



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Távközlési és Médiainformatikai Tanszék

Amazon Web Services

MSC - ÖNÁLLÓ LABORATÓRIUM 2

BESZÁMOLÓ

Készítette

Dániel Márk Kiss

Konzulens

István Pelle

November 24, 2023

Contents

Chapter 1	Introduction	Page 2
Chapter 2	Used AWS features and technologies	Page 4
2.1	Amazon Elastic Compute Cloud (EC2)	4
2.2	Amazon Simple Storage Service (S3)	4
2.3	Amazon RDS (Relational Database Service)	5
2.4	Amazon DynamoDB	5
2.5	AWS Lambda	6
2.6	AWS CodeCommit	6
2.7	Docker	7
2.8	MySQL	7
Chapter 3	Development	Page 8
3.1	Overview	8
3.2	Docker	9
	MySQL - User database — 9 • Express JS - Admin page — 9	
3.3	Amazon EC2 - Elastic Compute Cloud	10
3.4	Amazon S3 - Simple Storage Service	10
3.5	Amazon API Gateway	11
3.6	AWS Lambda	12
Chapter 4	Presentation of finished work	Page 13

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].

As I was browsing the different topics for my thesis I have found the AWS services very interesting. I heard that AWS is a very popular opinion for its robust cloud computing solutions. In conclusion, the decision to explore AWS services for my thesis is driven by the platform's reputation for reliability, innovation, and global accessibility. I thought it would be a great opportunity to familiarise myself with the different AWS services and technologies. This way I can also gain some experience in the cloud computing field. I have also done a course provided by Amazon. It contains a 14 module class where they teach the fundamentals of AWS. In today's standards it is crucial to have a minimal knowledge and also experience with one of the biggest cloud computing platforms.

For my project I have created a Book Tracking application. It has a separate admin page where the admins can register the users and another page where the users can track their books. This idea came to my mind while I was traveling to university. As I commute to the university

and back home I read a lot of books to pass the time. I thought it would be a great idea to create an application where I can track the books I have read and also the ones I am currently reading. This way I can keep track of my book collection as well. While I was developing the application I stumbled in an app called Goodreads. For this application I have used the following AWS services and technologies: AWS Lambda functions, Amazon API Gateway, Amazon DynamoDB, Amazon S3, Amazon EC2, Amazon CodeCommit, and other currently available technologies like: Docker, Express JS, MySQL, HTML, CSS, JavaScript, Bootstrap. Hopefully I have used Docker, Express JS, MySQL, HTML, CSS, JavaScript, Bootstrap before so I had some experience with them.

In the next chapter I will present the different AWS services and technologies I have used for my project. After that I will present the development phase of the project. In the last chapter I will present the finished application and showcase it.

Chapter 2

Used AWS features and technologies

Before I started the development of the application I have done a course provided by Amazon. It contains a 14 module class where they teach 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]. In the next sections I will present the different AWS services and technologies I have used for my project.

2.1 Amazon Elastic Compute Cloud (EC2)

The EC2 service [3] provides resizable compute capacity in the cloud, allowing users to run virtual servers, known as "instances," for various computing tasks. Key features:

1. Scalability: EC2 instances can be quickly scaled up or down based on demand. This means that users can add or remove instances to match the needs of their applications or workloads.
2. Variety of Instance Types: EC2 offers a wide range of instance types optimized for different use cases. These include instances optimized for compute-intensive workloads, memory-intensive tasks, storage-optimized applications, and more.
3. Operating System Flexibility: Users can choose from a variety of operating systems, including various Linux distributions, Microsoft Windows, and others, to run on their EC2 instances.

2.2 Amazon Simple Storage Service (S3)

S3 [4] offers scalable object storage for storing and retrieving data. It is commonly used for data backup, hosting static websites, and as a storage backend for applications. Amazon S3 is designed to provide durability, availability, and scalability at a low cost. Key features:

1. **Object Storage:** Amazon S3 stores data as objects, which consist of a file and its associated metadata. Each object is identified by a unique key, making it easy to access and manage.
2. **Buckets:** A bucket is a container for objects stored in Amazon S3. Buckets act as top-level folders or directories for organizing and managing objects. Each bucket has a unique name and is associated with a specific AWS region.
3. **Durability and Availability:** Amazon S3 is designed to provide 99.999999999% (11 9's) durability for objects over a given year. It achieves this by replicating data across multiple Availability Zones within a region, ensuring high availability.
4. **Scalability:** Amazon S3 can scale to accommodate virtually unlimited amounts of data. Users can easily upload, store, and retrieve any amount of data, making it suitable for a wide range of use cases.
5. **Access Control:** Users can control access to their buckets and objects through AWS Identity and Access Management (IAM) policies, bucket policies, Access Control Lists (ACLs), and signed URLs or cookies.
6. **Cross-Region Replication:** Users can configure Amazon S3 to automatically replicate objects from one bucket to another in a different AWS region. This helps achieve geographic redundancy and compliance with data residency requirements.
7. **Static Website Hosting:** Amazon S3 can be used to host static websites, providing a cost-effective solution for hosting web content.

2.3 Amazon RDS (Relational Database Service)

RDS [5] offers managed database services for various database engines, including MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. Key feature:

Managed Service: AWS handles the heavy lifting of database administration tasks such as hardware provisioning, database setup, configuration, patch management, and backups. This allows users to focus on application development rather than database management.

2.4 Amazon DynamoDB

Amazon DynamoDB [6] is fully-managed NoSQL database service that provides high performance and seamless scalability for applications that require low-latency access to data.

1. NoSQL Database: DynamoDB is a NoSQL database, which means it does not rely on a fixed schema like traditional relational databases. This allows for flexible data modeling, making it well-suited for applications with evolving data structures.
2. Managed Service: AWS handles all the operational aspects of DynamoDB, including hardware provisioning, setup, configuration, and maintenance. This frees developers from the burden of database management tasks.
3. Scalability: DynamoDB is designed to scale easily, both in terms of read and write throughput. It can handle massive volumes of traffic and automatically scales based on demand. Users can adjust read and write capacity units to accommodate their application's needs.

2.5 AWS Lambda

AWS Lambda [7] service allows users to run code without provisioning or managing servers. It executes code in response to specific events, making it a key component of serverless architecture.

1. Serverless Architecture: Lambda is a serverless computing service, which means developers do not need to manage the underlying servers. Instead, they write code and AWS handles the execution and scaling of that code.
2. Event-Driven Model: Lambda functions are triggered by events such as changes to data in an Amazon S3 bucket, updates to a DynamoDB table, or incoming HTTP requests via Amazon API Gateway. This event-driven model allows for real-time responses to changes in the environment.
3. Supported Runtimes: AWS Lambda supports multiple programming languages including Node.js, Python, Java, Go, Ruby, .NET Core, and custom runtimes. This flexibility enables developers to use their preferred programming language.
4. Pay-per-Use Pricing: With AWS Lambda, users pay only for the compute time consumed by their code. There is no charge when code is not running. This pricing model can lead to cost savings compared to traditional server-based architectures.

2.6 AWS CodeCommit

AWS CodeCommit [8] is a fully managed source control service provided by AWS. It is a secure and scalable Git-based repository hosting service designed to help teams collaborate on code development and version control. CodeCommit provides a secure and reliable platform for storing and managing code repositories.

Key features:

1. **Git Repository Hosting:** CodeCommit supports the Git version control system, providing a familiar interface for developers to manage their code repositories.
2. **Secure and Private:** CodeCommit repositories are secure by default and can be configured to be private. Access control policies, AWS Identity and Access Management (IAM) permissions, and encryption options ensure the confidentiality and integrity of code.
3. **Integration with AWS Services:** CodeCommit seamlessly integrates with other AWS services, such as AWS CodePipeline (is an automation pipeline for code builds), AWS CodeBuild (the service which does the automatic code build), AWS CodeDeploy (the service which does the automatic deployment), and more. This enables end-to-end continuous integration and continuous deployment (CI/CD) workflows.
4. **Version Control:** CodeCommit allows multiple developers to collaborate on projects by providing version control capabilities. Developers can create branches, merge code changes, and track commit history.

2.7 Docker

Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and are thus more lightweight than virtual machines. Containers are created from images that specify their precise contents. Images are often created by combining and modifying standard images downloaded from public repositories [9].

2.8 MySQL

MySQL is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language. A relational database organizes data into one or more data tables in which data types may be related to each other; these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database.

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. I have also version controlled the whole development process using git and publishing it on GitHub and also Amazons Code Commit system. Not only the source code can be found there but also the documentation of this project as I have written it using \LaTeX . The project I have created using AWS and other technologies is a Book Tracking website. It has a separate admin page where the admins can register the users and another page where the users can track their books. For the books there are three state. Not started, Reading and Finished. These books are presented in a card format in grid layout and it is only visible to the users after a successful login. To achive this I have used the following technologies:

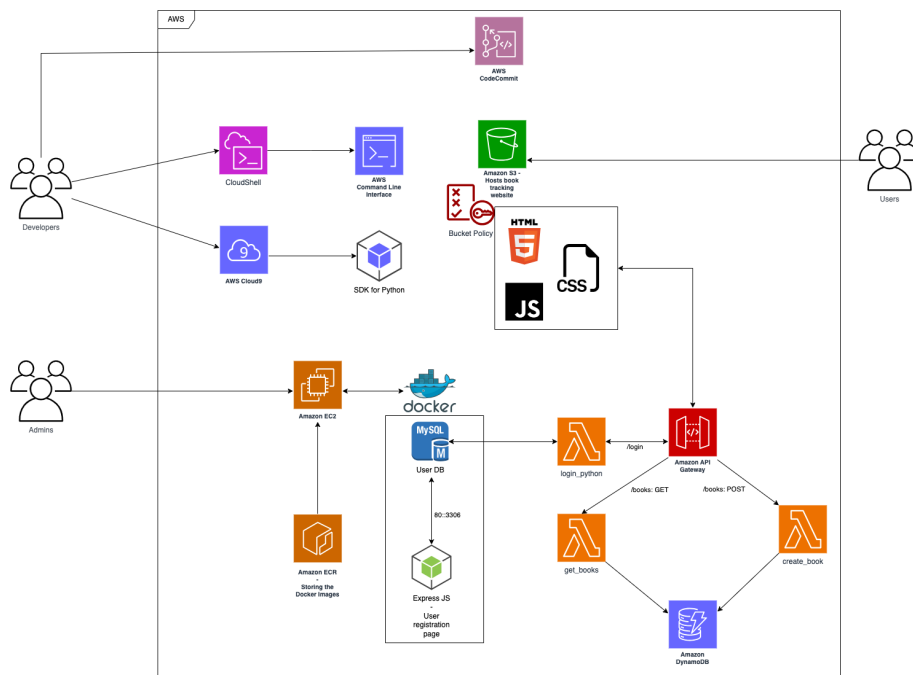


Figure 3.1: The architecture of the application.

3.2 Docker

I used this for the admin page creation and for storing the user data in a MySQL database. I created two containers, one for the admin page and one for the database. The admin page is written in Express JS and the database is MySQL.

3.2.1 MySQL - User database

I have set up this database using Docker and I have created a table for the users. The table has the following columns: username, password. To create the container I first created the image, which is created from the Dockerfile. The Dockerfile I composed contains the fundamentals like it used mysql 8.2.0 and it exposes its port to 3306. After the creation I used the following command to update the database.

```
docker run --name mysql_1 -p 3306:3306
-> -e MYSQL_ROOT_PASSWORD=pwd -d mysql_server
docker exec -i mysql_1 mysql -u root -ppwd < db.sql
```

This Docker command is used to After I have successfully created the container I tagged it and pushed it to the AWS ECR(Elastic Container Registry).

```
docker tag mysql_server
-> 554751627586.dkr.ecr.eu-north-1.amazonaws.com/mysql_1:latest
docker push 554751627586.dkr.ecr.eu-north-1.amazonaws.com/mysql_1:latest
```

3.2.2 Express JS - Admin page

The admin front end page was developed using Express JS. Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. On the webpage the admins can register new users and also delete them. The users and their passwords are stored in the MySQL database. I also applied Bootstraps to the page to make it more user friendly and responsive. After I have finished the development I have created the image and pushed it to the AWS ECR(Elastic Container Registry) also.

```
docker inspect network bridge
docker build --tag node_app .
docker run -d --name node_app_1 -p 3000:3000
-> -e APP_DB_HOST=172.17.0.2 node_app
```

I exposed the 3000 TCP port to the 3000 port of the host machine. I also set the APP_DB.HOST environment variable to the IP address of the MySQL container. This way the Express JS application can connect to the MySQL database.

3.3 Amazon EC2 - Elastic Compute Cloud

After I have pushed my containers to the AWS ECR I have created an EC2 instance. I have used the Amazon Linux 2 AMI (HVM), SSD Volume Type. I have also added a 30GB storage to the instance. I have also added that the instance can be accessed from anywhere. So I added a Security Group with the following rules:

- SSH - TCP - 22 - Anywhere
- HTTP - TCP - 80 - Anywhere
- HTTPS - TCP - 443 - Anywhere
- MYSQL/Aurora - TCP - 3306 - Anywhere

After the creation of the instance I have connected to it using SSH. I have also installed Docker and Docker Compose on the instance. After that I have pulled the images from the AWS ECR and started the containers. I also had to authorize the Docker to pull images from the AWS ECR. I have done this by running the following command:

```
Project on / main on - (us-east-1)
> aws configure
AWS Access Key ID [*****WQ5Q]: AKIAVCKOGCFBLPW4HR5W
AWS Secret Access Key [*****IPNa]: e28L1MT23iYUj+qKiCS0YjuUUio8KHYSu1cuo50
Default region name [us-east-1]: eu-north-1
Default output format [json]: json

Project on / main on - (eu-north-1) took 1m49s
> aws ecr get-login-password --region eu-north-1 | docker login --username AWS --password-stdin 554751627586.dkr.ecr.eu-north-1.amazonaws.com
Login Succeeded

Project on / main on - (eu-north-1) took 12s
> aws ecr create-repository --repository-name mysql_1
```

Figure 3.2: AWS EC2 configuration.

```
aws ecr get-login-password --region eu-north-1
docker login --username AWS --password-stdin
-> 554751627586.dkr.ecr.eu-north-1.amazonaws.com
docker pull 554751627586.dkr.ecr.eu-north-1.amazonaws.com/node-app
```

After this I started the front-end page of the Book Tracking app.

3.4 Amazon S3 - Simple Storage Service

The front-end page for the Book Tracking app is hosted on the Amazon S3. I have created a bucket for the page and I have uploaded the files to the bucket. I have also set the bucket to be public so anyone can access it. I have also set the bucket to be a static website hosting. I have also set the index.html as the index document. This way the page will be loaded when the user enters the URL

of the bucket. The page was creste with the help of HTML, CSS and JavaScript. I have also used Bootstrap to make the page more user friendly and responsive. First the user has to log in to the page. This data is stored in the MySQL database. After the successful login the user can see the books he/she has added to the list. The books are stored in the Amazon DynamoDB database. The user can also log out from the page. When the login happens or the Amazon DynamoDB retrieval the page sends a request to the Amazon API Gateway. This is a REST API which is connected to the AWS Lambda function. This function is written in Python and it is responsible for the communication between the front-end page and the Amazon DynamoDB database. The function is also responsible for the authentication of the users. In the next section I will explain these.

3.5 Amazon API Gateway

The Amazon API Gateway is used to host the different ReST APIs. There I have created the following APIs:

- GET: books
- POST: create_book
- GET: login

To create these APIs I have used the AWS Lambda functions. I have also set the API Gateway to be public so anyone can access it. I have also set the CORS to be enabled. This way the front-end page can access the APIs. I have also set the API Gateway to be a proxy. This way the API Gateway will forward the requests to the AWS Lambda functions. I have also set the API Gateway to be a REST API. This way the API Gateway will be able to handle the different HTTP methods.

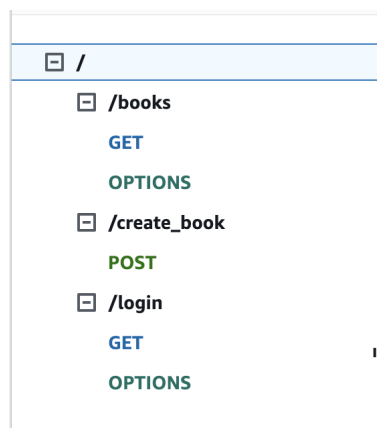


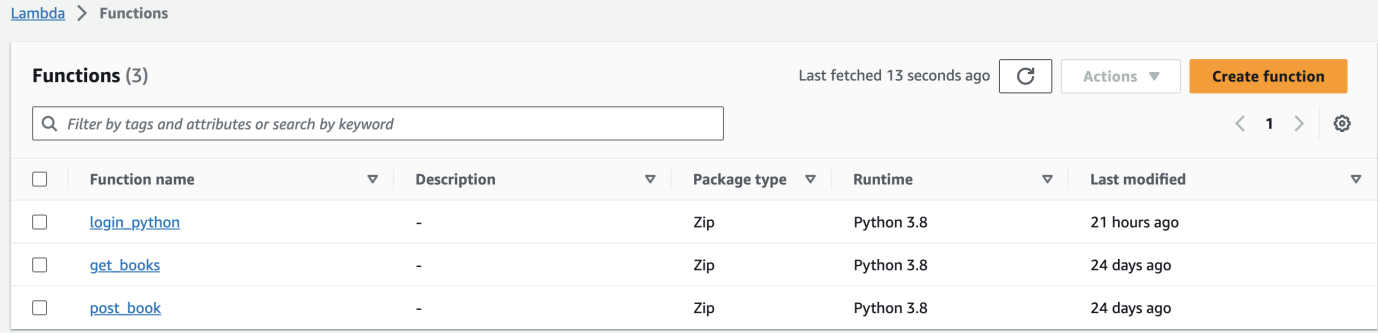
Figure 3.3: The APIs for the application.

3.6 AWS Lambda

For the APIs i have created the following AWS Lambda functions:

- `get_books` - is used for the GET: books API. It is responsible for retrieving the books from the Amazon DynamoDB database.
- `post_book` - is ued for the POST: create_book API. It is responsible for creating a new book in the Amazon DynamoDB database.
- `login_python` - is used for the GET: login API. It is responsible for the authentication of the users.

These lambda functions were created using Python. For the login python I had to create a Lambda Layer. The layer was needed to add `mysql.connector` to the Lambda function. This way the function can connect to the MySQL database. I have also set the timeout for the functions to be 30 seconds. This way the functions will not time out when they are retrieving the data from the database.



The screenshot shows the AWS Lambda console 'Functions' page. It displays a table with three functions: `login_python`, `get_books`, and `post_book`. All functions are using Python 3.8 as the runtime and Zip as the package type. The `login_python` function was last modified 21 hours ago, while the other two were last modified 24 days ago. The interface includes a search bar, a refresh button, and a 'Create function' button.

<input type="checkbox"/>	Function name	Description	Package type	Runtime	Last modified
<input type="checkbox"/>	login_python	-	Zip	Python 3.8	21 hours ago
<input type="checkbox"/>	get_books	-	Zip	Python 3.8	24 days ago
<input type="checkbox"/>	post_book	-	Zip	Python 3.8	24 days ago

Figure 3.4: The lambda functions for the APIs.

Chapter 4

Presentation of finished work

In this section I will present and showcase my finished application. I will include screenshots to have a better representation and understanding for the reader.

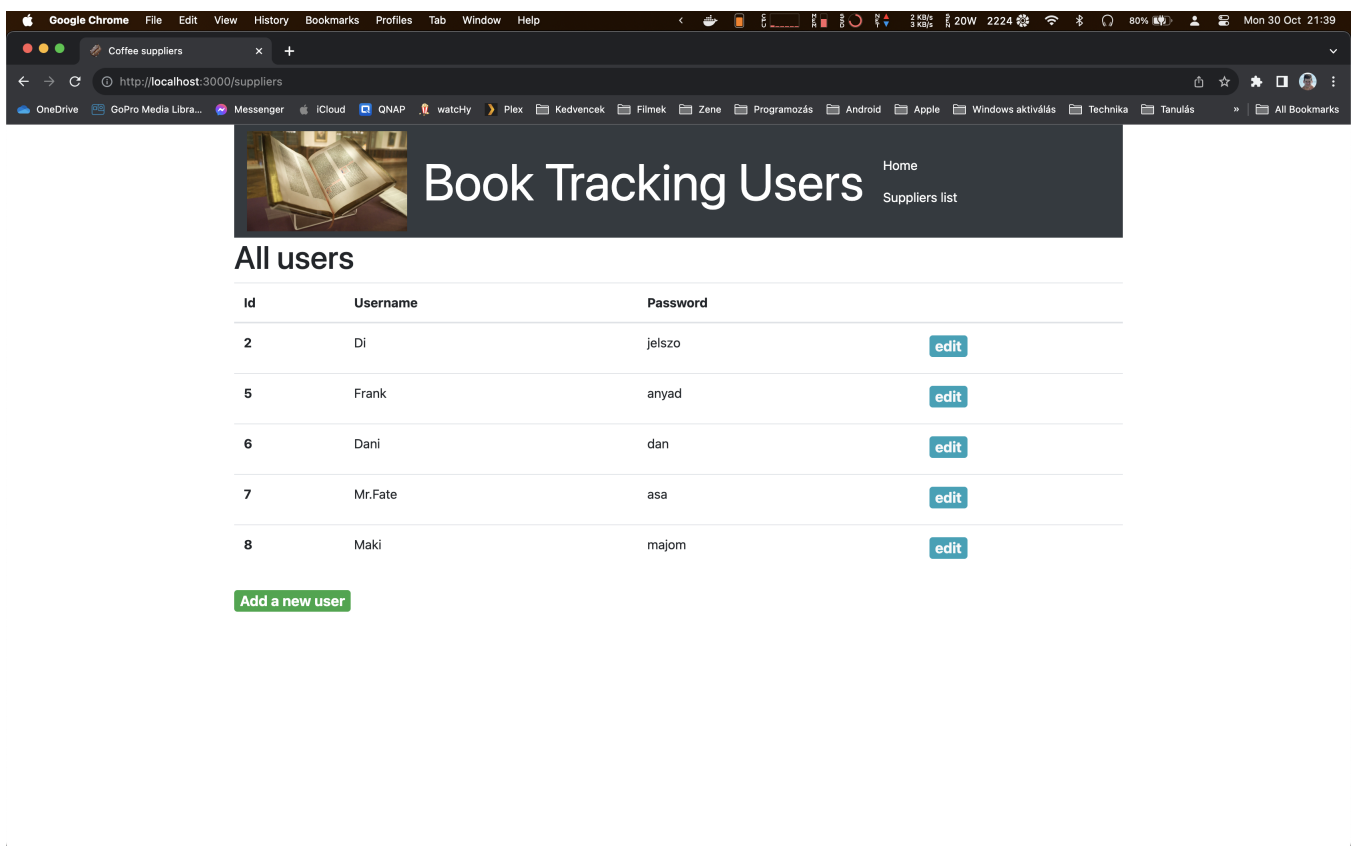


Figure 4.1: The main screen of the admin page.

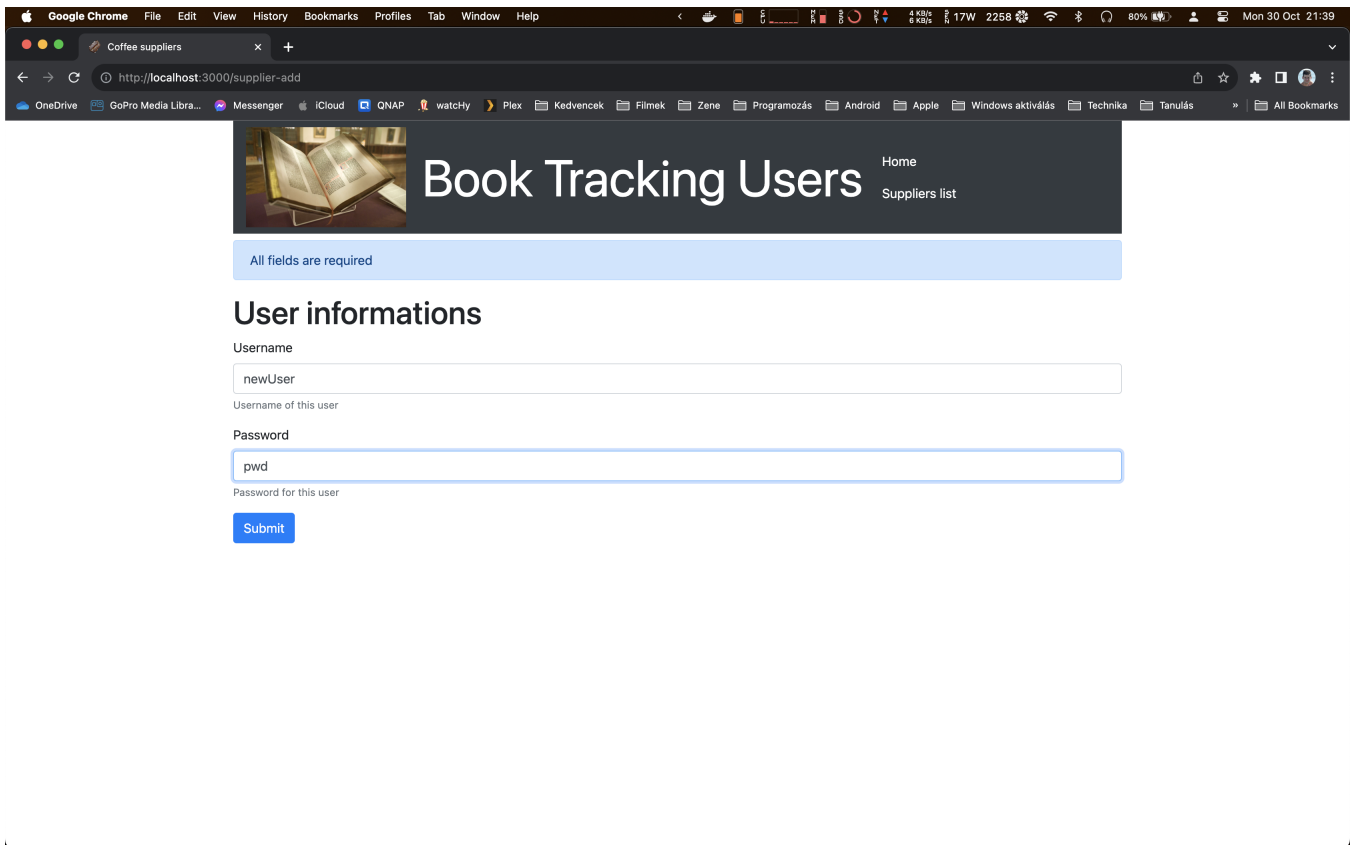


Figure 4.2: Creating a new user.

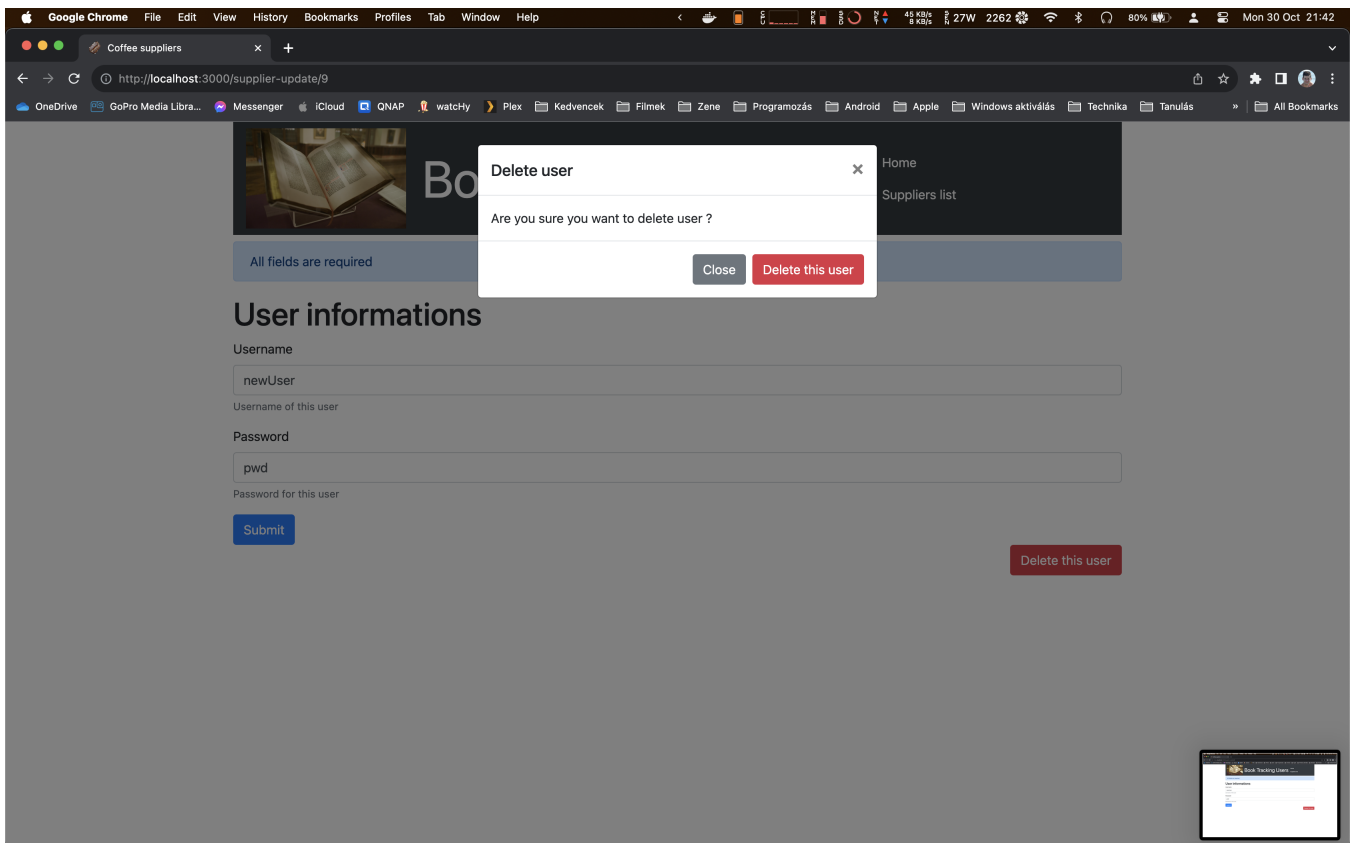


Figure 4.3: Deleting a user.

Book Tracking Online.

Username

Password

Sign in

© 2023 Kiss Daniel Mark All rights reserved.

Figure 4.4: The login page of the application.

Book Tracking Online.

Started: 2023.11.02

Ready

Bloodvampire
Darren Shane

Started: 2023.11.02

Reading

Steve Jobs
Walter Isaac

Started: 2023.11.02

Bookshelf

Kill on Orient
Agathe Christie

Log out

© 2023 Kiss Daniel Mark All rights reserved.

Figure 4.5: The main screen of the application.

Bibliography

- [1] TechTarget - Amazon Web Services: <https://www.techtarget.com/searchaws/definition/Amazon-Web-Services>
- [2] Amazon Web Services Academy: <https://aws.amazon.com/training/awsacademy/>
- [3] Amazon EC2: <https://aws.amazon.com/pm/ec2/>
- [4] Amazon S3: <https://aws.amazon.com/s3/>
- [5] Amazon RDS: <https://aws.amazon.com/rds/>
- [6] Amazon DynamoDB: <https://aws.amazon.com/dynamodb/>
- [7] Amazon Lambda: <https://aws.amazon.com/lambda/>
- [8] Amazon CodeCommit: <https://aws.amazon.com/codecommit/>
- [9] Docker: <https://www.docker.com>