Titan Challenge - Presentation

Objective - Requirements

Develop a robust API capable of searching for doctors within the USA using the NPI registry search API (https://npiregistry.cms.hhs.gov/search). Your solution should utilize AWS cloud services to enhance scalability and performance and integrate advanced searching techniques to optimize the accuracy and relevance of the search results.

Requirements

1. AWS Cloud Integration:

- Deploy the API on AWS, utilizing services such as AWS Lambda for serverless computing, and Amazon API Gateway.
- Use AWS Elastic Beanstalk or AWS ECS (Elastic Container Service) if a containerized deployment is preferred.

2. API Implementation:

- Develop an endpoint following REST practices:
- *GET /search/doctors: Accepts a doctor's name (first name, last name, and other fields you consider so that the result can be more accurate) and returns matching results.*
  Implement the solution in either TypeScript or Python.
- If using Python, FastAPI is recommended for its asynchronous support and rapid development features.
- If TypeScript is your choice, express-js-based frameworks are preferred.

3. Search Enhancement:
- Leverage the NPI registry search API as the primary data source.
- Explore advanced techniques such as fuzzy searching, embeddings, or large language models (LLM) to improve semantic matching. This is critical.
- Investigate additional data points that could refine search accuracy and result rankings. Narrowing down the geographical extent of the search could help (for example using the state, etc.)

4. Handling Different Search Outcomes:
- Ensure the API gracefully handles no results, a single result, and multiple results scenarios.

5. Front-End (Optional but Recommended):
   - Develop a TypeScript React-based front-end application to interact with your API, deployable on AWS for streamlined hosting and CI/CD integration.
   - The interface should allow users to search, view results, and use filters or sorting based on the data retrieved.

6. Documentation and Version Control:
   - Use GitHub for version control and sharing the final project.
     Include a README file with setup instructions, API usage examples, and an explanation of design choices.
   - Include a diagram with your solution architecture.
   - Include all source code and tests.
   - (Bonus) If you can setup Github Actions that deploy changes pushed to your main branch into the AWS Stack

7. Testing:
   - Include unit and integration tests to ensure application reliability, especially focusing on cloud deployments and data handling.

Solution Proposed and Implemented

APP URL: https://main.d2f8rmwuhugvil.amplifyapp.com/
APP Repo: https://github.com/FelipeGarcia911/doctors-engine-app

API URL: https://03v67xragj.execute-api.us-east-1.amazonaws.com/
API Swagger: https://03v67xragj.execute-api.us-east-1.amazonaws.com/api-docs/
API Repo: https://github.com/FelipeGarcia911/doctors-engine-api

API: General App Specs

- Language: Typescript
- Framework: NodeJS + ExpressJS
- Type: REST API
- Deployment: CI/CD using Github Actions
- Cloud: AWS Lambda + AWS Gateway
- Documentation: Swagger

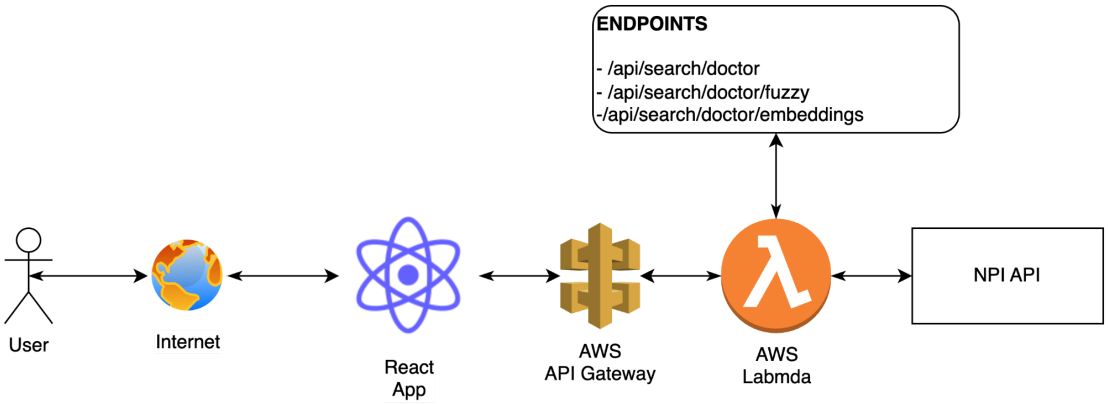

Image 01. Doctors App Architecture

API: Endpoints:

- Search Params:
  - number
  - first_name
  - last_name
  - postal_code
  - city
  - state
  - country
  - limit

- */api/search/doctor:*
  Simple and direct search against NPI API using the general request parameters.

- */api/search/doctor/fuzzy:*
  Sort NPI API results using fuzzy search.

- */api/search/doctor/embeddings*
  Sort NPI API results using a preconfigured embedding model.

- */api-docs*
  Swagger documentation UI

API: AWS Cloud Integration
A GitHub Action to deploy the code into an AWS Lambda once a pull request is merged into the main branch.

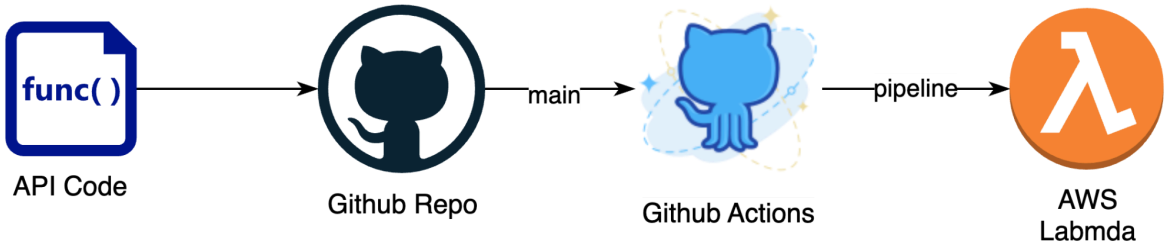

Image 2. API Cloud AWS Integration Architecture

API: Search Enhancement
For the search enhancements, we get the NPI Search Results and then we implement a sorting technique to retrieve, at the top, the most relevant results.

- **Embeddings**
  We implemented the model *sentence-transformers/paraphrase-xlm-r-multilingual-v1* to compare the similarity between a base sentence and sentences generated from API results. Using the parameters obtained from the user's query (such as first_name, last_name, etc.), we construct a ***baseQuery***. For each API result, we create a ***similarQuery*** to form an array of strings. These queries are then, inputted into the model along with the baseQuery, resulting in an array where each number represents the similarity score between the baseQuery and each *similarQuery*. Finally, we sort the API results based on these similarity scores.

  Score: Since it uses a cosine-similarity algorithm, 0 indicates a complete mismatch and 1 perfect match.

- **Fuzzy Search**
  Similar to the Embeddings implementation, we create a ***baseQuery*** using the user's params and the API Results to create a ***similarQuery*** for each API result; next, these queries are added into the FuzzySearch function and then we get the similarity score. Finally, we sort the API results based on these similarity scores.

  Score: "*Whether the score should be included in the result set. A score of 0 indicates a perfect match, while a score of 1 indicates a complete mismatch.*"

API: Documentation and Version Control

- Swagger was implemented to show and document all routes available in the application.
- Please check the repo listed at the beginning.

API: Testing

The Husky library was implemented to run the testing pipeline before adding a commit locally.

- A GitHub Action to run the testing pipeline on a pull request against the main branch

API: Key results and findings from the project

- Working with NodeJS + Express is straightforward since you can create an API quickly.
- I started using Tensorflow to implement the Embeddings feature but I couldn't complete it. It worked locally but I had several issues on AWS and then trying to compile a docker image.
- After moving outside TensorFlow, I found another JS library with a simple embedding implementation, but again, it had building issues.
- Embeddings and, usually IA models, could help a lot but it has some cost. The embeddings endpoint takes longer than the fuzzy search one. It's a good point to keep in mind if you are going to implement this on production and with thousands of users.
- I am glad there are plenty of JS AI libraries that help us a lot in implementing, very easy, different solutions.
- AWS Lambda offers a good serverless option to implement these kinds of APIs with "low" overhead.

API: Future enhancements and potential improvements.

- Implement authentication
- Implement pagination on all search endpoints
- Implement some AI models to improve the results sorting.
- Implement auditing
- Improve logger methods
- Configure Prettier and Linter

Frontend: App Specs

- Language: Typescript
- Framework: ReactJS
- UI: MaterialUI
- Deployment: Github + AWS Amplify

Frontend: AWS Cloud Integration

AWS Amplify was implemented to deploy the site. It's connected directly to the repo and it listens when a branch is merged into the main and then, starts the deployment process.

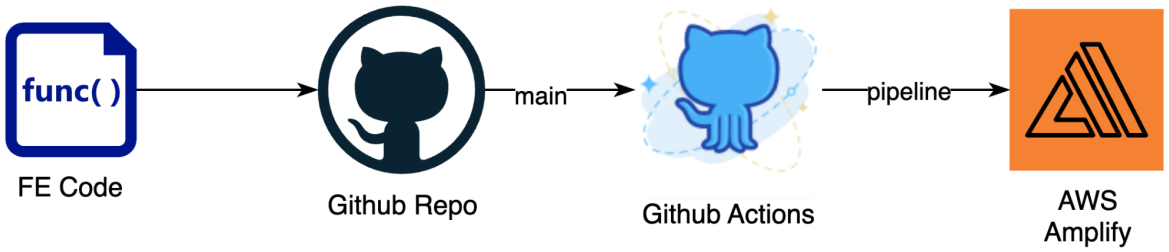

Image 3. Web App AWS Cloud Integration Architecture

Frontend: Key results and findings from the project

- Using MaterialUI helps a lot since it has most of the regular required components to create an application.
- ReactJS gives you enough tools to create from a simple app like this one or very complex applications.

Frontend: Future enhancements and potential improvements.

- Improve form validation.
- Improve UI for better UI/UX.
- Implement unit testing and integration testing.
- Improve error handling.
- Implement more advanced UI customization (Styles Component)
- Configure Prettier and Linter

AI Tools Used:

- Github Copilot: It was used as an integrated extension in VS Code. I have been using this tool for several months and it helps to solve and complete some small code blocks you're working on.
- OpenAI ChatGPT: It was used here to guide me in some scenarios I needed some help such as:
    - Compiling TensorFlow in a Docker Image
    - How to integrate Fuzzy Search and Embeddings in a Search Enhancement
    - How to create GitHub Actions to test and deploy into AWS
    - Resolve general development process questions

References:

- https://arxiv.org/abs/1908.10084
- https://aws.amazon.com/es/amplify/
- https://aws.amazon.com/es/lambda/
- https://huggingface.co/
- https://huggingface.co/stabilityai/stable-diffusion-3-medium
- https://www.fusejs.io/

Autor:
Arles Felipe Garcia Maya
Electronic Engineer - Universidad Tecnológica de Pereira
MSc Systems Engineering - Universidad Nacional de Colombia - Medellín
Team Lead - Fullstack Developer