

Recitation 10 - Final Project Team

Workflows + Websockets

Introduction

- In the previous recitations, you were introduced to Vue, a reactive framework for the web.
- In today's recitation, we'll be covering tips for the final project (particularly for working as a team) as well as exploring websockets, which may help some of your projects that need real time interactions.
- :) last recitation for the semester

Agenda

- Advice for team workflows
 - Splitting workload
 - Team dynamics
 - Deployment
 - Git workflow
- Project idea tips
- WebSocket example (Socket.IO)
 - Small messaging app
- Programming Resources

Splitting Workload

- So far, you've been designing and developing individually on your assignments. Now you'll be working in mostly teams of 4 (though some are 3 or 5, and we'll take that into account when evaluating the projects). There will be new challenges and
- **Team Contract**
 - **Before you begin your work as a team**, agree on a "team contract." Doesn't take that much time to discuss but greatly reduces the chance of derailment by misunderstandings and disagreements.
 - Open and candid discussion about commitments + aspirations

- how much time do you plan to put in? personal goals? how to resolve disagreements?
- **When thinking about assigning tasks**, you can divide the work for the project amongst team members as you please, with **2 important caveats**:
 - (1) the amount of work per team member must be *roughly* the same over the length of the project
 - **(2) each team member must participate in all software development activities (ie, design, coding, pitching, etc)**
 - **(IMPORTANT, different than what you may have experienced in internships)**
- **Design first! Remember diverge → converge framework. Iterate as you go.**

Advice for different parts of the framework:

 - **For diverge:** we recommend that you brainstorm design ideas first individually or in pairs, and then bring your ideas to the whole team to come up with new ideas and to filter your ideas down to the most promising three. Talk through the problem, purpose, and OP together.
 - **For converge:** A good concept design will be a solid foundation. Also, when you create a wireframe for your concepts, you can identify the interactive flow here and frontend components that you'll need to make.
 - Create as many reusable components as possible for consistent styling and so you don't have a lot of repeated code.
 - **For your alpha and betas:** Have a good idea of your resources (models, database tables/keys) and functionality (routes) before you start implementing. Making quick prototypes/proofs of concept with some hard-coding is good if you want to try something out
 - You can stub out unfinished functions. This is code that you're okay with throwing out if it doesn't work.

Team Dynamics

- **There are many deliverables and deadlines for the final project** throughout the next month. Here are some ideas/suggestions to help your team stay on track:
- **Have internal soft deadlines for smaller deliverables**
 - Ex: by midweek, have 1 concept implemented or the backend fully implemented for a concept (like the 5a and 6a assignment milestones)
 - Avoiding empty page syndrome
 - Soft deadlines/smaller deliverables particularly helpful for alpha + beta

- **Have one person be sort of a project manager**
 - Keeps everyone accountable for their deliverables for deadlines
- **Schedule recurring team meetings**
 - 1) Have synchronous meetings so everyone is on the same page
 - Check in on individual work, and plan for the next week
 - 2) Work session - helpful to work at the same time, even on Zoom / Discord call
 - Can ask questions / coordinate
 - Can help each other debug via screenshare/[VSCode liveshare](#)
- **Possibly use a system for project tracking and ETAs** (whether it's just a google doc, asana, jira, etc.)
 - Things often take longer than expected. What we think is easy to implement may take 2x longer than you originally planned.
 - Communicate clearly and early if things come up that prevent you from delivering a piece of code you signed up for.
- **Another idea is to have each person to be in charge of a concept (for implementation)**
 - Maybe decide concepts based on interests/what you want to learn
 - Since concepts may be synchronized with each other, you'll need to talk with one another/work together too
 - You should also be checking semi-regularly that the different pieces of code and features work together as expected.

Git workflow

- To make collaborating in one repo easier, some suggestions:
 - Working in feature branches (instead of on the main branch) - [Git Branches](#)
 - To merge your work into the main branch - [Squash-Rebase-Workflow](#)
 - if you want, before you rebase, you can create [pull requests](#) and [review](#) each other's code
 - Github will handle [merging a pull request](#)
 - (there are [many ways](#) to merge your work into the main branch like merge, squash-rebase, squash-merge etc. just agree on a protocol with your team)
- Reminder to commit often! (a lot of ppl have just been making one single commit when they hand in the assignment)

Deployment

- **Deployment Guide:** <https://61040-fa22.github.io/pages/deployment.html>
- **We recommend continuing to use Vercel** since you all have experience with it from the individual assignments.
 - We'll also only be able to guarantee staff support for this option.
 - The free plan only allows one user from each team to deploy.
 - To work around this, we recommend **creating a shared GitHub and Vercel account for your team**, so that all team members will be able to deploy. We recommend that you make your own Moira mailing list for your team (instructions are on deployment guide)
- Another option is Heroku, which is more flexible than Vercel.
 - If your team is trying to implement something that Vercel does not provide (e.g., **WebSockets**, which can help enable real time communication, or anything that requires more persistent connection), then deploying with Heroku can be useful.
 - We provided deployment guidance for Heroku on the deployment guide
 - Though websockets + real time communication may not be relevant for every project, it might be useful for a future webapp project you work on

Project Idea Tips

- **Do some background research into the problem space.**
 - What are the current apps in this problem space? What is missing (or apps that have imperfect solutions) and how could the user experience be improved?
 - Look back to the needfinding lecture—What are people's desire paths etc.?
 - Are there some datasets that exist or that you could create? For some apps, populating your app can lead to a richer experience for your users. (Some apps work well but don't have much data on them, not as interesting to explore.)
- **Keep in mind authentic apps vs. inauthentic apps**
 - <https://61040-fa22.github.io/assignments/assignment-p0>
 - An authentic app is one that is designed to respond to a genuine problem and that offers a novel and effective solution to it—or at least contributes ideas towards such a solution.
 - Is there a compelling need for your app?

- Would this be a project you'd be excited to show not only your friends but also people in your problem domain (ie. community members for the local community domain)? Would they be excited to really start using the app?
 - Does your project resemble one of the projects we encourage you not to do?
 - See inauthentic apps section
 - (Not really the goal of the final project to develop a meal swipe sharing app, or a loaning stuff on campus app, or a finding friends/pset partner app. Is there really an urgent/compelling problem or need for the app?)
- **Remember concept adoption, adjustment, invention from previous assignments**
 - It's fine to have concepts that already exist, but use them in an interesting way.


WebSockets Messaging App

- We'll go through a small messaging app that the TAs put together, which uses websockets (helpful for real time interactions). Particularly the socket.io library, which is compatible with Heroku but not Vercel.
- **WebSocket**
 - WebSocket is a protocol that supports bi-directional communication (either client or server can initiate communication).
 - It is useful for developing real-time application like chat system, or for interactions with different users.
 - In HTTP, server respond only when the client requested
 - For example, in your Fritter, when another user creates a freet, your feed doesn't get automatically updated. It is shown only after you have made a request to load the feed again.
 - In WebSocket, server and client can exchange data with much more freedom using the sockets. A socket is one endpoint for sending and receiving data across the network.
 - If you use WebSocket, you would be able to update the feed with freets that others have created in real time.
- **Socket.IO**
 - Socket.IO is a library that enables bidirectional communication
 - It is built on top of the WebSocket protocol

- **Exercise**

- We are going to walking through a very simple chat app that uses Socket.IO
- Connect on server side
- Connect on client side
- Sending a message
- Listening for a message
- Messaging app example: <https://github.com/61040-fa22/recitation-final>
 - npm run dev and npm run serve → localhost:8080
 - open 2 separate windows to test
 - chat and change your name

Resources

-  Programming Resources for 6.1040 Final Project
- [Deployment Guide](#)