System Integration
KEA SD22w

Group members:
Andreea Vasiliu - andr77c6@stud.kea.dk
Ani Boyadzhieva - anix0072@stud.kea.dk
Marcell Varga - marc308i@stud.kea.dk
Vlad-Alexandru Făget - vlad0764@stud.kea.dk

Choices

Azure

FTP server

We used an Azure Virtual Machine that has an Ubuntu image to host the FTP server, and the vsftpd package to create it. It wasn't a great choice to make it this way, but Render does not allow using different ports other than 80 (http) and 443 (https), and since it was a requirement to use port 21, we decided that this was the best choice. We have tried to move the http server on the virtual machine, but we lost around a week trying to make it operational, and due to the lack of time we decided to continue using Render as our http server host.

Render

Hosting

We host our API and website on Render. Until now we have worked only with Azure and each time it was a hassle to set up, to connect, to make changes, etc. The UI for Azure is extremely unintuitive and settings are spread throughout the Azure portal. We found Render as an alternative. Everything was set up in a few clicks. It has native support for node, and claims to have zero-downtime deploys, free tls certificates, support for large scale systems, and more. The prices are also a bit cheaper (about 40$ less a month for a system with 2 cores, 4 GB ram, 50 GB storage; we only checked for the web service prices).
The only downside so far is that we do not have a student subscription, but the free tier covers what we need.

Backend

Node.js

The backend service is developed in node.js. We did so as we prefer working in C# and, at a glance, nodejs is more similar to that. However we found that node has a few more quirks than C# and the implementation would probably be faster in python.

Passport

We opted to use passport to handle the authentication function. This was based on the previous choice. When searching for authentication middleware, passport comes as the first result. Given the popularity, we assumed that it was well documented or at least had plenty of community made guides. It was somewhat quick to implement, although it did require some poorly explained boilerplate code.

Nodemailer

We use this package to send invite emails. We chose it because it was the most simple to use. To send the email we created a google account and enabled third party app access.

Better-sqlite3

Instead of the default sqlite3 package, we used better-sqlite3. As the name suggests, it is better to work with. It uses proper async/await functions instead of callbacks, which makes life easier.

Feed

For generating the rss feed file we use the feed package. We found that there are very few choices to pick from when it comes to rss generation in node. This was the most recent package we found and it is 6 years old. Still, we did not want to format the feed by hand and it worked.

Express

Express is perhaps the most popular web app framework for node. We used it before and were familiar with it, making this an easy choice.

Swagger

We used swagger to make the api documentation. A very popular choice among developers and we used it before. Unfortunately, unlike python's FastApi, the documentation is not auto generated. There is a node package that claims to do so, but it requires some more code to set up. We had to manually type it, and the yaml format is unpleasant to work with. But once that is done, it works fine.

Socket.io

Socket.io is the most common library for creating a websocket server in Node.js. It is well documented, with great examples and community support on GitHub. It has automatic 'connect' and 'disconnect' events and it is easy to separate the server from the client by not having limitations regarding proprietary port number. Because we use socket.io, we have both the http server and the socket.io server listening on the same port with no limitations. It is easy to integrate by just listing the events that the sockets accepts and emits to the integration group,

with the required parameters that they need to send. Because it has great documentation, it was easy to understand how to emit events to a specific socket.

Frontend

Angular

For frontend we use angular. Most of us prefer angular as we consider it to be better documented than react, and is by far more compatible with older code. With react most guides we found are already obsolete if they are more than 6 months old. We only use angular default components and some bootstrap elements, so no other package was installed.