1. Problem — The raw idea, a use case, or something we’ve seen that motivates us to work on this

Problem statement: We want to develop a LEGO-starter kit project set for undergraduate cs students to learn about the software engineering process. The project should be controlled and constrained, yet *interesting* enough such that undergraduate students would want to complete the project and learn about the software engineering process.

What a typical undergraduate student taking in CSE 110 might know (prior knowledge)

- Data Structures
- Algorithms
- Limited frontend, web application building experiences
- Possibly some database experiences (SQL)
- Surely know Java, C, C++

Current Problems with CSE 110

- Certain variations of the course is very unconstrained, is more of a 10 week hackathon rather than a software engineering course
- Students don’t understand the software engineering in a repeatable, reliable way after the course
- Transition into industry, there is a disconnect between what is good vs. bad software engineering
- Students might succeed or fail because of unpredictable class behavior

2. Appetite — How much time we want to spend and how that constrains the solution

Time Constraints

- 6 person team
- Willing to spend 5-10 hours per week
- Around 7 Weeks of development
- Total number of hours to develop → from 35 to 70 hours per person
- Total team hours → 210 to 420 hours → 1 week to 2 weeks of development time

Technology Required

Frontend	Server/REST	Backend	Databases	Hosting	CI/CD
HTML CSS JavaScript React.js Angular.js	Node.js Flask (Python) Spring Boot (Java)	Java Python C++ JavaScript Socket.IO	Mongo SQL Google Firebase / Firestore Heroku PostgreSQL	GCP AWS Azure Heroku GitHub Pages	Github Actions Travis CI Git Pull Requests

*Make your case for why use React?
- Demise because deceptive in what it's all about
- Not great for accessibility (e.g. screenreaders) and SEO
- Why not Angular?
    - Hero unit who knows react?
    - High barrier of entry for learning webdev?

Coding was easy, but testing was hard with React

What is the actual use case? We should mock up what we want to do and *then* justify why we should use a specific tool/language based on what is needed/suitable

Matrix of end users (110 students, actual users)
- A learned probably wants something that beneficial to know and also easy to pick up

React:

- Motive

    - Resume point
- Risk assessment
    - Learning curve
    - Tutorial: https://reactjs.org/tutorial/tutorial.html
- Testing
    - React testing & CI
    - https://testing-library.com/docs/react-testing-library/intro

Node:

- Motive
    - Same language as the frontend, reducing learning curve
- Risk assessment
    - Learning curve
    - Tutorial: https://nodejs.dev/learn
- Testing
    - https://www.robinwieruch.de/node-js-jest

JavaScript:

- Motive
    - Widely used
    - Plain and simple
    - Same language as the frontend, reducing learning curve
- Risk Assessment
    - Learning curve
- Testing
    - Unit / Integration tests as we go

Firestore Database + Firebase Hosting:

- Motive
    - Tutorial has a chat app
        - We are still using Socket.IO because Jason has experience with it, and because we don't want the students to be able to build the chat app just by looking at the tutorial
    - https://www.raywenderlich.com/5359-firebase-tutorial-real-time-chat
    - https://codelabs.developers.google.com/codelabs/firebase-web#0

Github Actions / Pull Requests

- https://github.com/features/actions
- https://docs.github.com/en/free-pro-team@latest/actions/guides/building-and-testing-nodejs

- https://dev.to/dyarleniber/setting-up-a-ci-cd-workflow-on-github-actions-for-a-react-app-with-github-pages-and-codecov-4hnp
- Github pull requests for code reviews

Questions:

- To what extent do we need to test our project?
- Testing UI -- makes sense
- Testing Server End points
- Testing Calls to 3rd party apis?
- Testing Database calls?
- Integration testing (maybe React Testing Library)

3. Solution — The core elements we came up with, presented in a form that's easy for people to immediately understand

A list of messages, a list of strangers, search bar to find people

Brainstorm features for Chat App

1. Login (Login with Google and/or Facebook)
2. User Profiles
    a. Own registration page after logging in with google
    b. Create a user name
3. Private messaging / 1:1 messaging
4. Search bar by user name
5. Read receipt
6. Timestamps on each of the messages
7. Simple Notifications (UI on person)
8. Most recent chats
9. Contacts/friends list
10. Block list
11. Group messaging
12. (Online or offline) statuses
13. Typing emoji (library to integrate)
14. Typing indicator
15. Advanced Notifications (toast on @ messages)
16. UI Themes (especially dark mode)

4. Rabbit holes — Details about the solution worth calling out to avoid problems

Skillset

- Websockets - non common apis?
- Webrtc - p2p video/voice call
- React
- Databases
- Controller
- Hosting

Note: if we use something that students might not have knowledge of, include resources to derisk

5. No-gos — Anything specifically excluded from the concept: functionality or use cases we intentionally aren't covering to fit the appetite or make the problem tractable

Features

- Stories
- Video chat
- Money transfer
- Add-on games
- Multi-language (auto translate messages depending on your language settings)
- Chat roulette feature -- randomly throw people into a room together (maybe enabled only within group chats)
- Chatbot (for weather updates)
- Attach files/photos/videos?
- Facebook messaging integration support (and other applications)
- Group administrator (mute people, kick people out)
- Bookmark notifications for faster access to previous chat locations -- see all places where they @ you
- Reactions
- Automatically remove/mask profanity
- Geolocation