Documentation for Architecture & SQL Database

Database Overview:

I consulted the Microsoft Learn page to detail the process of creating a single database in Azure SQL Database via the Azure portal, Azure PowerShell, or Azure CLI. This guide includes setting up a new server, configuring the database, and selecting options such as authentication methods, database name, and compute + storage settings. It also provides steps for querying the newly set up database using Azure's Query editor and instructions for resource cleanup when they are no longer needed. I utilized the Azure Portal to create the database.

A screenshot of a computer

Description automatically generated

Figure 1 Snapshot of DB

I slightly modified the architecture to support the envisioned app version. To resolve compatibility issues, I reconfigured certain components, which involved separating the front end of the Mapalyze app from the React Planner app. The parser remained separate, but its alignment with the backend was enhanced. This architectural adjustment helped achieve our goals by facilitating rapid image uploads to the Mapalyze front end, edge detection processing by the parser, and subsequent image transmission to the React Planner backend for floor plan and 3D render conversion. We integrated the parser with the ReactPlanner backend as it was more practical, considering the RP backend's compatibility with Python, unlike our .NET-based React App (Mapalyze).

The challenge was to understand how this updated architecture operated while ensuring the SQL DB remained intact, particularly with Azure's subscription and payment issues. This report summarizes my focus during the last sprint, encapsulating our activities throughout the semester as we wrap up the project.

A diagram of a process flow

Description automatically generated

Figure 2 New Architecture of the App

* We use three microservices,
  + Pink – Mapalyze (FE & BE), written in ReactJS.
  + Blue – ReactPlanner (FE & BE), written in JavaScript.
  + Yellow – Parser, written in Python.

I modified the architecture slightly to support the app version we envisioned. To address compatibility issues, I reconfigured some components, which involved separating the front end of the Mapalyze app from the React Planner app. The parser remained separate, but we aligned it more with the back end. This architectural revision facilitated our objectives, enabling us to quickly upload a picture to the Mapalyze front end, receive edge detection from the parser, and send the image to the React Planner back end for conversion into a floor plan and a 3D render. We integrated the Parser with the ReactPlanner backend, as it was more feasible to do so. The RP backend is written in JavaScript, which integrates more smoothly with Python than our .NET React App (Mapalyze).