Problem 2 - Problem Solving 🤔

> Imagine that you are given an assignment to build an app that brings together 3 different platforms that LEE members use within one mobile-friendly interface. You are brand new at LEE, so don't know much about the 3 platforms beyond the platform and API documentation that the product managers have provided.

How would you go about building an app with limited information?

The primary goal of beginning this project would be to determine: what constitutes a MVP (minimum viable product)? The MVP will reflect the project goals, expectations, and user stories generated through conversation and consistent communication with the LEE staff/product managers. This will lead to an application that is testable in relation to the member market's use and experience.

What steps would you take to first start and then complete this assignment?

Starting a project

- Sprint Planning: User Stories

  MVP is all about understanding the minimal requirements to meet the use cases provided. I am accustomed to using a Trello Board for sprint planning and user story mapping but am familiar with ticketing systems as well. I find it is extremely helpful to have a tool like Trello to create an interactive environment for the developer and product managers to map out the key features and backlog related to standing up the MVP. This extends not only to key features but also to secondary or stretch features established with a ranking system. In terms of this projects prompt, this stage would be essential to starting an app with 'limited information'.

- Wireframe -> mock ups

  - Determine navigation / UX Flow

  In a mobile friendly interface (`react-native` deployed for development with Expo) there are established standards and guidelines to creating a user's navigation through the application. This, in many ways, simplifies the UX flow design. This step in the process would determine which features are visible throughout the application and when/where a user is able to access them. Are all three platforms visible and interactive?

  - Determine overall cohesive styling for the application

  Do we want to implement an existing UI kit or framework for consistent styling? What is the minimum amount of styling to communicate the key features?

  - Mapping out the user's journey

    - Who is a user?

    - What are a user's actions or tasks?

        - What is a successful journey? (Sign up for a workshop, connecting with other members, accessing a particular resource...)

- ERD - entity relationship diagram
    - What data needs to persist or be capable of CRUD functionality within the application?

  In this project we have members signing up and interacting with the mobile application which indicates a `users` table and login/signup/authentication/profile flow. Aside from this routing and storage interface there is the issue of the three platforms. How will users interact with the platforms and how will the platforms interact with each other? For example, a member may be able to join a resource sharing team that accesses information from one of the platforms. This will require a joins table as well as the generation of primary and foreign keys for members and teams. Communicating these interactions is essential to forming a solid ERD and provides the perfect template for designing the routes, handlers, models, and controllers for the backend (and may play an integral role in deciding which database to use and which backend language or framework to use (I really like Django for an application that requires a lot of 'admin' data input versus Go which is more useful for handling large amounts of requests very quickly and efficiently))

  The following is an ERD for a Food Truck application displaying several one-to-many relationships: Food Truck ERD

- If product managers aren't available...

  Diving into the documentation and setting up quick individual applications to test the functionality of the three platforms would be key to understanding the nature of the MVP application and how the platforms may interact. Much of my processes assume there is a lot of communication but if the instructions are "take these platforms and build a react-native app" I would determine user stories and backlog features by diving head first into the three platforms individually before trying to integrate them into one application.

Completing a project

- MVP integration of the three platforms involves testing (TDD)

  Communicating with the product managers about the platforms' behavior is greatly reinforced by reliable TDD. Test driven development as a process to best integrate the three API's would ensure the components function correctly. Having access to the documentation would give me enough information to integrate the platforms using a 'red, green' TDD process. I would use a familiar testing framework to at first fail and eventually pass the minimum amount of functionality and information through the component. TDD provides a great measuring stick for the project in that, as components/functions pass their tests they are a signal to the developer that the project is meeting the requirements (and if they start to fail as the project develops they provide an incentive to ask questions about the necessity of new features or functionality).

- Build, Measure, Learn

  Quickly building an MVP with an ingrained testing framework allows for instant feedback from product managers and the developer to improve the application toward a production level product for market. The project is completed when it satisfies all user stories and conforms to the tested success criteria. In this case, once all three platforms are integrated and working to the product managers' expectations, it may be time to test with member/users. Many criteria for an application are long-term in scope ("Have 3,000 members sign up for workshops through the mobile platform with 10,000 active monthly users total") which means the refactor process according to lean methodology requires asking if each feature is meeting basic requirements outlined in the initial planning phase.

A project is 'complete' once the team agrees that the MVP has been met. This is especially true when the project prompt is to build a mobile-friendly application integrating three platforms with minimal information provided from the product managers. Adhering to a LEAN/Agile methodology to quickly build a MVP with a lot of TDD is the best way to tackle this problem.