Editor in Azure

1. Develop Environment:

- VScode with Azure extensions for developing the application and have great support for web technologies and Azure services.

- Install SDKs and tools for chosen programming language and frameworks.

2. Application Architecture:

- Have front end and back end set up – done

- For 3D rendering, consider using libraries like Three.js or Babylon.js.

3. Create and Configure Azure Resources:

- Azure Portal/ Azure CLI -> create Azure App Service -> host web app.

- database: Azure SQL Database/Azure Cosmos DB

- Storing floor plans: Azure Blob Storage

4. Implement the 3D Layout Generation:

- Develop algorithms or use existing libraries to process floor plans and generate 3D layouts. This might involve complex image processing and geometric calculations.

- Ensure your app can appropriately mark stairs, hallways, and rooms, and has navigation features.

5. Develop the Web Application:

- Build the front end to allow users to upload floor plans and view the generated 3D layouts.

- Develop the back end to handle the processing of floor plans and serve the generated 3D models.

6. Integrate Azure Services:

- Azure Storage SDKs: integrate Blob Storage for storing floor plans and generated models.

- Implement any required database interactions.

7. Test Application:

- Thoroughly test the application, particularly the 3D generation logic and the user interface.

- Azure's integration with various CI/CD tools (Azure DevOps) for automated testing and deployment.

8. Deploy and Monitor:

- Deploy app to Azure App Service.

- Monitor your application by Azure Monitor and Application Insights to track performance and errors.

General process outline for creating API endpoint for web app:

1. Define the API Specification:

- Determine the functionalities your API needs to offer -> accept floor plan uploads and return a 3D layout.

- Define the endpoints -> request/response formats -> HTTP methods (POST for uploading floor plans and GET for retrieving 3D layouts).

2. Have Backend Framework:

3. Set Up the Backend Environment:

- new project in your chosen framework.

- Structure your project with routes, controllers, services, and models as needed.

4. Develop the API Endpoints:

- Implement a file upload endpoint to receive floor plans -> can involve multipart/form-data handling.

- Develop an endpoint to initiate the 3D layout processing and another one to retrieve the processed 3D layout.

- Ensure endpoints handle requests asynchronously, as the processing of floor plans to 3D layouts might take time.

5. Integrate 3D Layout Generation Logic:

- Incorporate the logic or service that processes the floor plans into 3D layouts. This might involve complex logic or the integration of a third-party library or service.

- Store the generated layouts in a retrievable format, possibly in cloud storage like Azure Blob Storage.

6. Test the API:

- Write unit and integration tests to ensure API behaves as expected.

- Test the API endpoints using tools like Postman or Swagger.

7. Deploy the API:

- Deploy API to a server or a cloud platform like Azure.

- deploy to Azure App Service/Azure Functions for a serverless approach.

9. Document the API:

- Provide clear documentation for API, including endpoint descriptions, request/response examples, and error handling.

- Swagger help auto-generate interactive API documentation.

10. Monitor and Maintain:

- Once deployed, monitor the API's performance and error rates.

- Implement logging and monitoring using tools like Azure Monitor or Application Insights.

11. Frontend Integration:

- ensure web app's frontend can interact with these API endpoints correctly, handling file uploads and displaying the 3D layouts.