Kodolányi János University

Department of Informatics

PUBLICATION

Application of Artificial Intelligence in Social Media

Social AI - Part 1 – Planning

Supervisors:

Dr. László Pitlik (https://orcid.org/0000-0001-5819-0319**),**

Dr. János Rikk (https://orcid.org/0000-0002-3846-6661**)**

Created by: István Vancsura (https://orcid.org/0000-0002-5402-8186**)**

Budapest

2023

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The goal is planning a software in the cloud (Microsoft Azure) using artificial intelligence (ChatGPT) technologies, which partially implements the automation of content creation on social media (Facebook) pages.

Throughout the development, I will utilize and integrate several Microsoft (e.g., Azure Functions, Azure DevOps), Google (e.g., Identity Platform), Meta (e.g., Meta for Developers) products and services, detailed in Table 1 and elaborated in section 1.1.

I aimed to keep the operational costs of the application low (around 0 HUF / month)

In the development process, I will employ numerous programming (e.g., Python, JavaScript), descriptive (e.g., HTML, CSS), and query (e.g., SQL) languages, summarized in Table 1. Subsequent chapters will extensively cover software design (see Chapter 1)

I would like to express my gratitude to all those who supported the preparation of this thesis and the development of the software, especially to Dr. János Rikk and Dr. László Pitlik.

1. PLANNED TECHNOLOGIES AND LANGUAGES

My goal is to create a modern and cost-effective solution that not only satisfies current needs but also accommodates future growth and increased demands while remaining stable and scalable.

"The most revolutionary and perhaps the most popular IT infrastructure solution of the decade is cloud technology." (Rubóczki Edit Szilvia, http://lib.uni-obuda.hu/sites/lib.uni-obuda.hu/files/Ruboczki_Edit_Szilvia_ertekezes.pdf, page 59, downloaded on 2023.10.07). Therefore, for the development and operation of the application, I will utilize a cloud-based infrastructure, Microsoft Azure (https://azure.microsoft.com). I will employ Azure's "Always Free" (https://azure.microsoft.com/en-us/pricing/free-services/) services and platforms, allowing for initial development and later scalability to meet growing demands.

For the backend system, I have chosen Azure Functions (https://azure.microsoft.com/hu-hu/products/functions/), enabling the development and operation of cloud-based functions. I have selected the Python language for Azure Function development, as Microsoft fully supports Python for Azure Function development (https://learn.microsoft.com/en-

us/azure/azure-functions/functions-reference-python?pivots=python-mode-decorators&tabs=asgi%2Capplication-level). Additionally, a key function of the application is a feature based on the ChatGPT model (https://openai.com/blog/chatgpt), for which an official OpenAI Python library is available (https://platform.openai.com/docs/libraries). For database management, I opted for Azure Cosmos DB (https://azure.microsoft.com/hu-hu/free/cosmos-db/), a horizontally scalable NoSQL database that can integrate with Azure Functions (https://learn.microsoft.com/en-us/azure/azure-functions/functions-add-output-binding-cosmos-db-vs-code?tabs=in-process%2Cv1&pivots=programming-language-python).

For the frontend (web interface), I chose the ReactJS framework (https://react.dev), which serves for creating and managing dynamic user interfaces. I use the JavaScript language in ReactJS, along with HTML and CSS for content display and styling. I selected the Azure Static Web Apps (https://azure.microsoft.com/en-us/products/app-service/static) platform for deploying the ReactJS application, as the platform fully supports the operation of React applications (https://learn.microsoft.com/en-us/azure/static-web-apps/getting-started?tabs=react).

In the authentication and user authentication domain, I integrate the Google Identity Platform (https://cloud.google.com/identity-platform), allowing users to log in easily with their Google accounts, eliminating the need for prospective users to register an account if they already have a Google account (gmail address).

For version control, I chose Git (https://git-scm.com), which is a distributed version control system. I use GitHub (https://github.com) and Azure DevOps (https://azure.microsoft.com/en-us/products/devops) platforms for storing source code, enabling collaboration, tracking the development process, and supporting CI/CD processes.

For continuous integration and continuous delivery (CI/CD) processes, I use GitHub Actions (https://github.com/features/actions) and Azure Pipelines (https://azure.microsoft.com/en-us/products/devops/pipelines), which allow for automated testing, build processes, storage of environmental variables, and rapid and continuous delivery of applications. I define CI/CD processes in YAML.

For application DNS management, I chose Azure DNS (https://azure.microsoft.com/en-us/services/dns/). Azure DNS is a cloud-based DNS service that enables the registration and management of domain names in the Azure environment. This is a crucial part of the cloud-

based infrastructure, as it ensures the resolution of domain names to the application's web interface.

For the image search engine, I selected the Pexels.com (https://www.pexels.com/) platform. API integration needs to be established in the application, as the Pexels API allows for searching, downloading, and displaying images on the application interface.

The following (Table 1) summarizes the planned technologies and languages.

Table 1.: Planned Technologies and Languages

Technical Components	**Planned Systems and Languages (URLs in the chapter 1)**
Cloud platform	Microsoft Azure
Domain Name System	Azure DNS
Backend Framework	Azure Functions
Languages of Backend Framework	Python, SQL, JSON
System of Backend System	Azure Function App
Frontend Framework	React JS
Languages of Frontend System	Javascript, HTML, CSS
System of Frontend System	Azure Static Web Apps
Frontend Autentication	Google Identity Platform
Source Controll	Git
Source Repository	GitHUB, Azure DevOps
CI/CD Platform of Backend System	GitHUB Actions
CI/CD Platform of Frontend System	Azure DevOps
Language of CI/CD Processes	YAML
Language of API Documentation	YAML
Database System	Azure Cosmos DB
Languages of Database	SQL, JSON
Social Media Platform	META (Facebook)
AI Engine	OpenAI
Search Engine of Images	Pexels
Compatible Runtime Environments	Linux, Windows, MacOS
Runtime Environment of CI/CD Processes	Linux

2. PLANNED DEPENDENCIES

During software development, it is common to use various external libraries and packages (see Table 2) to perform tasks more efficiently (in less development time) and achieve the desired functionality.

In the development of the Azure Functions backend system, I employ different Python libraries (see Table 2), and these libraries assist in implementing backend functionality.

During the development of the React frontend system, I use multiple Node Package Manager (NPM) packages (see Table 2). These packages aid in shaping frontend functions and the user interface, making it easier (e.g., accelerating) to develop React-based applications.

The external libraries and dependencies I use include:

Table 2.: Dependencies

Dependency types	**Dependencies**
Azure Functions Backend Preferred Installer Program (PIP) packages	openai, azure-cosmos, azure-functions, uuid
React Frontend System Node Package Manager (NPM) packages	axios, bootstrap, jwt-decode, react, react-dom, react-router-dom, react-scripts
React Frontend System scripts	Google GSI Client, Facebook SDK for Javascript

Azure Functions Backend System:

- openai: This Python library enables the use of the GPT-3.5 language model.
- azure-cosmos: This library assists in managing database operations in Azure Cosmos DB.
- azure-functions: This library allows for the development and management of Azure Functions applications in Python.
- uuid: This library aids in generating unique identifiers (UUID).

Embedded Scripts in the React Frontend System:

- Google GSI Client: This script is necessary for Google Sign-In integration, enabling users to log in with their Google accounts.

- Facebook SDK for JavaScript: This script is required for Facebook integration, allowing the retrieval of data from user-managed Facebook pages and requesting user consent for page management.

React Frontend System:

- bootstrap: This is a popular CSS and JavaScript framework that assists in designing simple and responsive web layouts.
- jwt-decode: This library helps decode JSON Web Tokens (JWT), which are used for user identification and authentication.
- react: This library is the foundation for developing React applications.
- react-dom: This library aids in rendering React components in the browser.
- react-router-dom: This library facilitates route management for React-based applications in the browser.
- react-scripts: This library includes developer tools and commands for running and building React applications.

3. ARCHITECTURE PLAN

Figure 1. illustrates the system architecture and the connections between individual components:

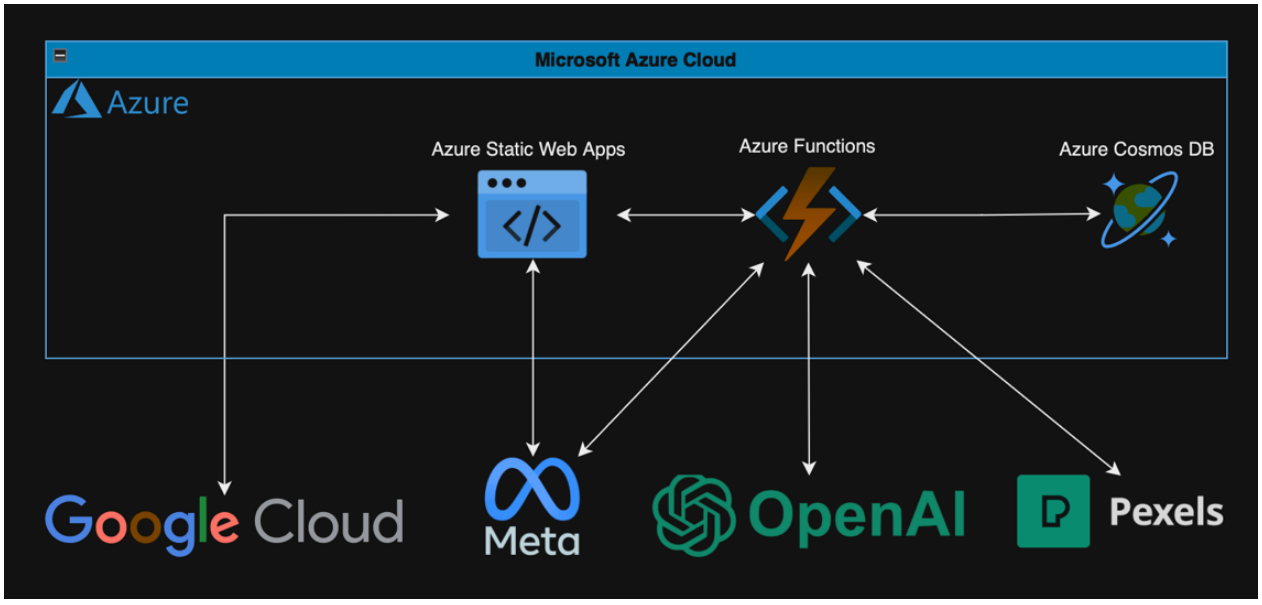

Figure 1.: System Architecture

Source: Self-made

4. API PLANNING

I used Swagger.io (https://editor.swagger.io) for the design and documentation of the API, an open-source tool for describing and documenting RESTful APIs. Swagger.io allowed me to specify API endpoints, parameters, responses, and data models in detail.

Below, I provide a more detailed explanation of each endpoint and its parameters:

GET /api/ImgGen:

This endpoint generates an image based on the provided text and returns the URL of the generated image. The endpoint expects the following parameters:

- text: The text for the generated image.
- code: The API authentication code. It is mandatory for endpoint authentication.

GET /api/UserAPI:

This endpoint returns data associated with the user based on the email input parameter from the database. The endpoint expects the following parameters for user identification:

- email: The email address for querying user data. This parameter is mandatory.
- code: The API authentication code. This parameter is also mandatory for endpoint authentication.

PATCH /api/UserAPI:

This endpoint allows updating user data in the database. The endpoint expects the following parameters:

- email: User's email address.
- billing_name: Billing name.
- billing_postalcode: Billing postal code.
- billing_town: Billing town.
- billing_taxnumber: Tax number.
- billing_address: Billing address.
- accessToken: Short-lived access token received from Facebook, see Figure 3.
- code: The API authentication code. It is mandatory for endpoint authentication.

GET /api/PostGen:

This endpoint generates text based on the provided parameters and returns the generated text and its associated image URL for social media posts. The endpoint expects the following parameters:

- company: Company name.
- website: Company website.
- activity: Company activity.
- about: Company description.
- language: Language of the post.
- code: The API authentication code. It is mandatory for endpoint authentication.

POST /api/PostGen:

This endpoint allows uploading generated text and images to social media platforms. The endpoint expects the following parameters:

- body: A JSON object containing data specifying the text to be uploaded, the authentication token, the social media page identifier, and the image URL. Mandatory fields include:
- textpost: The generated text.
- accesstoken: The authentication token.
- pageid: The identifier of the social media page.
- url: The URL of the image to be uploaded.
- code: The API authentication code. It is mandatory for endpoint authentication.

More detailed documentation can be found in the attachment of the publication

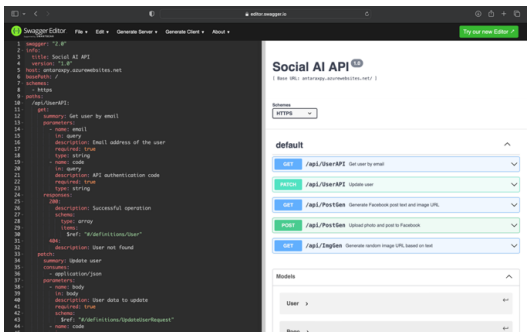

Figure 2.: API Documentation of the Software

Source: https://editor.swagger.io

5. FACEBOOK ACCESS TOKEN

The following documentation provides a detailed overview of the steps involved in requesting, renewing, and utilizing an Access Token, as well as managing permissions. This information assisted me in understanding and implementing functionalities related to integrating with Facebook:

https://docs.squiz.net/funnelback/docs/latest/build/data-sources/facebook/facebook-page-access-token.html

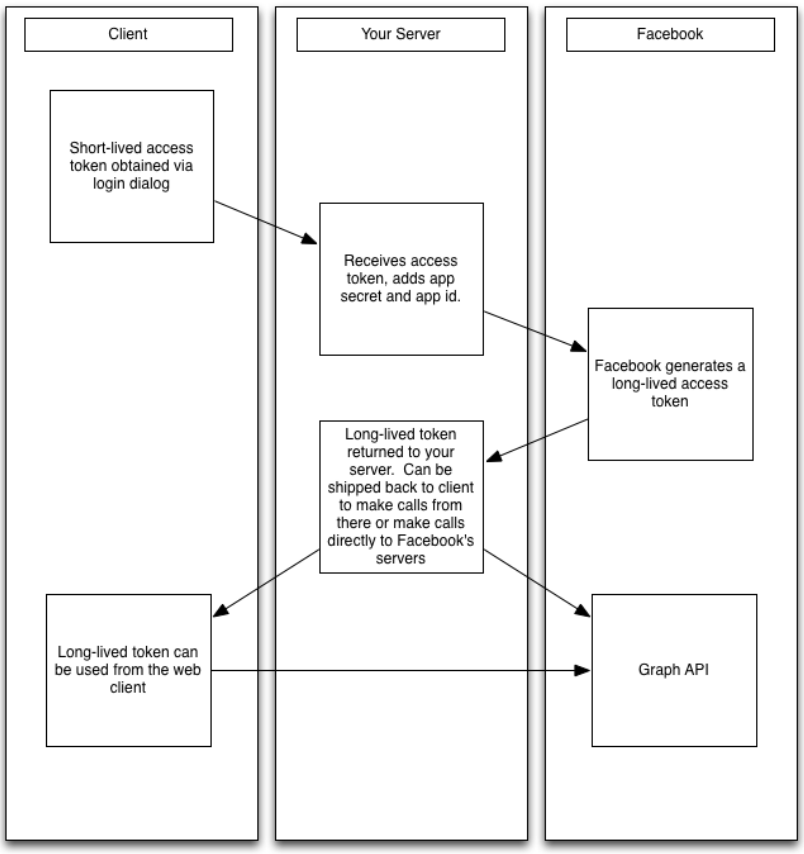

Figure 3.: Facebook Access Token process flow

Source: https://developers.facebook.com/docs/facebook-login/guides/access-tokens/get-long-lived

6. DATA MODELL

Azure Cosmos DB is a NoSQL database structured from JSON-format documents. The designed user object constitutes the attachment of the publication under the name "User.json" file. (see attachment number 2.)

CONCLUSION

In summary, the successful design of a cost-effective, cloud-based software solution leveraging ChatGPT for automating content creation on Facebook has been achieved. Integration of Microsoft, Google, and Meta services streamlined the development process. The goal of maintaining minimal operational costs (around 0 USD / month) was realized. Various programming and descriptive languages were effectively employed, as detailed in Table 1.

Special thanks to Dr. János Rikk and Dr. László Pitlik for their invaluable support in publication preparation.

REFERENCES

- Hungarian National Bank 4/2019. (IV.1) Recommendation on the usege of community and public cloud computing services, URL: https://www.mnb.hu/letoltes/4-2019-cloud-bg.pdf Downloaded: 2023.10.07.
- János Rikk, László Pitlik: ChatGPT-experiments-programming, 2023, URL: https://m2.mtmt.hu/gui2/?mode=browse&params=publication;34071055 Downloaded: 2023.10.08.
- Pitlik László: Token-based chatGPT3 – Example, 2023, URL: https://m2.mtmt.hu/gui2/?mode=browse&params=publication;34071014 Downloaded 2023.10.08.
- Rubóczki Edit Szilvia: „*The most revolutionary and perhaps the most popular IT infrastructure solution of the decade is cloud technology.", (translated with machine) URL: http://lib.uni-obuda.hu/sites/lib.uni-obuda.hu/files/Ruboczki_Edit_Szilvia_ertekezes.pdf page* 59. Downloaded 2023.10.07.

ATTACHMENTS

1. attachment: Social AI API.yaml

2. attachment: User.json