IDG2001 Assignment 1 – vCard application

09.04.2023

Group members

This group contains three group members:

- Alexandra Eloise Vanje
- Anosh Chaudhry
- Lisa Mari Myrene

Cloud service for this project

A vCard application stores and manages contact information in a digital format. To host and run such an application, there may be various services that may be suitable, such as PaaS, SaaS, and IaaS. The difference between these is how much freedom and responsibility you have with your application.

Since this is such a small application, a PaaS offering would be the best choice for hosting a vCard application, as it provides a pre-configured environment that includes an operating system, database, and web server. This can help reduce the setup time and maintenance required for the application. When choosing PaaS, we would only be managing the application and its data.

How the system works

This is a Flask application, a cloud based system that is running in the Railway cloud server. The application accepts a vCard file (contact file), sends it to an REST API endpoint, parses it server-side and adds it to a database. The system returns the contacts from the API as either JSON or vCard format. It is also possible to download everything from the database, locally to your computer.

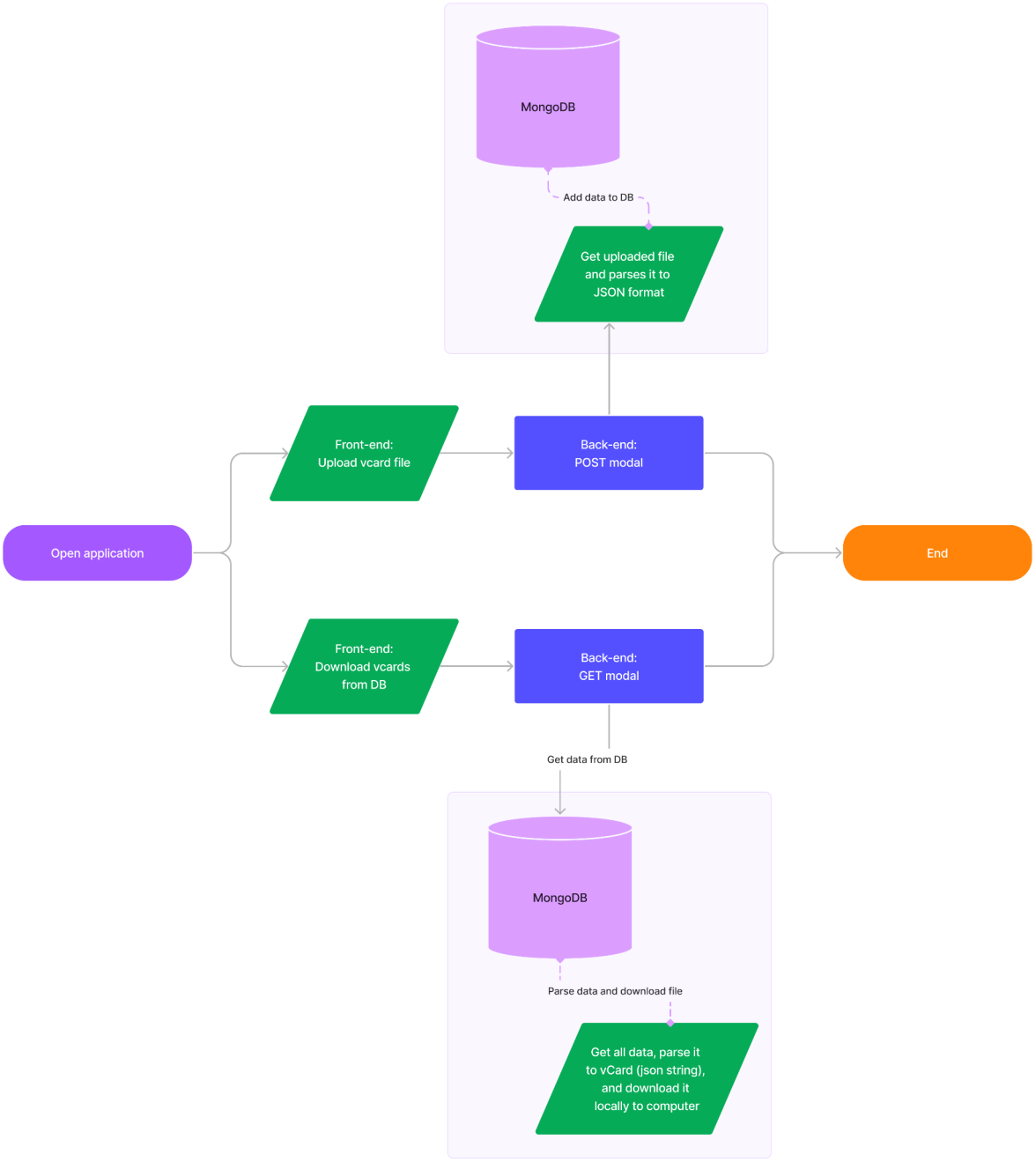
The project needs to be able to connect to the database and its collections. To make this possible, we needed to use the *pymongo* and *dotenv* modules. We put the *MONGO_URI* inside an *.env* file, to make the connection more secure. After the connection was set up, we could start creating the API endpoints.

All our API endpoints and routes are stored inside a main Python file. Here is where most of the functionality is. The Back-end system connects and talks to the Front-end side of the application, where a HTML form is, by creating REST APIs. Here is what the different API endpoints does:

API ENDPOINT	DESCRIPTION
`@app.route('/')`	This set the home route for the application. When the user is in the home path, they will see the form rendered on the website.
`@app.route('/contacts', methods=['POST'])`	This route takes in the uploaded file from the user, parses it from vcf to json, and then pushes the result to the database.
`@app.route('/contacts', methods=['GET'])`	This route finds all contacts in the database and displays it to the user.
`@app.route('/contacts/<id>', methods=['GET'])`	This route finds a certain contact in the database based on id, and displays it to the user.
`@app.route('/contacts/vcard', methods=['GET'])`	This route gets all the contacts in the database, parses it from json back to vcf (in json format), and then displays it to the user.
`@app.route('/contacts/<id>/vcard', methods=['GET'])`	This route gets one specific contact in the database based on id, parses it from json back to vcf (in json format), and then displays it to the user.

All the parsing functionalities that are used in the API endpoints are in separate Python files. When the user wants to download everything from the database locally to their computer, we created an asynchronous function in JavaScript that creates an AJAX request to exchange data with the server, so then creates a ‘blob’ which is a file-like object for file handling.

Here is a simplified diagram of how the processes works when doing different actions:



Choices made during development

In the very beginning of the development process, there was some debate on whether to keep all of our code in a single Python file or if we should separate our project into smaller bits. We decided to go for the latter option to establish a separation of concerns. This keeps the project more organized and easily readable for others. We have a *main.py* file with all our routes and API endpoints, and its needed modules, and separate Python files for different functionalities and database connection.

We also decided to develop this project using the web app framework Flask and Pymongo, a tool for working with Python and MongoDB. We chose Flask over other frameworks because it is rather easy to learn and we had some knowledge already. There are also a lot of resources about Flask which made it appealing. However, FastAPI also seemed like a good option (in hindsight, maybe better as it is specifically for creating APIs), so we will consider using that for future projects. Pymongo is one of the most used tools for working with mongoDB and Python, so it was natural to choose this. Because it is an official MongoDB Python driver it has a lot of documentation which we found to be important.

When coding, we decided to have a “trial and error” approach. As we are not that familiar with Python, we used this project as a learning opportunity to get a better understanding of it while also fulfilling the task requirements. Essentially, we tried a lot of different things with the code until we finally found something that works. Our code is probably not optimal for anything large scale, but it works and produces wanted results, which is what we aimed for with our level of knowledge.

The most difficult part of this assignment was to find out how to do the parsing from JSON to vCard, and back. We tried a lot of things, such as using only one Python file to parse everything depending on the state of the file, but it turned out to be too complex. Therefore we divided the different parses into 3 separate Python files that made it more organized.