System Integration
KEA SD22w

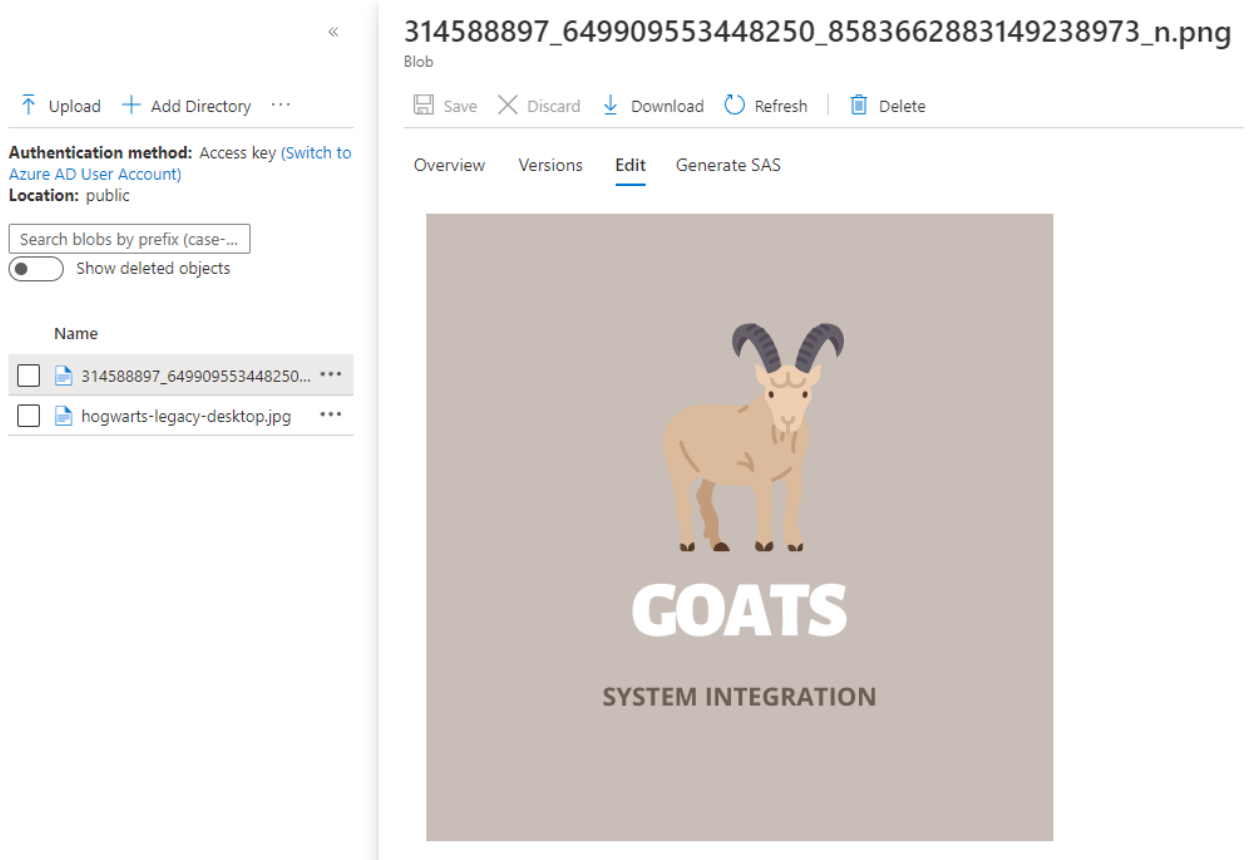
Group members:
Andreea Vasiliu - andr77c6@stud.kea.dk
Ani Boyadzhieva - anix0072@stud.kea.dk
Marcell Varga - marc308i@stud.kea.dk
Vlad-Alexandru Făget - vlad0764@stud.kea.dk

How To

Expose:

CDN: Andreea

We created an Azure Storage account that hosts the logo for the other group. We have a public container that contains the logo. For the partner group to have access to the logo, we made it public.



FTP Server: Andreea & Ani

We created an Azure Virtual Machine running Ubuntu. From there, we created a ftp server using vsftpd package with : `sudo systemctl restart vsftpd` . Then, we modified the vsftpd.conf file to make the server passive with ports ranging from 30000-31000

```
user_sub_token=$USER
local_root=/home/$USER/ftp
pasv_min_port=30000
pasv_max_port=31000
```

After that, we configured the firewall to accept connections on port 21 and 30000-31000 for passive mode

```
To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
21/tcp                     ALLOW       Anywhere
20/tcp                     ALLOW       Anywhere
80/tcp                     ALLOW       Anywhere
443/tcp                    ALLOW       Anywhere
Nginx HTTP                 ALLOW       Anywhere
OpenSSH                    ALLOW       Anywhere
30000:31000/tcp            ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
21/tcp (v6)                ALLOW       Anywhere (v6)
20/tcp (v6)                ALLOW       Anywhere (v6)
80/tcp (v6)                ALLOW       Anywhere (v6)
443/tcp (v6)               ALLOW       Anywhere (v6)
Nginx HTTP (v6)            ALLOW       Anywhere (v6)
OpenSSH (v6)               ALLOW       Anywhere (v6)
30000:31000/tcp (v6)       ALLOW       Anywhere (v6)
```

And also on the VM network settings:

Priority	Name	Port	Protocol	Source	Destination	Action	
100	HTTP	80	TCP	Any	Any	Allow	•••
300	SSH	22	TCP	Any	Any	Allow	•••
320	HTTPS	443	TCP	Any	Any	Allow	•••
350	FTP	21	TCP	Any	Any	Allow	•••
351	FTP2	20	TCP	Any	Any	Allow	•••
361	FTP3	30000-31000	TCP	Any	Any	Allow	•••

Then, we created a folder called files with root access for the ftp user to have permissions to put the file in:

```
threeam@threeam:~/ftp/files$ ls -la
total 92
drwxr-xr-x 2 threeam threeam  4096 Dec 11 14:22 .
drwxrwxr-x 3 nobody  nogroup  4096 Dec  2 22:41 ..
```

GraphQL: Vlad

We use a GraphQL server to query data from the product database. We host this service on Render and we write this manually in JavaScript using the express, express-graphql, and graphql packages. We define a schema for the data structure then write the query functions that are in basic SQL. Lastly, we define an endpoint where it can be accessed by the other group. We have 3 different queries:

```
type Query {
    Search(keyword:String): [Product],
    GetProduct(id:String): Product,
    GetAllProducts: [Product]
  }
```

1. Search(keyword:String): searches the provided keyword in the product_name product_description, main_category and sub_category columns in the database. And returns an array of products
2. GetProduct(id:String): searches for a specific product based on it's id and returns one product object
3. GetAllProducts: returns all products in the database as an array of Product

Friends: Andreea, Marcell, Vlad

To make invitations possible we have an “invite” table where we store all invitations. We create a unique token so that we can identify the new user and automatically add them as friends or have the invitation expire and delete the invitation altogether. We also implement the email invitations which is as simple as using the nodemailer package. It only requires a valid email account and it can send emails anywhere. Then we just configure an endpoint to access it and it is ready to use for the other group.
Then we use websockets (socket.io) to display the user status:

1. When a user is connected, it creates the socket
2. When the ‘login’ event is received, it searches for the user’s invites, and, if the invite is not accepted, it emits an ‘invitation-pending’ event with the email of the person that is invited. Then, it searches if the user has friends. If it does, it searches if the user’s friends have an open socket and if it’s found, it emits an ‘online’ event with the user’s username as a message.
3. When the ‘logout’ event is received, it searches if the user has friends. If it does, it searches if the user’s friends have an open socket and if it’s found, it emits an ‘offline’ event with the user’s username as a message.
4. When a user that is logged in disconnects, the same process as the login happens

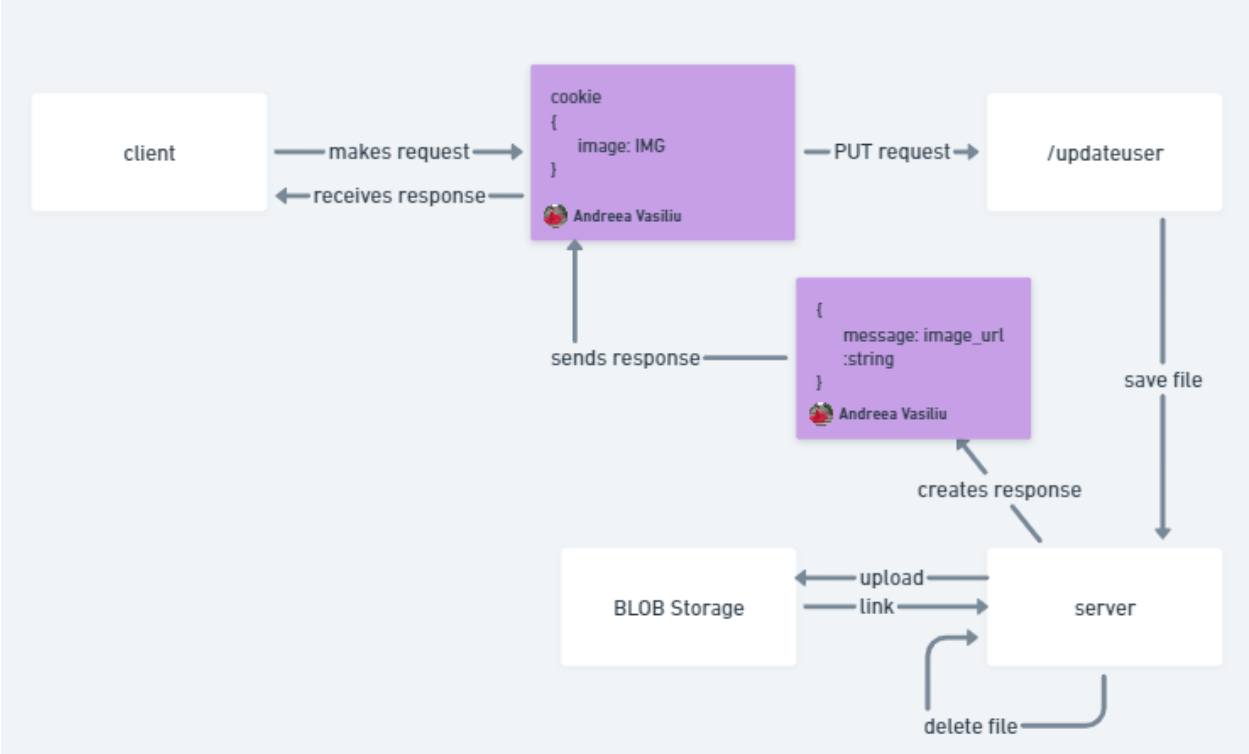
Authentication: Andreea, Marcell, Vlad

For registration and login we use Passport and the users table in our database. We host it using an express server and to hash the password we use crypto. We have constraints for the username and password that are enforced on the server level. The server stores the session in a database table and sends it as a cookie. The cookies are saved in the browser and it’s required to send a cookie with the sessionid to access all the other endpoints except /login/password and /signup. Here, we had some problems when our partner group integrated our system because CORS should accept credentials (the cookie) and we had some trial-and-error before we could find a solution that does not compromise on security.

Profile picture: Andreea

In the same Azure Storage, we have created a new container named the same as our integration group (goat) where we connect it using a SAS(shared access signature) with a token to upload the picture that comes from the /updateprofile with a mutimedia/form-data type. Then, the url of the picture from the storage account is saved in the database along with the access token. For this, we had to use the multer package to store the file in the server, then upload it in the container using azure/storage-blob package and then multer deletes it from the server.

Diagram of the flow:



RSS: Vlad

The RSS feed generates a list of the latest wishes added to the database by any user. Whenever a call is made to the API to add or remove a wish, it also forwards a call to re-generate the rss feed file. There is room for improvement as it feels wrong to show for anyone to see what a user wishes for. It might be better to store it in a database as a json object and have it individual for each user, then to retrieve it when loading the front page if a user is logged in.

Integration:

Website:

The website is created in Angular and deployed on Render. The UI is designed using Bootstrap. It is a single page application.
For every request that we make, the exposed endpoints require a Ocp-Apim-Subscription-Key that we had to create

Logo: Marcell, Ani

The logo is pulled from the other group's CDN with a link and then displayed on the webpage.

Web scraper: Marcell, Vlad

The web scraper is not hosted. It is only run once locally to create the database file and send to the other group's FTP server. We use axios to crawl through websites and cheerio to scrape them. We look at the website's HTML code and find the elements we want to get from each page. We specify these for cheerio and it automatically extracts the data from all the pages we want to scrape.
Then, we sent the generated database to the partner group's FTP server using the provided credentials.

Authentication: Andreea, Marcell, Ani

We use the other group's login/register system by calling their endpoint. When we register a user, their server sends back a custom API key so we can request the data from the database.

Product Display: Vlad

We gather the products via a Graphql query. Unfortunately, the query does not retrieve the product images, so we must recursively get the details for each individual product. It considerably increases the loading time. It can be improved by adding pagination.