Opti'tour: Architectural and Design Choices

Elise BACHET, Andy GONZALES, Louis LABORY, Jason LAVAL, William MICHAUD, Lou REINA-KUTZINGER

INSA Lyon – October 2025

Abstract

This document presents and explains the main **Architectural and Design Choices** made during the development of the Opti'tour project. It focuses on the reasoning behind both the **backend (FastAPI)** and the **frontend (React)** architectures, as well as their interactions within an Agile context.

1. Back-end Architecture (FastAPI)

The backend follows the **Model–View–Controller (MVC)** architectural pattern. While **FastAPI** does not explicitly enforce this structure, we adopted it to improve **modularity**, **separation of responsibilities**, and **maintainability**, which perfectly suits an **Agile project**.

- **Model layer:** Defined in `schemas.py`, this layer contains the data models and class definitions that represent the entities of our system.
- **View layer:** Exposed through FastAPI endpoints, directly accessed by the frontend to send and retrieve data.
- **Controller layer:** Contains the **business logic**, orchestrating interactions between models and views, validating inputs, managing state changes, and returning structured responses.

This structure provides clear boundaries between layers and facilitates **independent development, testing, and refactoring** throughout the project's lifecycle.

2. Front-end Architecture (React)

The frontend is built using **React** and follows a **component-based architecture**. The user interface is composed of independent, reusable components organized hierarchically, ensuring clarity and scalability. Each component manages its own **logic**, **rendering**, and **styling**, which promotes modularity and maintainability.

Although React does not inherently rely on classical object-oriented patterns, our design incorporates aspects of the **State Design Pattern**. In traditional **OOP**, this pattern enables systems to change behavior depending on their current state. In React, this concept is implemented through **Hooks** (such as `useState` and `useEffect`), which allow components to react dynamically to state changes and ensure smooth, real-time updates of the UI.

3. Integration Between Back-end and Front-end

Communication between the two layers occurs through **RESTful API calls**. The React frontend sends HTTP requests to FastAPI endpoints, which process data and return JSON responses. This approach ensures:

- A clear **separation of concerns** between logic and presentation.
- Simpler debugging and testing, as each layer can be validated independently.
- Easier future migration to microservices or additional client interfaces.

Nota Bene: The combination of **FastAPI (backend)** and **React (frontend)** provides a flexible, scalable, and maintainable architecture. It supports **rapid iteration**, **clear communication between components**, and aligns perfectly with the principles of the **Agile methodology**.