Eötvös Loránd University
Faculty of Informatics
Department of Programming Languages
and Compilers

Use of Cloud Technologies in a Modern Scalable Web Application

Dr. Viktória Zsók
Lecturer

Basir Doost Mohammadi
Computer Science BSc

Budapest, 2020

Contents

Chapter 1

INTRODUCTION

1.1 Motivation

In this project we create a web application to serve as a platform for a host institution to announce their conferences, and to streamline the process of registering to a conference both for the user and the institution. This project has been created using modern web development principles and designs, to create a rich and seamless user experience and to comply with the current standards set for web applications.

Over the past decade, internet access has rapidly spread across people in all continents and with popularization of smartphones, the web has become a very important part of most people's lives. Consequently, many new concepts, methods, and technologies have been created over the past few years, to provide a better web experience with the significant increase of demand. Web applications and websites as a whole have changed to an extent that now browsing most webpages from the 2000's often feels outdated and obsolete. Since all current major web applications such as Facebook or Gmail now leverage many of these new technologies to offer their services, they technologies are updated frequently to provide the best experience possible for users and developers. Utilization of these tools in our project helps with many of the challenges we face, and makes the development process more efficient and structured. It also provides us with a solid infrastructure to build the application on, and to extend it with new use cases and features.

Cloud technology is used in this project to contain and process all of the application resources, and offers on-demand storage and processing power that we require for the application. This removes the need for buying or renting servers to hold the project, and eliminates the burden of having to maintain and configure the servers and lets us focus only on development on a reliable infrastructure. It also provides automatic scaling, to allocate more processing power whenever needed to deal with the amount of traffic on the application [1][2].

Using the Cloud, we only pay for the amount of computational resources we use, instead of having a fixed amount of resources for a fixed fee. This ultimately reduces the cost of development and running the project, especially since our application does not require any intense processing. Apart from that, our Cloud provider offers a certain amount of usage on each of the services for free each month. At the time of writing the application has not been receiving traffic from the public, but there is a live deployment of the application which is used for development, testing, and demo purposes. Our total Cloud costs for this project including the cost of the live application and running the build and deployment system over about three months has been $0, since we do not pass any of our Cloud provider's limits to actually be charged for the resources consumed. With the amount of traffic we expect a project like this to receive after being publicly released, it is still very unlikely to incur costs more than a very insignificant amount.

The user interface of the application has also been made using modern frameworks and design principles, to deliver a quick and intuitive interaction with the user. It is implemented as a Single Page Application using React, an open-source UI framework for JavaScript maintained by Facebook and community developers. React allows us to create a website, using a nested structure of so called components. These components are defined with a syntax called JSX, which resembles HTML but provides the functionalities of JavaScript within it, to define the user interface of the application.

One of the main focuses in the design process of this project has been to maintain separation of concerns. This combined with the strong error-logging capabilities of our Cloud system, makes the process of finding the root cause of bugs and patching them easier, and allows for a better overall developer experience. Adding new features and tools to the website can also be easily done, by defining the necessary backend operations, performing those operations in a Lambda function in response to API requests, and finally, extending the user interface to utilize the new operations to allow the user to interact with the new features.

1.2 Thesis Structure

Chapter 2 of this thesis contains a user documentation, which provides a description of the application and its use cases, a brief overview of the technical methods used in the implementation and how it affects the user experience, and a complete user's guide for all of the available features of the application.

Chapter 3 is a developer documentation, containing all the technical details of the application. In this chapter we specify all the tasks and challenges of the development process, and provide detailed information about the concepts and components used in the application - including how they function, and how they connect with each other to shape the functionality of the application.

API documentation of the main code component of the application can be found in the appendices, which provides detailed information about the classes and functions used in the application.

Chapter 2

USER DOCUMENTATION

2.1 Project Description

The goal of this project is to create a website named Confer, to manage registrations for an institution's conferences. Visitors can find information about the offered conferences, and sign in to make a registration through the website. There are also tools and features implemented for the host institution's management, so that they can create new conferences, modify existing ones, and manage payments and registrations submitted by users.

All visitors are able to view the conferences, but creating an account is required to make a registration on the system. During registration, users can also request meal preparation and assistance with accommodation; in which case they will be contacted later by the host institution for more details. After payment for the registration has been processed, a PDF document containing the user's entrance ticket is sent to the email provided during registration.

Each registered user of the application has a user type of either basic, moderator, or admin. An account created using the website will have the type basic, and allows the user to make registrations and access their previous registrations and payments. Additionally, the host institution can grant a user moderator or admin rights, allowing them to use the moderator tools for managing the system. Currently the admin and moderator users can perform the same operations, however they have been implemented separately to allow extending the application's use cases in the future with two different levels of management rights.

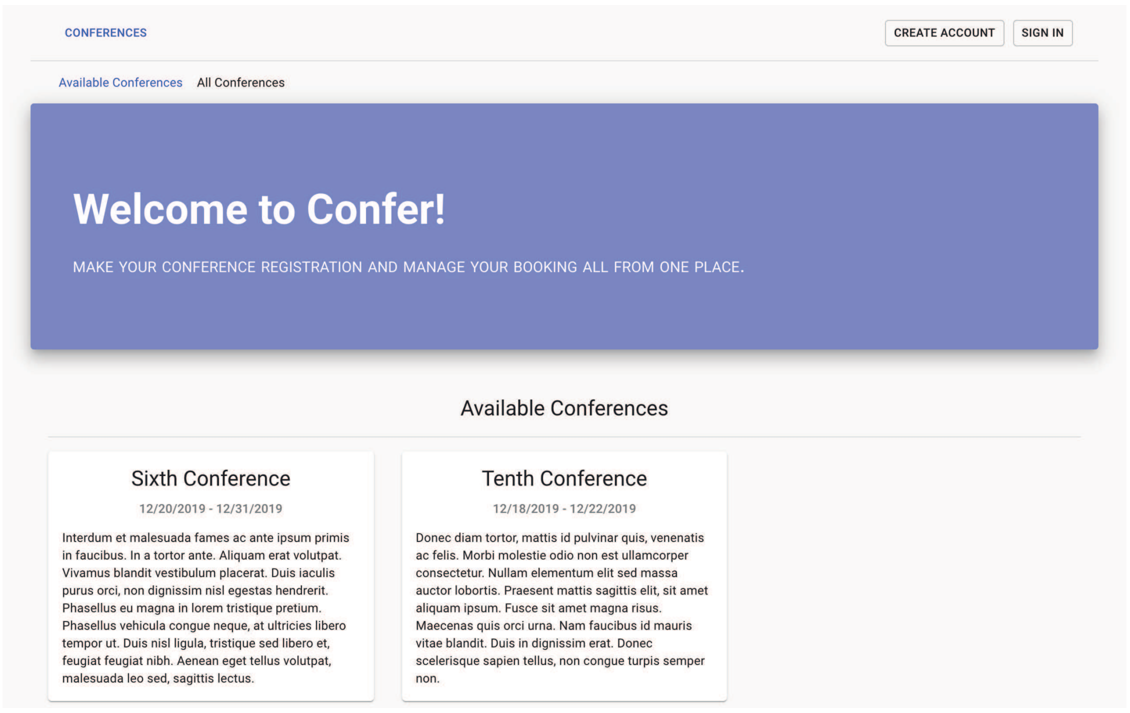
2.2 Technical Information

In this project, we have used the Serverless computing technology based on Amazon Web Services. In this architecture, the application runs inside managed containers that are hosted on the Cloud provider's servers, which eliminates the need for developers to manage and maintain the servers themselves since all server configuration and maintenance is done by the provider. Using a serverless solution, we are able to deliver a scalable and highly available application, to ensure the system is always accessible to users even when experiencing higher-than-expected traffic on the website. Also with the rapid development process resulting from serverless, we have the means to introduce new features and solve bugs in the application more quickly to provide a better user experience [3][4]. More information about the technical stack of the application can be found in the developer documentation.

2.3 Usage Information

In this section we provide the information needed for a user to interact with the application. The website can be accessed using the URL:
`https://master.d6nxzuw99ek2a.amplifyapp.com/`.

2.3.1 Getting Started



Welcome to Confer!

MAKE YOUR CONFERENCE REGISTRATION AND MANAGE YOUR BOOKING ALL FROM ONE PLACE.

Available Conferences

Sixth Conference
12/20/2019 - 12/31/2019
Interdum et malesuada fames ac ante ipsum primis in faucibus. In a tortor ante. Aliquam erat volutpat. Vivamus blandit vestibulum placerat. Duis iaculis purus orci, non dignissim nisl egestas hendrerit. Phasellus eu magna in lorem tristique pretium. Phasellus vehicula congue neque, at ultricies libero tempor ut. Duis nisl ligula, tristique sed libero et, feugiat feugiat nibh. Aenean eget tellus volutpat, malesuada leo sed, sagittis lectus.

Tenth Conference
12/18/2019 - 12/22/2019
Donec diam tortor, mattis id pulvinar quis, venenatis ac felis. Morbi molestie odio non est ullamcorper consectetur. Nullam elementum elit sed massa auctor lobortis. Praesent mattis sagittis elit, sit amet aliquam ipsum. Fusce sit amet magna risus. Maecenas quis orci urna. Nam faucibus id mauris vitae blandit. Duis in dignissim erat. Donec scelerisque sapien tellus, non congue turpis semper non.

Upon opening the website, the available conferences page is displayed as depicted above; containing a list of all upcoming conferences that the user can register for. Clicking on any of the conferences redirects the user to a page containing more information about the conference.

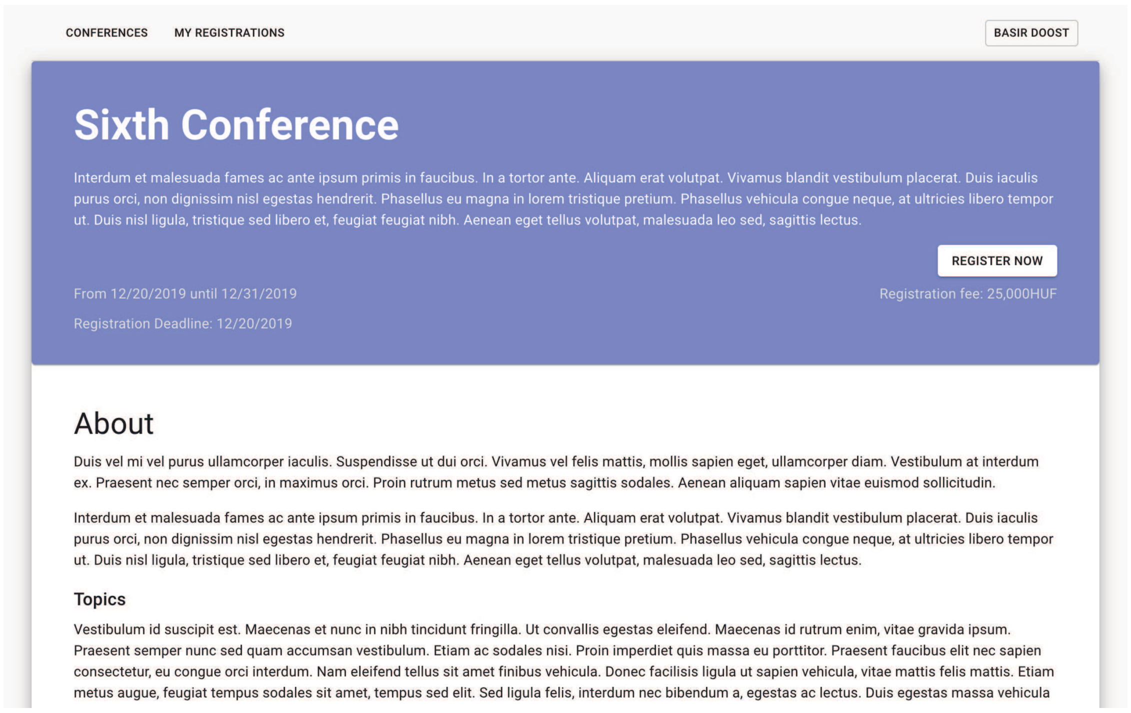
A navigation bar is always present on top of the screen which the user can use to navigate between the different sections of the application, or to sign in or out of their account. In the Conferences section, the user can switch between upcoming conferences and all conferences on the website - including those that are past their registration deadline - using the navigation bar.

To use most features of the website, creating an account is required. To create a new account, the user can click on the "Create Account" button in the navigation bar; after which they will be presented with a form containing information needed for creating a new account. A valid email address, user's full name, and a password - containing minimum 8 characters and composed of at least one uppercase letter, one lowercase letter, and one digit - are required when creating an account.

After an account is successfully created on the system, the verification form is shown on the website and the user will receive an email to confirm their email address. This email contains a code, which has to be typed into the verification form along with the email address used to create the account. In case the user closes the verification form after account creation, they can access it again by clicking on the "Sign In" button in the navigation bar and pressing "Verify your account".

In case the user has already created an account on the website but has forgotten their password, they can navigate to the "Sign In" page and press "Forgot your password", after which a form is displayed that asks for the email address that was used when creating the account. Upon submission of this form, the user receives an email containing a confirmation code that is needed to change their password, and a new form opens on the website for the user to enter their new password. Now the user can enter their email address, the confirmation code they just received, and a new password, to gain access back into their account.

2.3.2 Registering for a Conference



Sixth Conference

Interdum et malesuada fames ac ante ipsum primis in faucibus. In a tortor ante. Aliquam erat volutpat. Vivamus blandit vestibulum placerat. Duis iaculis purus orci, non dignissim nisl egestas hendrerit. Phasellus eu magna in lorem tristique pretium. Phasellus vehicula congue neque, at ultricies libero tempor ut. Duis nisl ligula, tristique sed libero et, feugiat feugiat nibh. Aenean eget tellus volutpat, malesuada leo sed, sagittis lectus.

From 12/20/2019 until 12/31/2019

Registration fee: 25,000HUF

Registration Deadline: 12/20/2019

About

Duis vel mi vel purus ullamcorper iaculis. Suspendisse ut dui orci. Vivamus vel felis mattis, mollis sapien eget, ullamcorper diam. Vestibulum at interdum ex. Praesent nec semper orci, in maximus orci. Proin rutrum metus sed metus sagittis sodales. Aenean aliquam sapien vitae euismod sollicitudin.

Interdum et malesuada fames ac ante ipsum primis in faucibus. In a tortor ante. Aliquam erat volutpat. Vivamus blandit vestibulum placerat. Duis iaculis purus orci, non dignissim nisl egestas hendrerit. Phasellus eu magna in lorem tristique pretium. Phasellus vehicula congue neque, at ultricies libero tempor ut. Duis nisl ligula, tristique sed libero et, feugiat feugiat nibh. Aenean eget tellus volutpat, malesuada leo sed, sagittis lectus.

Topics

Vestibulum id suscipit est. Maecenas et nunc in nibh tincidunt fringilla. Ut convallis egestas eleifend. Maecenas id rutrum enim, vitae gravida ipsum. Praesent semper nunc sed quam accumsan vestibulum. Etiam ac sodales nisi. Proin imperdiet quis massa eu porttitor. Praesent faucibus elit nec sapien consectetur, eu congue orci interdum. Nam eleifend tellus sit amet finibus vehicula. Donec facilisis ligula ut sapien vehicula, vitae mattis felis mattis. Etiam metus augue, feugiat tempus sodales sit amet, tempus sed elit. Sed ligula felis, interdum nec bibendum a, egestas ac lectus. Duis egestas massa vehicula

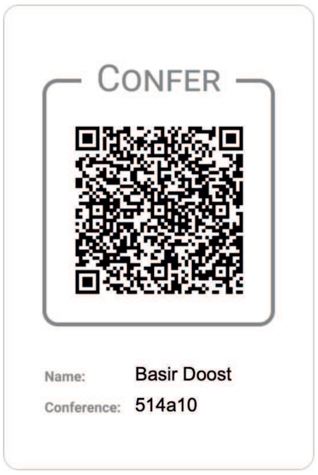
Users can register for a conference by navigating into the details page for a given conference (depicted above), and pressing the "Register Now" button which can be found below the conference description, on the bottom right hand side of the blue overview dialogue. After that they are presented with a form, containing three steps needed for registration. On the first step, the user needs to input the full name and email address of the person for whom the registration is made, along with separate billing information including full name, email, and address of billing. Additionally, the user can choose to use the name and email provided in their account for making the registration by checking the "Use name and email from profile" box, and to have the same billing information as in the registration with the "Use name and email from previous section" checkbox.

After pressing next, the user is asked to enter whether they want additional services and assistance for the conference, including meal preparation and help with accommodation. They are also able to enter notes for the meal, such as dietary restrictions or allergies, and provide their date of arrival to the conference and the number of days they will need accommodation for, so that the host institution can provide assistance with it. It should be noted that filling this information is not required for registration.

On the last step, the user can review all the submitted information and verify if they are all correct, and finally submit the registration. After processing the registration, a new page is displayed, stating that the registration has been successful, and containing important information that the user must note to be able to make the payment. This page includes a payment ID, a detailed invoice of the total amount they need to pay, and payment instructions and bank account details to help the user transfer the registration fee.

Note: The payment ID needs to be included as a comment in the bank transfer to highlight which payment the transfer is made for.

After the success of the payment is confirmed by the moderators, the user receives an email address with a PDF document attached, containing the ticket they need for entry to the conference. The ticket looks similar to the figure below:

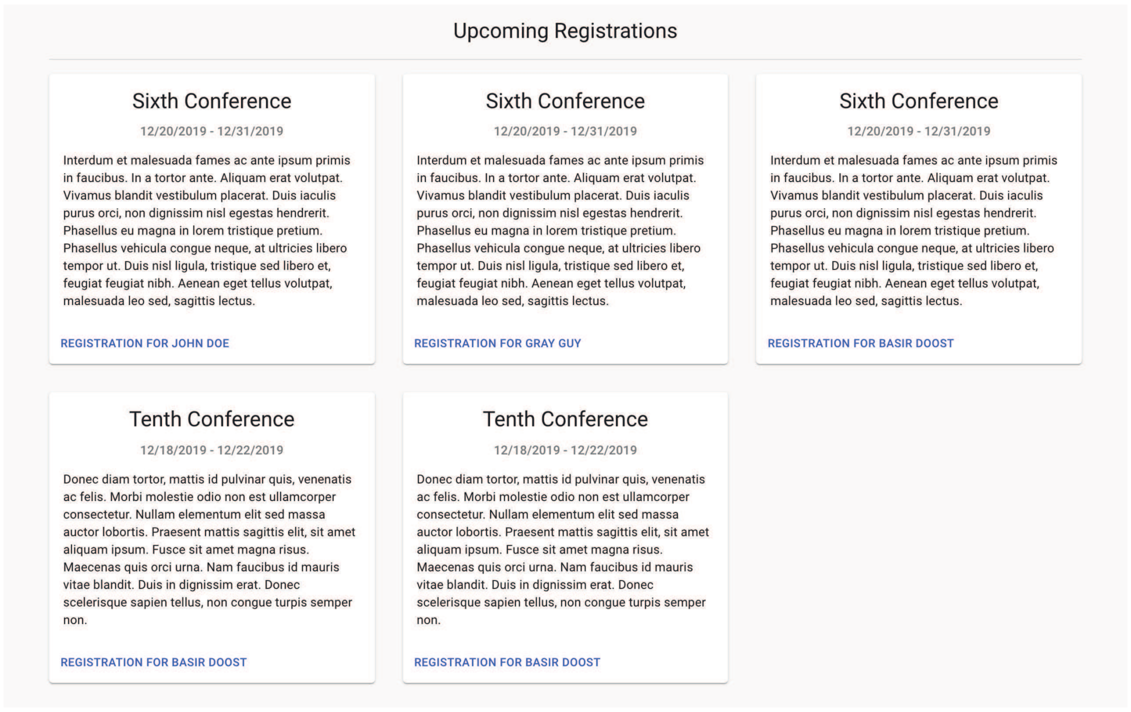


The user needs to have this document with them when they enter the conference. The document does not need to be printed, and can simply be shown from the attendee's phone; however, if for any reason this is not possible for the attendee, they can choose to print the document and have it with them at the conference.

A user can make multiple registrations for the same conference. This way, a coordinator in an institution or a business who wants to register a number of their colleagues to a conference can make a separate registration for each person using their own account, on behalf of the attendees. In this case, the user can provide the name and email address of the attendee during registration, and the website will send the ticket document to the attendee's email.

2.3.3 User Registrations

Users can find all their past registrations by clicking on "My Registrations". The page displayed afterwards consists of two lists, containing user's upcoming and past registrations. The lists look similar to the figure on the next page.


Upcoming Registrations

Sixth Conference

12/20/2019 - 12/31/2019

Interdum et malesuada fames ac ante ipsum primis in faucibus. In a tortor ante. Aliquam erat volutpat. Vivamus blandit vestibulum placerat. Duis iaculis purus orci, non dignissim nisl egestas hendrerit. Phasellus eu magna in lorem tristique pretium. Phasellus vehicula congue neque, at ultricies libero tempor ut. Duis nisl ligula, tristique sed libero et, feugiat feugiat nibh. Aenean eget tellus volutpat, malesuada leo sed, sagittis lectus.

REGISTRATION FOR JOHN DOE

Sixth Conference

12/20/2019 - 12/31/2019

Interdum et malesuada fames ac ante ipsum primis in faucibus. In a tortor ante. Aliquam erat volutpat. Vivamus blandit vestibulum placerat. Duis iaculis purus orci, non dignissim nisl egestas hendrerit. Phasellus eu magna in lorem tristique pretium. Phasellus vehicula congue neque, at ultricies libero tempor ut. Duis nisl ligula, tristique sed libero et, feugiat feugiat nibh. Aenean eget tellus volutpat, malesuada leo sed, sagittis lectus.

REGISTRATION FOR GRAY GUY

Sixth Conference

12/20/2019 - 12/31/2019

Interdum et malesuada fames ac ante ipsum primis in faucibus. In a tortor ante. Aliquam erat volutpat. Vivamus blandit vestibulum placerat. Duis iaculis purus orci, non dignissim nisl egestas hendrerit. Phasellus eu magna in lorem tristique pretium. Phasellus vehicula congue neque, at ultricies libero tempor ut. Duis nisl ligula, tristique sed libero et, feugiat feugiat nibh. Aenean eget tellus volutpat, malesuada leo sed, sagittis lectus.

REGISTRATION FOR BASIR DOOST

Tenth Conference

12/18/2019 - 12/22/2019

Donec diam tortor, mattis id pulvinar quis, venenatis ac felis. Morbi molestie odio non est ullamcorper consectetur. Nullam elementum elit sed massa auctor lobortis. Praesent mattis sagittis elit, sit amet aliquam ipsum. Fusce sit amet magna risus. Maecenas quis orci urna. Nam faucibus id mauris vitae blandit. Duis in dignissim erat. Donec scelerisque sapien tellus, non congue turpis semper non.

REGISTRATION FOR BASIR DOOST

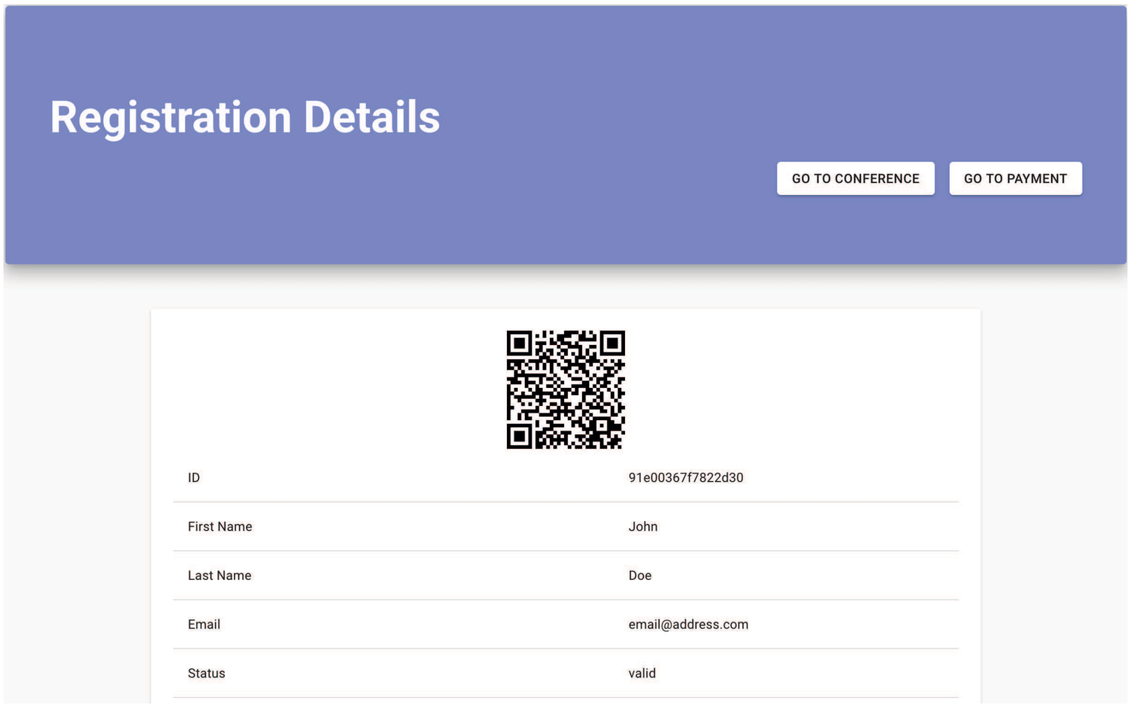
Tenth Conference

12/18/2019 - 12/22/2019

Donec diam tortor, mattis id pulvinar quis, venenatis ac felis. Morbi molestie odio non est ullamcorper consectetur. Nullam elementum elit sed massa auctor lobortis. Praesent mattis sagittis elit, sit amet aliquam ipsum. Fusce sit amet magna risus. Maecenas quis orci urna. Nam faucibus id mauris vitae blandit. Duis in dignissim erat. Donec scelerisque sapien tellus, non congue turpis semper non.

REGISTRATION FOR BASIR DOOST

By clicking on "Registration for {attendee's name}" on each list item, the user can navigate to a page containing the registration's details and QR code. This page can be used to verify the registration's provided information and contact the institution in case any of the information needs to be changed, and also to save the QR code needed for entry in case they do not have access to the ticket document that was emailed to them. Part of this page's content can be found in the figure below.


Registration Details

ID	91e00367f7822d30
First Name	John
Last Name	Doe
Email	email@address.com
Status	valid

The "Go to payment" button can be used to navigate to a page containing the details of the payment associated with this registration, displayed in a format similar to the current page.

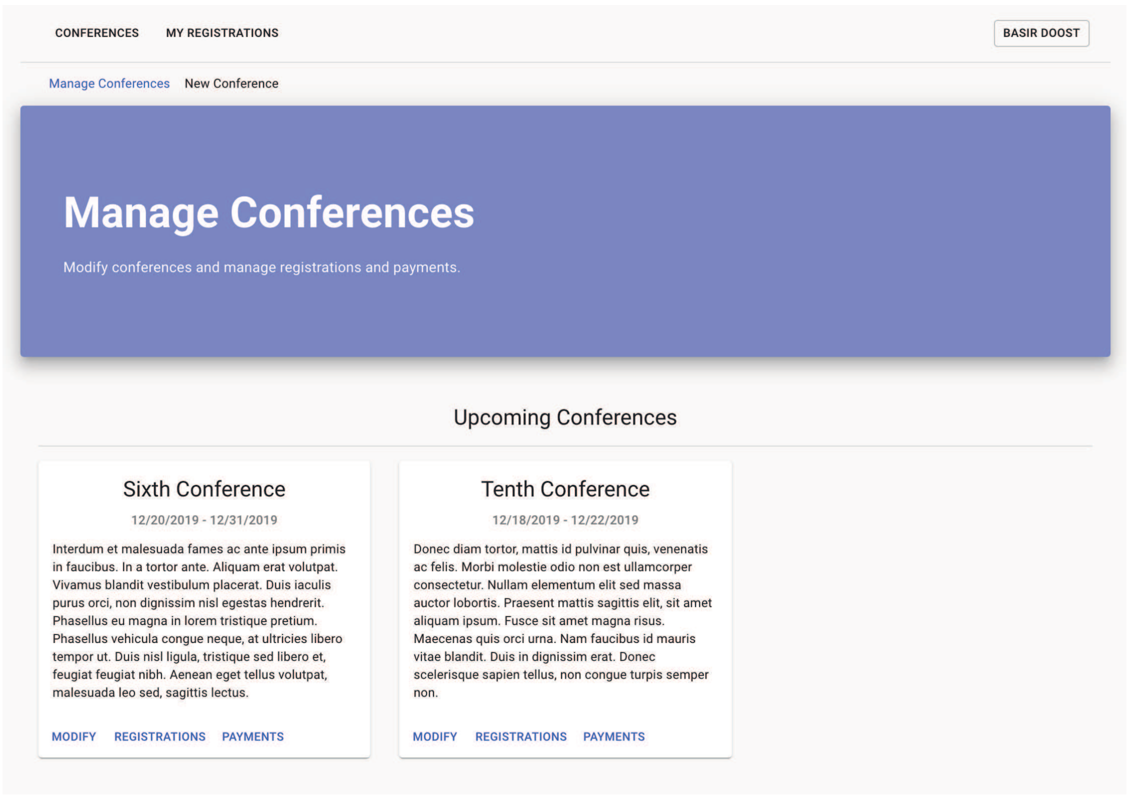
The "Go to conference" button can also be used to navigate to the details page of the conference corresponding to the registration.

2.4 Moderator Usage Information

The moderator section of the website can be accessed using the URL: `https://master.d6nxzuw99ek2a.amplifyapp.com/moderator`. Accessing this part of the website is only possible if the user is already signed into an account with moderator or admin type. The navigation bar in this section contains a button to display a list of all existing conferences to manage, and a button that navigates to the conference creation page.

2.4.1 Managing Conferences

The conference management page is displayed when the user first opens the moderator section of the website, and it can also be accessed from anywhere within the moderator section by clicking on the "Manage Conferences" button in the navigation bar. This page looks like the figure below.



Manage Conferences

Modify conferences and manage registrations and payments.

Upcoming Conferences

Sixth Conference

12/20/2019 - 12/31/2019

Interdum et malesuada fames ac ante ipsum primis in faucibus. In a tortor ante. Aliquam erat volutpat. Vivamus blandit vestibulum placerat. Duis iaculis purus orci, non dignissim nisl egestas hendrerit. Phasellus eu magna in lorem tristique pretium. Phasellus vehicula congue neque, at ultricies libero tempor ut. Duis nisl ligula, tristique sed libero et, feugiat feugiat nibh. Aenean eget tellus volutpat, malesuada leo sed, sagittis lectus.

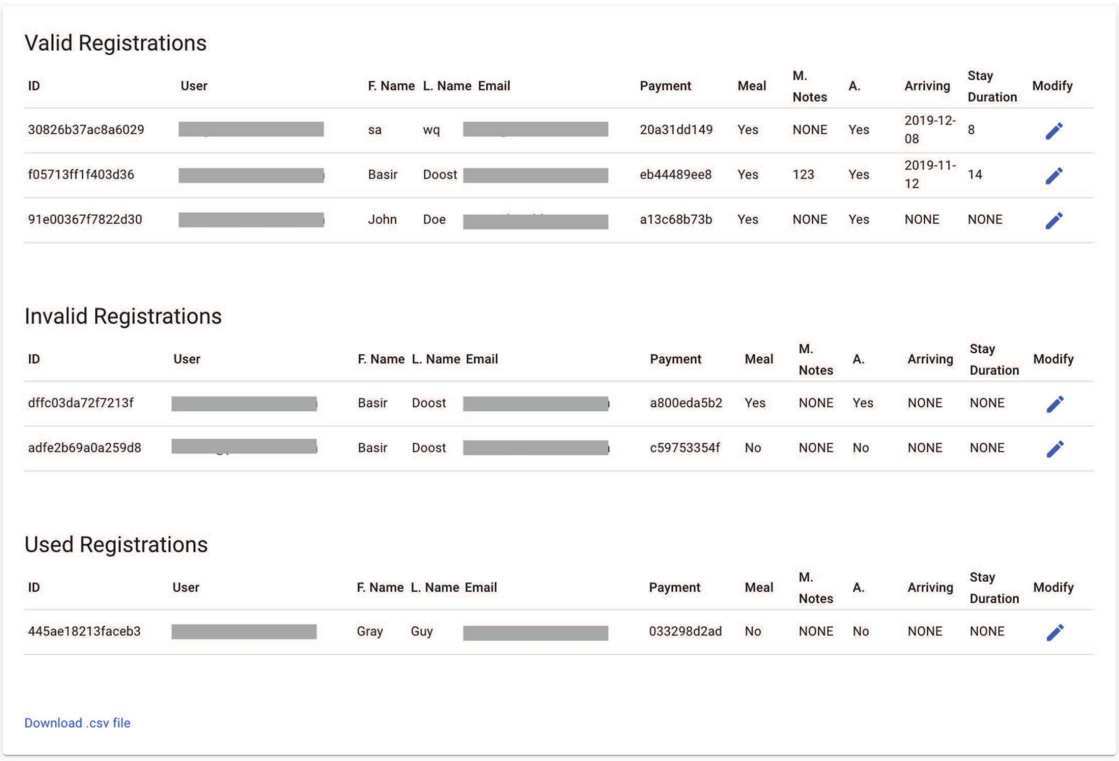
Tenth Conference

12/18/2019 - 12/22/2019

Donec diam tortor, mattis id pulvinar quis, venenatis ac felis. Morbi molestie odio non est ullamcorper consectetur. Nullam elementum elit sed massa auctor lobortis. Praesent mattis sagittis elit, sit amet aliquam ipsum. Fusce sit amet magna risus. Maecenas quis orci urna. Nam faucibus id mauris vitae blandit. Duis in dignissim erat. Donec scelerisque sapien tellus, non congue turpis semper non.

This page contains all the conferences created on the website, split into upcoming and past conference. The moderator can edit the details of a conference by clicking on "Modify" below a conference. This displays a form with all of the conference details, which the moderator can modify and submit. More information about this form and its parameters can be found in Creating a New Conference.

Clicking on "Registrations", opens a page containing a table with all of the submitted registrations for the conference. This table is depicted below.



Valid Registrations

ID	User	F. Name	L. Name	Email	Payment	Meal	M. Notes	A.	Arriving	Stay Duration	Modify
30826b37ac8a6029	[redacted]	sa	wq	[redacted]	20a31dd149	Yes	NONE	Yes	2019-12-08	8	✎
f05713ff1f403d36	[redacted]	Basir	Doost	[redacted]	eb44489ee8	Yes	123	Yes	2019-11-12	14	✎
91e00367f7822d30	[redacted]	John	Doe	[redacted]	a13c68b73b	Yes	NONE	Yes	NONE	NONE	✎

Invalid Registrations

ID	User	F. Name	L. Name	Email	Payment	Meal	M. Notes	A.	Arriving	Stay Duration	Modify
dffc03da72f7213f	[redacted]	Basir	Doost	[redacted]	a800eda5b2	Yes	NONE	Yes	NONE	NONE	✎
adfe2b69a0a259d8	[redacted]	Basir	Doost	[redacted]	c59753354f	No	NONE	No	NONE	NONE	✎

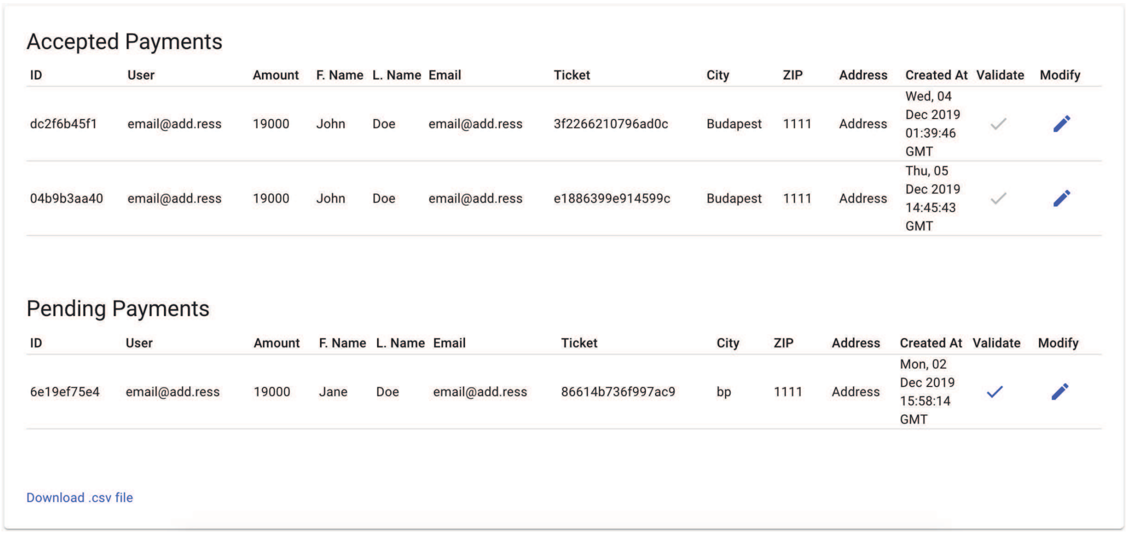
Used Registrations

ID	User	F. Name	L. Name	Email	Payment	Meal	M. Notes	A.	Arriving	Stay Duration	Modify
445ae18213faceb3	[redacted]	Gray	Guy	[redacted]	033298d2ad	No	NONE	No	NONE	NONE	✎

Download .csv file

All information associated with a registration can be found in this table, which is used by the host institution to be able to plan the conference according to the number of attendees and assist the users who requested accommodation and meals. This information can also be exported into a .csv file to be opened with Microsoft Excel or similar applications. Also, the moderator is able to modify the information of a registration by clicking on the button under the modify column of the table. This opens a dialog inside the current page, containing a form with all of the registration's data which the moderator can modify and submit.

Clicking on "Payments" below a conference also opens a page similar to the conference registrations page, with a table containing information regarding all of the payments created for that conference. This table is shown in the figure on the next page.


Accepted Payments

ID	User	Amount	F. Name	L. Name	Email	Ticket	City	ZIP	Address	Created At	Validate	Modify
dc2f6b45f1	email@add.ress	19000	John	Doe	email@add.ress	3f2266210796ad0c	Budapest	1111	Address	Wed, 04 Dec 2019 01:39:46 GMT	✓	✎
04b9b3aa40	email@add.ress	19000	John	Doe	email@add.ress	e1886399e914599c	Budapest	1111	Address	Thu, 05 Dec 2019 14:45:43 GMT	✓	✎

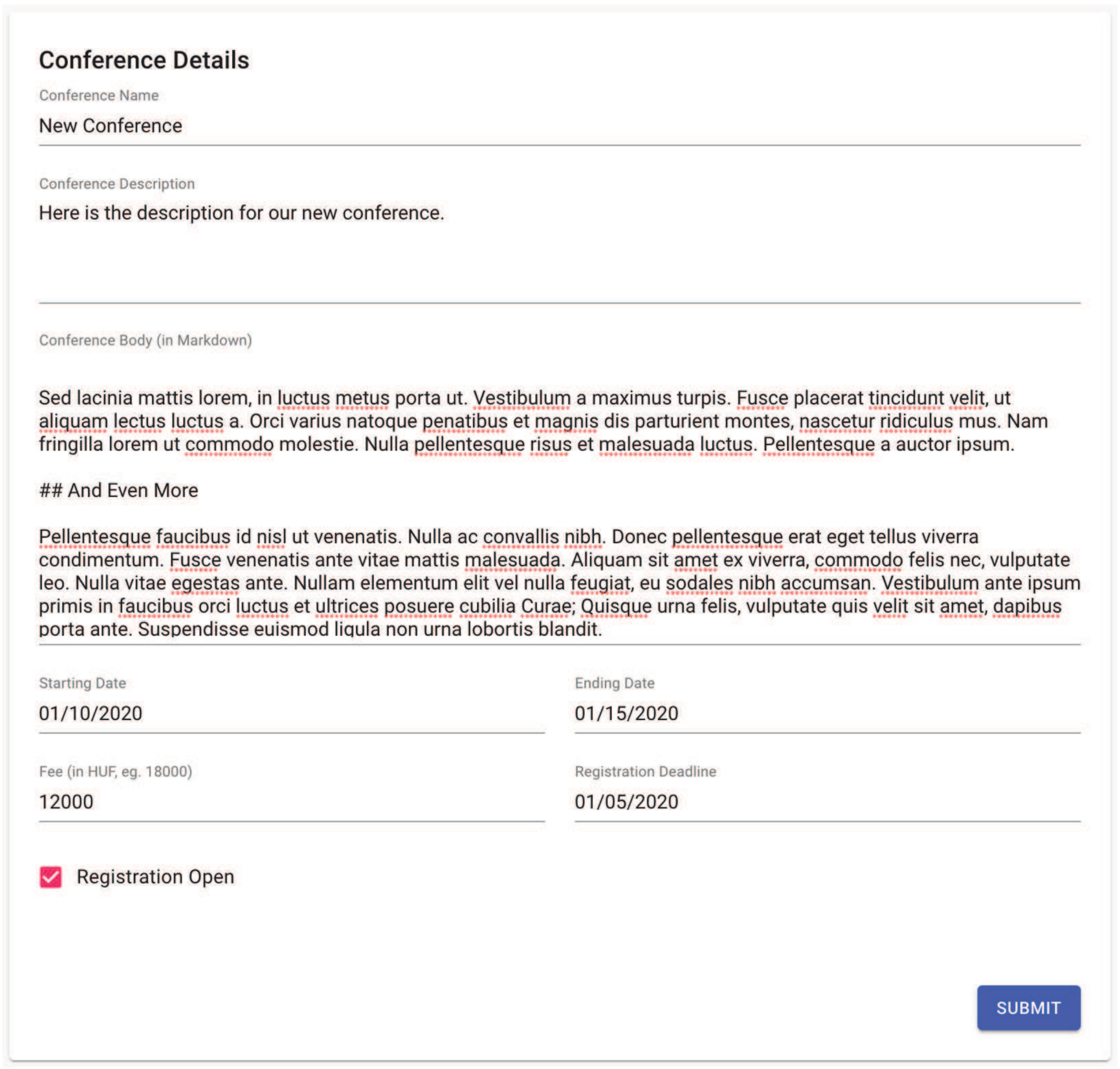
Pending Payments

ID	User	Amount	F. Name	L. Name	Email	Ticket	City	ZIP	Address	Created At	Validate	Modify
6e19ef75e4	email@add.ress	19000	Jane	Doe	email@add.ress	86614b736f997ac9	bp	1111	Address	Mon, 02 Dec 2019 15:58:14 GMT	✓	✎

The payment amount and the billing information can be found in this table. Additionally, the moderator can use this page to validate the status of a payment after confirming the bank transfer. The modify button can be used to edit the payment details when needed, such as when a user's billing information has changed, or the amount of payment needs to be changed due to a special pricing.

2.4.2 Creating a New Conference

A new conference can be created by clicking on "New Conference" in the moderator section's navigation bar, and the user is navigated to a page with a form containing the fields needed to create a conference. The moderator needs to submit a name, a description, the page content, starting and ending date of the conference, the registration fee, and the registration deadline. The name, description, and dates of the conference are displayed in the main page of the website, and the content can be styled using Markdown to be displayed in the conference details page. Fee is the amount users will be charged for a registration, and the registration deadline is the last date that registration is possible for the conference. Also the "Registration Open" checkbox can be used to control whether or not registration is open and the conference is displayed in the website. Moderators can uncheck this box to make the conference private, which hides the conference from the website and closes registration. Note that if either the deadline is passed or the conference is set to closed, users will not be able to register for the conference. This form is depicted on the next page.



Conference Details

Conference Name
New Conference

Conference Description
Here is the description for our new conference.

Conference Body (in Markdown)

Sed lacinia mattis lorem, in luctus metus porta ut. Vestibulum a maximus turpis. Fusce placerat tincidunt velit, ut aliquam lectus luctus a. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nam fringilla lorem ut commodo molestie. Nulla pellentesque risus et malesuada luctus. Pellentesque a auctor ipsum.

And Even More

Pellentesque faucibus id nisl ut venenatis. Nulla ac convallis nibh. Donec pellentesque erat eget tellus viverra condimentum. Fusce venenatis ante vitae mattis malesuada. Aliquam sit amet ex viverra, commodo felis nec, vulputate leo. Nulla vitae egestas ante. Nullam elementum elit vel nulla feugiat, eu sodales nibh accumsan. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Quisque urna felis, vulputate quis velit sit amet, dapibus porta ante. Suspendisse euismod ligula non urna lobortis blandit.

Starting Date: 01/10/2020	Ending Date: 01/15/2020
Fee (in HUF, eg. 18000): 12000	Registration Deadline: 01/05/2020

✓ Registration Open

2.4.3 Scanning a Ticket

Moderators can scan the QR code on a ticket document to validate the state of registration. For this, first the moderator has to be signed into the website on the device they are using to scan the QR code. Scanning the QR results in the following URL:

`https://master.d6nxzuw99ek2a.amplifyapp.com/moderator/qr/{ticketId}`

After opening this link, the status of the the ticket is displayed as either valid, invalid, or used. Along with this, a table containing all of the registration details is included. Scanning a valid ticket will also set its status to used, which will be shown the next time that the ticket is scanned.

Chapter 3

DEVELOPER DOCUMENTATION

3.1 Problem Specification

The goal of this project is to create a modern and user friendly platform to host and manage an institution's conferences. This platform is offered as a web application, and we have used the placeholder name "Confer" during the development process to refer to it - however this name can be adjusted to the host institution's needs for commercial usage. The main challenges of the project can be summarized as following:

- Separating the application into a "backend" part that contains all of the system's data and allows operations on them, and a "frontend" part that allows users to interact with the system.
- The application needs to be able to handle information that can be categorized into conferences, users, registrations, and payments.
- Providing a secure way of authenticating users into the system.
- Providing an accessible and fast method of data storage.
- Creating a modern web user interface.
- Maintaining a list of all conferences and sharing them on the website.
- Adding a purchase flow for registering to a conference.
- Automatically generating a PDF document as a ticket for every successful registration.

- Means to send this PDF document to the provided email address on registration.
- Offering the ability to scan and verify the authenticity of a ticket document.
- Offering a moderator interface for management.
- Keeping all billing information required for invoicing payments.
- Keeping information about whether the user requires extra services such as accommodation and meal preparation, and displaying the requests to moderators to prepare accordingly.
- Allowing a user to register other people with a different name and email address, to allow easier batch registration from businesses or institutions for their employees.
- Creating a logical structure that can be easily maintained and extended with new features.
- Using the right architectural practices to minimize Cloud resource usage and execution times to reduce costs, and to improve API response times for a quick and seamless user experience.

3.2 Project Structure

3.2.1 Overview

Confer is a web application consisting of a frontend and a backend layer. The frontend is a web client that runs on the user's browser and performs server-side operations through HTTP calls to the backend's REST API. This API is the interface through which accessing and modifying internal data is achieved - such as retrieving the list of offered conferences, or making a registration. Authentication in the website and authorization for API requests is possible by configuring a user pool that allows the application to securely create, store, and retrieve users on the backend. A Cloud Database stores the system's data about the users, conferences, tickets, and payments; and this data is consumed and updated by the API endpoints to form the usage flow of Confer.

3.2.2 Used Technologies and Methods

Backend

The backend is implemented using AWS Serverless Application Model (SAM), an open-source framework for development of serverless applications on AWS, providing all resources required [5]. SAM is an extension to CloudFormation, an Infrastructure-as-Code system which allows defining and provisioning all of the Cloud resources of the application from a single template file [6].

Having a template file define all the resources and infrastructure of the application makes configuration and management of the resources much easier, and also allows us to easily use a CI/CD pipeline for maintaining build and deployment. We use AWS CodeStar, CodePipeline, CodeBuild, and CodeDeploy for our CI/CD system. More information on this can be found in section 3.3 Build and Deployment.

For data storage we use DynamoDB; a fully-managed, scalable, and distributed NoSQL database on AWS. DynamoDB comes with built-in security and encryption features and has good integration capabilities with the rest of our stack [7].

A Cognito UserPool is used for authentication. It provides a secure and easy way to perform user registration and login, [8] and it allows the user to receive an identity token that can be used to authorize API requests [9].

The business logic is implemented as microservices through AWS Lambda, a serverless computation tool that allows execution of a function written in one of the supported programming languages - Node.js in this project - in response to events.

Frontend

The frontend is implemented using React, a web framework based on JavaScript that helps us create stylish and fast user interfaces on a web browser. Reacts follows the *Single Page Application* (referred to as SPA) design.

SPAs are a common practice in modern websites and can be seen in major products such as Gmail, Facebook, Airbnb, and many more. They function differently from static websites - such that instead of a new page being fetched from the server and rendered in the browser every time the user navigates, the application is fetched into the browser at once and the content of the page is handled dynamically using JavaScript. This approach offers a richer interaction between the user and the application, by removing a lot of loading time on user actions since we don't have to fetch a new HTML from the server on every navigation to a new page. This approach does increase initial loading time, however, this should be quite insignificant in most currently used computers and smartphones. But overall, this does not outweigh the advantage of offering a more continuous and smooth user experience.

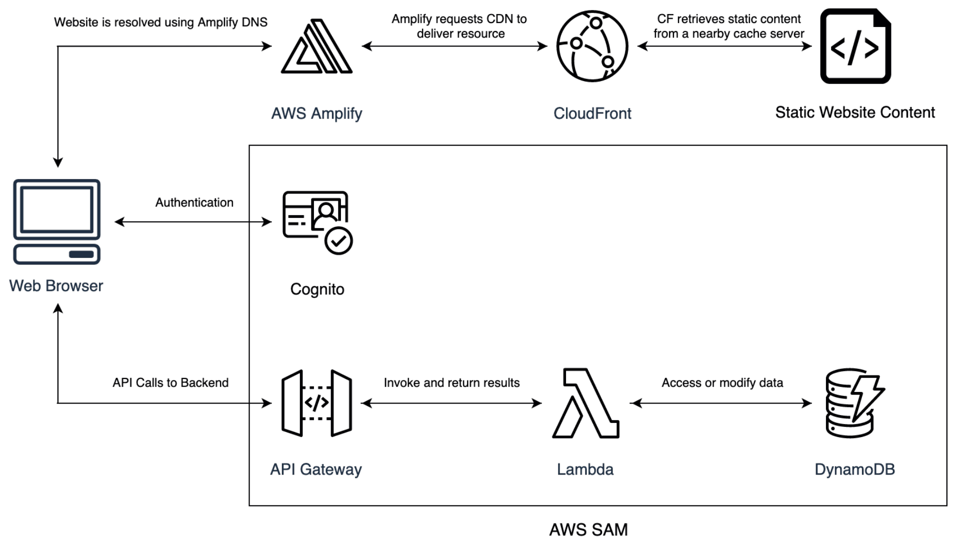
React implements this by creating a tree of elements and inserting them into a "root" node in the browser DOM. Every time a React element on the UI changes, that element and its children are re-rendered and the tree is modified according to that. To display those changes on the browser, React uses the package React-DOM that performs a diff operation to find what actually changed between what is currently on the UI and the new tree, and updates only what's necessary [10]. This process is referred to as Reconciliation, and uses an algorithm based on a list of heuristics for common use cases to update the UI faster while keeping updates predictable [11].

React uses a syntax extension called JSX, which resembles HTML and is used to define UI elements. JSX is not simply an HTML representation, as it supports JavaScript language capabilities and allows a loose coupling between the JavaScript logic of the UI and the UI element itself in each React component. This way, the markup and the logic of UI components do not have to be separated which improves the development experience in React [12].

We also utilize AWS Amplify for the frontend in this project. Amplify is the collection of a development framework and other developer services that provide a fast and easy way to create mobile and web applications. Amplify offers tools to control AWS resources to create fullstack applications [13], although in this project we are only using Amplify in our frontend to interact with the backend infrastructure, and to deploy the frontend.

3.2.3 Abstract Architecture of Resources

In this section we describe the resources of the project and how they interact with each other to create the functionality of the application. The figure below demonstrates an overall view of the project's abstract structure.


Website is resolved using Amplify DNS
Amplify requests CDN to deliver resource
CF retrieves static content from a nearby cache server
AWS Amplify
CloudFront
Static Website Content
Authentication
Web Browser
Cognito
API Calls to Backend
Invoke and return results
Access or modify data
API Gateway
Lambda
DynamoDB
AWS SAM

When the user opens the website in their browser, Amplify retrieves the static website content in form of HTML, JavaScript, and CSS files from a nearby available cache server in the CloudFront network, and returns it to the user.

When the website is loaded in user's browser, it can directly communicate with our Cognito user pool to authenticate users and retrieve a token validating their identity.

The website sends HTTP requests to the REST API to interact with the backend system - supplying the identity token received from Cognito if necessary, and these requests are handled by API Gateway. Using Lambda integration, API Gateway invokes the Lambda microservice with an event containing the request properties. Lambda processes the request, by retrieving and modifying data from our database when needed. After processing the request, Lambda returns a response to API Gateway; and API Gateway in turn creates an HTTP response corresponding to the response received from Lambda and sends it to the user's browser. Finally, this response in displayed as part of the website in user's browser.

3.3 Build and Deployment

The source code of the application is stored on the "Confer" and "confer-website" repositories on GitHub, for the backend and frontend projects respectively. All of the application's resources are deployed on AWS, and we use a set of deployment tools to publish changes from the master branch of the repositories to the live deployment of the application.

3.3.1 Backend Deployment Process

Since all the resources of the backend part are defined using CloudFormation (CF), we can also use CF for provisioning those resources using what is known as "stacks". A stack contains all the resources defined in the CF template file and can be managed by CF. In this project, the CloudFormation template is defined within the "template.yml" file. This file essentially contains the configuration of all the backend resources.

Continuous Integration and Continuous Delivery are achieved using CodeStar. The CodeStar console provides a unified interface to manage the backend's build and deployment process, and uses CodePipeline to create a pipeline for deployment to production. The pipeline in our project consists of the Confer GitHub repository as the source, a build stage, and a deploy stage. CodePipeline connects to the GitHub repository, and whenever a change is pushed to the master branch, the pipeline stores the code in a source artifact on AWS S3 and starts executing the build and deploy stages.

CodeBuild handles the build stage by retrieving the source artifact made by CodePipeline in the last step, and executing the build according to the instructions from the "buildspec.yml" file. The build stage is made up of four phases: install, pre-build, build, and post-build. In this stage, we first install any dependencies we might have using npm, and we also install the "services" package in the "LambdaLayers/services/nodejs/" directory to be in the correct folder structure to be used as a Lambda Layer. Then, we package the application from the template.yml using the AWS CloudFormation CLI, and create a build artifact with a CF export template that is ready to be deployed.

CodeDeploy handles the deployment by first using a CloudFormation GenerageChangeSet action on the CF stack and the build artifact created in the previous stage. This actions compares the current stack with the new artifact, and creates a JSON representation of all the changes of the stack's resources. Finally, an ExecuteChangeSet action is triggered, which uses the previously generated change set to update the stack's resources and release the changes to production.

3.3.2 Frontend Deployment Process

Amplify offers a smooth development experience, by handling the whole deployment process of the frontend and producing a versioned and robust final deployment of the website - with a few simple configuration steps. It offers a complete set of CI/CD tools, supports build and deployment of React projects by default, and hosts the website itself through a Content Delivery Network (CDN).

CloudFront is used by Amplify as a CDN to host the static content of the website. It caches the website content in many servers in different regions of the world to deliver to end users, which results in low latency and high availability for the website [14].

Amplify is connected to the "confer-website" GitHub repository and every time a change is pushed to the master branch, it creates a new build using the instructions in the "amplify.yml" file. The build process is fairly simple, and consists of a "npm ci" command in and a "npm run build" command to perform a clean install on all the project dependencies and to build the application using the build script defined in the "package.json" file. Finally, the build artifacts are used to deploy the website to Amplify.

3.4 Project Resources

In this section we will provide detailed information on each of the resources of the application and how they function.

3.4.1 User Pool

We created a user pool for this project on Cognito that handles all of the authentication needs of the project. This user pool securely stores the user authentication data on Amazon servers, and the website can interact with it using the Amplify JavaScript framework to perform actions such as signing in, creating an account, or resetting a forgotten password. It also takes care of securely storing the login session on user's browser automatically, and refresh the session if necessary.

The user's email address is used as the username to make registration and signing in easier for users. Also, the user must provide their full name when registering, and the password must be at least 8 characters long and include uppercase and lowercase letters and at least one digit.

3.4.2 Databases

We use four DynamoDB tables to store the system's data: Users, Conferences, Tickets, and Payments. Details about each of these tables can be found below. Wherever a date is used, the format is considered to be the ISO 8601 Date format i.e. "YYYY-MM-DD". The id column is set as the primary key for all the tables.

On a simple table, retrieving an item or items is only possible by specifying the primary keys or scanning all of the items. To be able to perform a query using conditions on a different column, we need to have a Global Secondary Index (GSI) on the table for that column [15]. The Tickets and Payments tables each have a GSI on the "conference" column. This allows us to query the table with to retrieve all the ticket or payment items which match a given conference ID value. This method can be seen in use in the Confer moderator console.

Users

The Users table holds all data that we need to store for each user. Currently, it stores each user's name, user type, and their purchased tickets. The columns of this table can be found below.

Column	Data Type	Description
id	String	User ID
family_name	String	User family name

Column	Data Type	Description
given_name	String	User given name
conferences_attending	List	A list containing a map for each user ticket

Notes: ID of a user is considered to be their email address used when creating their account. Also in the conferences_attending column, each ticket is stored in the list as map containing two keys "id" and "conference", holding the ticket ID and the conference ID respectively.

Conferences

The Conferences table holds data regarding each conference. It stores all information needed to display and register for a conference.

Column	Data Type	Description
id	String	Conference ID
name	String	Conference name
description	String	Conference description
page_content	String	Main content of the conference page
fee	Number	Registration fee (HUF)
starts_at	String	Starting date
ends_at	String	Ending date
deadline	String	Registration deadline date
is_open	Boolean	Is conference shown on website

Notes: ID of a conference is the string representation of a 24 bit number in base 16. The page content column is meant to contain the content in Markdown format to allow customizing style in the editor. The is_open property can be set to false to hide conference from the website homepage and also close registration.

Tickets

The Tickets table holds data regarding the tickets created on the system. It stores information about the owner of the ticket and their submitted preferences during purchase, and also the status of the ticket.

Column	Data Type	Description
id	String	Ticket ID
conference	String	ID of the conference corresponding ticket
user	String	ID of submitting user

Column	Data Type	Description
given_name	String	Ticket owner's given name
family_name	String	Ticket owner's family name
email	String	Ticket owner's email
status	String	Ticket status; "valid", "invalid", or "used"
payment_id	String	ID of the corresponding payment
created_at	String	Date of invoice
requested_meal	Boolean	Whether user requested meal preparation
meal_notes	String	Notes for meal
requested_accommodation	Boolean	Whether user requested accommodation help
arriving_time	String	Date of arrival in case of accommodation
stay_length	String	Duration of stay in case of accommodation
entered_at	String	Time of user entrance scan

Payments

The Payments table holds data regarding the payments created on the system. It stores the billing information, payment amount, and the status of the payment.

Column	Data Type	Description
id	String	Payment ID
conference	String	ID of the conference payment is made for
user	String	ID of the user submitting the paper
given_name	String	Billing given name
family_name	String	Billing family name
email	String	Billing email
address	String	Billing address
postalcode	String	Billing postal code
city	String	Billing city
country	String	Billing country
amount	String	Billing city
conference_name	String	Name of the conference payment is made for
ticket_id	String	ID of the corresponding ticket
type	String	Payment type; currently only "transfer"
status	String	Payment status; "waiting" or "accepted"
transactions	Array	Online transaction IDs; Currently only empty
created_at	String	Date of invoice

3.4.3 Lambda Functions

The Lambda functions are where all of the backend operations of the application is processed. All of the Lambda functions in the project except "ConferPostUserVerification" and "ConferSendTicketAsEmail" are used to process requests on an API endpoint, and we refer to them collectively as API Lambdas for the sake of simplicity.

We have created a package called "services" to be used by the API Lambdas to define all the valid operations within the system. It provides the main functionalities and the logic of the application, and allows performing actions on the whole system while keeping system's information consistent. Functions in one of the services can also use functions from another service to implement their behaviour. This package contains a set of Service classes, listed below:

- IdService: Used to generate IDs compliant with our system for conferences, tickets, and payments.
- ResponseService: Used to create a success or error response in a format compatible with API Gateway's Lambda integration to be returned from Lambdas.
- EventService: Used to retrieve information and supplied parameters in the API request from the event object passed to Lambda
- ConferenceService: Used to perform defined operations that mainly involve the conferences on the system, such as creating a new conference or retrieving overviews for all current conferences.
- PaymentService: Used to perform defined operations that mainly involve the payments on the system, such as retrieving a payment's details or validating one.
- TicketService: Used to perform defined operations that mainly involve the tickets on the system, such as creating a ticket or scanning its QR code.
- TicketService: Used to retrieve information about a user, such as getting a user's type or full name.

A detailed API documentation of this package containing all functions, their exact behaviour and structure, and their required parameters and response type can be found in Appendix A.

The services package is used inside the API Lambdas through Lambda Layers. Lambda Layers is a feature of Lambda that allows sharing code between different Lambdas without including the code in each one's deployment package [16]. This way, the services package is deployed once to AWS as a layer, and the same layer is included in the API Lambdas so that they can use the services package just as they normally would with a package installed in the node_modules directory.

We have also created another layer used by ConferSendTicketAsEmail to provide the dependencies we use for ticket PDF document generation. Although these packages are not shared between different Lambdas, it is still useful to use Layers for this purpose since this keeps the Lambda deployment package small.

A list of all the Lambda functions and their functionality can be found below.

API Lambdas

The API Lambdas take their input values from the event object passed to them when they are called by API Gateway. This object contains the API request's body and path parameters, and also in case of protected endpoints, claims from the authorizer containing information about the identity of the user who made the request. We extract request information from this event using the EventService class. Functions that require the user ID or a specific user type will return with an error if the user ID is missing or user does not have permission for the given operation. Similarly, the function will return with an error if a required body property or path parameter is missing. We also define some body properties as optional, and if these properties are missing from the API request they will be evaluated to string "NONE".

They implement their functionality using a combination of the functions provided by the services package, and return a standard response using the ResponseService module at the end of their execution. The function returns the following after a successful execution:

```
{
  statusCode: 201,
  body: JSON.stringify({
    Result: result,
    Reference: awsRequestId
  }),
  headers: {
    'Access-Control-Allow-Origin': '*',
  },
}
```

And in case of an error during the execution:

```
{
  statusCode: 500,
  body: JSON.stringify({
    Error: errorMessage,
    Reference: awsRequestId,
  }),
  headers: {
    'Access-Control-Allow-Origin': '*',
  }
}
```

JSON.stringify in the body denotes that the JavaScript object given as the parameter is converted into its JSON string representation in order for it to be transferable inside the HTTP response body. These return values are parsed by API Gateway to form the HTTP response of API endpoints. The Result property can contain any JavaScript object, to be able to return responses of different types for different endpoints; however, the Error property is meant to be used to send the error message as a string.

ConferGetConferenceDetails

This function returns details for a conference given in the event's path parameters. This operation can be used by guests.

Requires Path Parameter: conferenceId

Result type ConferenceService~ConferenceDetails

ConferGetCurrentConferences

This function returns a list of overviews for conferences that are currently available for registration, meaning the conference is open and registration deadline has not passed. This operation can be used by guests.

Result type Array.<ConferenceService~ConferenceOverview>

ConferGetOpenConferences

This function returns a list of overviews for all publicly open conferences, even if registration deadline has passed. This operation can be used by guests.

Result type Array.<ConferenceService~ConferenceOverview>

ConferModeratorCreateConference

This function creates a conference in the system with the given properties. This operation can only be used by moderators.

Requires Body Parameters: name, description, page_content, fee, starts_at, ends_at, is_open, deadline

Result type `{id, name}` - Created conference's ID and name

ConferModeratorGetAllConferences

This function returns a list of overviews for all conferences on the system, including the ones that are not open. This operation can only be used by moderators.

Result type Array.<ConferenceService~ConferenceOverview>

ConferModeratorGetConferencePayments

This function returns a list of all payments for a given conference. This operation can only be used by moderators.

Result type Array.<PaymentService~PaymentDetails>

ConferModeratorGetConferenceTickets

This function returns a list of all tickets for a given conference. This operation can only be used by moderators.

Result type Array.<TicketService~TicketDetails>

ConferModeratorModifyPayment

This function modifies the properties of the payment with the provided ID. This operation can only be used by moderators.

Requires Path Parameter: paymentId

Requires Body Parameters: given_name, family_name, email, amount, city, postalcode, country, address

Result type `{given_name, family_name}` - Updated payment's name properties

ConferModeratorModifyTicket

This function modifies the properties of the ticket with the provided ID. This operation can only be used by moderators.

Requires Path Parameter: ticketId

Requires Body Parameters: given_name, family_name, email, requested_accommodation

Optional Body Parameters: meal_notes, arriving_time, stay_length

Result type `{given_name, family_name}` - Updated ticket's name properties

ConferModeratorScanQR

This function is used for scanning a ticket's QR. It takes the ticket ID and gives back the ticket information to moderator and sets its state to used. This operation can only be used by moderators.

Requires Path Parameter: ticketId

Result type TicketService~ScanResult

ConferModeratorUpdateConference

This function is used to update the properties of the conference with the provided ID. This operation can only be used by moderators.

Requires Path Parameter: conferenceId

Requires Body Parameters: name, description, page_content, fee, starts_at, ends_at, is_open, deadline

Result type `{id, name}` - Updated conference's ID and name

ConferModeratorValidateTransferPayment

This function is used to confirm a payment by bank transfer has been made. It sets the state of the payment and its corresponding ticket to valid, adds the ticket to user's profile, and at the end of its execution, invokes Lambda function ConferSendTicketAsEmail asynchronously to send the ticket document to its owner. This operation can only be used by moderators.

Requires Path Parameter: paymentId

Requires Body Parameters: name, description, page_content, fee, starts_at, ends_at, is_open, deadline

Result type `{paymentId, status}` - The updated paymentId and its status containing string "valid".

ConferPreparePayment

This function is used when submitting a registration for a conference, and creates a new payment and ticket in the system, with status "waiting" and "invalid" respectively. This operation can be used by all registered users.

Requires Path Parameter: conferenceId

Requires Body Parameters: given_name, family_name, email, invoice_given_name, invoice_family_name, invoice_email, invoice_address, invoice_postalcode, invoice_city, invoice_country, payment_type, requested_meal, requested_accommodation

Optional Body Parameters: meal_notes, arriving_time, stay_length

Result type `{conference, user, payment_id, given_name, family_name}` - ID of the conference, user, and the newly created payment, and the newly created ticket's name properties.

ConferUserGetConferences

This function returns all of the tickets in the user's profile. This operation can be used by all registered users.

Result type `{id, conference, given_name, family_name}` - An object containing ID of the the ticket, overview of type ConferenceService~ConferenceOverview for the conference corresponding to ticket, and the ticket's name properties.

ConferUserGetPaymentDetails

This function returns details for a payment given in the event's path parameters. Only the owner of the payment can access the result, and the function return with an error otherwise.

Requires Path Parameter: paymentId

Result type PaymentService~PaymentDetails

ConferUserGetTicketDetails

This function returns details for a ticket given in the event's path parameters. Only the owner of the ticket can access the result, and the function return with an error otherwise.

Requires Path Parameter: ticketId

Result type TicketService~TicketDetails

ConferUserGetUserType

This function returns the user type of the currently signed in user. This operation can be used by all registered users.

Result type `{user_type}` - User type of the sender of the API request.

Helper Lambdas

ConferPostUserVerification

This function is used as a post-confirmation trigger for our Cognito user pool. It is invoked by Cognito after a user completes their registration by confirming their email address. It adds the new user to the Users database table with "basic" user type and their registration information.

ConferSendTicketAsEmail

Upon invocation, this function creates a PDF file as the ticket document and sends it to the email address provided on the ticket. It is meant to be invoked as a separate microservice after the ticket is added to the user's profile. We use npm packages "qrcode", "pdfkit", and "nodemailer" in this function through the Lambda layer ticketemailtools. The qrcode package is used to generate a QR code containing a link to the moderator scan page on the website with the ticket's ID. Next, pdfkit is used to generate a PDF document using a static template and the previously generated QR code. Finally, this PDF document is emailed to the email address on the ticket (not the user's email) using the nodemailer package and throught the AWS SES mailing service.

This function takes parameters id, name, and conference_id in its event, representing the ticket ID, ticket owner's full name, and the conference ID respectively.

3.4.4 API Endpoints

API Gateway is used to create the REST API of the backend which makes communication between the website's frontend and backend possible. API Gateway uses Lambda Integration to forward the API call's HTTP request to the Lambda function, and send the results back to the client. CORS is also enabled through API Gateway to allow access to our API from the website's origin.

One of the important features of API Gateway for our project is the integration with our Cognito user pool. A Cognito Authorizer has been set up on Confer's API, which guards our protected API endpoints from unauthorized access. On protected endpoints, API Gateway checks the "Authorization" header of the sent request and rejects the request if it is not present. A valid Authorization header on a request is the identity token returned from Cognito for the Confer user pool. API Gateway decodes this token and verifies its signature using the user pool's public RSA key, to verify the identity of the request sender. After receiving the identity information, it attaches them to the request before sending it to the Lambda function. This way, we can find the user ID of the request sender, to enforce user permissions on API requests.

Using Lambda integration, when an HTTP request is received by an endpoint, API Gateway creates an event containing all the necessary information for Lambda - including the request headers, body, and path parameters, and results from the Cognito Authorizer - and invokes the corresponding Lambda function with that event. After the Lambda returns with a successful or failed response, API Gateway parses it into a valid HTTP response with the correct response status code, the necessary CORS headers, and a string representing a JSON object containing the invocation result or error; and finally sends that response to the submitter of the request.

A list of all API endpoints and their details can be found below. All endpoints require the Authorization header, unless specified otherwise.

- **/conferences** - *GET:* ConferGetCurrentConferences (No authorization required)
  - **/open** - *GET:* ConferGetOpenConferences (No authorization required)
  - **/{conferenceId}** - *GET:* ConferGetConferenceDetails (No authorization required)
    - **/payments** - *POST:* ConferPreparePayment

- **/moderator**
  - **/conferences** - *GET:* ConferModeratorGetAllConferences, *POST:* ConferModeratorCreateConference
    - **/{conferenceId}** - *PUT:* ConferModeratorUpdateConference
      - **/payments** - *GET:* ConferModeratorGetConferencePayments
      - **/tickets** - *GET:* ConferModeratorGetConferenceTickets
  - **/payments**
    - **/{paymentId}** - *PUT:* ConferModeratorModifyPayment
  - **/qr**
    - **/{ticketId}** - *GET:* ConferModeratorScanQR
  - **/tickets**
    - **/{ticketId}** - *PUT:* ConferModeratorModifyTicket
  - **/transfer_payments**
    - **/{paymentId}** - *PUT:* ConferModeratorValidateTransferPayment
- **/user**
  - **/conferences** - *GET:* ConferUserGetConferences
  - **/payments**
    - **/{paymentId}** - *GET:* ConferUserGetPaymentDetails
  - **/tickets**
    - **/{ticketId}** - *GET* ConferUserGetTicketDetails
  - **/type** - *GET:* ConferUserGetUserType

3.4.5 Website

The website is a user interface made using React that allows users to interact with Confer. React projects are made up of components that can be defined as ES6 classes or functions, and in this project we have chosen the functional components approach. In earlier versions of React, using class components offered states and lifecycle hooks that were missing in functional components; but since the introduction of React Hooks, these features can be achieved using useState and useEffect hooks in functional components.

We use the Material-UI as one of the main dependencies of the website. Material-UI is a framework that provides React components based on the popular open-source

Material design system, and helps us create a modern and stylish user interface efficiently.

The main files in the project are index.html, index.js, App.js, Moderator.js, and the views, components, and Services directories. We will explain how each of these affect the application.

The index.html file is the entry point of the project. It has a body with only an empty div element with id "root", and contains the index.js file as a script. When the index.js file is loaded into the user's browser, it renders the App component into the element with id "root" in the index.html file by modifying the browser DOM using ReactDOM. The App component is the parent component of our React application. It includes the main navigation bar of the website which is present in all pages, and it defines the body of the web page depending on the URL entered in the browser using React router. The router checks if the URL entered in the browser matches any of the defined routes, and renders the component of the matching route, or the not found page if the URL didn't match any of the routes. As an example, when opening URL `https://master.d6nxzuw99ek2a.amplifyapp.com/conferences/all` the URL matches the route `/conferences/all`, and React router renders the associated component AllConferencesPage in the body.

In the App component there is also a route defined for `/moderator`, which renders the ModeratorApp component. ModeratorApp is meant to be a main component over the moderator section of the application, and includes its own Router to redirect to specific pages of the moderator section. In this component we have a "userType" state, and when the component is first mounted - without displaying the moderator section just yet - we first check the logged in user's type by sending an API call to our backend endpoints. Once we receive the response, we update the userType state, which re-renders the component with the new state. Now if the user type is moderator or admin, the component renders the requested moderator page; and otherwise the user does not have permission to access the moderator section and will be redirected to a not authorized page.

The files inside the views directory each contain a React component corresponding to the body content of a page in the application. A view component is the parent component of the content of a page, and defines its user interface using Material-UI elements and our custom components defined in the components directory.

In the components directory, we have defined a set of components specific to this project using Material-UI that are used to display a specific part of a page's data, or self-contained dialogues or forms that implement a certain part of the website's functionality and user interface. The aim was to encapsulate any certain functionality of the website in a separate component to allow easier maintenance and re-usability in the code.

The Services directory contains two service modules ”AuthService” and ”ApiService”. They both use the Amplify framework as a dependency and create a connection to the backend of our application. AuthService is used for interacting with our Cognito user pool, to receive the identity token of the currently signed in user or to perform operations such as signing in or creating a new account. ApiService contains three classes: ConferencesApi for guest actions, UserApi for user actions, and ModeratorApi for moderator actions. Each contains functions for sending an HTTP request to one of our backend REST API endpoints, and they can be used by our React components to retrieve data from the backend or to perform operations.

The website can be started locally by navigating to the confer-website repository using the terminal and first running the command `npm install` to install all of the project's dependencies, and next running `npm start` to build the website and start a local deployment on localhost. Now, the website can be used by navigating to `http://localhost:3000/` in a browser.

All pages of the website and their corresponding URL can be found below. The notation `/:parameter` in a URL defines the route parameter of the route. As an example, the `/conference/:conferenceId` opens the Conference Details page for the conference with ID equal to `:conferenceId`. Also, the terminology ”Ticket” from the backend has been changed to ”Registration” in the website design to better resonate with users; however this does not cause any inconsistencies since the frontend and backend projects are encapsulated from each other. All of the pages of the application require the user to be signed in, unless specified otherwise in the list below.

- **AvailableConferencesPage:** Used to display conferences available for registration. This page is considered as the main page of the application, and can be used by guests.

  Route: `/` or `/conferences`

- **AllConferencesPage:** Used to display all publicly open conferences. This page can be used by guests.

  Route: `/conferences/all`

- **ConferenceRegisterPage:** The registration page containing the forms required for registering to a given conference.

  Route: `/conferences/:conferenceId/register`

- **RegistrationsPage:** Used to display the current user's successful registrations.

  Route: `/user/registrations`

- **RegistrationDetailsPage:** Used to display a given registration's details.

  Route: `/user/registrations/:registrationId`

- **PaymentDetailsPage:** Used to display a given payment's details.

  Route: `/user/payments/:paymentId`

- **ModeratorConferencesPage:** This page displays all open and non-open conferences on the system and allows modification and moderation capabilities for them. This page is considered as the main page of the moderator section of the website.

  Route: `/moderator /moderator/conferences`

- **ModeratorNewConferencePage:** This page is used to create a new conference on the system.

  Route: `/moderator/conferences/new`

- **ModeratorModifyConferencePage:** This page is used to modify details of an existing conference.

  Route: `/moderator/conferences/:conferenceId`

- **ModeratorConferenceRegistrationsPage:** This page is used to display information about all the registrations submitted for a given conference. It also provides the ability to modify any of those registrations.

  Route: `/moderator/conferences/:conferenceId/registrations`

- **ModeratorConferencePaymentsPage:** This page is used to display information about all the payments submitted for a given conference. It also provides the ability to modify any of those payments.

  Route: `/moderator/conferences/:conferenceId/payments`

- **ModeratorConferencePaymentsPage:** This page is meant to be used in the QR code of the ticket, such that an admin can open it after scanning the QR code to check the validity of a ticket upon user entrance to a conference.

  Route: `/moderator/qr/:registrationId`

- **NotAuthorizedPage:** This page is displayed when a signed in user with type basic attempts to open any of the valid moderator pages.

- **NotFoundPage:** This page is used when the route entered is invalid and does not correspond to any of the website pages, or when a guest user tries to access any of the protected parts of the website that require the user to be signed in.

Chapter 4

CONCLUSION

With the level of quality that is currently offered by popular websites and applications, there is no doubt that a successful solution needs to comply with current standards to be able to satisfy users. In this project we achieved our goal of creating a fast and elegant web application, and delivering the promised functionalities. The approaches chosen in this project are far from being the most straightforward solution for creating a web application; however, they provide a solid and scalable infrastructure, and a structured development process needed for large scale complex applications.

AWS provided many useful services to make this project possible, and helped tremendously with the development process by encapsulating much of the server configuration and allowing us to focus on development of the application itself. Apart from that, using AWS allowed us to run a live deployment of the application without incurring any costs. Since many of the used services were released not too long ago, they have not yet been widely adopted by the community, so there is relatively fewer articles and sources of information that can be found for them apart from the official documentation. Due to this, some of these tools - especially the less explored sections of them - have a steep learning curve, and it might take a while to get familiar with their definitions and methods.

Because of the full-stack nature of this project, many core concepts in different fields of Computer Science had to be thoroughly explored to be able to have a clear vision of each used component's behaviour, resulting in valuable personal learnings on web development. During the development process of this project, extensive background research was done on foundation topics such as NoSQL databases, the HTTP protocol, REST APIs, DevOps, and the functionality of web browsers - to be able to utilize the components of the application to their full potential.

APPENDICES

Appendix A

Services Package API Documentation

The services package contains all of the service classes used in Confer backend. This package is used as a layer in Lambda functions to provide their main functionalities and create a consistent interface through which the data of the system is accessed an modified - such as a standard format to return responses, and perform operations to modify or retrieve data on the database. In this appendix, we provide detailed information about the classes and functions that each of the modules in this package contains.

A.1 services module

The services module is defined in the index.js file, which is set to be the 'main' module in the package using the package.json file. This means that this module's exports will be returned as a result of using require on the services package. This module exports an object used as a namespace containing all the other service modules.

Example usage: `const ResponseService = require('services').ResponseService;`

- services
  - .ConferenceService ⇒ ConferenceService
  - .EventService ⇒ EventService
  - .IdService ⇒ IdService
  - .PaymentService ⇒ PaymentService
  - .ResponseService ⇒ ResponseService
  - .TicketService ⇒ TicketService
  - .UserService ⇒ UserService

services.ConferenceService ⇒ ConferenceService

Exports the ConferenceService class

services.EventService ⇒ EventService

Exports the EventService class

services.IdService ⇒ IdService

Exports the IdService class

services.PaymentService ⇒ PaymentService

Exports the PaymentService class

services.ResponseService ⇒ ResponseService

Exports the ResponseService class

services.TicketService ⇒ TicketService

Exports the TicketService class

services.UserService ⇒ UserService

Exports the UserService class

A.1.1 IdService

This module creates IDs to be used by the rest of the application.

IdService.generate(bytes) ⇒ string

Creates a hexadecimal identifier with the given number of bytes.

Kind: static method of IdService

Param	Type
bytes	number

A.1.2 ResponseService

This module provides a way to generate a valid success or error response with API Gateway integration from Lambda functions.

- ResponseService
  - .error(errorMessage, awsRequestId) ⇒ Object
  - .success(result, awsRequestId) ⇒ Object

ResponseService.error(errorMessage, awsRequestId) ⇒ Object

Returns an error response object with a 500 status code and the given error message in the body. The response body also contains awsRequestId for debugging.

Kind: static method of ResponseService

Param	Type
errorMessage	string
awsRequestId	string

ResponseService.success(result, awsRequestId) ⇒ Object

Returns a successful response object with a 201 status code and the given result in the body. The response body also contains awsRequestId for debugging.

Kind: static method of ResponseService

Param	Type
result	any
awsRequestId	string

A.1.3 EventService

This module provides Lambda functions with the ability to parse and extract data from the API request event. It includes three classes Event, Body and PathParams. Event is the only class that is exported from this module, and the PathParams and Body classes can be accessed using the PathParams() and Body() functions of the Event class.

- EventService
  - ~Event
    - new Event(event)
    - .getAuthenticatedUser() ⇒ string
    - .PathParams() ⇒ PathParams
    - .Body() ⇒ Body
  - ~Body
    - new Body(body)
    - .getFor(key) ⇒ string
    - .getForOptional(key) ⇒ string
  - ~PathParams
    - new PathParams(pathParams)
    - .getFor(key) ⇒ string

EventService~Event

A class representing an API request event sent to a Lambda function. This class is exported from the module and can be accessed as a static member of EventService. Example usage: const event = new EventService.Event(event);

Kind: inner class of EventService

- ~Event
  - new Event(event)
  - .getAuthenticatedUser() ⇒ string
  - .PathParams() ⇒ PathParams
  - .Body() ⇒ Body

new Event(event) Constructor for the Event class.

Param	Type
event	object

event.getAuthenticatedUser() ⇒ string Returns the current authenticated user's ID.

Kind: instance method of Event

event.PathParams() ⇒ PathParams Returns an instance of the PathParams helper class instantiated with the event data from the parent Event object. Example usage: const pathParams = new EventService.Event(event).PathParams();

Kind: instance method of Event
Returns: PathParams - An instance of the PathParams class made from this.event.pathParameters

event.Body() ⇒ Body Returns an instance of the Body helper class instantiated with the event data from the parent Event object. Example usage: const pathParams = new EventService.Event(event).PathParams();

Kind: instance method of Event
Returns: Body - An instance of the Body class made from this.event.body parsed to JSON

EventService~Body

A class representing body of the API request event. This class is not exported from the module and is meant to be instantiated only from within this module, so it is not accessible through "require("services").EventService.Body". Creating a new instance is only possible through "new EventService.Event({event}).Body()".

Kind: inner class of EventService
Access: protected

- ~Body
  - new Body(body)
  - .getFor(key) ⇒ string
  - .getForOptional(key) ⇒ string

new Body(body) Constructor for the Body class. Can only be used from within this module.

Param	Type
body	object

body.getFor(key) ⇒ string Returns the body property from the API request with the given key or throws an error if a body property with the given key does not exist. Is used when a body property is required in the API request and we need to fail the request in case it is missing.
Kind: instance method of Body
Throws:

- Error with message: "Body does not have key {key}"

Param	Type
key	string

body.getForOptional(key) ⇒ string Returns the body property from the API request with the given key or "NONE" if a body property with the given key does not exist. Is used when a body property is optional in the API request. The "NONE" placeholder string is used instead of the empty string since DynamoDB does not allows insertion of empty strings in the database.
Kind: instance method of Body
Throws:

- Error with message: "Body does not have key {key}"

Param	Type
key	string

EventService~PathParams

A class representing path parameters of the API request event. This class is not exported from the module and is meant to be instantiated only from within this module, so it is not accessible through ”require(”services”).EventService.PathParams”. Creating a new instance is only possible through ”new EventService.Event({event}).PathParams()”.

Kind: inner class of EventService
Access: protected

- ~PathParams
  - new PathParams(pathParams)
  - .getFor(key) ⇒ string

new PathParams(pathParams) Constructor for the PathParams class. Can only be used from within this module.

Param	Type
pathParams	object

pathParams.getFor(key) ⇒ string Returns the path parameter with the given key or throws an error if a path parameter with the given key does not exist.

Kind: instance method of PathParams
Throws:

- Error with message: ”Path parameters does not have key {key}”

Param	Type
key	string

A.1.4 ConferenceService

This module provides Lambda functions with services and the ability to retrieve data and perform operations mainly involving Conference items on our database. All the functions in this module are asynchronous, and therefore need to be called using the 'await' keyword. Example usage: await ConferenceService.getAllOverviews();

- ConferenceService
  - *static*
    - .createConference(id, name, description, page_content, fee, starts_at, ends_at, is_open, deadline)
    - .getAllOverviews() ⇒ Array.<ConferenceOverview>
    - .getAllCurrentOverviews() ⇒ Array.<ConferenceOverview>
    - .getAllOpenOverviews() ⇒ Array.<ConferenceOverview>
    - .getOverviewForConference(conferenceId) ⇒ ConferenceOverview
    - .getAllOpenOverviews(conferenceIds) ⇒ Array.<ConferenceOverview>
    - .getDetailsForConference(conferenceId) ⇒ ConferenceDetails
    - .getDetailsForConference(conferenceId) ⇒ Array.<TicketDetails>
    - .getPaymentsForConference(conferenceId) ⇒ Array.<PaymentDetails>
  - *inner*
    - ~ConferenceOverview : Object
    - ~ConferenceDetails : Object

ConferenceService.createConference(id, name, description, page_content, fee, starts_at, ends_at, is_open, deadline)

(Async) Creates a Conference in the database with the given parameters. This function can also be used to update an existing Conference, since it performs a Put operation on the database.

Kind: static method of ConferenceService

Param	Type
id	string
name	string
description	string
page_content	string
fee	string
starts_at	string
ends_at	string
is_open	boolean
deadline	string

ConferenceService.getAllOverviews() ⇒ Array.<ConferenceOverview>

(Async) Returns with an array containing the overview objects of all the Conferences on the database.

Kind: static method of ConferenceService

ConferenceService.getAllCurrentOverviews() ⇒ Array.<ConferenceOverview>

(Async) Returns with an array containing the overview objects of all the available Conferences on the database. A conference is available when the registration deadline is not over and the conference is open to public.

Kind: static method of ConferenceService

ConferenceService.getAllOpenOverviews() ⇒ Array.<ConferenceOverview>

(Async) Returns with an array containing the overview objects of all the Conferences on the database that are open to public.

Kind: static method of ConferenceService

ConferenceService.getOverviewForConference(conferenceId) ⇒ ConferenceOverview

(Async) Returns with an object containing the overview object of the Conference on the database with the given ID.

Kind: static method of ConferenceService

Param	Type
conferenceId	string

ConferenceService.getAllOpenOverviews(conferenceIds) ⇒ Array.<ConferenceOverview>

(Async) Returns with an array containing the overview objects of the Conferences on the database with the given array of IDs.

Kind: static method of ConferenceService

Param	Type
conferenceIds	Array.<string>

ConferenceService.getDetailsForConference(conferenceId) ⇒ ConferenceDetails

(Async) Returns with an object containing the all the properties of the Conferences with the given ID on the database.

Kind: static method of ConferenceService

Param	Type
conferenceId	string

ConferenceService.getDetailsForConference(conferenceId) ⇒ Array.<TicketDetails>

(Async) Returns with an array containing the TicketDetails objects that are associated with the given Conference ID.

Kind: static method of ConferenceService

Param	Type
conferenceId	string

ConferenceService.getPaymentsForConference(conferenceId) ⇒ Array.<PaymentDetails>

(Async) Returns with an array containing the PaymentDetails objects that are associated with the given Conference ID.

Kind: static method of ConferenceService

Param	Type
conferenceId	string

ConferenceService~ConferenceOverview : Object

Kind: inner typedef of ConferenceService

Properties

Name	Type	Description
id	string	Conference ID
name	string	Conference name
description	string	Description of the conference
starts_at	string	Starting date
ends_at	string	Ending date
deadline	string	Registration deadline

Name	Type	Description
is_open	boolean	Whether conference is discoverable to public

ConferenceService~ConferenceDetails : Object

Kind: inner typedef of ConferenceService

Properties

Name	Type	Description
id	string	Conference ID
name	string	Conference name
description	string	Description of the conference
page_content	string	Content displayed on the conference page
fee	string	Conference fee
starts_at	string	Starting date
ends_at	string	Ending date
deadline	string	Registration deadline
is_open	boolean	Whether conference is discoverable to public

A.1.5 PaymentService

This module provides Lambda functions with services and the ability to retrieve data and perform operations mainly involving Payment items on our database.

- PaymentService
  - ~Payment
    - new Payment(id)
    - .createTransferPayment(conference, user, given_name, family_name, email, address, postalcode, city, country, ticket_id)
    - .getPaymentDetails() ⇒ PaymentDetails
    - .validateTransferPayment()
    - .modifyPayment(given_name, family_name, email, amount, city, postalcode, country, address)
  - ~PaymentDetails : Object

PaymentService~Payment

A class that holds a payment ID as a property and provides services involving that payment. All the functions in this class are asynchronous, and therefore need to be called using the 'await' keyword.

Example usage: `await new PaymentService.Payment({id}).getPaymentDetails();`
Kind: inner class of PaymentService

- ~Payment
  - new Payment(id)
  - .createTransferPayment(conference, user, given_name, family_name, email, address, postalcode, city, country, ticket_id)
  - .getPaymentDetails() ⇒ PaymentDetails
  - .validateTransferPayment()
  - .modifyPayment(given_name, family_name, email, amount, city, postalcode, country, address)

new Payment(id) Constructor for the Payment class. Takes the Payment's ID as a parameter.

Param	Type
id	string

payment.createTransferPayment(conference, user, given_name, family_name, email, address, postalcode, city, country, ticket_id) (Async) Creates a payment with type 'transfer' in the database with the object's id parameter and the given parameters.
Kind: instance method of Payment

Param	Type	Description
conference	string	ID of the conference payment is made for
user	string	ID of the user submitting the paper
given_name	string	Billing given name
family_name	string	Billing family name
email	string	Billing email
address	string	Billing address
postalcode	string	Billing postal code
city	string	Billing city
country	string	Billing country
ticket_id	string	ID of the corresponding ticket

payment.getPaymentDetails() ⇒ PaymentDetails (Async) Returns with a PaymentDetails object corresponding to the payment details on the database.

Kind: instance method of Payment

payment.validateTransferPayment() (Async) Sets the status of the payment to accepted, validates the ticket, and finally adds the registration to user profile and emails it to the provided email address. This function is meant to be used after verifying the user has transferred the payment.

Kind: instance method of Payment
Throws:

- Error Throws an error if the database update fails or payment type is not transfer.

payment.modifyPayment(given_name, family_name, email, amount, city, postalcode, country, address) (Async) Modifies the details of the payment in the database with the given parameters.

Kind: instance method of Payment

Param	Type	Description
given_name	string	Billing given name
family_name	string	Billing family name
email	string	Billing email
amount	string	The amount of payment in HUF
city	string	Billing city
postalcode	string	Billing postal code
country	string	Billing country
address	string	Billing address

PaymentService~PaymentDetails : Object

Kind: inner typedef of PaymentService
Properties

Name	Type	Description
id	string	Payment ID
conference	string	ID of the conference payment is made for
user	string	ID of the user submitting the paper
given_name	string	Billing given name
family_name	string	Billing family name
email	string	Billing email

Name	Type	Description
address	string	Billing address
postalcode	string	Billing postal code
city	string	Billing city
country	string	Billing country
amount	string	Billing city
conference_name	string	Name of the conference payment is made for
ticket_id	string	ID of the corresponding ticket
type	string	Payment type; currently only ”transfer”
status	string	Payment status; ”waiting” or ”accepted”
transactions	Array	Online transaction IDs; Currently only empty
created_at	string	Date of invoice

A.1.6 TicketService

This module provides Lambda functions with services and the ability to retrieve data and perform operations mainly involving Ticket items on our database.

- TicketService
  - ~Ticket
    - new Ticket(id)
    - .createTicket(conference, user, given_name, family_name, email, payment_id, requested_meal, meal_notes, requested_accommodation, arriving_time, stay_length)
    - .modifyTicket(given_name, family_name, email, requested_meal, meal_notes, requested_accommodation, arriving_time, stay_length)
    - .getTicketDetails() ⇒ TicketDetails
    - .getTicketName() ⇒ Name
    - .validateTicket()
    - .addToUserProfile()
    - .scanTicket() ⇒ ScanResult
  - ~TicketDetails : Object
  - ~Name : Object
  - ~ScanResult : Object

TicketService~Ticket

A class that holds a ticket ID as a property and provides services involving that ticket. All the functions in this class are asynchronous, and therefore need to be called using the ’await’ keyword. Example usage: await new TicketService.Ticket({id}).getTicketDetails();

Kind: inner class of TicketService

- ~Ticket
  - new Ticket(id)
  - .createTicket(conference, user, given_name, family_name, email, payment_id, requested_meal, meal_notes, requested_accommodation, arriving_time, stay_length)
  - .modifyTicket(given_name, family_name, email, requested_meal, meal_notes, requested_accommodation, arriving_time, stay_length)
  - .getTicketDetails() ⇒ TicketDetails
  - .getTicketName() ⇒ Name
  - .validateTicket()
  - .addToUserProfile()
  - .scanTicket() ⇒ ScanResult

new Ticket(id) Constructor for the Payment class. Takes the Payment’s ID as a parameter.

Param	Type
id	string

ticket.createTicket(conference, user, given_name, family_name, email, payment_id, requested_meal, meal_notes, requested_accommodation, arriving_time, stay_length) Creates a ticket in the database with the object’s id parameter and the given parameters.

Kind: instance method of Ticket

Param	Type	Description
conference	string	ID of the conference corresponding ticket
user	string	ID of submitting user
given_name	string	Ticket owner’s given name
family_name	string	Ticket owner’s family name
email	string	Ticket owner’s email
payment_id	string	ID of the corresponding payment
requested_meal	boolean	Whether user requested meal preparation
meal_notes	string	Notes for meal

Param	Type	Description
requested_accommodation	boolean	Whether user requested accommodation help
arriving_time	string	Date of arrival in case of accommodation
stay_length	string	Duration of stay in case of accommodation

ticket.modifyTicket(given_name, family_name, email, requested_meal, meal_notes, requested_accommodation, arriving_time, stay_length) Creates a ticket in the database with the object's id parameter and the given parameters.

Kind: instance method of Ticket

Param	Type	Description
given_name	string	Ticket owner's given name
family_name	string	Ticket owner's family name
email	string	Ticket owner's email
requested_meal	boolean	Whether user requested meal preparation
meal_notes	string	Notes for meal
requested_accommodation	boolean	Whether user requested accommodation help
arriving_time	string	Date of arrival in case of accommodation
stay_length	string	Duration of stay in case of accommodation

ticket.getTicketDetails() ⇒ TicketDetails (Async) Returns with a TicketDetails object corresponding to the ticket details on the database.

Kind: instance method of Ticket

ticket.getTicketName() ⇒ Name (Async) Returns with an object containing the given name and family name on the ticket. This functions is preferred over using getTicketDetails for retrieving the name since the implementation provides better database response time.

Kind: instance method of Ticket

ticket.validateTicket() (Async) Sets the state of the ticket to 'valid'. This function is meant to be used after validation of payment.

Kind: instance method of Ticket

ticket.addToUserProfile() (Async) Adds the ticket to be user profile by appending the ticket as an 'AttendingConference' object (from module UserService) to the conferences_attending property of the user, and sends an email containing the ticket document to the provided address. This function is meant to be used after validation of the ticket.

Kind: instance method of Ticket

ticket.scanTicket() ⇒ ScanResult (Async) Sets the state of the ticket to used, and returns the an object containing the ticket details and time of execution. This function is meant to be used for scanning a ticket's QR code.

Kind: instance method of Ticket

TicketService~TicketDetails : Object

Kind: inner typedef of TicketService

Properties

Name	Type	Description
id	string	Ticket ID
conference	string	ID of the conference corresponding ticket
user	string	ID of submitting user
given_name	string	Ticket owner's given name
family_name	string	Ticket owner's family name
email	string	Ticket owner's email
status	string	Ticket status; "valid", "invalid", or "used"
payment_id	string	ID of the corresponding payment
created_at	string	Date of invoice
requested_meal	boolean	Whether user requested meal preparation
meal_notes	string	Notes for meal
requested_accommodation	boolean	Whether user requested accommodation help
arriving_time	string	Date of arrival in case of accommodation
stay_length	string	Duration of stay in case of accommodation

TicketService~Name : Object

Kind: inner typedef of TicketService

Properties

Name	Type	Description
given_name	string	Ticket owner's given name

Name	Type	Description
family_name	string	Ticket owner's family name

TicketService~ScanResult : Object

Kind: inner typedef of TicketService

Properties

Name	Type	Description
id	string	Ticket ID
conference	string	ID of the conference corresponding ticket
user	string	ID of submitting user
given_name	string	Ticket owner's given name
family_name	string	Ticket owner's family name
email	string	Ticket owner's email
status	string	Ticket status; "valid", "invalid", or "used"
payment_id	string	ID of the corresponding payment
created_at	string	Date of invoice
requested_meal	boolean	Whether user requested meal preparation
meal_notes	string	Notes for meal
requested_accommodation	boolean	Whether user requested accommodation help
arriving_time	string	Date of arrival in case of accommodation
stay_length	string	Duration of stay in case of accommodation
time	string	Scan time

A.1.7 UserService

This module provides Lambda functions with services regarding a user item in the database.

- UserService
  - ~User
    - new User(id)
    - .getUserType() ⇒ string
    - .getUserName() ⇒ Name
    - .getTickets() ⇒ Array.<AttendingConference>
  - ~AttendingConference : Object
  - ~Name : Object

UserService~User

A class that holds a user ID as a property and provides services and information about the user. All the functions in this class are asynchronous, and therefore need to be called using the 'await' keyword.

Example usage: `await new UserService.User({id}).getUserType();`

Kind: inner class of UserService

- ~User
  - new User(id)
  - .getUserType() ⇒ string
  - .getUserName() ⇒ Name
  - .getTickets() ⇒ Array.<AttendingConference>

new User(id) Constructor for the user class. Takes the user's ID as a parameter.

Param	Type
id	string

user.getUserType() ⇒ string (Async) Returns with the user type. The result can be 'basic', 'admin', or 'moderator'.

Kind: instance method of User

user.getUserName() ⇒ Name (Async) Returns with an object containing the user's name.

Kind: instance method of User

user.getTickets() ⇒ Array.<AttendingConference> (Async) Returns with an array containing the AttendingConference objects from all the tickets that have been added to user's profile.

Kind: instance method of User

UserService~AttendingConference : Object

Kind: inner typedef of UserService

Properties

Name	Type	Description

Name	Type	Description
id	string	Ticket's ID
conference	string	Conference's ID

UserService~Name : Object

Kind: inner typedef of UserService

Properties

Name	Type	Description
given_name	string	User's given name
family_name	string	User's family name

REFERENCES

[1] P. Srivastava and R. Khan, "A review paper on cloud computing", *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 8, p. 17, 2018-06. DOI: 10.23956/ijarcsse.v8i6.711.

[2] What is cloud computing?, [Online]. Available: https://aws.amazon.com/what-is-cloud-computing/ (accessed on 2019-12-04).

[3] M. Stigler, "Understanding serverless computing", in *Beginning Serverless Computing: Developing with Amazon Web Services, Microsoft Azure, and Google Cloud*. Berkeley, CA: Apress, 2018, pp. 1–14, ISBN: 978-1-4842-3084-8. DOI: 10.1007/978-1-4842-3084-8_1.

[4] ——, "Amazon web services", in *Beginning Serverless Computing: Developing with Amazon Web Services, Microsoft Azure, and Google Cloud*. Berkeley, CA: Apress, 2018, pp. 41–81, ISBN: 978-1-4842-3084-8. DOI: 10.1007/978-1-4842-3084-8_3.

[5] What is the aws serverless application model (aws sam)?, [Online]. Available: https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html (accessed on 2019-11-25).

[6] What is aws cloudformation?, [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html (accessed on 2019-11-25).

[7] What is amazon dynamodb?, [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html (accessed on 2019-11-25).

[8] What is amazon cognito?, [Online]. Available: https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html (accessed on 2019-11-25).

[9] Control access to a rest api using amazon cognito user pools as authorizer, [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-integrate-with-cognito.html (accessed on 2019-11-25).

[10] Rendering elements, [Online]. Available: `https://reactjs.org/docs/rendering-elements.html` (accessed on 2019-12-01).

[11] Reconciliation, [Online]. Available: `https://reactjs.org/docs/reconciliation.html` (accessed on 2019-12-01).

[12] Introducing jsx, [Online]. Available: `https://reactjs.org/docs/introducing-jsx.html` (accessed on 2019-12-01).

[13] Aws amplify faq, [Online]. Available: `https://aws.amazon.com/amplify/faqs/` (accessed on 2019-12-02).

[14] Aws amplify faq - hosting, [Online]. Available: `https://aws.amazon.com/amplify/faqs/#Hosting` (accessed on 2019-12-02).

[15] Dynamodb developer guide: Global secondary indexes, [Online]. Available: `https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html` (accessed on 2019-12-02).

[16] New for aws lambda use any programming language and share common components, [Online]. Available: `https://aws.amazon.com/blogs/aws/new-for-aws-lambda-use-any-programming-language-and-share-common-components/` (accessed on 2019-12-02).