Moving From Your Project Proposal to a Requirements Document

Congratulations! Your idea has been sketched out and approved. The next step is to create a Requirements Document that codifies all of the details of your project. How much detail? The idea is that you could hand this document to another developer (or team) and they would be able to build using only this document *and* this document would be used by the client to assess whether you built what you said you were going to build or not.

Yikes! That sounds like a lot! It is, but here is a strategy to get you started - and to keep you from becoming overwhelmed.

User Stories

1. Start with your MVP (Must Have) features or user scenarios and create user stories from them. As you work, you will probably discover hierarchical levels of user access. It is helpful to give these users capitalized distinctions and use them consistently. Instead of, "An admin can create a room that a user can join for meetings", by specifying, "A Teacher can create a Room that a Parent can join for meetings" provides more context for how this app will be used. Note that "Room" is also capitalized - but "meeting" is not. These subtleties communicate whether something is a major entity in your app or simply a descriptive.
2. With the Must Have user stories jotted down, move to your Should Have user stories. When those are completed, do the Nice-to-haves.
3. From your User Stories, make a list of what you will need from the other components: **data sources**, **site architecture**, **data flow diagrams**, and **interface designs**.
4. Once you have completed your lists, you can start to flesh out the actual information for each of those components that is needed in the Requirements Document.
5. This (like everything in web) will be an iterative process, so don't be too hard on yourself if you have to make changes, you are going to have to make (many) changes. The important thing to keep in mind is that, if you change an aspect of one component, what other components also need to be updated?

Example

Here is an excerpt from a proposal for a web development To Do list:

Feature

"A user can add a task to their task list, select a category for the task, and set a due date for it."

User Stories from this feature

- A Developer can see all of their assigned Tasks.
- A Developer can create a new Task for themselves.
- A Developer can select a category for a Task.
- A Developer can set a due date for a Task.
- A Developer can mark a Task as completed.
- A Developer can edit or delete their own Tasks.

To data sources: From the user stories, we can see that full CRUD functionality is required for the Developer user. Furthermore, "A Developer can create a new Task for themselves." From this user story, we need a table of Developers because they can only create a Task for themselves. We also need a table for Tasks. If there were an API of developers or tasks, we could specify that. In this case, we will create our own data source. From our other user stories, we should have tables for tasks and categories. A Task (so far) needs fields for the assigned Developer, its category, its due date, and whether it has been completed or not.

Data Sources: DB Tables

- Developers:
- Tasks: Developer | Category | Date | Completed
- Categories:

From the user stories, we can infer what we need for data flow diagrams and wireframes…

Site Architecture & Data Flow

- Developer login
- Developer sees Tasks
- Developer creates a new Task
- Developer edits or deletes existing Tasks
- A Developer marks a Task as completed

Interface Design (wireframes)

- Home page
- Login screen
- See all Tasks (list view)
- Create/Edit Tasks (details view)

Again, just start with bullet lists, then you can tackle the individual components and get down to the details. This will help you understand your app from a high level and will make your later decisions much easier.

It is very important to connect all of the dots through the document. The data flow should not indicate a user step that does not have a matching element in the interface, the interface should not have an input for something not stored in the database; the database should not have a field for data that is not referenced in a user story.

In the above To Do List scenario, later on while creating user stories for the Lead Developer (admin) role, I may choose to say that a task is not completed until it is reviewed by the Lead Developer. This means, I need to add a Role field to the Developers table; a Status field to the Tasks table; a review step in the data flow diagram, and an "approve" button to the Lead Developer Task interface - and probably an approved icon to the Developer interface. What else do I need to do? Keep iterating and updating.