CO565
Web Services

CW1

21611431

16th May 2018

Computing & Web Development
Buckinghamshire New University

Summary

Part A

Research

Section 1

Technology - Wider Context

1.1 Considerations on the challenges when building distributed systems

When creating distributing systems, different challenges had to be overcome by developers in other to develop such systems:

- Inherent complexities: Inter-node communication needs different mechanisms than intra-node communication, as well as policies and protocols. Synchronization and coordination is more complicated, and problems such as latency, jitter, transient failures and overload are also problems to consider.
- Accidental complexities: Those happen because of limitations with software tools and development techniques. Most of these happen because of choices made when the time arose to build the system.
- Inadequate methods and techniques: For a long time, software development studies and practices have focused on single-process and single-threaded applications, which has lead to a lack of expertise on software techniques distributed systems.
- Continuous re-invention and re-discovery of core concepts and techniques: For a long time, time has been used on solving the same problems over and over again, instead of trying to develop techniques that would facilitate the reuse of code across different platforms, which lead to less time being available to invest in innovation.

1.2 Different Distributed Computing technologies

1.2.1 Ad hoc Network Programming

Ad hoc network programming was one of the first solutions to allow the development of distributed computing systems. That technology used Interprocess Communication mechanisms. In IPC, there was a use of shared memory, pipes and sockets, which lead to a big coupling between the application code and the socket API, as well as paradigms mismatches, since local communication techniques use object-oriented techniques and remote communications use function-oriented techniques.

1.2.2 Structured communication

This technology was an improvement to ad hoc since it doesn't couple application code to low-level IPC mechanisms, offers higher-level communication to distributed systems, encapsulates machine-level details and embodies types and communication style closer to the application domain. One particular example of this technology are Remote Procedure Call platforms. They have the following characteristics:

- Allow distributed applications to cooperate with one another by:
  - Invoking functions on each other
  - Passing parameters along with each invocation
  - Receiving results from the function that was invocated

With this technique, however, components are still aware of their peers' remoteness, therefore it does not fulfil the following distributed systems requirements:

- Location-independence of components
- Flexible component deployment

1.2.3 Middleware

Middleware is used as a bridge between the remote system and the application, being a distribution infrastructure software. It resides between an application and the operating system, network or database. Different types of middleware have appeared over time, which are listed below.

1.2.3.1 Distributed Object Computing

Also known as DOC, it represents the confluence of two major information technologies: RPC-based distributed computing systems and object-oriented design and programming.

COBRA 2.x and Java RMI are examples of DOC. They focus on interfaces, which are contracts between clients and servers that define a location-independent means for clients to view and access object services provided by a server. Such technologies also define communication protocols and object information models to enable interoperability.

Their key limitations are:

- Lack of functional boundaries: This type of middleware treats all interfaces as client/server contracts and don't provide standard assembly mechanisms to decouple dependencies among collaborating object implementations. This leads to a requirement for the explicit discovery and connection to objects that are only dependencies for the actual object that the application is trying to reach.
- Lack of software deployment and configuration standards: Results in systems that are harder to maintain and software component implementations that are much harder to reuse.

1.2.3.2 Component middleware

This technology emerged to address the following limitations of DOC:

- Lack of functional boundaries: Allows a group of cohesive component objects to interact with each other through multiple provided and required interfaces and defines standard runtime mechanisms needed to execute these component objects in generic application servers.
- Lack of standard deployment and configuration mechanisms: Component middleware specifies the infrastructure to packages, which makes it easier to customize, assemble and disseminate components throughout a distributed system.

Examples of this technology are Enterprise JavaBeans and COBRA Component Model.

For this technology, a set of rules and definitions has been applied:

- Component is an implementation entity that exposes a set of named interfaces and connection points that components use to communicate with each other.
- Containers provide the server runtime environment for component implementations. Contains various pre-defined hooks and operations that give components access to strategies and services, such as persistence, event notification, transaction,

replication, load balancing and security. Each container is also responsible for initializing and providing runtime contexts for the managed components.

Component middleware also automates aspects of various stages in the application development lifecycle like component implementation, packaging, assembly and deployment. This approach enables developers to create applications more rapidly and robustly than DOC.

1.2.3.3 Publish/Subscribe and Message-Oriented Services

RPC, DOC and component middleware are all based on request/response communication. Aspects of this type of communication are:

- Synchonous communication: The client waits for a response and stops the rest of its process
- Designated communication: The client must know the identity of the server, which leads to tight coupling between the application code and the API
- Point-to-Point communication: Client talks to just one server at a time

In order to fix the problems that that type of communication created for some systems, the following alternatives were created:

- Message-Oriented Middleware: Applied in technologies such as IBM MQ Series, BEA MessageQ and TIBCO Rendezvous
- Publish/Subscribe middleware: Applied in technologies such as Java Messaging Service (JMS), Data Distribution Service (DDS) and WS-NOTIFICATION

Specifications of Message-Oriented:

- Support for asynchronous communication
- Transactional properties

On top of that, Publish/Subscribe also allows:

- Anonymous communication (loose coupling)
- Group communication

On the Publish/Subscribe paradigm:

- Publishers are the source of events. The may need to describe the type of event they generate sometimes
- Subscribers are the event sinks; they consume data on topics of interest to them. They may need to declare filtering information sometimes
- Event channels: Components in the system that propagate events. They can perform services such as filtering and routing, Quality of Service enforcement and fault management.

In these types of systems, events can be represented in many ways, and the interfaces can also be generic or specialized.

1.2.3.4 Service-Oriented Architectures and Web Services

Service-Oriented Architecture (SOA) is a style of organizing and utilizing distributed capabilities that may be controlled by different organizations and owners. It provides a uniform means to offer, discover, interact with and use capabilities of loosely coupled and interoperable software services to support the requirements of the business processes and application users.

From the emergence of this technology along with the increasing popularity of the World Wide Web, SOAP was created. It is a protocol to exchange XML-based messages over a network, normally using HTTP. SOAP spawned a popular new variant of SOA called Web Services. It allows developers to package application logic into services whose interfaces are described with the Web Service Description Language (WSDL). WSDL-based services are often accessed using higher-level Internet protocols. They can also be used to build an Enterprise Service Bus(ESB), which is a distributed computing architecture that simplifies interworking between disparate systems.

Web services have established themselves as the technology of choice for most enterprise business applications. They complement earlier successful middleware technologies and provide standard mechanisms for interoperability. Examples of Web Services technologies are:

- Microsoft Windows Component Foundation (WCF)
- Service Component Architecture (SCA)

Web Services combine aspects of component-based development and Web technologies. They also provide black-box functionality that can be described and reused without concern for how a service is implemented. They are not accessed using the object model-specific protocols, but rather using Web protocols and data formats.

Web service technologies focus on middleware integration & allow component reuse across an organization's entire application set, regardless of the technology implemented for those. In general, web services make it relatively easy to reuse and share common logic with such diverse clients as mobile, desktop and web applications. The broad reach of web services is possible because they rely on open standards that are ubiquitous, interoperable across different computing platforms and independent of the underlying execution technologies.

All web services use HTTP and leverage data-interchange standards like XML and JSON, as well as common media types, but differ on the way they may use HTTP:

- As an application protocol to define standard service behaviours
- HTTP as a transport mechanism to convey data

In Web services, there are two prominent design models:

- SOAP
  - Language, platform and transport independent
  - Works well in distributed enterprise environments
  - Provides significant pre-build extensibility in the form of WS* standards
  - Built-in error handling
  - XML message-format only
  - Automation when used with certain language products
  - Uses WSDL as a description language
- REST
  - Uses easy to understand standards like swagger and OpenAPI Specification 3.0
  - Can use different types of message formatting like JSON
  - Fast (no extensive processing required)
  - Close to other Web technologies in design philosophy
  - Uses WADL as a description language

Part B

Report

Section 2

Project Proposal

2.1 Project Name

The name for the project is CryptoStats

2.2 Overview of the Project

The aim of the project is to provide users with relevant information regarding cryptocurrency exchanges. By analysing the data provided by the external API on the values of the cryptocurrencys, the service will provide the user with information about the patterns in the value of the cryptocurrency and data such as the average number of days that the currency's value declines and increases in one go, if there is any resemblance in the fluctuation betweeen the currency and other major currencies.

The goal with the project is to be able to provide users with enough information to help them make informed decisions when doing an exchange for a particular coin, and also gather information about what makes cryptocurrenncies fluctuate in value. Rather than just showing the value of the coin, the service aims to provide way more valuable information by analysing the data in the currency's history, which allows the user to posess very important and essential information when trading coins.

The information processed will be presented to the users in two different ways: The first in a web site that will display the data in a user-friendly manner, built using ASP.NET Core 2 for the back-end and Angular and Bootstrap for the front-end. The second is by presenting the processed information to the user through a RESTful API with a JSON response, in order to facilitate the integration of the information with other applications or services.

For the submission of the project, I only intend to provide information for major coins

such as Bitcoin, Ethereum, Litecoin and Bitcoin Cash, since they are the most known and there is many information and exchanges that trade these coins.

2.3 External Web Services

In order to obtain information about the coins, I will be using the GDAX exchange API.

2.4 Data Exposed By The Service

The service aims to expose the following data:

- Average fluctuation for the last day, week, month, and year
- Average time when in growth
- Average time when in decline
- Difference between highest value and latest value

2.5 Database Schema

2.6 User Interface Ideas

The main concept for the User Interface of the web page is for it to be displayed on a single page, and using Angular components allow the interface to be responsive and show the right information that the user is looking for without having to load the page again or another page.

The first sketches for the interface are the following:

The Web application will have some additional pages explaining the project and how it came about, as well as a form allowing users to leave messages and an API page explaining how to work with the API.

The following images are wireframes that were produced based on the sketches:

Figure 2.1: Database Schema

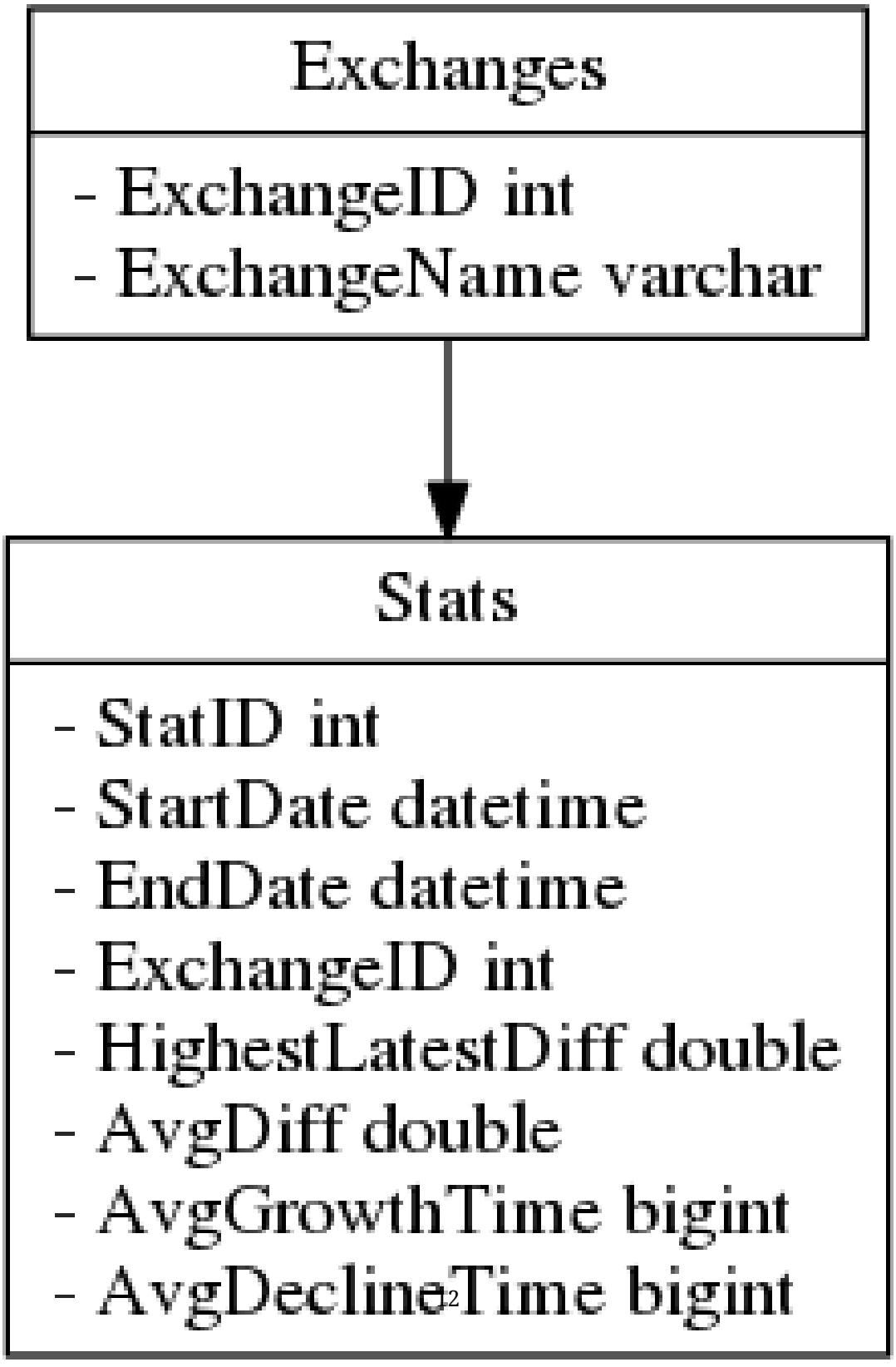

Figure 2.2: 1st Sketches

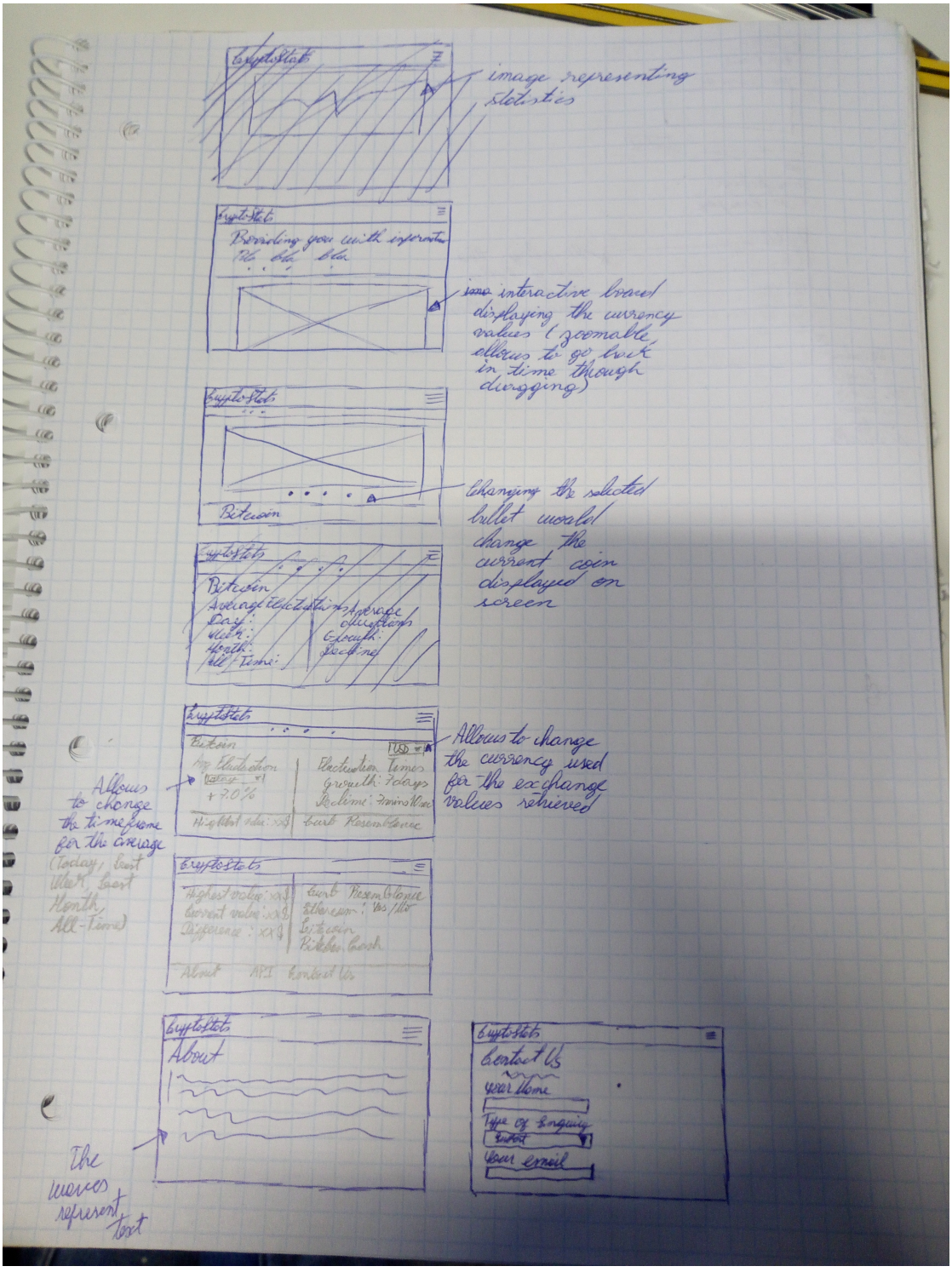

Figure 2.3: 2nd Sketches

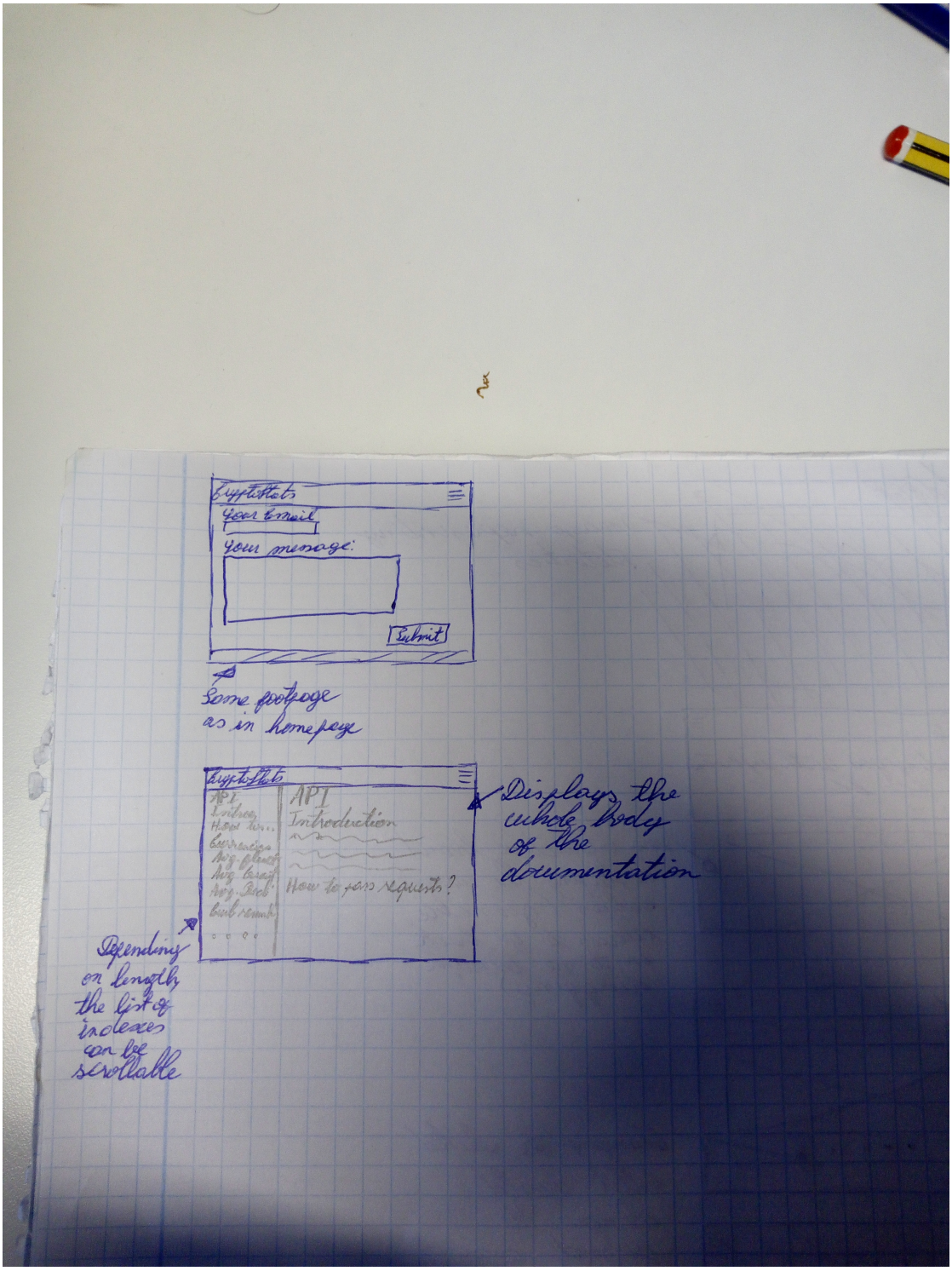

Figure 2.4: Homepage

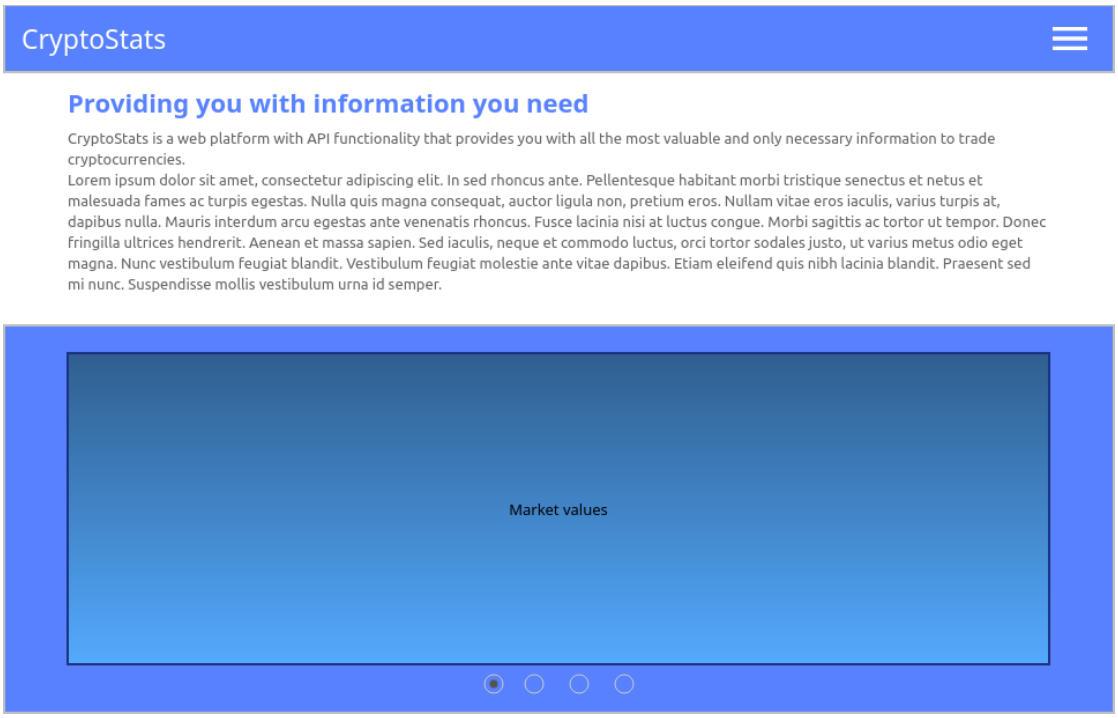

Figure 2.5: Homepage Bottom

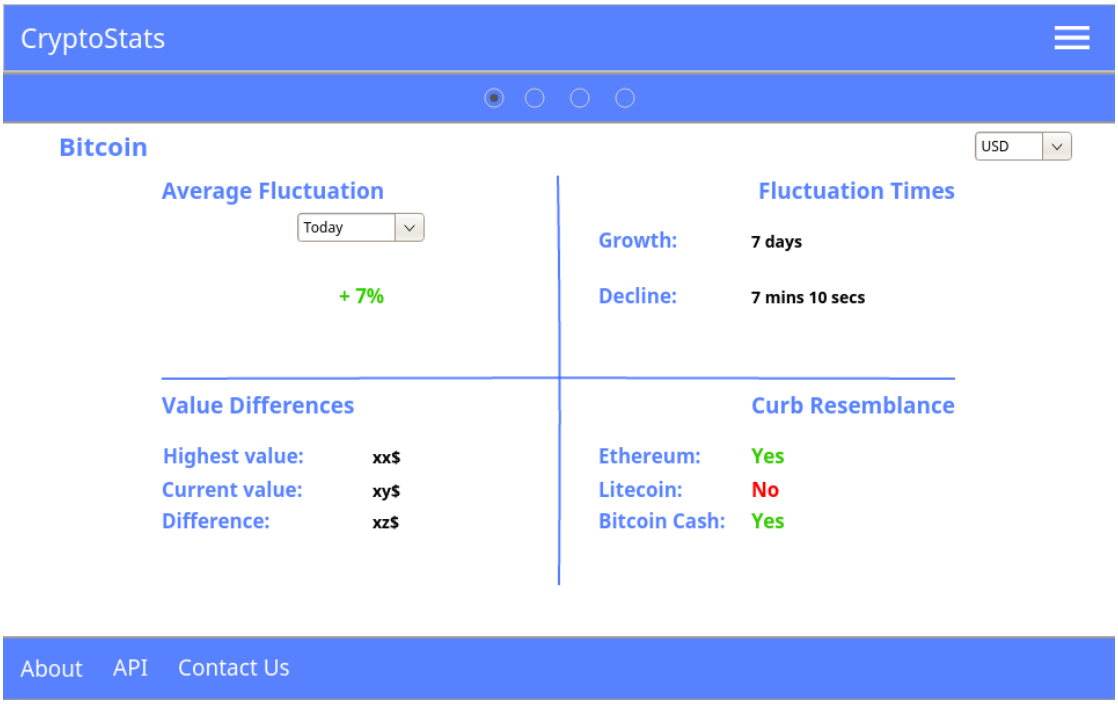

Figure 2.6: About Page

Figure 2.7: Contact Us Page

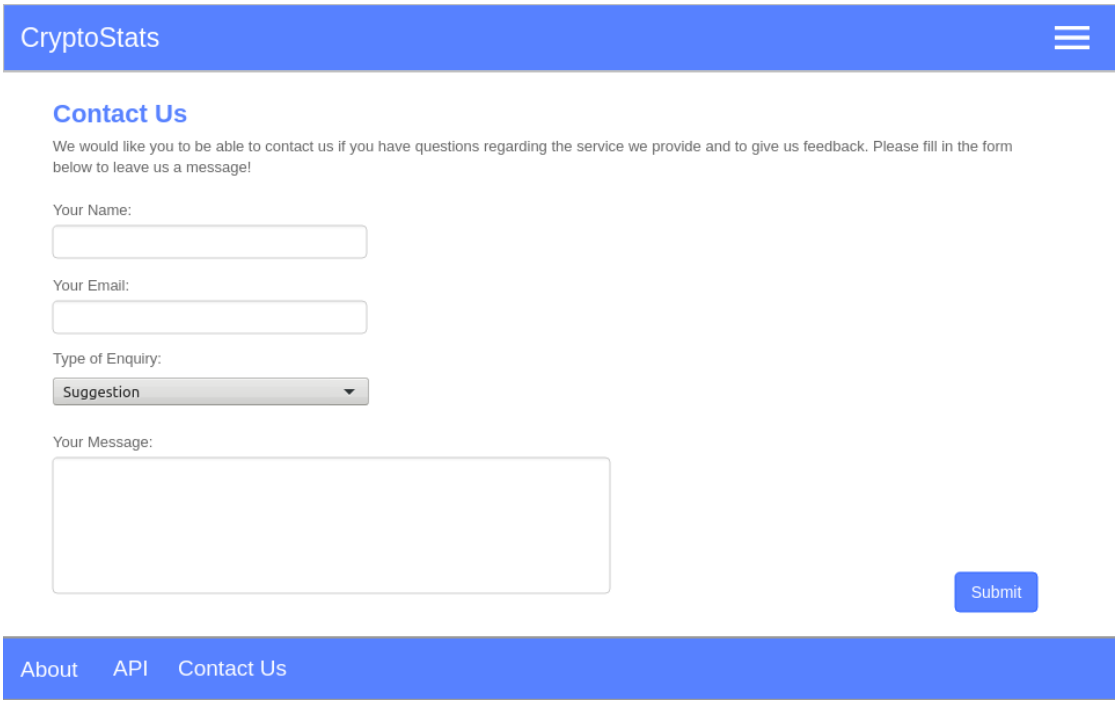

Figure 2.8: API Page

Section 3

Engineering Approach

For this project, I worked in an agile and iterative manner, based on the SCRUM methodology.

3.1 Requirements specification

When specifying the requirements for the project, I started by choosing what I wanted the project to offer to the users initially. I started by thinking of the core functionality that I wanted for the project and what data I wanted my API to expose to the user. After deciding on the data that would be exposed, I started specifying the requirements that would lead to the exposure of the data, by doing a list of the core functionality that the API would have:

- Display all the exchanges and statistics
- Display exchanges and statistics by id
- Display statistics by start date
- Display statistics by end date
- Display statistics by start and end date
- Display statistics related to a single exchange
- Display statistics related to a single exchange by id
- Display statistics related to a single exchange by start date
- Display statistics related to a single exchange by end date
- Display statistics related to a single exchange by start and end date

Most of this funcionality would allow the user to retrieve different series of data by the criteria that they choose (id and or range of dates) and the type of data that they are looking for (exchange or statistics).

3.2 Designing

I started designing the implementation of the features on paper, by writing down the parts of the algorithms that would implement the features that were the most difficult to implement.

In order to implement the features, I chose to go with a Model-Controller approach, where the data would be initially stored and retrieved through the models and the business logic and presentation of the data would occur in the controller. The business logic that would allow the API to seed the database and add data to it would also work in the models.

3.3 Implementation

For the implementation, I started by implementing the models and the database context that would allow the connection to the database, storage and retrieval of the data.

After that, I implemented the controllers and added the functionality that would display the JSON results to the user.

After all of that was done, I moved on to implement the features that would retrieve data from the input API and process it. For that, I studied and modified a library client that was available for the API I was using, so that only the functions that I needed for my project would be available, and created new namespaces for those functionalities.

After implementing the seeding and storage of new data functinoalities, I started implementing the front-end of my project. Based on my previous UI design and wireframes, I used the planned technologies to build a front-end that would resemble the original design as much as possible.

3.4 Testing

This project was throughly tested as it was being built. As soon as a new feature would be implemented, the project would be tested to make sure that everything was working as expected and that everything was also behaving and looking in an expected way.

Section 4

Project Management

In order to manage the project, the code as well as the project tasks were stored in a GitHub repository.

That choice was made because I already had previous experience storing my code and repositories in GitHub but also because the way GitHub does project management was very suitable for the approach that I was using to develop the project.

Using SCRUM terms, GitHub facilitates the planning of sprints and tasks by its use of issues and milestones, but also because it makes it very easy to keep track of how the project is evolving through the use of Kanban boards. It is also possible to assign the different tasks to different sprints and to users, to that everyone that is involved in the project knows where the project is and what they need to do.

In this particular case, I developed the whole project myself, but I still found it very useful to use the features provided by GitHub, especially because it allows me to comment on the tasks as I am getting them done, allowing me to have a log of the progress that I have made over time, as well as register the difficulties that I had to face and how I overcame them.

The project code and tasks can be found here.

Section 5

Development

5.1 Requirements elicitation

5.1.1 API

- Display all stored exchanges and statistics
- Display exchanges with a specific id
- Display statistics with a specific id
- Display the statistics starting at a specific date
- Display the statistics ending at a specific date
- Display the statistics with a specific start and end dates
- Fetch all the statistics related to a specific exchange
- Fetch the statistic related to a particular exchange with a specific id
- Fetch all the statistics related to a specific exchange with a specific start date
- Fetch all the statistics related to a specific exchange with a particular end date
- Fetch a statistic related to a particular exchange with specific start and end dates

5.1.2 Front-End

- Display for a particular exchange:
  - Average fluctuation
  - Average time in growth

  - Average time in decline
  - Difference between highest and latest values
- Allow display of information for a period of the last day, week, month and year.

5.2 Design

5.3 Implementation

Here is the code for the API:

5.3.1 Main Program

Figure 5.1: Program.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;
using CryptoStats.Models;

namespace CryptoStats
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = BuildWebHost(args);

            using (var scope = host.Services.CreateScope())
            {
                var services = scope.ServiceProvider;

                try
                {
                    SeedData.Initialize(services);
```

```
                }
                catch (Exception ex)
                {
                    var logger = services.GetRequiredService<ILogger<Program>>();
                    logger.LogError(ex, "An error occurred seeding the DB.");
                    Console.Write(ex.Message);
                }
            }

            host.Run();
        }

        public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .Build();
    }
}
```

Figure 5.2: Startup.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Sqlite;
using CryptoStats.Models;

namespace CryptoStats
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }
```

```
24          public IConfiguration Configuration { get; }

25          // This method gets called by the runtime. Use this method to add services to
26          public void ConfigureServices(IServiceCollection services)
27          {
28              services.AddDbContext<CryptoContext>(opt => opt.UseSqlite("Data Source=ap
29              services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_
30          }

31          // This method gets called by the runtime. Use this method to configure the H
32          public void Configure(IApplicationBuilder app, IHostingEnvironment env)
33          {
34              if (env.IsDevelopment())
35              {
36                  app.UseDeveloperExceptionPage();
37              }
38              else
39              {
40                  app.UseHsts();
41              }

42              app.UseHttpsRedirection();
43              app.UseMvc();
44          }
45      }
46  }
```

Figure 5.3: appsettings.json

```
1  {
2    "ConnectionStrings":{
3      "DefaultConnection": "DataSource=app.db"
4    },
5    "Logging": {
6      "LogLevel": {
7        "Default": "Warning"
8      }
9    },
10   "AllowedHosts": "*"
11 }
```

5.3.2 Models

Figure 5.4: Exchange.cs

```
using System.Collections.Generic;

namespace CryptoStats.Models
{
    public class Exchange
    {
        public int ExchangeId{get;set;}
        public string Name{get;set;}

        //Setting list for foreign key
        public List<Stat> Stats{get;set;}
    }
}
```

Figure 5.5: Stat.cs

```
using System;

namespace CryptoStats.Models
{
    public class Stat
    {
        public int StatId{get;set;}
        public DateTime startDate {get;set;}
        public DateTime endDate{get;set;}
        public decimal HighestLatestDiff{get;set;}
        public decimal avgDiff {get;set;}
        public TimeSpan avgGrowthTime {get;set;}
        public TimeSpan avgDeclineTime {get;set;}

        //Setting variables for foreign key
        public int ExchangeId{get;set;}
        public Exchange Exchange {get;set;}
    }
}
```

Figure 5.6: SeedData.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using System;
using System.Linq;
using System.Collections.Generic;
using CryptoStats.GDAX;
using CryptoStats.GDAX.Services.Products.Models;
```

```
8  using CryptoStats.GDAX.Services.Products.Types;
9  using CryptoStats.GDAX.Shared.Types;

10 namespace CryptoStats.Models
11 {
12     public static class SeedData
13     {
14         private static GDAXClient client = new GDAXClient();
15         public static void Initialize(IServiceProvider serviceProvider)
16         {
17             using(var context = new CryptoContext(serviceProvider.GetRequiredService<
18             {
19                 if(context.Exchanges.Any() && context.Stats.Any())
20                 {
21                     return;
22                 }

23                 if(context.Exchanges.Any() == false)
24                 {
25                     var products = client.ProductsService.GetAllProductsAsync().Resul
26                     foreach(Product prod in products)
27                     {
28                         Console.Write(prod.Id.ToString());
29                         context.Exchanges.Add(new Exchange(){Name = prod.Id.ToString(
30                     }
31                     context.SaveChanges();
32                 }

33                 if(!context.Stats.Any())
34                 {
35                     foreach(Exchange e in context.Exchanges)
36                     {
37                         seedStat(e.ExchangeId, DateTime.Now.AddDays(-7), DateTime.Now
38                     }

39                 }
40             }
41         }

42         public static void seedStat(int exchangeID, DateTime startDate, DateTime endD
43         {
44             using(context)
45             {
46                 foreach(Product p in client.ProductsService.GetAllProductsAsync().Res
47                 {
```

```
48                  System.Threading.Thread.Sleep(3000);
49                  if(p.Id.ToString().Equals(context.Exchanges.Find(exchangeID).Name
50                  {
51                      TimeSpan t = endDate.Subtract(startDate);
52                      int days = (int)t.TotalDays;
53                          for(int i = 0; i < days; i++)
54                              {
55                                  System.Threading.Thread.Sleep(3000);
56                                  var candles = client.ProductsService.GetHistoricR
57                                  System.Threading.Thread.Sleep(1000);
58                                  Stat adder = new Stat(){
59                                      startDate = candles.First().Time,
60                                      endDate = candles.Last().Time
61                                  };

62                                  //calculate highestLatestDiff
63                                  decimal high = 0.00M;
64                                  foreach(Candle c in candles)
65                                  {
66                                      if(c.High > high)
67                                      {
68                                          high = (decimal)c.High;
69                                      }
70                                  }
71                                  adder.HighestLatestDiff = (decimal)candles.Last()

72                                  //calculate avgDiff
73                                  //Use difference between open and close of candle
74                                  decimal candlesDiff = 0.00M;
75                                  foreach(Candle c in  candles)
76                                  {
77                                      candlesDiff += Convert.ToDecimal(c.Close - c.
78                                  }
79                                  adder.avgDiff = candlesDiff / candles.Count();

80                                  //calculate average growth and decline times
81                                  //store candles in a temporary list to retrieve t
82                                  //growth and decline times are stored in a list o
83                                  //average is calculated afterwards
84                                  List<TimeSpan> growthTimes = new List<TimeSpan>()
85                                  List<TimeSpan> declineTimes = new List<TimeSpan>(
86                                  bool? inGrowth = null;
87                                  List<Candle> growthCandles = new List<Candle>();

88                                  foreach(Candle c in candles)
```

```
{
    if(c.Close > c.Open && inGrowth == false)
    {
        inGrowth = true;
        declineTimes.Add(growthCandles.Last().Tim
        growthCandles.Clear();
        growthCandles.Add(c);
    }
    else if((c.Close > c.Open && inGrowth == true
    {
        growthCandles.Add(c);
        inGrowth = true;
    }
    else if(c.Close < c.Open && inGrowth == true)
    {
        inGrowth = false;
        growthTimes.Add(growthCandles.Last().Time
        growthCandles.Clear();
        growthCandles.Add(c);
    }
    else if((c.Close < c.Open && inGrowth == fals
    {
        growthCandles.Add(c);
        inGrowth = false;
    }
}
if(inGrowth == true)
{
    growthTimes.Add(growthCandles.Last().Time.Sub
}
else if (inGrowth == false)
{
    declineTimes.Add(growthCandles.Last().Time.Su
}

double totalTimes = 0;
foreach(TimeSpan time in growthTimes)
{
    totalTimes += t.TotalMilliseconds;
}
adder.avgGrowthTime = TimeSpan.FromMilliseconds(t

totalTimes = 0;
foreach(TimeSpan time in declineTimes)
{
```

```
            totalTimes += t.TotalMilliseconds;
        }
        adder.avgDeclineTime = TimeSpan.FromMilliseconds(
        context.Stats.Add(adder);
        startDate.AddDays(1);
    }

    startDate.AddDays(-days);
    for(int i = 0; i < days; i++)
    {
        System.Threading.Thread.Sleep(1000);
        var candles = client.ProductsService.GetHistoricR
        System.Threading.Thread.Sleep(1000);
        Stat adder = new Stat(){
            startDate = candles.First().Time,
            endDate = candles.Last().Time
        };

        //calculate highestLatestDiff
        decimal high = 0.00M;
        foreach(Candle c in candles)
        {
            if(c.High > high)
            {
                high = (decimal)c.High;
            }
        }
        adder.HighestLatestDiff = (decimal)candles.Last()

        //calculate avgDiff
        //Use difference between open and close of candle
        decimal candlesDiff = 0.00M;
        foreach(Candle c in  candles)
        {
            candlesDiff += (decimal)(c.Close - c.Open);
        }
        adder.avgDiff = candlesDiff / candles.Count();

        //calculate average growth and decline times
        //store candles in a temporary list to retrieve t
        //growth and decline times are stored in a list c
        //average is calculated afterwards
        List<TimeSpan> growthTimes = new List<TimeSpan>()
        List<TimeSpan> declineTimes = new List<TimeSpan>(
        bool inGrowth = false;
```

```
173             List<Candle> growthCandles = new List<Candle>();

174             foreach(Candle c in candles)
175             {
176                 if(c.Close > c.Open && inGrowth == false)
177                 {
178                     inGrowth = true;
179                     declineTimes.Add(growthCandles.Last().Tim
180                     growthCandles.Clear();
181                     growthCandles.Add(c);
182                 }
183                 else if(c.Close > c.Open && inGrowth == true)
184                 {
185                     growthCandles.Add(c);
186                 }
187                 else if(c.Close < c.Open && inGrowth == true)
188                 {
189                     inGrowth = false;
190                     growthTimes.Add(growthCandles.Last().Time
191                     growthCandles.Clear();
192                     growthCandles.Add(c);
193                 }
194                 else if(c.Close < c.Open && inGrowth == false
195                 {
196                     growthCandles.Add(c);
197                 }
198             }
199             if(inGrowth)
200             {
201                 growthTimes.Add(growthCandles.Last().Time.Sub
202             }
203             else
204             {
205                 declineTimes.Add(growthCandles.Last().Time.Su
206             }

207             double totalTimes = 0;
208             foreach(TimeSpan time in growthTimes)
209             {
210                 totalTimes += t.TotalMilliseconds;
211             }
212             adder.avgGrowthTime = TimeSpan.FromMilliseconds(t

213             totalTimes = 0;
214             foreach(TimeSpan time in declineTimes)
```

```
                                        {
                                            totalTimes += t.TotalMilliseconds;
                                        }
                                        adder.avgDeclineTime = TimeSpan.FromMilliseconds(
                                        context.Stats.Add(adder);
                                        endDate.AddDays(-1);
                                    }
                                    context.SaveChanges();
                                }
                            }
                        }
                    }
                }
            }
```

Figure 5.7: DbContext.cs

```
using Microsoft.EntityFrameworkCore;

namespace CryptoStats.Models
{
    public class CryptoContext: DbContext
    {
        public CryptoContext(DbContextOptions<CryptoContext> options) : base(options)
        {
        }

        public DbSet<Exchange> Exchanges{get;set;}
        public DbSet<Stat> Stats{get;set;}
    }
}
```

5.3.3 Controllers

Figure 5.8: ExchangesController.cs

```
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Linq;
using System;
using CryptoStats.Models;

namespace CryptoStats.Controllers
{
    [Route("api/[controller]")]
```

```
    [ApiController]
    public class ExchangesController : ControllerBase
    {
        private readonly CryptoContext _context;

        public ExchangesController(CryptoContext context)
        {
            _context = context;
        }

        [HttpGet]
        public ActionResult<List<Exchange>> GetAll()
        {
            return _context.Exchanges.ToList();
        }

        [HttpGet("{id}", Name="GetExchange")]
        public ActionResult<Exchange> GetById(int id)
        {
            var item = _context.Exchanges.Find(id);
            if(item == null)
            {
                return NotFound();
            }
            return item;
        }
    }
}
```

Figure 5.9: ExchangeStatsController.cs

```
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Linq;
using System;
using CryptoStats.Models;

namespace CryptoStats.Controllers
{
    [Route("api/")]
    [ApiController]
    public class ExchangeStatsController : ControllerBase
    {
        private readonly CryptoContext _context;
```

```
13          public ExchangeStatsController(CryptoContext context)
14          {
15              _context = context;
16          }

17          [HttpGet("exchanges/{id}/stats")]
18          public ActionResult<List<Stat>> GetAll(int id)
19          {
20              var exchange = _context.Exchanges.Where(e => e.ExchangeId == id).First();
21              if(exchange == null)
22              {
23                  return NotFound();
24              }
25              else
26              {
27                  return exchange.Stats;
28              }
29          }

30          [HttpGet("exchanges/{id}/stats/id/{statId}")]
31          public ActionResult<Stat> GetByID(int id, int statId)
32          {
33              var exchange = _context.Exchanges.Where(e => e.ExchangeId == id).First();
34              if(exchange == null)
35              {
36                  return NotFound();
37              }
38              else
39              {
40                  return exchange.Stats.Where(s => s.StatId == statId).First();
41              }
42          }

43          [HttpGet("exchanges/{id}/stats/startDate/{startDate}")]
44          public ActionResult<List<Stat>> GetByStartDate(int id, DateTime startDate)
45          {
46              var exchange = _context.Exchanges.Where(e => e.ExchangeId == id).First();
47              if(exchange == null)
48              {
49                  return NotFound();
50              }
51              else
52              {
53                  return exchange.Stats.Where(s => s.startDate.Equals(startDate)).ToLis
54              }
```

```
55          }

56          [HttpGet("exchanges/{id}/stats/endDate/{endDate}")]
57          public ActionResult<List<Stat>> GetByEndDate(int id, DateTime endDate)
58          {
59              var exchange = _context.Exchanges.Where(e => e.ExchangeId== id).First();
60              if(exchange == null)
61              {
62                  return NotFound();
63              }
64              else
65              {
66                  return exchange.Stats.Where(s => s.endDate.Equals(endDate)).ToList();
67              }
68          }

69          [HttpGet("exchanges/{id}/stats/dates/{startDate}/{endDate}")]
70          public ActionResult<Stat> GetByDates(int id, DateTime startDate, DateTime enc
71          {
72              var exchange = _context.Exchanges.Where(e => e.ExchangeId == id).First();
73              if(exchange == null)
74              {
75                  return NotFound();
76              }
77              else
78              {
79                  return exchange.Stats.Where(s => s.startDate.Equals(startDate) && s.e
80              }
81          }
82      }
83  }
```

Figure 5.10: StatsController.cs

```
1   using Microsoft.AspNetCore.Mvc;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System;
5   using CryptoStats.Models;

6   namespace CryptoStats.Controllers
7   {
8       [Route("api/[controller]")]
9       [ApiController]
10      public class StatsController : ControllerBase
```

```
11      {
12          private readonly CryptoContext _context;

13          public StatsController(CryptoContext context)
14          {
15              _context = context;
16          }

17          [HttpGet]
18          public ActionResult<List<Stat>> GetAll()
19          {
20              return _context.Stats.ToList();
21          }

22          [HttpGet("/id/{id}")]
23          public ActionResult<Stat> GetById(int id)
24          {
25              var item = _context.Stats.Find(id);
26              if(item == null)
27              {
28                  return NotFound();
29              }
30              return item;
31          }

32          [HttpGet("/startDate/{startDate}")]
33          public ActionResult<List<Stat>> GetByStartDate(DateTime startDate)
34          {
35              var items = _context.Stats.Where(s => s.startDate.Equals(startDate)).ToLi
36              if(items == null)
37              {
38                  return NotFound();
39              }
40              return items;
41          }

42          [HttpGet("/endDate/{endDate}")]
43          public ActionResult<List<Stat>> GetByEndDate(DateTime endDate)
44          {
45              var items = _context.Stats.Where(s => s.endDate.Equals(endDate)).ToList()
46              if(items == null)
47              {
48                  return NotFound();
49              }
50              return items;
```

```
51          }

52          [HttpGet("/dates/{startDate}/{endDate}")]
53          public ActionResult<Stat> GetByDates(DateTime startDate, DateTime endDate)
54          {
55              var item = _context.Stats.Where(s => s.startDate.Equals(startDate) && s.e
56              if(item == null)
57              {
58                  return NotFound();
59              }
60              return item;
61          }
62      }
63  }
```

5.3.4 GDAX Client

Figure 5.11: GDAXClient.cs

```
1   using System;
2   using CryptoStats.GDAX.Network.HttpClient;
3   using CryptoStats.GDAX.Network.HttpRequest;
4   using CryptoStats.GDAX.Services.Products;
5   using CryptoStats.GDAX.Shared.Utilities.Clock;
6   using CryptoStats.GDAX.Shared.Utilities.Queries;

7   namespace CryptoStats.GDAX
8   {
9       public class GDAXClient
10      {
11          public GDAXClient(bool sandBox = false) : this( new HttpClient(), sandBox){}

12          public GDAXClient(IHttpClient httpClient, bool sandBox = false)
13          {
14              var clock = new Clock();
15              var httpRequestMessageService = new HttpRequestMessageService(clock, sand
16              var queryBuilder = new QueryBuilder();

17              ProductsService = new ProductsService(httpClient, httpRequestMessageServi
18          }

19          public ProductsService ProductsService{get;}
20      }
21  }
```

Figure 5.12: ProductsService.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;
using CryptoStats.GDAX.Services;
using CryptoStats.GDAX.Network.HttpClient;
using CryptoStats.GDAX.Network.HttpRequest;
using CryptoStats.GDAX.Services.Products.Models;
using CryptoStats.GDAX.Services.Products.Models.Responses;
using CryptoStats.GDAX.Services.Products.Types;
using CryptoStats.GDAX.Shared.Types;
using CryptoStats.GDAX.Shared.Utilities.Extensions;
using CryptoStats.GDAX.Shared.Utilities.Queries;

namespace CryptoStats.GDAX.Services.Products
{
    public class ProductsService: AbstractService
    {
        private readonly IQueryBuilder queryBuilder;

        public ProductsService(IHttpClient httpClient, IHttpRequestMessageService htt
        {
            this.queryBuilder = queryBuilder;
        }

        public async Task<List<Product>> GetAllProductsAsync()
        {
            return await SendServiceCall<List<Product>>(HttpMethod.Get, "/products");
        }

        public async Task<ProductsOrderBookResponse> GetProductOrderBookAsync(
                        ProductType productId,
                        ProductLevel productLevel = ProductLevel.One)
        {
            var productsOrderBookJsonResponse = await SendServiceCall<ProductsOrderBo
            var productOrderBookResponse = ConvertProductOrderBookResponse(productsOr

            return productOrderBookResponse;
        }

        public async Task<ProductTicker> GetProductTickerAsync(ProductType productId)
```

```
38          {
39              return await SendServiceCall<ProductTicker>(HttpMethod.Get, $"/products/{
40          }
41          public async Task<ProductStats> GetProductStatsAsync(ProductType productId)
42          {
43              return await SendServiceCall<ProductStats>(HttpMethod.Get, $"/products/{p
44          }
45          public async Task<IList<IList<ProductTrade>>> GetTradesAsync(
46              ProductType productId,
47              int limit = 100,
48              int numberOfPages = 0)
49          {
50              var httpResponseMessage = await SendHttpRequestMessagePagedAsync<ProductT
51              return httpResponseMessage;
52          }
53          public async Task<IList<Candle>> GetHistoricRatesAsync(
54                          ProductType productPair,
55                          DateTime start,
56                          DateTime end,
57                          CandleGranularity granularity)
58          {
59              const int maxPeriods = 300;
60              var candleList = new List<Candle>();
61              DateTime? batchEnd = end;
62              DateTime batchStart;
63              var maxBatchPeriod = (int)granularity * maxPeriods;
64              do
65              {
66                  if (batchEnd == null) {
67                      break;
68                  }
69                  batchStart = batchEnd.Value.AddSeconds(-maxBatchPeriod);
70                  if (batchStart < start) batchStart = start;
71                  candleList.AddRange(await GetHistoricRatesAsync(productPair, batchSta
```

```
72                  batchEnd = candleList.LastOrDefault()?.Time;
73              } while (batchStart > start);
74              return candleList;
75          }
76          private async Task<IList<Candle>> GetHistoricRatesAsync(
77                          ProductType productId,
78                          DateTime start,
79                          DateTime end,
80                          int granularity)
81          {
82              var isoStart = start.ToString("s");
83              var isoEnd = end.ToString("s");
84              var queryString = queryBuilder.BuildQuery(
85                  new KeyValuePair<string, string>("start", isoStart),
86                  new KeyValuePair<string, string>("end", isoEnd),
87                  new KeyValuePair<string, string>("granularity", granularity.ToString(
88              return await SendServiceCall<IList<Candle>>(HttpMethod.Get, $"/products/{
89          }
90          private ProductsOrderBookResponse ConvertProductOrderBookResponse(
91              ProductsOrderBookJsonResponse productsOrderBookJsonResponse,
92              ProductLevel productLevel)
93          {
94              var askList = productsOrderBookJsonResponse.Asks.Select(product => produc
95              {
96                  OrderId = productLevel == ProductLevel.Three
97                      ? new Guid(askArray[2])
98                      : (Guid?)null,
99                  NumberOfOrders = productLevel == ProductLevel.Three
100                     ? (decimal?)null
101                     : Convert.ToDecimal(askArray[2], CultureInfo.InvariantCulture)
102             }).ToArray();
103             var bidList = productsOrderBookJsonResponse.Bids.Select(product => produc
104             {
105                 OrderId = productLevel == ProductLevel.Three
106                     ? new Guid(bidArray[2])
107                     : (Guid?)null,
108                 NumberOfOrders = productLevel == ProductLevel.Three
109                     ? (decimal?)null
```

```
110                         : Convert.ToDecimal(bidArray[2], CultureInfo.InvariantCulture)
111                 });

112                 var productOrderBookResponse = new ProductsOrderBookResponse(productsOrde
113                 return productOrderBookResponse;
114             }
115         }
116     }
```

For the GDAX Client, an instance of HttpClient, and JSON deserializers have been used to send HTTP requests to the input API and deserialize the data sent back from the request. The output API was built using REST technology, and sends JSON responses that the system automatically serializes from the ActionResults that the controllers return.

5.4 Testing

Testing as been done alongside the build of the project, as features were being implemented to make sure that everything was being working as it was supposed to.
Testing as been done on different browsers (Google Chrome, Chromium, Firefox) and also using Sqliteman to make sure that the database was being populated properly.
For debugging, when the API wouldn't throw an exception to the browser, but the application wouldn't behave as expected, then I would use the debugger from Visual Studio Code for C# and put breakpoints in the most critical and error-prone parts of the code until I stumbled upon the exception or behaviour that would be making the application behave in an unexcepted manner, to then later on fix it.

References