Backend Development

CS 485/698: AI-Assisted SE

Today’s Agenda

- P3 grading standards discussion
- P4 deadline discussion
- Team meeting (~15 minutes)
  - Standup meeting: check in with your team, discuss anything blocking you
  - Michael and I will be coming by to check in with your teams
- In-class activity: making matches in the backend for “AI Tinder”

P3 Grading Standards

- We released P3 grades a few hours ago
- Grades were bimodal: ~half of you got nearly full credit, ~half got significant deductions

P3 Grading Standards

- We released P3 grades a few hours ago
- Grades were bimodal: ~half of you got nearly full credit, ~half got significant deductions
- Most deductions were for UIs that ***didn't actually work***
  - That is, when we tested them ourselves, it was clear that they were incomplete: buttons that didn't work, missing pages, etc

P3 Grading Standards

- We released P3 grades a few hours ago
- Grades were bimodal: ~half of you got nearly full credit, ~half got significant deductions
- Most deductions were for UIs that ***didn't actually work***
  - That is, when we tested them ourselves, it was clear that they were incomplete: buttons that didn't work, missing pages, etc
- Our standard (for P3 and going forward): if a contractor/freelancer gave us this work, would we hire them again?
  - That means we're rewarding completeness, attention to detail

P3 Grading Standards

There will not be “carried through error” in P4 grading. We will expect everything in the UI to be working.

- We released P3 grades a few hou
- Grades were bimodal: ~half of you got nearly full credit, ~half got significant deductions
- Most deductions were for UIs that ***didn’t actually work***
  - That is, when we tested them ourselves, it was clear that they were incomplete: buttons that didn’t work, missing pages, etc
- Our standard (for P3 and going forward): if a contractor/freelancer gave us this work, would we hire them again?
  - That means we’re rewarding completeness, attention to detail

P4 Deadline

- The P4 deadline is currently Friday AoE
  - Reasoning: I don’t want to assign you work over spring break

P4 Deadline

- The P4 deadline is currently Friday AoE
  - Reasoning: I don’t want to assign you work over spring break
- If the class prefers it, we can extend the deadline:
  - …to Sunday AoE
  - …to the end of spring break (i.e., 3/22 AoE)
  - …to something else

P4 Deadline

- The P4 deadline is currently Friday AoE
  - Reasoning: I don’t want to assign you work over spring break
- If the class prefers it, we can extend the deadline:
  - …to Sunday AoE
  - …to the end of spring break (i.e., 3/22 AoE)
  - …to something else
- Let’s take a secret vote. Close your eyes…

Team Meeting (~15 min)

- Standup meeting:
  - Check in with your team
  - Discuss anything blocking you
  - Be ready to answer our question: **what can you demo by Wednesday?**
- Michael and I will be coming by to check in. Ask us about:
  - P3 grading
  - P4 expectations

Backend: Data

Backend: Data

- A common reason that you may need a separate backend for an application is **persistent or shared data**

Backend: Data

- A common reason that you may need a separate backend for an application is **persistent or shared data**
  - For example, our AI Tinder example needs to keep track of who has liked/rejected/super-liked whom to identify matches
    - But we don't want that data on either client - it's important to avoid telling users that someone else likes them until they've liked them back (core promise of the app!)

Backend: Data

- A common reason that you may need a separate backend for an application is **persistent or shared data**
  - For example, our AI Tinder example needs to keep track of who has liked/rejected/super-liked whom to identify matches
    - But we don't want that data on either client - it's important to avoid telling users that someone else likes them until they've liked them back (core promise of the app!)
  - Other examples: messages in Discord, posts on Reddit, …

Backend: Data

- A common reason that you may need a separate backend for an application is **persistent or shared data**
  - For example, our AI Tinder example needs to keep track of who has liked/rejected/super-liked whom to identify matches
    - But we don't want that data on either client - it's important to avoid telling users that someone else likes them until they've liked them back (core promise of the app!)
  - Other examples: messages in Discord, posts on Reddit, …
- Typical solution: some kind of **database**

Backend: Databases

- You probably have taken (or will take) a databases class
  - At NJIT, this is CS 331

Backend: Databases

- You probably have taken (or will take) a databases class
  - At NJIT, this is CS 331
- This class probably focused on ***relational databases***
  - SQL for queries, transaction support, etc.
  - Examples: Sqlite, MySQL, Postgres, …

Backend: Databases

- You probably have taken (or will take) a databases class
  - At NJIT, this is CS 331
- This class probably focused on ***relational databases***
  - SQL for queries, transaction support, etc.
  - Examples: Sqlite, MySQL, Postgres, …
- However, a relational database is not the only option for data storage in a webservice
  - In-memory cache (e.g., Redis) for short-term data
  - Blob storage (e.g., S3) for rarely-accessed data
  - …

Backend: Databases

Choose the right storage for the needs of your application!

- You probably have taken (or will take) a databases class
  - At NJIT, this is CS 331
- This class probably focused on ***relational databases***
  - SQL for queries, transaction support, etc.
  - Examples: Sqlite, MySQL, Postgres, …
- However, a relational database is not the only option for data storage in a webservice
  - In-memory cache (e.g., Redis) for short-term data
  - Blob storage (e.g., S3) for rarely-accessed data
  - …

Backend: Durability

- Typical metric for data storage: ***durability*** (how much of your data can you still retrieve after a fixed time has passed?)

Backend: Durability

- Typical metric for data storage: ***durability*** (how much of your data can you still retrieve after a fixed time has passed?)
- How to achieve durability in practice?
  - ***Replication***: store multiple copies, keep up-to-date
  - Pay someone to do it for you (this is e.g., AWS' business model)
  - Note different concerns at scale (e.g., how often do hard drives fail?)

Backend: Durability

- Typical metric for data storage: ***durability*** (how much of your data can you still retrieve after a fixed time has passed?)
- How to achieve durability in practice?
  - ***Replication***: store multiple copies, keep up-to-date
  - Pay someone to do it for you (this is e.g., AWS' business model)
  - Note different concerns at scale (e.g., how often do hard drives fail?)
- Storing data also has legal/security implications:
  - GDPR, similar regulations may require deletion on request
  - Best practice: **encrypt data at rest**: it's harmless even if it leaks

Backend: Interacting with the Frontend

- Often, our business logic is going to require us to interact with the frontends of various users
  - E.g., a ride-hailing app needs to notify the user that a driver is approaching
  - Often this involves the data of more than one user
    - Have to make sure we actually interact with the correct frontends - we don’t want to notify the wrong users!

Backend: Interacting with the Frontend

- Often, our business logic is going to require us to interact with the frontends of various users
  - E.g., a ride-hailing app needs to notify the user that a driver is approaching
  - Often this involves the data of more than one user
    - Have to make sure we actually interact with the correct frontends - we don't want to notify the wrong users!
- Different kinds of applications use different strategies for the ***notifications*** that they send to users

Backend: Two Ways to Do Notifications

Backend: Two Ways to Do Notifications

- ***Frontend poll***: every *n* seconds, the frontend calls a backend endpoint to see if there are any new matches
  - How's this for battery life (especially if you're not popular)?

Backend: Two Ways to Do Notifications

- ***Frontend poll***: every *n* seconds, the frontend calls a backend endpoint to see if there are any new matches
  - How's this for battery life (especially if you're not popular)?
- Whenever the backend discovers a match, it sends a ***push notification*** to the frontend UI of each user
  - The user who swiped last is easy to notify
  - What if the other user isn't currently using the app?
    - Wait til they turn on the app
    - Force their computer to notify them now

Backend Development: In-Class Activity

- Find person with same letter/number as you
  - Early in the alphabet/smaller number = front of room
- If you have a **letter**, you're doing a **frontend poll**; if you have a **number**, you're doing **push notifications**
  - Either way, specification/instructions on next slide
- Pick one of your codebases from last Wednesday to serve as a base
  - Doesn't matter which, you decide
- **Pair programming** activity: two people, one computer
  - Close all but one computer!

Polling (letters)

- Make the frontend poll the backend every 10 seconds for matches.
- The backend needs a function to receive poll requests.
- Mock the ‘likes’ database so that you’re likely to get a match during testing
- Bonus: make a real local ‘likes’ database

Pushes (numbers)

- Frontend UI checks for Push API support.
- Register a service worker to listen for notification events.
- Ask the user for permission to send push notifications.
- Subscribe to notifications by creating a subscription object using the applicationServerKey.
    - An applicationServerKey is generated by the backend and passed to the client during subscription. It has a public key and a private key for encrypting notification messages.
- Send the subscription object to the backend.
- When there’s an event, the backend’s Push Manager will send push events to all relevant subscriptions.
- Frontend’s service worker will listen for the event and trigger a push notification in the frontend UI.

In-Class Activity: Show and Tell

- Explain, compare, and contrast your backend with another pair
  - see table for assignments -------->
- Start by spending ~2-3 minutes explaining your backend to the other pair
- Then, discuss + share tooling tips:
  - Which was easier to build? Why?
  - Which was easier to test? Why?
  - How much does the level of detail in the spec matter?

A	**1**
B	**2**
C	**3**
D	**4**
E	**5**
F	**6**
G	**7**
H	**8**
I	**9**
J	**10**

In-Class Activity: Discussion

- Which backend was easier for the LLM to implement?
  - Does the complexity of the specification matter?
  - Does the amount of detail in the specification matter?
- Which backend is better? How do you know?
- Are “better” backends always harder to implement (using an LLM)? Why or why not?
- Security: how can you tell that only the correct frontends can see each match?
  - How would you test this?

Wrapup and Reminders

- A4 due Wednesday before class
- P4 due Friday AoE