MSc - Önálló laboratórium 2

Amazon Web Services

Daniel Mark Kiss

2023

Contents

Chapter 1 **Introduction** Page 2

Chapter 2 **Current techologies** Page 3

2.1 Amazon Simple Storage Service (S3) 3

2.2 Amazon RDS (Relational Database Service) 3

2.3 Amazon DynamoDB 3

2.4 AWS Lambda 3

2.5 AWS CodeCommit 3

Chapter 3 **Development** Page 5

3.1 Overview 5

3.2 APILayer Rest API 5

3.3 SwiftUI 5

3.4 ARKit and RealityKit 7

Chapter 4 **Presentation of finished work** Page 10

Chapter 5 **Sources** Page 16

Chapter 1

Introduction

Amazon Web Services (AWS) is a comprehensive cloud computing platform provided by Amazon. It includes a mixture of infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS) and packaged-software-as-a-service (SaaS). It offers a vast array of cloud services, including computing power, storage, databases, machine learning, analytics, and more. AWS allows individuals and organizations to access and utilize computing resources over the internet, without the need to own or maintain physical servers and infrastructure. Amazon launched its first web services in 2002 from the internal infrastructure that Amazon.com built to handle its online retail operations. In 2006, it began offering its defining IaaS services. AWS was one of the first companies to introduce a pay-as-you-go cloud computing model that scales to provide users with compute, storage or throughput as needed. AWS is separated into different services; each can be configured in different ways based on the user's needs. Users can see configuration options and individual server maps for an AWS service. AWS provides a scalable and flexible environment for businesses to build, deploy, and manage applications and services. It offers a pay-as-you-go pricing model, which means users only pay for the resources they consume, making it cost-effective and efficient for a wide range of use cases. [1]

Chapter 2

Current techologies

Before I started the development of the application I have done a course provided by Amazon. It conatains a 14 module class where they teache the fundamentals of AWS. It was roughly 5 weeks for me to finish and complete all laboratory exercises and module closing quiz. Throughout the course I have familiarised myself with the different AWS services and theoretical background. It proved to be a good repetition for my previous knowledge like REST API.

2.1 Amazon Simple Storage Service (S3)

...

2.2 Amazon RDS (Relational Database Service)

...

2.3 Amazon DynamoDB

...

2.4 AWS Lambda

...

2.5 AWS CodeCommit

...

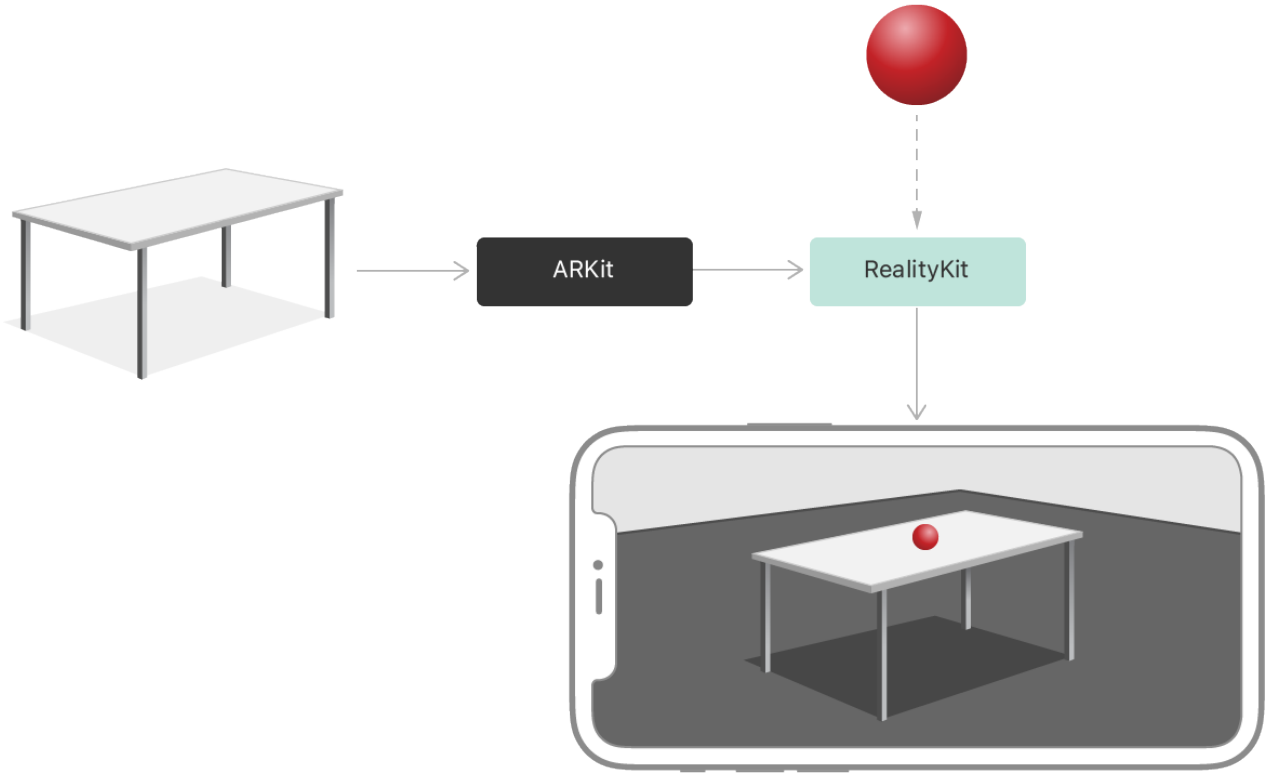

Figure 2.1: RealityKit and ARKit usage.

Chapter 3

Development

3.1 Overview

For the development phase of the project I used AWS's Cloud9 integrated developer environment(IDE) as the main developer platform for the front-end. For testing I used an iPhone 11 with dual camera system. I have also version controled the whole development process using git and publishing it on GitHub. Not only the source code can be found there but also the documentation of this project as I have writen it using LaTeX.

3.2 APILayer Rest API

The fundamental part of the application is the data it displays. For retriving the displayed informations I used APILayer's Exchange Rates Data API an open and available for free financial API. The only problem is that you can only do 250 queries per month in the free version. I used 2 endpoints. The first is '/convert'. With this endpoint, we have any amount conversion from one currency to another. The output of this enpoint is the following JSON.

The other endpoint used is '/fluctuation'. This endpoint returns the fluctuation data between specified dates. The data can be for all available currencies or for a specific set.

3.3 SwiftUI

SwiftUI is Apple's brand new framework for building user interfaces for iOS, tvOS, macOS, and watchOS. Apple introduced SwiftUI in 2019 and the framework has been evolving ever since. Unlike UIKit, SwiftUI is a cross-platform framework. The key difference with UIKit and AppKit is that SwiftUI defines the user interface declaratively, not imperatively. What does that mean?

Using UIKit you create views to build the view hierarchy of your application's user interface. That is not how SwiftUI works. SwiftUI provides developers with an API to declare or describe

Listing 3.1: JSON from /convert endpoint

```
{
  "success": true,
  "query": {
      "from": "EUR",
      "to": "HUF",
      "amount": 1
  },
  "info": {
      "timestamp": 1682930463,
      "rate": 373.180303
  },
  "date": "2023-05-01",
  "result": 373.180303
}
```

what the user interface should look like. SwiftUI inspects the declaration or description of the user interface and converts it to your application's user interface. SwiftUI does the heavy lifting for you.

One of the most challenging aspects of user interface development is synchronizing the application's state and its user interface. Every time the application's state changes, the user interface needs to update to reflect the change. During the development phase, this was a challenge that had to be overcome. Despite the fact that I have already used and developed an iOS application with SwiftUI, it was excellent practice to deepen my knowledge of user state management. I used ObservableObjects to solve this problem.

I used a common state management technique, the MVC pattern, to control the data and model. MVC (Model-View-Controller) is a pattern in software design commonly used to implement user interfaces, data, and controlling logic. It emphasizes a separation between the software's business logic and display. This "separation of concerns" provides for a better division of labor and improved maintenance. Sticking to convention, I created a CurrencyController, CurrencyView and a CurrencyModell class. The CurrencyModel class contains the generated 3D models and their associated values. The task of the CurrencyController class is to query the data and update the information displayed on the View. In the CurrencyView class, it deals with the code defining the appearance of the application and the display of the given dataset.

Listing 3.2: JSON from /fluctuation endpoint


```
{
  "base": "EUR",
  "end_date": "2018-02-26",
  "fluctuation": true,
  "rates": {
    "JPY": {
      "change": 0.0635,
      "change_pct": 0.0483,
      "end_rate": 131.651142,
      "start_rate": 131.587611
    },
    "USD": {
      "change": 0.0038,
      "change_pct": 0.3078,
      "end_rate": 1.232735,
      "start_rate": 1.228952
    }
  },
  "start_date": "2018-02-25",
  "success": true
}
```


3.4 ARKit and RealityKit

To operate augmented reality and display the 3D generated graph, I used the ARKit and RealityKit frameworks provided by Apple.

The CurrencyARViewContainer is responsible for displaying the AR view.

```
struct CurrencyARViewContainer: UIViewRepresentable {

    @StateObject var controler:CurrencyController

    func makeUIView(context: Context) -> ARView{
        AR.view = ARView(frame: .zero)
        return AR.view
```

```
    }

    func updateUIView(_ uiView: ARView, context: Context) {
        print("updating view - \(controler.timerHappened)")
        uiView.scene.anchors.removeAll()
        ...
    }
}
```

To generate the texts and columns, I used the .generateBox() and .generateText() functions of the built-in MeshResource class. The MeshResource class stores the points defining the shapes. In order for this to become a 3D model, a texture must also be specified. I used the SimpleMaterial() function for this. We also need an AnchorEntity, which defines the center of our model in the 3D world. After defining these variables, we can create the ModelEntity and place it in the AR world using the AnchorEntity.

```
    func updateUIView(_ uiView: ARView, context: Context) {
        uiView.scene.anchors.removeAll()

        let cylinderMeshResource = MeshResource.generateBox(size: SIMD3(x: 1.2, y:

        let myMaterial = SimpleMaterial(color: .gray, roughness: 0, isMetallic: tr
        let radians = 90.0 * Float.pi / 180.0

        let kozeppont = AnchorEntity(world: SIMD3(x: 0.0, y: 0.0, z: 0.0))
        let axisXEntity = ModelEntity(mesh: cylinderMeshResource, materials: [myM

        let coneXEntity = ModelEntity(mesh: coneMeshResource, materials: [myMater
        coneXEntity.orientation = simd_quatf(angle: radians, axis: SIMD3(x: 0, y:

        axisXEntity.addChild(coneXEntity)
        coneXEntity.setPosition(SIMD3(x: 0.6, y: 0.0, z: 0.0), relativeTo: axisXE



        kozeppont.addChild(axisXEntity)
        uiView.scene.addAnchor(kozeppont)
        ...
```

}

To be able to move the different elements together, all 3D models are children of the axes. Thus, if the axis moves, the connected elements will also move due to the parent-child relationship. In its current version, MeshResource does not support the generation of cones by default, so I was able to achieve this by using an external library package. After importing the RealityGeometries library, I was able to easily generate cones, which I eventually used to draw axes.

Chapter 4

Presentation of finished work

In this section I will present and showcase my finished appliction. I will include screenshots to have a better representation and understanding for the reader. The camera function in the application allows you to capture the current state of the graphs, which in this case came in handy for documentation.

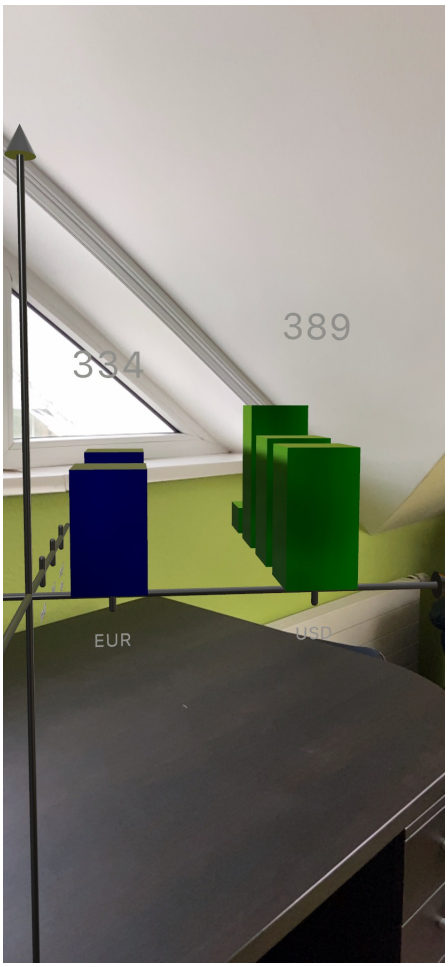

Figure 4.1: The main screen of the application.

As the attached images clearly show, the virtual 3D graph can be easily walked around and

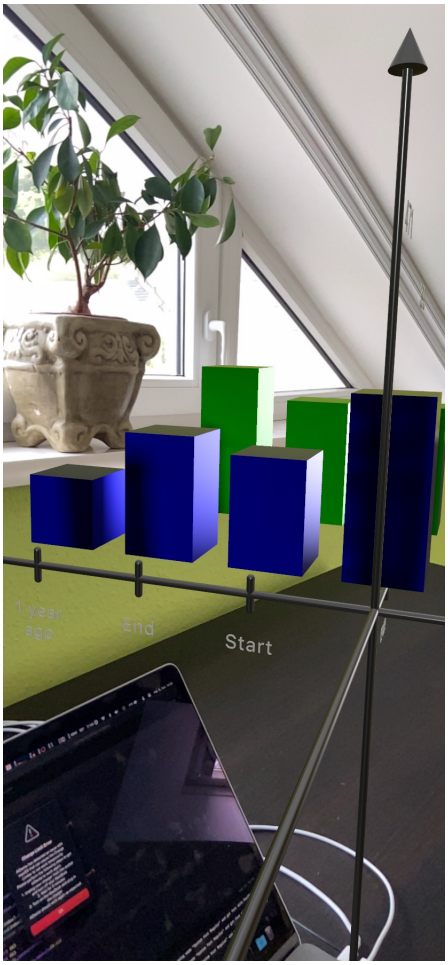

Figure 4.2: The main screen of the application.

viewed from different angles, thereby giving users a new comparative perspective. It is possible to move and rotate the entire graph, as well as move the current exchange rate columns to make it easier to compare with other metrics. The move and rotate functions are only available in Spectate mode, for this you have to stop Live mode (or otherwise start Live mode) by pressing the button located in the upper left corner.

Currently, 2 currencies are available in the application, but of course this can be easily expanded at any time in the future. These two are EUR to HUF and USD to HUF. These can be displayed after selection and confirmation from the bottom bar. If the given exchange rate is already placed, it can no longer be added to the graph twice.

Figure 4.3: The main screen of the application.

Figure 4.4: The main screen of the application.

Figure 4.5: The main screen of the application.

Figure 4.6: The main screen of the application.

Chapter 5

Sources

Below I list the external links and sources used during the project and the report.

Bibliography

[1] TechTarget - Amazon Web Services: https://www.techtarget.com/searchaws/definition/Amazon-Web-Services

[2] Leslie Lamport (1994) *LaTeX: a document preparation system*, Addison Wesley, Massachusetts, 2nd ed.

TechTarget - Amazon Web Services

Apple - Augmented Reality

Apple - RealityKit

Kodeco - RalityKit tutorials

Apple Forum - Where to start ARKit

Apple - Adding procedural assets to a scene

Medium - Adding 3D text to scene

BetterProgramming - Taking AR view snapshot

YouTube - Placing models

GitHub - FocusEntity

BetterProgramming - Update model entity

RapidAPI

Yahoo Financial API Guide

APILayer REST API

RealityGeometries - Kúpinsta

YouTube - RealitySchool: Place, Interact with, and Remove AR Objects in RealityKit

Cocoacast - SwiftUI

Hacking with Swift - ObservedObject

Mozilla - MVC