Meta

Meta University Eng Project Plan Template

Fill in blanks (enclosed by brackets []) and remove red text as you work through writing your project plan. Your project plan should be a living document and can be changed as you progress through the internship. Make sure to work on this document together with your manager to get feedback, as well as ensuring your project meets the requirements and expectations in the Project Guide.

[Project Name]

Intern: Eric Lin
Intern Manager: Yao Li
Intern Director: [Name]
Peer(s): Xiaoying Li, Silvia Wang
GitHub Repository Link: [Link]

CHANGES

- Updated overview by providing an example of a website, and slight clarifications.
- Updated user stories, moving couldn't solve problems to stretch features.
- Add more stretch features (better question suggestion algorithm, username + public profile)
- Changed second technical challenge to authenticating via GitHub
- Write out the plan for the 6 weeks.

Overview

As a competitive programmer, I try to find new ways to improve my question solving skills. However, while there are websites that list questions (example) and their Elo

ratings (a number that determines a user's relative standing to every other user), there is no website that actively suggests questions or provides an all-in-one site for practice content such as books, content creators, or websites to practice on. My project aims to solve this problem.

- Category: Sports stats
- Story: The main goal is to provide the user with suggested problems at their estimated Elo rating, maybe with targeted question topics.
- Market: Competitive programmers aiming to improve
- Habit: For people actively trying to improve, probably daily.
- Scope: The scope is to have a one-in-all website for competitive programmers to improve their skills.

Product Spec

Based on the app description, this section goes into more detail about what the app should do, and what functionalities it must provide to the users.

User Stories

User stories are actions that the user should be able to perform in your app.

First, focus and identify functionality that is required for your MVP (Minimum Viable Product) that conforms to all the project requirements and expectations. Make sure your technical challenges are part of your MVP.

You should also identify optional / nice-to-have functionalities that would be done as stretch goals during MU Week 8 and 9. Remember, *technical challenges should not be optional features*, they must be code complete before the end of Week 8!

Required

AC - accepted submission

Login	• Users can log in with GitHub OAuth ◦ Login should be persistent • Users can link their account from the competitive programming site

Profile	• Top of the profile is suggested problems (contradicts the screen archetype as it's located at the bottom but I felt its more important for it to be at the topic) • Users can see their latest submissions • Users can see their submission stats (# of attempts and AC, in numbers and percentages) • Users can see the topic breakdown of the questions they solved ○ Users can see the percentage of questions with certain topics in their Elo range • Users can see their number of submissions and AC over a set amount of time (daily, weekly, monthly) • There should be a refresh button that fetches the latest info from the API endpoints

Suggested problems	• Website should be able to suggest a problem in their Elo range or slightly above.
	  ○ User can refresh the problem if they do not want to solve it
	  ○ Users can specify which topic the question should be picked from.
	  ○ The difficulty of the suggested questions should be adjusted after seeing if the user can't AC, or AC after X number of submissions
	• There should be a refresh button that fetches the latest info from the API endpoints

General	• Should list the upcoming contests • Should link to all of the other tabs (questions/resources/profile)

Resources	• Community board of suggested resources (textbook pdfs, content creators, websites, etc…), available for all users to see
	• If logged in
	  ○ User can create a post linking resource and include tags relevant to the resource
	  ○ User can upvote/downvote other resources
	  ○ User can search for post titles or filter by most recent, most upvotes, tags
	  ■ Posts will be sorted by most upvotes first

Questions	• Should list all of the questions, their Elo, and topic tags ○ The questions should be linked, allowing the user to click the title to go to the question • Users can filter by latest question, Elo range, question topic • Users can search for question titles

	• If logged in, questions will be marked green if completed.

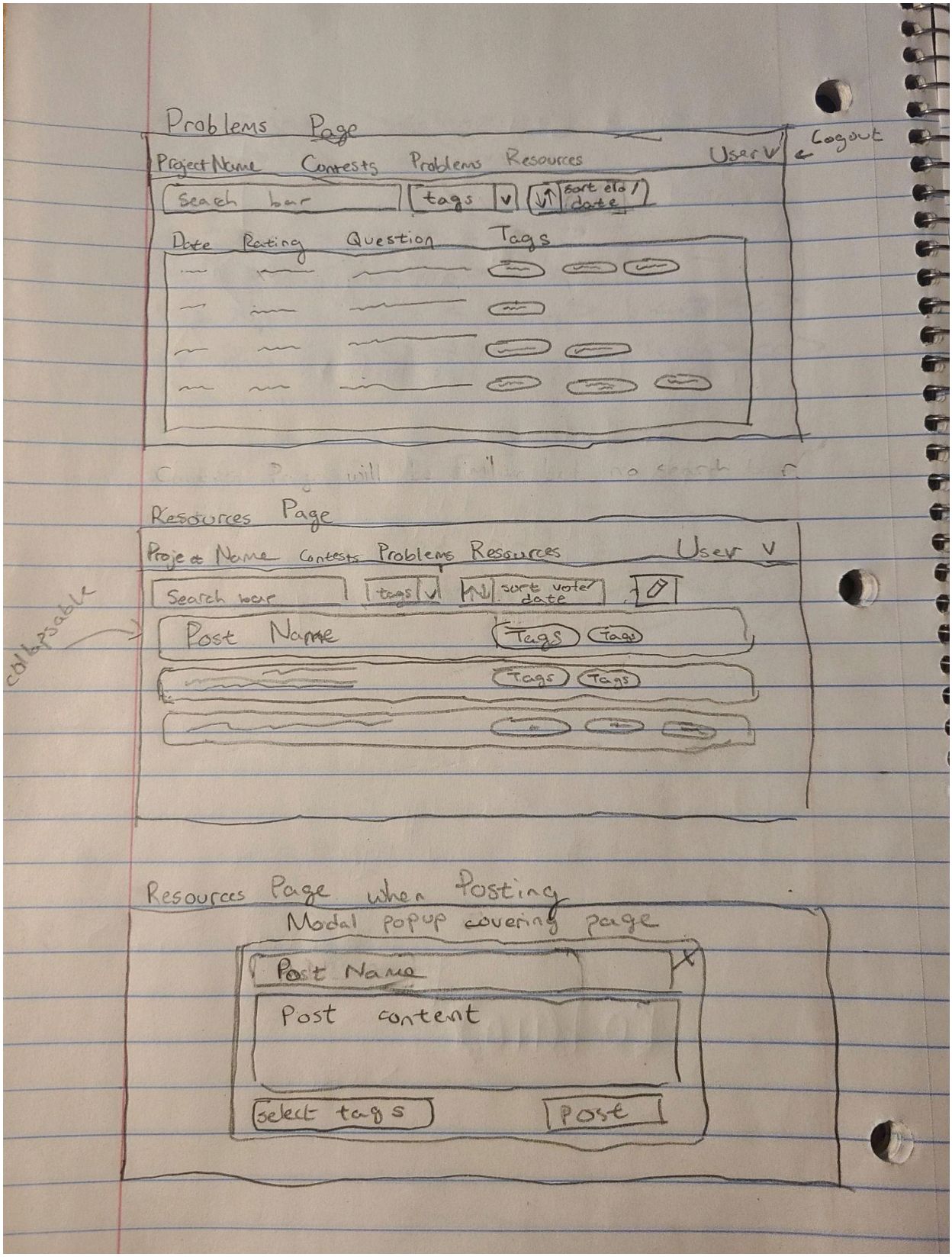
Stretch	• Support multiple competitive sites (required is codeforces)
	• Better polish the question suggestion algorithm
	• In suggested questions:
	  ○ Users can mark the problem as “Couldn’t solve”
	  ■ Couldn’t solve questions should be stored for future attempts
	  ■ Couldn’t solve should also store questions that user attempted in the past but failed
	• Users can create a custom username
	  ○ If not logged in, user can see the general stats of another user via their username (website will look something like …com/profile/USERNAME)
	• Visualize user profile stats with graphs

Screen Archetypes

[Describe the different screens that, together, compose the full experience of your app. You can leverage anything you want, such as diagrams and mocks.]

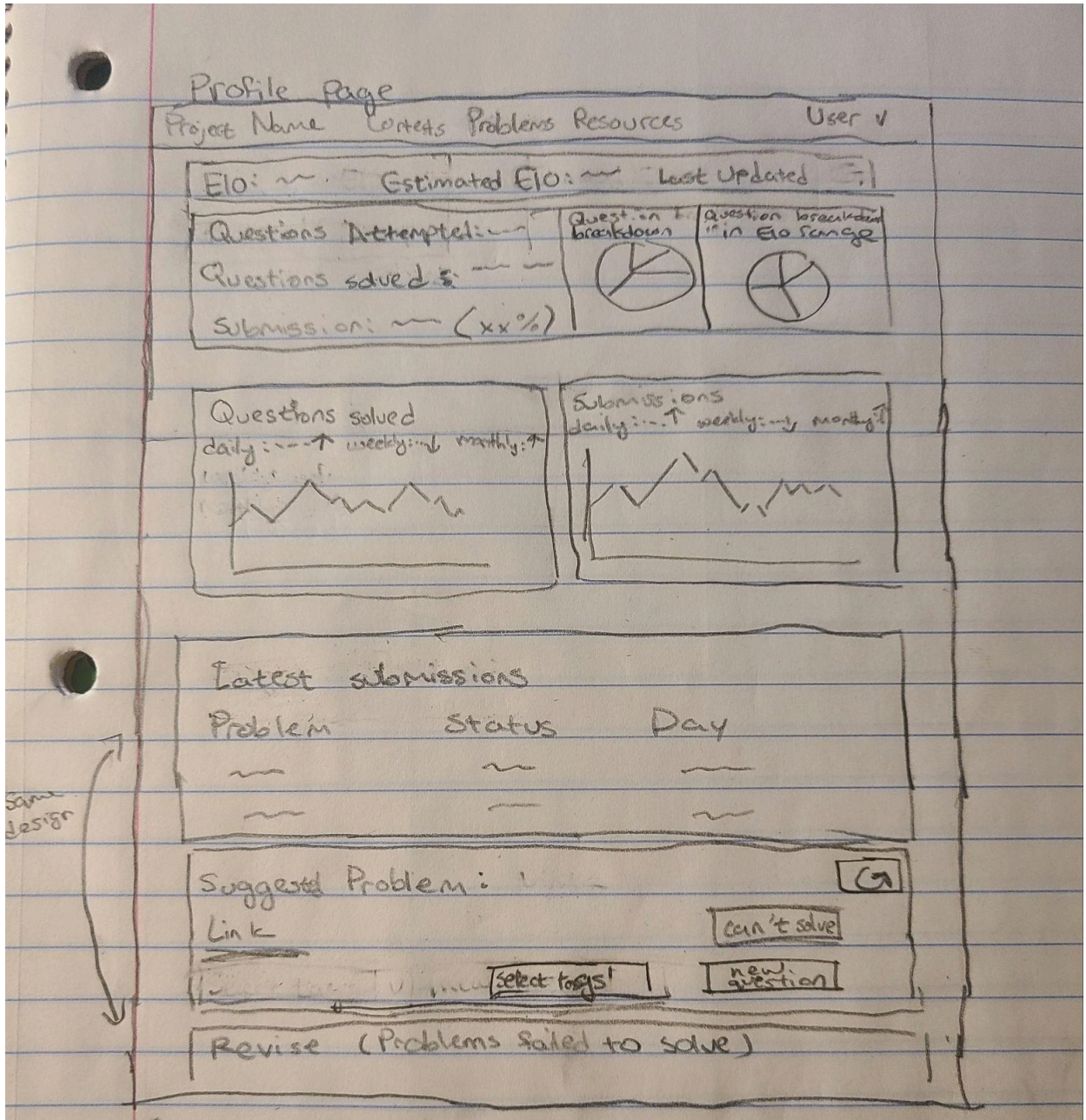
[Using diagrams you can also describe how navigation and presentation of these screens will work on a high-level.]

[These are just high-level representations though. Don’t spend too much time building mocks.]



Problems page heavily inspired by clist.by

Profile Page



Cards should all be collapsable and collapse be persistent

Data Model

[Describe the data you're going to need to back your application. This can include database models (like tables), or external data you'll require from some API.]

User

id	int	Autogenerated by Prisma
Name (Stretch)	String	Username. When created, default username can be user_{id}
cfHandle	string	Can be null if user didn't register their cf account
cfEstimatedElo	int	Estimated ELO. If linked at first, use ELO if they have, else populate with 1200
cfSolved	[[Question, Int]]	An array storing questions solved and how many attempts it took
cfAttempted	[[Question, Int]]	An array storing questions attempted and how many attempts
cfSolvedMonth	Int[31]	Number of questions solved during the last month
cfSubmissionsMonth	Int[31]	Number of questions attempted during the last month
cfIncomplete (Stretch)	[Question]	Array of questions user failed to solve
postLiked	[Post]	Array of posts that the user has liked. When user creates a post, they automatically like it, similar to Reddit

cfCurrentQuestion	[Question, timeAssigned]	The current suggested question and time it was assigned. This is to ignore submissions made in the past

Question

id	int	Autogenerated by Prisma
contestId	int	ID of contest
index	string	Question index in contest
ratings	int	ELO of question
tags	String[]	String array of question topics

Submission

userId	User	A relation to user (handled by prisma)
id	int	Autogenerated by Prisma
question	Question	Stores the Question related to this submission
timeSubmitted	string	UNIX time
verdict	string	One of “AC”, “WA”, “TL”, “ML”, “RE”, “CE” (WA: wrong answer, AC: accepted, TL: Time Limit Exceed, ML: Memory Limit Exceeded, RE: Runtime error, CE: Compilation error)

Post

id	int	Autogenerated by Prisma

authorId	int	Id of author user
title	string	Post title
content	string	Post body
creationTime	string	UNIX time
tags	string []	Tags associated with post
likes	int	Number of likes

Server Endpoints

[Describe the endpoints that your application is going to consume from your server. If you're using REST, then you'll probably want to include the method (GET/POST/etc) and the expected parameters (query parameters, body parameters, etc.)]

I would also need to figure out OAuth but that Codepath is going over that next week.

Request	Name	Description
GET	user/:id	Should return the contents of the current user id
GET	questions	Should return all of the questions. Should support query parameters for filter & sort
GET	posts	Should return all of the posts. Should support query parameters for query and sort
POST	posts	Create a new post, passing in all but the likes field on the Post table
POST	user/:id	(Handles can't solve)

		Update the cfIncomplete fields
POST	postLike/:postId	Updates the like/dislike in a post and the user who liked/disliked the post
POST	linkCF/:userId	Should handle the linking of the CF account after authentication

Navigation

Project Requirements

[Based on the Project Guide, describe how your project is going to be fulfilling each of the base project requirements.]

Technical Challenges

For your project, you should demonstrate that you can apply what you've learned so far and expand on that knowledge to write code and implement features that go beyond the scope of the projects you worked on during CodePath.

Based on the general idea and direction of your project requirements, your intern manager will create at least two (2) Technical Challenges for you. This section is all about explaining what they are and how you're planning to tackle them - you'll work together with your manager to fill it out.

Technical Challenge #1 - [Name/Small Description]

What

What problem are you solving, and what parts go beyond what you learned in CodePath?

The challenge will be to sync up data from external API endpoints to my database every set amount of time if the user doesn't request a data refresh first. This ensures that the website should have the most up to date information. Also, Codeforces has a one API call per 2 second rate limit so I need to handle that.

How

Explain in words how you'll solve this problem.

After doing a brief research, it looks like I will have to look into schedulers (BullMQ) and integrate them into my server. These schedulers can do external API calls perhaps once a day? to the external endpoints to update the information on user/new questions. BullMQ uses redis so there might also be a challenge of connecting the two together.

Technical Challenge #2

What

Everyone dislikes creating new accounts and trying to memorize the different passwords that they need. Because competitive programmers are first and foremost programmers, and therefore are highly likely to have a GitHub account, I want to authenticate a user via GitHub instead.

How

I will look into GitHub OAuth and look into some tutorials to get a login via GitHub working.

Database Integration

[Describe what you are using for database storage. For example, Parse, MongoDB, Sequelize, etc.]

PostgreSQL with Prisma.

External APIs

[Describe at least one external API you're using for your project. For example, Google Maps, Spoonacular, OpenWeather, etc.]

Codeforces	• https://codeforces.com/api/user.status?handle=[USER]
	  ○ Returns QuestionID, contestID, names, rating, tags, and verdict (Passed/failed/Compilation Error)
	• https://codeforces.com/api/problemset.problems?tags=implementation
	  ○ Returns a problem list (older problems do not have as many stats as newer problems (particularly no website title so will have to handle that))
	• https://codeforces.com/api/contest.list (all contests)

Authentication

[Describe how user authentication is handled for your project, including logging in and signing up. Also describe any kind of cookie / session management you're doing and how you're implementing it, and how this affects navigation between different screens by the same user.]

GitHub OAuth

Visuals and Interactions

[Provide details on how your app is fulfilling the following UI craft requirements, and how these are technically accomplished.]

- Interesting Cursor Interaction
- UI Component with Custom Visual Styling
- Loading State

Timeline

Project execution will start in Week 4 of MU. Based on the previously defined requirements, user stories and technical challenges, use the following table to scope

out and plan a timeline for deliverables over Week 4 - 9. You can be as detailed as you need, ranging from simply mentioning the user stories, or dividing them into sub-tasks.

You are free to modify the table, add / remove rows or columns, whatever fits your style! The important thing here is that you focus and prioritize certain aspects of your project so you don't get behind and are ready to deliver the MVP - remember your required features should be code complete before the end of Week 8, including both technical challenges!

We also encourage you to leverage project tracking tools such as GitHub Issues or Meta's internal Tasks / GSD tooling to keep manage individual units of work.

MU Week	Project Week	Focus	Tasks
4	1	Focus on the components that will serve as the skeleton of your project. You will probably be using most of what you learned in CodePath to set up things like the client and server repositories, initial routing, login / registration, creating a database with object models, etc.	• Set up project structure • Create the Prisma schema and database • Set up API endpoints fetching both user and questions stats from CF • Implement GitHub OAuth and persistent login
5	2	Week 5 and 6 should be where you focus on the specific requirements of your project.	• Create basic profile page UI • Process the questions from Codeforces (counting the topics in ELO range) and storing them in DB, updating data if too old • Create linking method to link user account to website • Create submission stats (attempts, AC, percentages)
6	3	By this point, you should be getting started with your technical challenges as well.	• Implement the question suggestion algorithm (doesn't have to be a good algorithm) ○ Should adjust the estimated ELO whether a user can solve it and their attempts • Add problem refresh • Implement topic targeted suggest problems • Get started with setting up BullMQ task schedulers to constantly update information of users and questions if data is too old (updating every day)

7	4	You should focus on finishing your MVP and core requirements. By this point, you should be done with at least one of your technical challenges.	• If task schedulers are not finished, finish that.
			• Create a questions list with Elo, topic tags, and link them to questions
			  ○ Create search, sorts and filtering for the questions
			  ○ If logged in, implement the solved functionality (question box turns green) for their solve questions

			• Get started with the resources board (search, sorts, and filtering will be similar so I just need to implement the posting of resources)
8	5	Continue work on finishing touches and stretch goals for your MVP. By this point, your core functionality and both TAPs should all be in place. It is also a good point to start working on stretch goals that could further expand on the functionality (and technical complexity) of your project. This week you also have to submit your self-review, make sure you allocate enough time for this alongside your final submission for your project!	• Finish the resources board if not finished (probably need to implement like/dislike feature if not complete) • Work and implement the general page, listing the upcoming and previous contests. STRETCH FEATURES • Visualize user stats with graphs via a graph library • Add a refresh button into the profile, fetching the latest data ○ This will need to work with task schedulers so as to not spam Codeforces API's rate limit
9	6	It's time to show others what you have built! Work on a presentation and demo that you will present to other interns to showcase your work. You are also free to continue polishing and expanding on your project!	• Implement "couldn't solve", storing the questions the user failed to solve in the past • Implement the creation of a username ○ This can provide the resource board with an author name • Implement public profile pages, allowing for foreign users to see a user's public stats (total solved, total submissions, graphs)
10	7	For this week, we have a bunch of extra activities prepared to give you a quick dive of what it is to work at Meta. You will find activities around using internal tools and frameworks, and even committing code to our internal repositories.	