
- Program a simple RESTful API.

2 Questions

What are microservices? Describe in detail the pros and cons of microservices architecture by giving examples.

Answer:

Microservices are a software architecture style in which a large software application is divided into smaller, independent components that communicate with each other through well-defined APIs (Application Programming Interfaces). Each microservice is designed to perform a specific function, and can be developed, tested, and deployed independently of the other microservices.

The main advantage of using microservices is that they allow for greater flexibility and scalability in the development and maintenance of software applications. Because each microservice is self-contained and independently deployable, it is easier to make changes to a specific microservice without affecting the entire application. This can make it easier to add new features, fix bugs, and perform other maintenance tasks. Microservices are typically developed using a variety of programming languages and technologies, depending on the specific requirements of the application. They can be deployed to run on a variety of platforms, including cloud-based platforms such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform.

Overall, microservices are a popular approach to building software applications that are designed to be flexible, scalable, and easy to maintain.

Advantages: There are several advantages to using microservices in software development:

1. Improved flexibility: Microservices allow for greater flexibility in the development and maintenance of software applications. Because each microservice is self-contained and independently deployable, it is easier to make changes to a specific microservice without affecting the entire application.

2. Enhanced scalability: Microservices can be scaled up or down independently of each other, which allows for more efficient use of resources. This can be particularly useful for applications that experience varying levels of traffic and demand.

3. Enhanced scalability: Microservices can be scaled up or down independently of each other, which allows for more efficient use of resources. This can be particularly useful for applications that experience varying levels of traffic and demand.

4. Easier maintenance: Because each microservice is focused on a specific function, it is easier to identify and fix issues that arise. This can save time and resources compared to traditional monolithic software architectures, where issues may be more difficult to locate and fix.

5. Greater resilience: Microservices are designed to be modular and independent, which makes them less prone to failures. If one microservice goes down, the rest of the application can continue to function normally, resulting in improved availability and reliability.

6. Better support for continuous delivery and deployment: Microservices are typically designed to be developed, tested, and deployed independently of each other, which makes it easier to implement continuous delivery and deployment processes. This can speed up the development process and allow for faster time-to-market.

Overall, the use of microservices can provide a number of benefits in the development and maintenance of software applications.

Disadvantages: While microservices offer many advantages in software development, there are also some potential drawbacks to consider:

1. Complexity: Microservices can add complexity to the development process, as they require the use of APIs (Application Programming Interfaces) to communicate with each other. This can require additional time and resources to set up and maintain.

2. Debugging and testing: Debugging and testing can be more challenging with microservices, as issues may span multiple microservices and require a more extensive debugging process.

3. Deployment: Deploying microservices can be more complex than deploying a traditional monolithic application, as each microservice must be deployed independently. This can require additional time and resources.

4. Integration: Integrating microservices can be more complex than integrating a traditional monolithic application, as it requires the use of APIs to communicate between the various components.

5. Security: Microservices can pose additional security risks, as they may rely on multiple APIs and communication channels. This can increase the risk of vulnerabilities and require additional security measures to be put in place.

Overall, while microservices offer many benefits, it is important to carefully consider the potential drawbacks and ensure that they are the right fit for a particular project.

3 Exercises

Exercise a: In this exercise, you will learn how to develop a simple RESTful API for the application of your choice.

1. Think about an application, for example, an online record keeping application; or an online Cloud monitoring service such as CloudHarmony; or a music service such as last.fm that keep tracks of users music taste profile. Select a service that you wish to create on similar lines.

2. Use Docker container to deply your service. Use Dockerfile to build your image.

3. Use microservices as an architectural paradigm to design a simple service(s). Describe and document that service.

4. Create a RESTful API to implement your service. Your RESTful API should implement atleast two verbs, for example GET and POST. For GET, you should implement atleast one path,

Solution

Our solution for this is a Crypto currency related webApp. This provides users with real-time or historical data on the price of various cryptocurrencies, which allow users to view the current market value of a particular cryptocurrency, as well as

its price history over a certain period of time. Users of this website can use the price data to track the value of their own cryptocurrency holdings or to inform their decision-making when buying or selling cryptocurrencies. We felt price tracking is important for understanding the overall health of the cryptocurrency market. By analyzing price trends and changes, investors can get a sense of the market sentiment and identify potential buying or selling opportunities. Additionally, price tracking can help identify potential market manipulations or other irregularities, which can be important for maintaining the integrity of the market.

Our web application serves two main purposes:

1. It allows users to simulate trading Bitcoin and track the prices of various cryptocurrencies in real-time. Some key features of our website include the ability to track cryptocurrency prices in real-time, view historical price data using charts for periods of different lengths of days.

2. Use an investing simulator where users can start with a certain amount of cash and trade BTC for Euro or vice versa using real-time data.

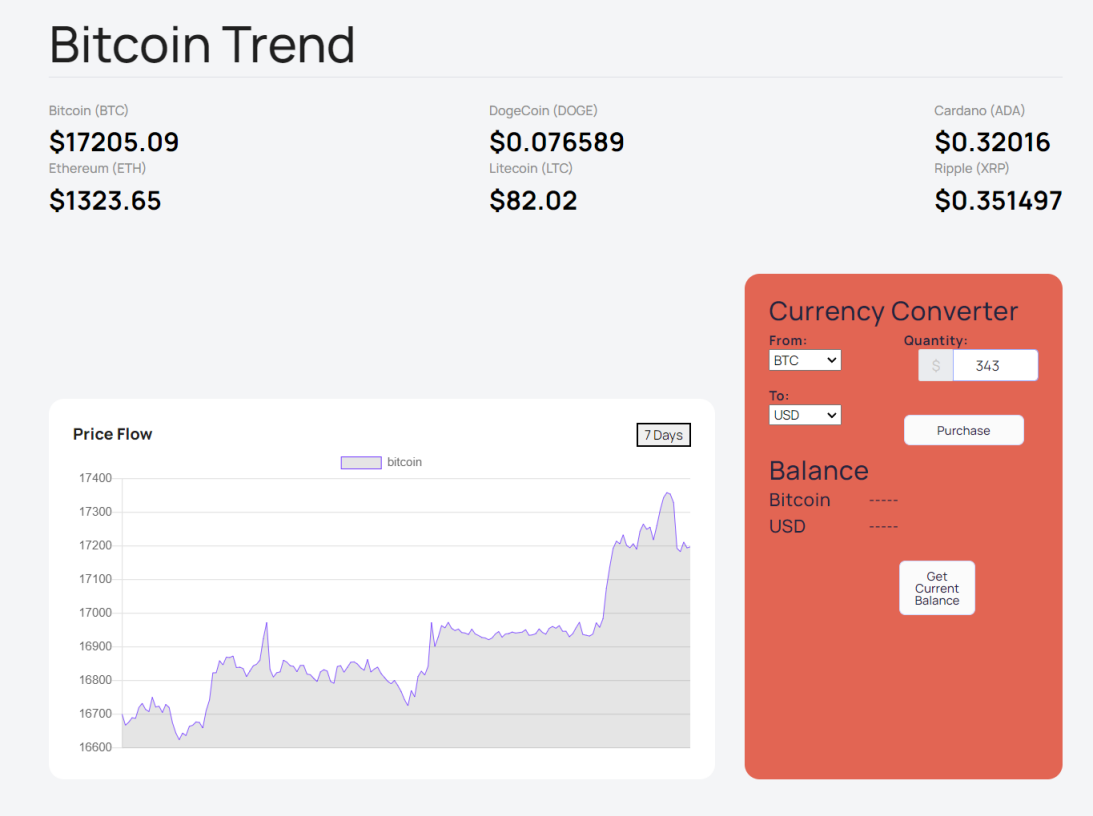

Figure 1: The WebApp

For the REST framework, Python and Flask were utilized. In the frontend, Javascript (chart.js, ajax), HTML, and CSS are utilized. We chose mongoDB as our database because it offers a configurable schema, automation, redundancy, and scalability that is ideal for microservice design. A Docker container is used for deployment. In order to direct port 80 to the port where Flask is executing, we additionally utilized nginx as a reverse proxy.

Architecture: The website retrieves the value of BTC from API-1, which accesses CoinGekoAPI. The website then stores the transaction by making a post request to API-2, which manages a non-relational database. The website can also retrieve the latest transaction to display the current balance.

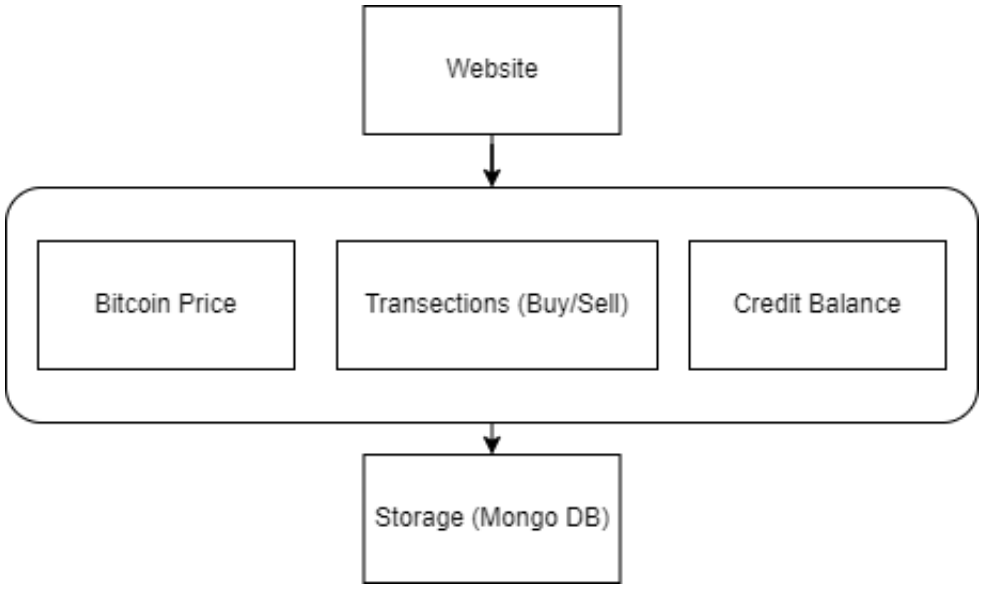

Figure 2: Cloud design Architecture

API-1 retrieves the price of a cryptocurrency by making a GET request to coingecko.com, a website that provides cryptocurrency-related data. The retrieved data is then processed and displayed in the user frontend, which is deployed in Docker container 1. Container 2 contains API-2, which is used to read and write data in the database. The database (mongoDB) is deployed in Docker container 3.

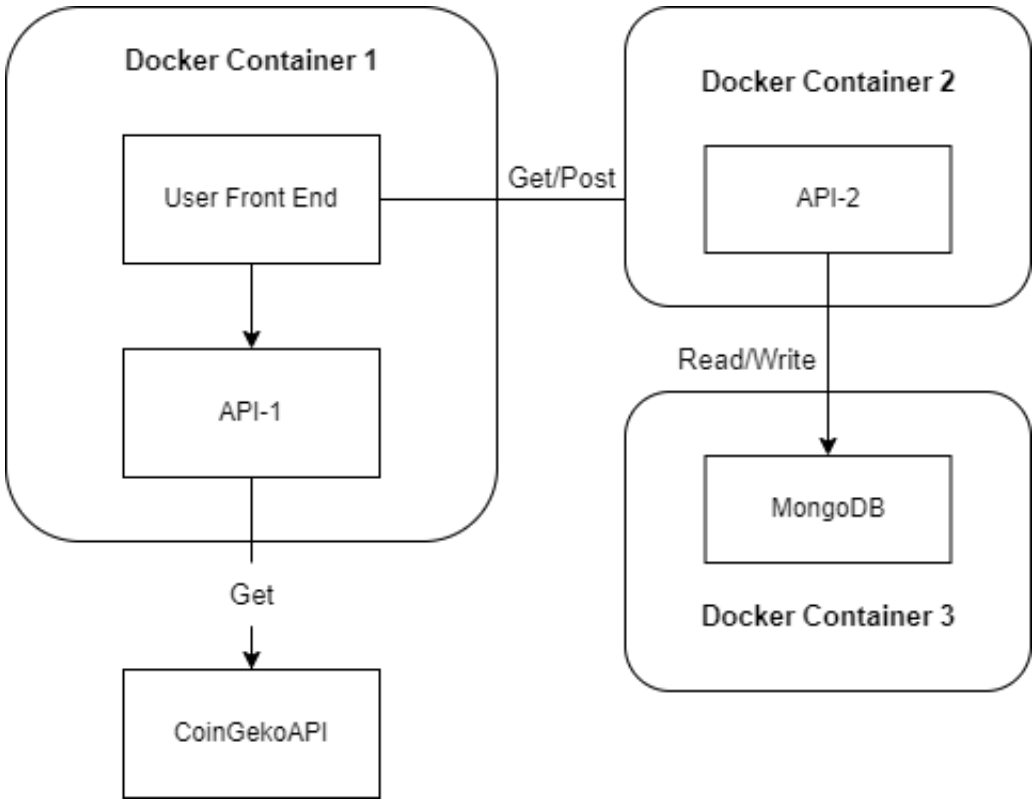

Figure 3: Architecture of the Docker Containers

In this project, we used microservices to develop the application in a modular way, which made it easy to combine the different parts. We have now gained a good understanding of how microservices work and the benefits of using them to build applications. Our project covers all the objectives of the lab. We implemented both GET and POST methods, used a database, and integrated external APIs as well as the ones we created. In the future, we plan to add additional features such as login functionality and the ability to track a user's custom portfolio. We also hope to build a trading bot that can trade assets automatically using price prediction.

References

one, howpublished = http://16.170.202.83:8000/,