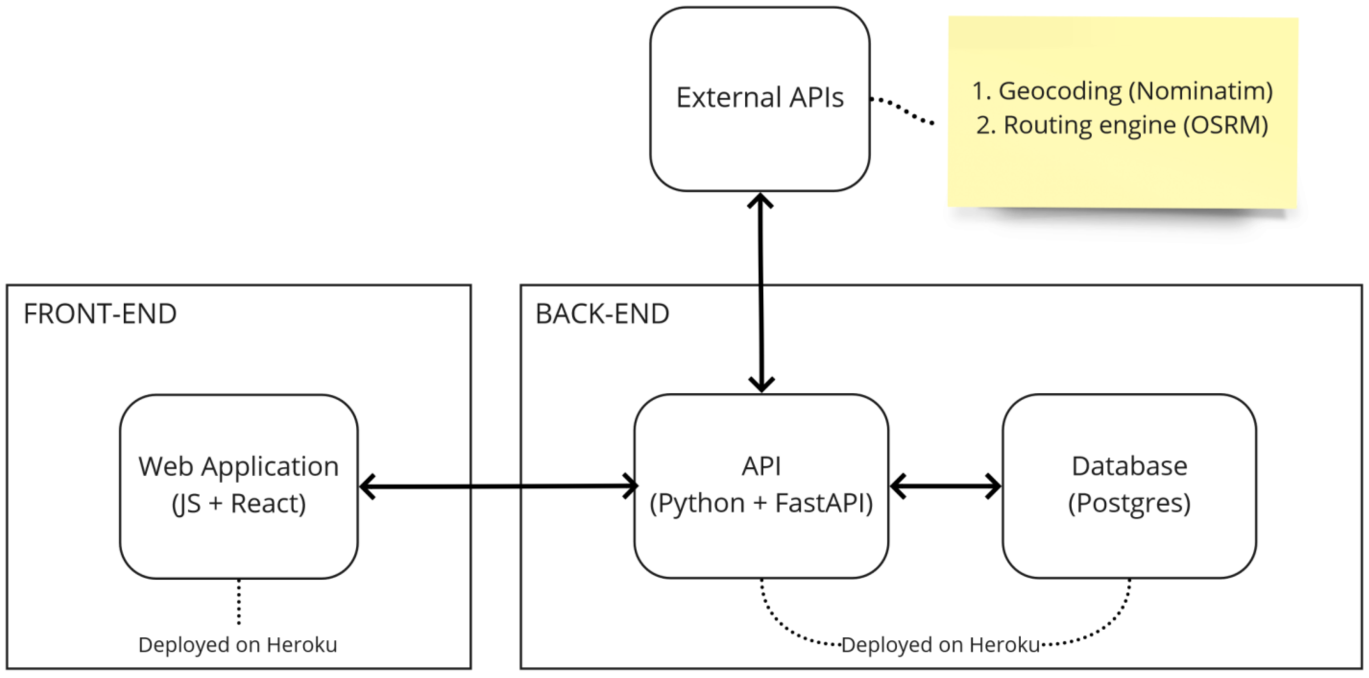
Delivery Application Challenge

Answers

Question 1

I've separated the project into front-end and back-end, developed and deployed them independently. The backend consists of only one microservice, built as a RESTful API. The adoption of separate components with React and FastAPI allows maintainability, quick delivery, and the project to be scaled in the next stages.

1. Frontend (React)
   - Provides a web application with user interface
   - HTTP requests to Backend
2. Backend / API (FastAPI)
   - Write in Python with FastAPI
   - Provides an API to communicate to front-end
   - Communication with Database
   - Request to External APIs
3. External APIs
   - Geocoding (Nominatim)
   - Shortest route (OSRM)
4. DataBase (Postgressql)
   - Store all searched routes



Question 2

The project adopted the micro-service architecture, for the reasons below:

1. Independence: The components are isolated from each other, allowing different teams or professionals to develop them independently and allowing to adopt different tools that better solve the problem;
2. Fast deploy: It's better to adopt microservices to achieve CI/CD, letting each service have its own smaller amount of automated tests and code coverage for example;
3. The components communicate through HTTP/REST protocol, simpler and easier to set up;
4. Looking further: It would be agile to scale the application by developing a different feature (like user registration) into a new microservice without any interference with the already existing microservices;
5. Project Functionalities: besides the idea of microservices comprising multiple services working independently, each one with your database, the project proposal allowed me to develop with only one database, one microservice, and one user interface.

Question 3

I would go with extreme programming because of three main reasons:

1. The team iterations are shorter, turning the development more agile and the feedback more often;
2. The features can be mutable along the development, by modifying them or even adding new ones. This gives more flexibility to requirements gathering;
3. The team members adopt pair programming and have more responsibilities with continuous integration, by automating testing, refactoring, and code standards.

Question 4

I would adopt trunk-based development to keep commits and branches smaller and merge with the main branch daily, making coding review easier and turning the project development more agile through code integration and small releases more frequently.

Question 5

That's difficult to answer. It will depend on schedule, budget, and size of each project. I think for proof of concept projects, smaller and more versatile teams are better to take the project out of paper. For example, accumulating the role of a software engineer and product owner.

Along the project evolves, the first thing to do out of a prototype stage would be to get a product owner onboard, this role ensures better communication between all stakeholders and most importantly, that the project will be delivered in time and generate real value to the customer.

Depending on the schedule versus the size of the project, adding more developers and distributing tasks along the team helps to meet the deadline delivery without sacrificing the quality of the project. Nonetheless, this decision needs to be evaluated and planned by the product owner at the early stage of a project, considering that not always do bigger teams do a better job, the communication can be harder, and members added to an ongoing project will go through a learning curve until achieving good deliverability.

Question 6

Independently of the project stage, the adoption of automated tests (unit, functional, integration…) should be mandatory to spend less time finding code mistakes or doing manual testing. Furthermore, automated tests are an important practice to get projects into production quickly with quality assurance.