Frontend Development with LLMs: Intro

CS 485/698: AI-Assisted SE

Today’s Agenda

- Team meeting (~20 minutes)
  - Sprint planning
- Short lecture on frontend development + mocking
- In-class activity and discussion
- Time for you to fill out a survey on how we’re doing

Sprint Planning

- Spend the next ~20 minutes with your project team
- This is our first official ***sprint planning meeting***:
  - Decide on what your overall goals for P3 are. What will you deliver by then?
  - Which backlog items are needed for this goal?
    - Do you need to create any?
  - Who is responsible for accomplishing each item?
    - When will you work on each item? Are there internal deadlines? How will you know if someone is stuck?

Frontend Development

- Questions we want to answer:
  - Can LLMs speak HTML and CSS?
  - Can LLMs read and render a DOM?
  - Can LLMs program in JavaScript?
  - Can LLMs tell you what to put into a session cookie?
  - Can LLMs design a web page? A whole web site?
  - Can LLMs *critique* a web site design and implementation?

Frontend Development

- Questions we want to answer:
  - Can LLMs speak HTML and CSS?
  - Can LLMs read and render a DOM?
  - Can LLMs program in JavaScript?
  - Can LLMs tell you what to put into a session cookie?
  - Can LLMs design a web page? A whole web site?
  - Can LLMs *critique* a web site design and implementation?
- Before we actually try it, what does the class think?

From Requirements Eng. to Frontend Dev.

- We’ve already looked at some requirements engineering processes:
  - User Story → UX Storyboard → Wireframe
  - User Story → Development Specification

From Requirements Eng. to Frontend Dev.

- We've already looked at some requirements engineering processes:
  - User Story → UX Storyboard → Wireframe
  - User Story → Development Specification
- Today, we're going to actually *use* some of these artifacts to generate code:
  - User Story + Wireframe + Dev Spec → Frontend Code

From Requirements Eng. to Frontend Dev.

- We've already looked at some requirements engineering processes:
  - User Story → UX Storyboard → Wireframe
  - User Story → Development Specification
- Today, we're going to actually *use* some of these artifacts to generate code:
  - User Story + Wireframe + Dev Spec → Frontend Code

First, though, a brief detour to discuss ***software architecture***

Software Architecture

- Recall that architecture and design are the “**glue**” between what your software is supposed to do and the code you actually write

Software Architecture

- Recall that architecture and design are the "**glue**" between what your software is supposed to do and the code you actually write
- Purpose of architecture: explain the **overall structure** of the system without getting bogged down in the details

Software Architecture

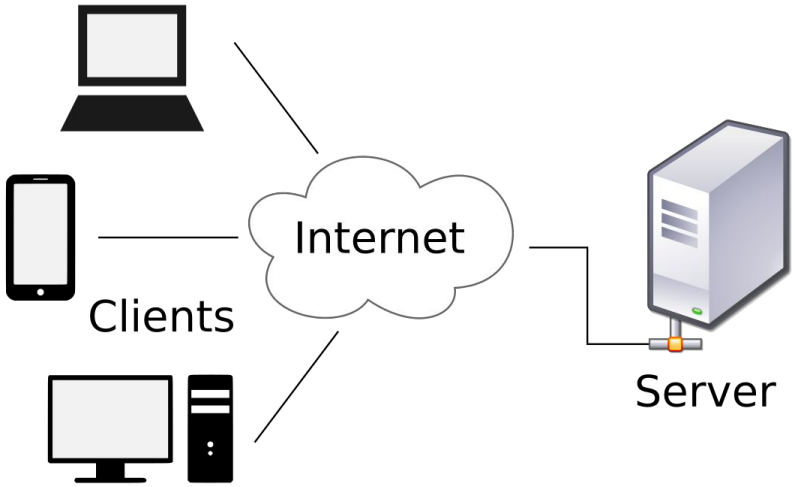
- Recall that architecture and design are the "**glue**" between what your software is supposed to do and the code you actually write
- Purpose of architecture: explain the **overall structure** of the system without getting bogged down in the details
  - "***Frontend***" vs "***Backend***" is an architectural shorthand:
    - "frontend" = parts of the system that users see
    - "backend" = parts of the system that they don't

Software Architecture

- Recall that architecture and design are the "**glue**" between what your software is supposed to do and the code you actually write
- Purpose of architecture: explain the **overall structure** of the system without getting bogged down in the details
  - "***Frontend***" vs "***Backend***" is an architectural shorthand:
    - "frontend" = parts of the system that users see
    - "backend" = parts of the system that they don't
  - Typically webservices use a ***client-server architecture***
    - client is the frontend, server is the backend

Typical Webservice Architecture

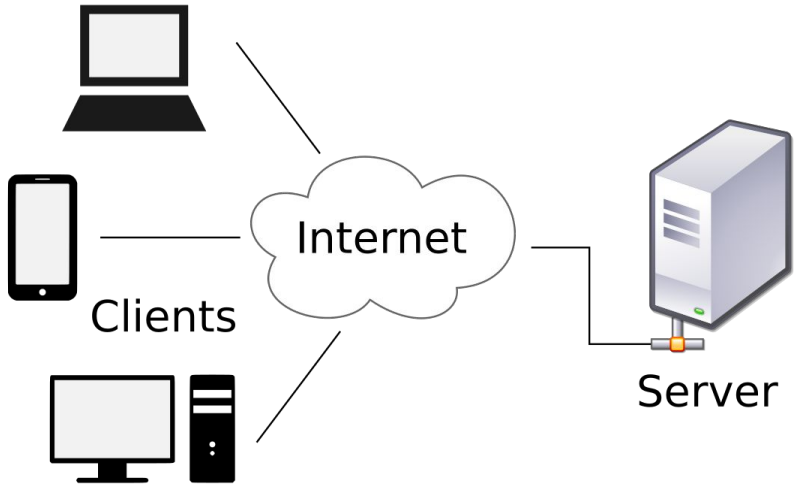
Definition: a ***client-server architecture*** partitions tasks or workloads between the providers of a resource or service (***servers***) and service requesters (***clients***) [Wikipedia]



Typical Webservice Architecture

Definition: a ***client-server architecture*** partitions tasks or workloads between the providers of a resource or service (***servers***) and service requesters (***clients***) [Wikipedia]

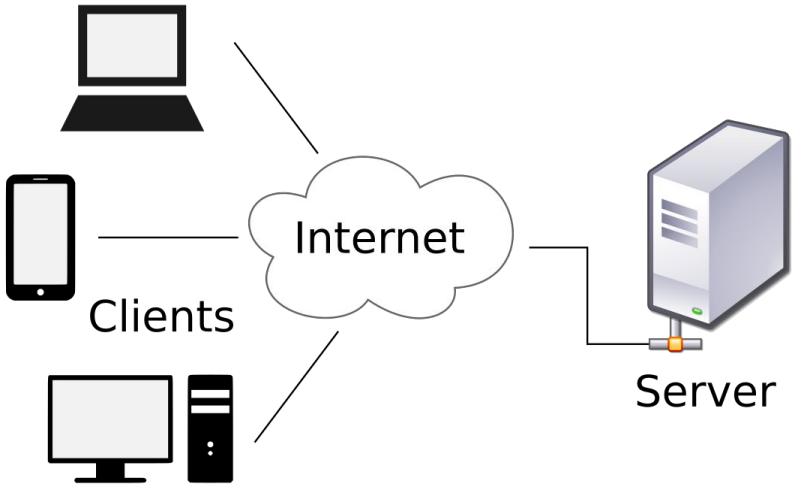
- network doesn't have to be the internet (client and server can even be on the same machine!)



Typical Webservice Architecture

Definition: a ***client-server architecture*** partitions tasks or workloads between the providers of a resource or service (***servers***) and service requesters (***clients***) [Wikipedia]

- network doesn't have to be the internet (client and server can even be on the same machine!)
- example of decomposition: server has its **own architecture** internally, but we don't see it



Why Architecture? Decomposition

- Decomposing a webservice into the frontend and backend lets us work on them ***independently***
  - And, potentially, in parallel (though we won't do that today)

Why Architecture? Decomposition

- Decomposing a webservice into the frontend and backend lets us work on them ***independently***
  - And, potentially, in parallel (though we won't do that today)

Why would we care about the ability to work on them in parallel in the real world?

Why Architecture? Decomposition

- Decomposing a webservice into the frontend and backend lets us work on them ***independently***
  - And, potentially, in parallel (though we won't do that today)

Why would we care about the ability to work on them in parallel in the real world?
Can assign to **different teams**!

Why Architecture? Decomposition

- Decomposing a webservice into the frontend and backend lets us work on them ***independently***
  - And, potentially, in parallel (though we won't do that today)
- There's only one problem…

Why Architecture? Decomposition

- Decomposing a webservice into the frontend and backend lets us work on them ***independently***
  - And, potentially, in parallel (though we won't do that today)
- There's only one problem…
  - …the frontend has ***dependencies*** on the backend
    - e.g., it needs data to display

Why Architecture? Decomposition

- Decomposing a webservice into the frontend and backend lets us work on them ***independently***
  - And, potentially, in parallel (though we won't do that today)
- There's only one problem…
  - …the frontend has ***dependencies*** on the backend
    - e.g., it needs data to display
- Question: how do we work on the frontend without actually having a backend implementation yet?

Why Architecture? Decomposition

- Decomposing a webservice into the frontend and backend lets us work on them ***independently***
  - And, potentially, in parallel (though we won't do that today)
- There's only one problem…
  - …the frontend has ***dependencies*** on the backend
    - e.g., it needs data to display
- Question: how do we work on the frontend without actually having a backend implementation yet?
  - Answer: ***mocking***

Mocking

Definition: ***Mocking*** uses “fake” APIs to test the behavior of some other part of a software system.

Mocking

Definition: ***Mocking*** uses “fake” APIs to test the behavior of some other part of a software system.

- analogy: use a crash test dummy instead of real human to test automobiles

Mocking example: Web API Dependency

- Suppose we're writing a single-page web app
- The API we'll use (e.g., Speech to Text, an LLM, etc.) hasn't been implemented yet or costs money to use
- We want to be able to write our frontend (website) code without waiting on the server-side developers to implement the API and without spending money each time
- What should we do?

Mocking example: Web API Dependency

- Solution: make our own “fake” (“mock”) implementation of the API
- For each method the API exposes, write a substitute for it that just returns some **hardcoded data** (or any other approximation)
    - Why does this work?

Frontend Development Approach

- Our **cunning plan** for frontend development:
  - define the interface between frontend and backend
    - in particular, this requires us to specify the APIs well

Frontend Development Approach

- Our **cunning plan** for frontend development:
  - define the interface between frontend and backend
    - in particular, this requires us to specify the APIs well
  - mock the backend
    - hardcode API responses (we'll implement properly later)

Frontend Development Approach

- Our **cunning plan** for frontend development:
  - define the interface between frontend and backend
    - in particular, this requires us to specify the APIs well
  - mock the backend
    - hardcode API responses (we'll implement properly later)
  - design and build a nice frontend, with all the usual nice things (taking advantage of lots of web frontend code in LLM training data):
    - navigation, responsive layout, etc.
    - use standard web frameworks (e.g., React, Svelte)

In-class Activity: User Story -> Frontend Code

Goal: turn this user story into a front end with your project team:

As a student, I want to be able to send an email to my parents every day to let them know I'm ok and not to worry about me. It would relieve me if I could be sure I didn't miss a day.

In-class Activity: User Story -> Frontend Code

Goal: turn this user story into a front end with your project team:

As a student, I want to be able to send an email to my parents every day to let them know I'm ok and not to worry about me. It would relieve me if I could be sure I didn't miss a day.

Activity steps:

1. Use Figma Make to create a UX storyboard with suggested screen mockups.
2. Use Figma Make to turn one storyboard screen mockups into a wireframe.
3. Use an LLM to turn the user story and UX storyboard into a development spec.
4. Use an LLM-enabled IDE to turn the dev spec and wireframe into a new front-end project that uses React. Mock the backend with a well-defined REST API.
5. If you finish ahead of time, an additional activity: use an LLM-enabled IDE to turn the dev spec and wireframe into a new frontend project that uses Svelte instead of React. Mock the backend with a well-defined REST API. What differs?

Whole-class Discussion

- How did you pass graphics to the LLMs?
- What did your code generation LLM need as inputs to ensure it generated the right front-end code?
  - Did anyone forget to give it something? What happened?
- Did the generated code match your expectations?
- How many rounds did it take for the LLM to generate good code?
- What are the differences between the project using React vs using Svelte?

How are we doing?

Don’t forget to sign up for A3 reflection topics!

https://forms.gle/cBCAQBY3cE68h2v46