INSTITUT SUPERIEUR INFORMATIQUE ISI
المعهد العالي للإعلامية

UTM
جامعة تونس المنار
UNIVERSITÉ DE TUNIS EL MANAR

Internship Report

Name: Mohamed Farouk Hamadi
Studying for: A Licence in Computer Science
Location: Amen Bank
Supervisor: Mr. Oussama Ben Cheikh
Duration: 2 Months

بنك الأمان
AMEN BANK

Acknowledgement

First I would like to thank Mr. Oussama Ben Cheikh for giving me the opportunity to do an internship within Amen Bank.
Although quite short, for me this was a great experience I could learn from. It helped me to explore my skills and increased my interest in web development and making better software overall.
Special thanks to Mr. Oussama Ben Cheikh and Mr. Mohamed Amine Bacha for being so accommodating and understanding. I do appreciate their patience and help.

Abstract

The key to having successful and fully functional banking web applications is their accessibility, easy-to-use interfaces and security. It is of no surprise that the latter is a popular topic of research and constant development, as is the main goal at Amen Bank, where I had the honor to spend my first training period. A solution has been proposed for a web app that fills all those standards and uses the most advanced technologies to do so. Taking into account web standards and planning the application architecture beforehand, a solution has been put forward explained in detail in this report.

Contents

1. Introduction

As a full time student at ISI Ariana I have decided to get an internship this summer. The degree I'm currently studying is a Bachelor of Computer Science. The course at IT is quite challenging for me for a few reasons: Firstly, the programming languages and development tools I was using at my university were new for me, hence I had to quickly integrate not only with my new living environment but also academically. This was very beneficial for me as at the end I could compare what I've learnt with what I already knew and find a connection between the two. Since I find myself more interested in the area of web development and design, I was happy to be assigned the position of a trainee in web development in Amen Bank. This report is a description of my 2 month internship carried out as an optional component of the course at university. In the following chapter details of the activities I was given. Afterwards, I explain my role and tasks as a trainee and give specific technical details about my main tasks. Finally, a conclusion is drawn from the experience. You can find the code in my Github Repository.

1.1.Context of Project

The software development team at Amen Bank is responsible for several projects including a desktop cashier client for bank insiders and a banking website available for public users. The team there is responsible for planning out the architecture, developing the most optimal solution and implementing a full stack solution.

1.2. Context of Internship

I have been assigned the task to make a full stack cashier application and suggest a microservices solution that is secure, well architected, scalable and containerized. The provided solution had to be a cloud native application composed of a collection of small, independent and loosely coupled services. Security wise, I had to make authentication as secure as possible using the best hashing algorithms for every task without sacrificing speed by using cache databases(Redis).When it comes to architecture, I chose to go with a loosely coupled application revolving around small services, transactions were independent by making each one run on a server for a couple reasons such as if teams work on my app in the future, they only have to handle specific services rather than the app as a whole.Last but not least scalability and containerization are deeply related and dependent of each other, the app is fully containerized which means that it’s portable and it can run between different platforms and clouds.

1.3. Technical Aspect

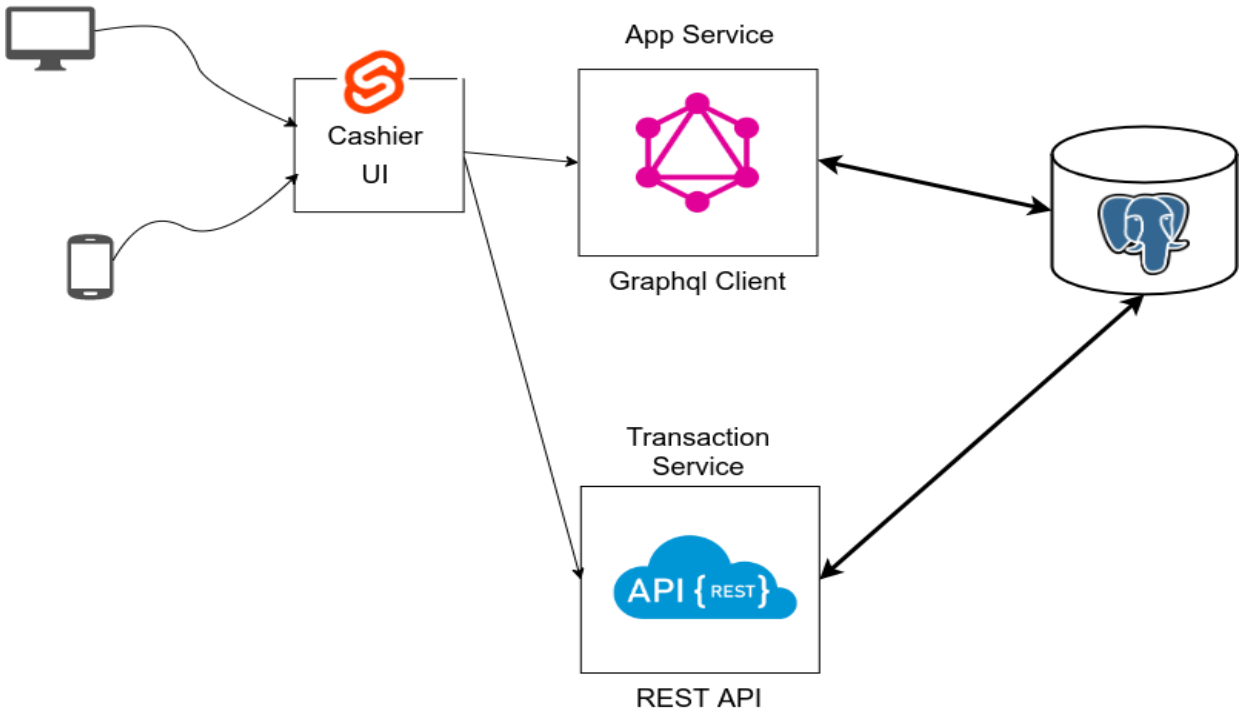
For the completion of the tasks the following web languages and scripts have been used:
Svelte Kit the web framework for the layout and javascript logic, vanilla css and SMUI for the design and styling of the layout, TypeScript as my full stack programming language, Bash for automation, Docker to containerize microservices(Docker Compose), PostgresQL as my main database and redis for cache.

2.Solutions

2.1. Diagrams and Planning

Planning my idea and making it as graphical as possible and as user friendly as possible was one of the things I worked on during my internship at Amen Bank. Having a clear graphical roadmap helped me figure out what I should do next.

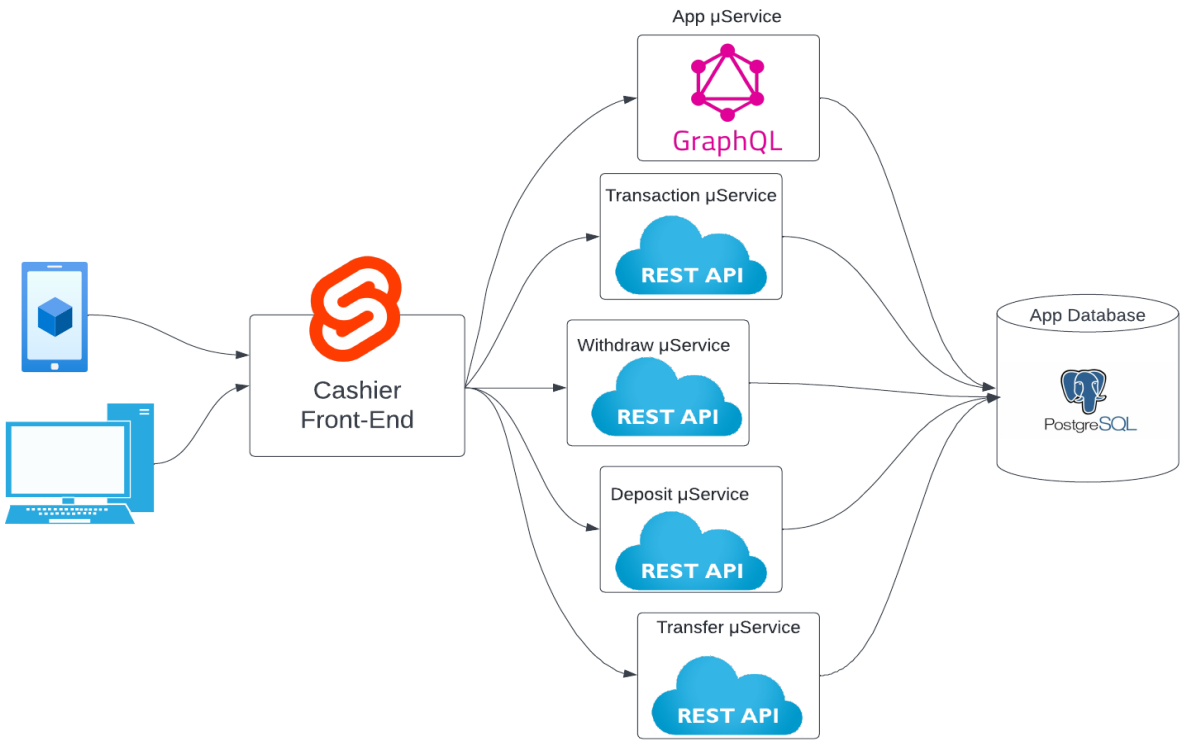
2.1.1. Initial Application Diagram



As seen on the above figure, my initial plan consisted of having a Sveltekit front-end that calls two apis or microservices that in order get data from the database. It was intended that the UI had to be simple since a banking application is supposed to look simple. About the backend, at first I built a graphql api that had everything excluding transactions, this service provided information about users, authentication logic, operations on users and tellers etc and I also had a Transaction service that was a REST api and it did

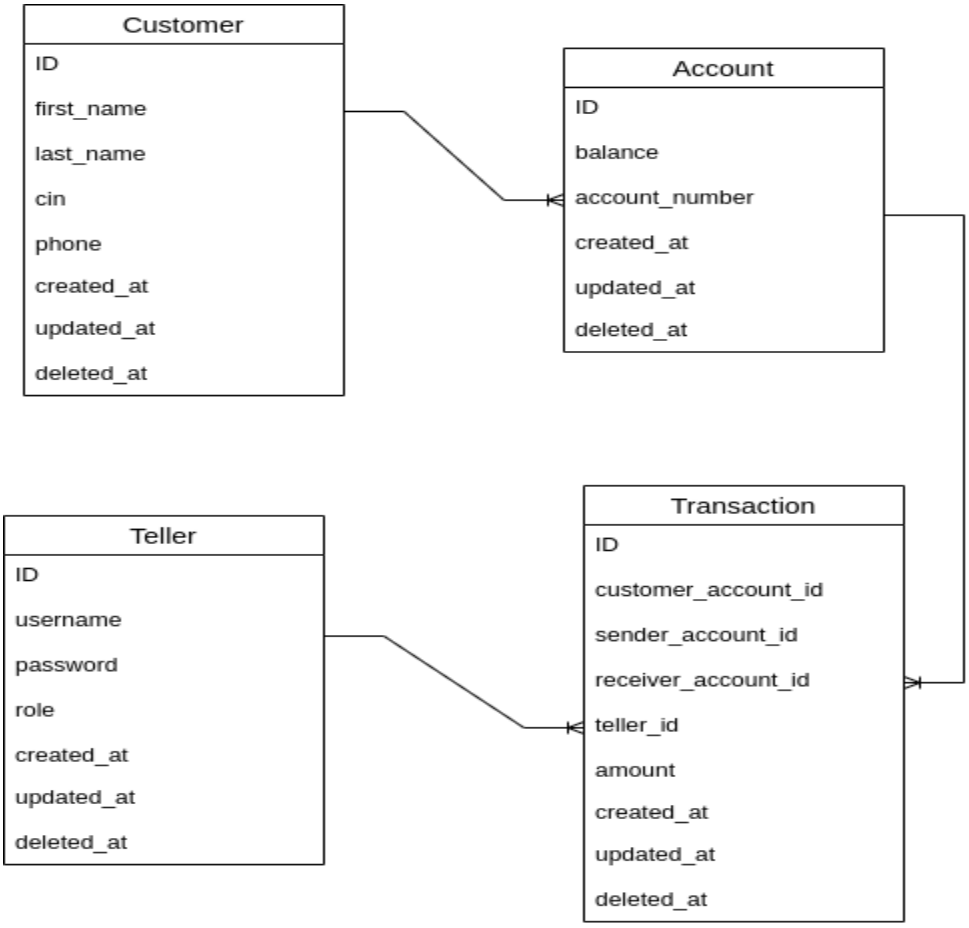
everything related to Banking Transactions from making operations such as depositing money, withdrawing and transferring it to getting transactions and information about them. These two services were loosely coupled so that if one service is down, the other one is still fully functional which is very beneficial to such a critical system. Last but not least both microservices communicate with a single Postgres database.

2.1.2. Final Application Diagram



After having a conversation with my mentor Oussama Ben Cheikh in which I was told that my “micro”services weren’t loose enough, I decided to split banking operations. Each operation was running on its own service so that for example if the withdrawal service is down we can still make other types of banking operations so this makes for a more stable system overall.

2.1.3. Database Diagram



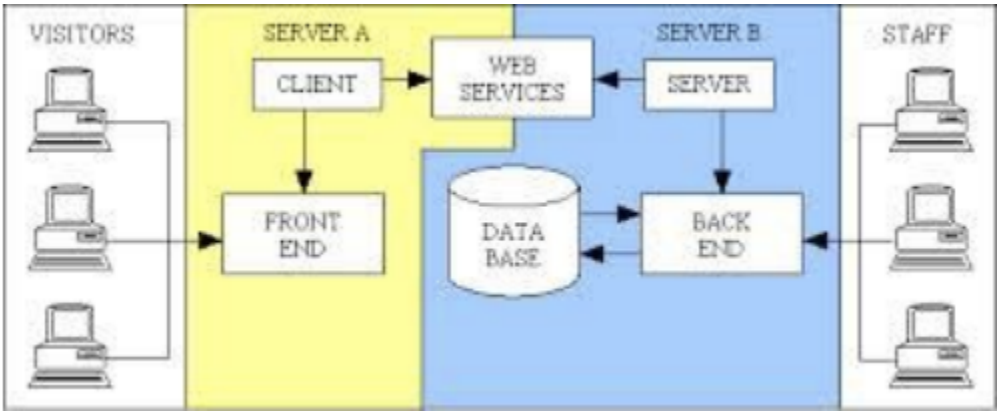
The database schema I picked was very simple and well thought out, every table has an auto-incremented primary key called ID and 3 characteristic columns that define important dates in a banking system so that tellers are able to track that information.

3. Discussion

3.1. Overview

The Bankify application had been planned to consist of two parts: front-end and back-end development. The front-end is the part of the web that you can see and interact with (e.g. Client-side programming). While front-end code interacts with the user in real time, the back-end interacts with a server to return user ready results. The front-end is a combination of HTML,CSS and JavaScript coding which are combined in the Svelte kit framework. By using JavaScript, modifications of the design of a webpage can be made immediately, however only temporary and visible only by the user. Normally the user would not have rights to modify web content dynamically on the server side. Logically, administrators are the ones who deal with back-end modification of databases for example, as they often contain sensitive data which should not be available to see or modify by the general public.
Back-end programming languages include Nodejs, Golang, Python, Ruby and others.I have focused on the front-end and the back-end development of the Bankify app. On figure 3 it can be clearly seen how front-end and back-end development differ and where is their common point. Further explained you will find out more about each of the components in the below-described module.

Fig. 3 - Front-end and back-end development



3.2. GraphQL

GraphQL is a query language I used extensively for my API, it's a runtime for fulfilling those queries with my existing data. It Provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.
This is what a GraphQL schema looks like:

```
type Query {
 hero: Character
}

type Character {
 name: String
 friends: [Character]
 homeWorld: Planet
 species: Species
}


type Planet {
 name: String
 climate: String
}
```

3.3. JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language. This technology was used in my Transaction api microservices as a way to communicate between the front-end and the backend.
This is what a JSON response looks like:

```
{
  "employees":[
        { "firstName":"John", "lastName":"Doe" },
        { "firstName":"Anna", "lastName":"Smith" },
        { "firstName":"Peter", "lastName":"Jones" }
  ]
}
```

3.4. SvelteKit

3.4.1. HTML

HTML in Sveltekit is similar to regular html except it's built into the framework which is the only difference. One major concern of a banking app is privacy and availability only to Tellers.It is considered best practice to add "noindex" and "nofollow" attributes so that the page is not included in search results and also it tells crawlers to not follow the links on the page. The lang attribute is also set as english (lang="en") in order to inform the browser of the default human language of the script, which is essential for the proper reading of the web page by certain technologies for the disabled.

3.4.2. Style/ CSS

In Svelte, you can write CSS in a stylesheet like you normally would on a typical project. You can also use CSS-in-JS solutions, like styled-components and Emotion, if you'd like. It's become increasingly common to divide code into components, rather than by file type. Every component in Bankify has a scoped stylesheet so that there is no style-overriding happening. I've also used the SMUI library along with vanilla css since styling was not my main priority and it would take forever to get a good looking website with as many components as mine.

3.5. Svelte(with TypeScript)

Svelte is a free and open-source front end compiler, I used it for my dynamic templating and data rendering. Svelte applications and components are defined in `.svelte` files, which are HTML files extended with templating syntax that is similar to JSX.
This is what svelte code looks like:

```
<script>
```

```
    let count = 1;

    $: doubled = count * 2;
</script>

<p>{count} * 2 = {doubled}</p>

<button on:click={() => count = count + 1}>Count</button>
```

3.6. TypeScript

All functionality of the front-end has been programmed in TypeScript which is just JavaScript with syntax for types, it gives developers better tooling at scale and enables them to write bug free production code. TypeScript was heavily used in my app to write any sort of logic like performing checks on inputs, pagination, authenticating users and much more.

3.7. Nodejs(with TypeScript)

Server code has been written in Nodejs with typescript compilation. If you can write Javascript you can definitely write Node since it's just a runtime outside of the browser.
The logic in servers such as verifying requests on the backend, getting data from the database was done in this language. It really helped the fast development of the app since it's very easy to write and work with because of the huge ecosystem around it.

3.8. Docker

Docker is an open platform for developing, shipping, and running applications. Docker enabled me to separate my application from my infrastructure so I could test and deliver it quickly.

Every microservice in Bankify was containerized so that it’s independent of other services. This makes for a good development environment across operating systems.

4. Problems and Proposed Solutions

4.1.Authentication

Problem: In a banking application, the process of verifying a user's identity and their ability to access a requested account is the most important aspect of the app since you don’t want an intrusive person to get into your system and start making transactions. Consequently, I had to find a way to make authentication as secure and as fast as possible.

Proposed Solution: After checking out multiple authentication methods, I decided to go for session authentication where a signed cookie is stored in the browser, and an unsigned version of it is stored in the key-value database Redis. When a user makes a request to the server, the cookie gets decrypted and checked against the redis-session value. For passwords, I used the Argon2 hashing algorithm because it’s dedicated to password encryption and doesn't have any uses apart from that.

4.2.Microservices Architecture

Problem: Modern applications rely heavily on microservices, without this architecture scaling is complicated and expensive, testing times are slow because you have to test the whole codebase at once and you have to use a single programming language that could get outdated.

Proposed Solution: I tried out multiple technologies after sticking to a graphql server and 4 REST apis. My first attempt at doing microservices was a gRPC based backend which I was stuck figuring out for two weeks but one thing I didn't like about it is that a gRPC heavily uses HTTP/2 and that's impossible to call from a web browser directly so I had to mess with Envoy Proxy which is a proxy layer that translates HTTP/2 to HTTP/1.1 and I felt like complicating things on a small application wasn't worth it. Second, I went for Apollo Federation which is a powerful, open architecture for creating a supergraph that combines multiple GraphQL APIs. And since I like graphql I thought that I would have a great experience with this technology but it turns out that it wasn't mature enough and it lacked documentation and technical help and resources on the internet plus it didn't have good integration with TypeScript so I dropped it. My final solution that worked was just running microservices on different APIs that communicate with one database to get data.

5. Conclusion

In a nutshell, this internship has been an excellent and rewarding experience. I can conclude that there has been a lot I've learned from my project at Amen bank. Needless to say, the technical aspects of the work I've done are not flawless and could be improved. As someone with little experience in web development whatsoever I believe my time spent in researching and discovering new languages was well worth it and contributed to finding an acceptable solution to an important aspect of web design and development. Two main things that I've learned the importance of are time-management skills and self-motivation. Although I have often stumbled upon these problems at University, they had to be approached differently in a working environment. I have yet to complete another two years of studies, in order to achieve a licence in Computer Science. Working with web development technologies has increased my interest in web development, hence prompting me to focus on this field more and try to get as good as I can.