# Secret-api

## Table of contents



## About

This platform allows users to add and retrieve confidential information. Additionally, it offers the option to share this information via email, providing a way communicate securely. You can check out the site here. I use only free solutions for hosting, so I recommend to clone the project from my GitHub and try it out in a development environment as well if some features are slow.

## Technologies

- Java Script
- React
- Css
- Python
- Flask
- SQLAlchemy
- Vite
- Postgresql

# Features

- Store secrets with a unique identifier
- Get the secrets with the unique identifier
- Share the secret with an email.
- Retrieve the secret with a random url.

# Setup guide

First clone the project from my GitHub repository. After you cloned the project open the project folders with your IDE of choice or if you don't have one open it in terminal. I recommend to open the **backend** and the **frontend** folders separately.

## Backend

**dependencies**

- Python 3

If you don't have python install it first.

After that open the **secret-api** folder in the **secret-server** folder. If your IDE didn't configure the project or you opened the project in a terminal (cmd or powershell on windows) run the following command:

Install the dependencies

```
pip install -r requirements.txt
```

then

```
python app.py
```

With the command above you can start your backend server. After starting the sever it should automatically connect to the database.

## Frontend

**dependencies**

- nvm
- npm

Open the **secret-frontend** folder in the **Secret-server** folder with your IDE or in terminal. If you are using windows it's possible that nvm is not installed on your computer, so you should install it first.

If you are not familiar how to do this follow the guide in the following link:

Link to node version manager installation guide

After that run the command bellow in a terminal.

```
npm install
```

This will install all the dependencies that you need to run the program. After that run the code bellow to start the React development server.

```
npm run dev
```

You can check out the site on the link bellow:

http://localhost:5173/

# Implementation

## backend

For the secret api I used **Python** with **Flask** and **SQLAlchemy** for data management. I hosted the database on supabase.com which is a free solutions for hosting databases. It uses PostgresSQL for sql dialect. The secrets are being saved here. The backend server is hosted on render.com. It's a free solutions for hosting web services or static websites. The service is connected to a dedicated branch on GitHub and auto deploys the server if changes are committed to the branch. It can send out email which is done with pythons smtplib trough google's smtp server.

**Server status codes**

The server responds with the following status codes:

- **200** : secret has been added or retrieved successfully
- **404** : secret not found
- **500** : internal server error wit the related error message

**Code snippets and explanation**

- The endpoints:

I created the endpoints with flask's Blueprint class. I created a Blueprint instance named secret_blueprint. All the endpoint expects a json request, but they can respond based on the Accept header in json and in Xml to.

```
secret_blueprint = Blueprint("secret", __name__, url_prefix="/secret")
```

This way I could creat my endpoint in a different file, so I could organize my code in a more organizable and expandable way.

The endpoint for adding a secret:

```
@secret_blueprint.post('')
def add_new_secret():
    secret_data = request.get_json()
    accept_headers = request.headers.get("Accept")
    return secret_controller.handle_new_secret(secret_data, accept_headers)
```

This endpoint receives the secret data and teh Accept header and passes it to the related method in the SecretController class. After that a random generated ID (UUID) is generated for the secret, and it is saved in te database. After that the server sends out the id to the provided email address for the user. This is not the most convenient way, but I taught this is the most secure because this way only the user can access te ID.

The end point for getting a secret:

```
  @secret_blueprint.get('/<hash_id>')
def get_secret(hash_id):
    accept_headers = request.headers.get("Accept")
    return secret_controller.handle_get_secret(hash_id, accept_headers)
```

Trough this endpoint the user can retrieve a secret with it related hash_id if it's not expired. This validated in the related secret service class. If the secret is not valid the server sends back the "secret expired" message.

The end point for sharing a secret:

```
@secret_blueprint.post('/share')
def share_secret():
    shared_secret_data = request.get_json()
    accept_header = request.headers.get('Accept')
    return secret_controller.handle_share(shared_secret_data, accept_header)
```

Trough this endpoint the user can share his secret with another user via e-mail. The server sends out a random link to the provided email address. This link contains a random share_id. This id is not identical to the

secrets hash_id. This id is also stored in the database, but it is regenerated with every share request

The end point for getting a shared secret:

```
@secret_blueprint.get("/share/<shared_id>")
def get_shared_secret(shared_id):
    accept_header = request.headers.get('Accept')
    return secret_controller.handle_get_shared_secret(shared_id, accept_header)
```

Trough this link the users can access teh secret that are shared with the on a random link if the secret is not expired and teh shared id in the url is correct.

## frontend

The frontend is implemented in **React**, and it is hosted on Netlify.com It communicates with the secret-api trough json format. The frontend has a responsive navbar, but it is only optimized for desktop use at the moment.

**The pages**

**The landing page:**

```
https://your-secret-app.netlify.app/
```

On the landing page there is some minimal information about how the site functions.

**The add secret page:**

```
https://your-secret-app.netlify.app/add-secret
```

On this page the user can add a secret trough a form. After clicking the add button its sends the data to the api which saves it to the database. If some error occurs it gives an alert message and logs the error to the browser development console. If it's successful it gives a toast message that the secret is saved

**The get secret page:**

```
https://your-secret-app.netlify.app/get-secret
```

On this page the user can retrieve a secret with the provided hash id. If the id is correct and the secret is not expired the secret appears in a secret box if its expired the secret expired message appears instead. If an error occurs the page gives an alert message and the error is logged to the browser development console.

**The share secret page:**

```
https://your-secret-app.netlify.app/share-secret
```

This page is only accessible through the get secret page. Here the user can share his secret with another user via email. In a form the user can give his name, the subject of the email and the email address, and it is sent to the secret api. This way it can send out more personalized emails with the random link for the secret. After sharing the site redirect the user to te landing page.

**The get shared secret page:**

```
https://your-secret-app.netlify.app/share-secret/:sharedID
```

This page receives through the url a unique share id. With this id it fetches the related secret from the api and presents it to the user. If the secret is expired the secret expired message appears in the place of the secret. If some problems occur the page gives an alert message and logs the error in the console. This page can't be accessed through the site.

**The contact page:**

```
https://your-secret-app.netlify.app/contacts
```

A page with my contacts.

# Tests

I wrote unit test for my secret service class because that class contained most of the logic using pythons unittest library. I also wrote postman tests to check te api responds in the proper format based on the request accept header. I included the postman requests for these in the test folder next the unit tests.

## Postman test

```
pm.test("Response Body is XML", function () {
    pm.response.to.have.header("Content-Type", "application/xml")
});
```

Test for xml response.

```
pm.test("response body is json", function () {
  pm.response.to.have.jsonBody();
});
```

Test for json response.

# Upgrade possibilities

- Refine frontend, add some animations
- Work on a more convenient to get the id for his secret
- More refined secret sharing.

If you run into any problems please contact me on **nick.balazs18@gmail.com**