# Pace Group C - Requirements and Design Report

## High-Level Project Description with Correct Identification of User Groups

### General Description

The software we are building is to create a web-based Mentorship platform that would allow users to register as a mentor or a mentee. In addition, users from the organization using our product can register and sign in as administrators. After registering, a mentor or mentee are required to fill a mentorship form for identification. Then, the platform shows viewable calendar for availability of mentors from which mentees can request for a meeting, and, on their version of the site, mentors can view and manage mentees' mentorship requests.

### How the three user groups use the system

- Mentor: System requires mentors to create their own accounts and fill out the sign-up form in order to approve mentee's matching requests via emails. In addition, the system requires mentors to provide email address and availability of their schedule for mentees to book an appointment.
- Mentee: System does not require mentees to have their own account. Instead, they are required to fill out the sign-up form in order to view availability of mentors they are going to match with. Also, they need to provide contact info while filling the form.
- Admins: The role of an Admin is to decide how to host the form, edit the form, and approve Mentors. The Admin could have the ability to remove Mentors, and possibly pass on Admin privileges to another user if that Admin decides to leave the platform.

## System Architecture in DFD

### DFD Level 0:

A description of this DFD can be found at the bottom of the linked image.
DFD Level 0

### DFD Level 1:

- This DFD shows the system architecture within more detail.
- It is encouraged to download the image to view it with more ease.
- This DFD has the same three external entities as the previous one, but the mentor and mentee are duplicated in a few different places in order to avoid data flows crossing one another. Furthermore, it goes deeper into showing how accounts are actually created, and matches actually made. In addition, this DFD also shows the system's data stores, as opposed to its level 0 counterpart.
- Any feature included in a red rectangle is to be completed for the Peer Review #2 milestone.
- Any feature included in a dotted rectangle (or shown in a dotted fashion itself) is to be completed for the Final Project milestone.
  DFD Level 1

## Breakdown of the Target Components for Each Milestone

### Target Components: According to the course website, the major target milestones within the course are as follows:

- Technical Requirements - October 21st, 2020
- Peer Testing #1 – December 2nd, 2020
- Peer Testing #2 - March 3rd, 2021
- Final Report, Code Repo, Video Demo - April (during Final Exam Period)

We hope to accomplish the tasks outlined by each major milestone a week in advance of the milestone due date. Below are the functional requirements for each milestone.

## Functional Requirements

### 1) Peer Testing #1

FR1. Create a static form that can be embedded into a website via iFrame (with Name, email, time-zone, availability, ranked skills, and user type [Mentee vs. Mentor]).

FR2. The form will be split into subsections, using a “next” button to navigate to the next section, and a "back" button to view the previous one.

FR3. Both mentors and mentees will use a ranking system to indicate either their skills or areas seeking mentorship. This, as well as time-zone selection and availability will be used to weigh matching mentees to mentors.

FR4. The form should restrict the age of Mentors (anyone 17 and under should not be able to become a mentor).

FR5. Once the user submits the publicly hosted form, a popup will appear. The popup will state the domain address to access the Mentorship Portal, but it will redirect the user to the hosting organization’s website.

FR6. The “Mentorship Portal” will host 3 functionalities: Login, Forgot Password, and Register. A member of an organization will click on Register to select their user-type (either Mentor or Organization) to enter information about themselves and create an account within the system.

FR7. To register Organizations and Mentors successfully, a unique email must be entered. If that email has already been used in the database, an error message will be displayed.

FR8. To correlate the company to the user, the domain name of the email will be used (i.e john.doe@UBC.ca signifies that John Doe’s organization is UBC).

FR9. To complete organization registration, the user will create an account using the same email address, which will be their username. They will also need to provide a password (in the format of at least 6-10 characters, with at least one number). That user will then be allocated admin privileges.

FR10. The Admin will publicly host the form (by embedding the form within their website via iFrame) for Mentee registration.

FR11. Account registration for Mentors requires a unique email address, and a password. To register an account successfully, the email must not be associated to a pre-existing account but must match with an email used in the mentorship form (which will be accessible through email invite). If the entered email has not been associated to a form, an error message directing the user to fill out a form will be displayed.

FR12. An email will be sent to the address provided in account registration. The user must click that link to be able to login to their account.

FR13. Mentors and Admins will be able to login from the Mentorship Portal. The login will be unsuccessful if either the email or password credentials do not align with what’s in the database (in which case, an error message will be displayed).

FR14. Mentors and Admins will be able to reset their password by clicking on “Forgot your password?” on the Mentorship Portal. The user will be required to enter an email address which has been associated with an account in the database. If successful, an email will be sent to that address, which the user can click to be brought to a separate page to reset their password.

FR15. Mentors will be required to fill in the form through email invite.

FR16. Mentors and Admins will be able to logout of their account, to which they will return to the index of the mentorship portal, where they can login, retrieve their password, or register.

FR17. Logged in Mentors on the Mentorship Portal will be able to accept Mentees that they have matched with (based off of similar skills, availability, and timezone).

FR18. Once a Mentor has accepted a Mentee, an email will be sent to the Mentee to notify them of a successful match. The two can then correspond through email.

### 2) Peer Testing #2

FR19. The Admin will decide how to host the Mentorship form (either publicly, or internally through email invite). The source code for publicly embedded forms will be generated by clicking a "Generate" button so that Admins can add the iFrame tag to their organization's public website.

FR20. The form will be dynamic in the way that Admins can edit the fields of the form to suit their needs.

FR21. The form will be dynamic in the way that depending on the user filling out the form on public sites (i.e either mentor or mentee), once the radio button is selected for the type of user, the form will display appropriate information for each.

FR22. The email that notifies Mentees that they have been matched to a Mentor will link to a third-party calendar (such as Google Calendar) with the Mentor's availability. Mentees will be able to create an appointment with Mentors through that calendar.

FR23. To "unmatch" Mentees and Mentors, the Mentee can remove themselves from the calendar, cancelling all future appointments. The Mentor could then remove them for their dashboard on the Mentorship Portal.

FR24. Mentors who validate their email address will be able to login, however, they will not be able to view any matched Mentees until an Admin approves their account.

FR25. Admins will be able to approve Mentors through the Mentorship Portal.

### 3) Final Report

FR26. Ensure all components of the system are functional, fixing any final bugs.

FR27. Implement “bonus features” (a.k.a features that are not required of the system, but that would be nice to have):
- Allow Mentees to create an account with the system, and develop surrounding UI, with additional features.
- Allow Mentors the ability to continue receiving Mentee requests, or put them on hold.
- Allow a user to be both a Mentor and Mentee
- Allow Admins to be associated with more than one organization
- Allow the system to support more than one language.
- Allow Admins to pass their admin privileges to another user if that current admin no longer wishes to sustain an Admin role.

## User Stories

Mentee:

- I want to be able to be matched with a Mentor who is willing to share their accomplishments and experiences within their organization to enable myself to gain knowledge in various lifelong skills.
- As a mentee, I want to be able to share information about myself, including my preferences and schedule so that I can be matched with a Mentor who is available based on my timeframe and skill interests.
- Once I’ve been matched with a Mentor, I want to be able to communicate through email so that we can stay in touch and book appointments around our schedules. I want to be able to have the flexibility of gaining knowledge without being constrained to schedules that don’t work for me.

Mentors:

- I want to be able to share my knowledge and skills with individuals looking for mentorship, in the areas I have experience and am comfortable in sharing.
- I want to be able to create an account within my organization so that I can view and accept possible mentees to mentor, as well as current mentees that I have accepted to mentor.
- I want the ability to retrieve and/or change my account's password.
- I want the flexibility of Mentoring around my availability. I want to be able to send the information of my availability through a calendar's interface to my matched Mentees and have them book an appointment accordingly.

Admins:

- I want my organization to be part of a program/system that allows individuals with experience and knowledge in our organization to provide Mentorship to other individuals seeking guidance.

- I want to have the ability to create an account so that I can monitor accounts for Mentors and Mentees within my organization to ensure guidelines are being adhered to for security.
- I want the flexibility of verifying, approving or rejecting Mentors within my organization based on our requirements.
- I want the capability and/or functionality to host forms either publicly or internally through my organization's email invite system. This would enable easier communication.
- I want to tailor the generic form to my organization's specifications in order to streamline the information process towards our needs.

References:

- https://www.visual-paradigm.com/guide/agile-software-development/what-is-user-story/
- https://www.theserverside.com/featureThe-7-user-story-guidelines-every-Agile-developer-should-know?_ga=2.191533999.44778962.1602895908-2114816780.1602895908

---

## Non-Functional Requirements and Environmental Constraints

### Efficiency

System requires the least time/steps it will take users to perform their task such as registration and matching mentorship because the software we are building is a mentorship tool that makes mentors' and mentees' lives easier.

### Safety

Errors that users make is undoable and should be simple to operate. (e.g. undo button)

### Utility

System provides sufficient functionality for users to accommodate range of users tasks. For example, availability of mentor on calendar, registration, sign up form, personal account modification module, etc.(core functions)

### Testability

System should be able to be tested by non-staff users to collect feedback.

### Extensibility

System should be easily editable if a new feature is going to be added because again this should be a helpful tool and keep updating system to be more useful by users feedback if possible.

### Stability

System should not meet any conflicts when users are tasking.

### Data reliability

System collects users' personal information when accessing the software. Backend data management should be safe and stable to store personal information. (e.g. users who have registered should not be required to register again because of loss of data.)

---

## Tech Stack Chosen and Rationale

### Options, pros and cons

| Tech type | Tech name | Pros | Cons |
|---|---|---|---|
| FE | React + UI libraries such as: Ant design, bootstrap, material UI, bulma, tailwind CSS | - Client first choice - Widely supported and documented - Facebook-backed | - A bit harder to learn than Vue |
| FE | Vue | - JP has experience - Easy learning curve | - Not as much support and documentation - No major company backing |
| FE | Angular | - Most mature of all three frameworks | - Steep learning curve |
| BE | Node with Express.JS | - Client first choice - Uses JS, like the front-end - JP has experience - V8 engine (very fast) | - Callback syntax can become complex - Slower than Python if trying to do heavy computation tasks |

| Tech type | Tech name | Pros | Cons |
|---|---|---|---|
| BE | Python with one of Django, Flask, Nameko | - Python syntax (subjective advantage) | - Doesn't natively support asynchronous development - Slower than node |
| DB | Firebase | - Client first choice - JP has experience - Easy to learn and use - Large community | - Expensive when scaling |
| DB | AWS | - Large suite of tools (authentication, data processing) - More capable & customizable than Firebase | - Steeper learning curve - Harder to set up |
| DB | MongoDB | - More predictable pricing | - Harder to set up than AWS |

## Final choice

### The final tech choice will be the following:

- Front-end: React + UI libraries
- Back end: NodeJS
- Database and authentication: Firebase

### Rationale:

Our client told us that the tech stack we use is, eventually, up to us. However, the client did seem to have a preference for the tech stack above. This is a big reason why we opted for this final choice. In addition, the table above shows some of the significant advantages that our chosen tech stack presents.

In addition to React, we will be using various front-end libraries, such as: Ant design, bootstrap, material UI, bulma, or tailwind CSS. Again, this is a suggestion from our client. Using such front-end libraries has a few drawbacks. First, it will add a bit of complexity to our setup, and dependencies on external codebases, which means we are vulnerable to the future potential lack of maintenance or breaking changes of libraries. However, using such libraries will help us craft a professional and good-looking front-end, with code that is less error-prone than ours, since it is "tried-and-tested" by many developers. In addition, we believe that using well-supported libraries with large communities of developers relying on them (such as Bootstrap or Ant design) will mitigate the issue of being dependent on other developers to maintain the libraries.

---

## Test Strategy

### Front-end

On the front-end, we want to use React, with a few libraries as well.

To test our React code, we will use Jest and React-testing-library. As per this link, that is the standard way of implementing testing in a React application.

Jest is specifically aimed at unit testing JavaScript and, although it is not the only way to do so, it is the tool recommended by React, and it was developed by Facebook. Jest has other advantages. As per this article, for instance: "One of the advantages of Jest is speed. Jest runs tests in parallel which makes running the whole test suite so much faster. Also, you've got the possibility to use "--watch" and only run the tests affected by your changes in the editor. The coverage feature is impressive too. It gives you a nice and simple rendering directly in your console after you run the coverage command (...)". Jest also has snapshot testing, which, according to this article, "make[s] testing basic UI components extremely simple with one line of code: expect(component).toMatchSnapshot();."

React-testing-library is a good way to test React. A significant advantage of this library is that it tests actual HTML elements within the DOM, and is thus very reliable, and tests the system similarly to how the user would "test" it (because the user sees HTML elements, not React technicals, such as whether or not a component is in a specific state).

### Back-end

On the back-end, we are using Node.JS.

On the back-end, we will mostly be doing unit testing. To do so, we will use Jest. There is a large number of frameworks which allow unit testing with JavaScript, but we chose Jest because it has the advantages described above, and we believe that using the same testing framework on the back-end and front-end reduces the amount that developers have to learn, and improves collaboration between the front-end and back-end "experts" of our team.

### Database and authentication

The solution we will use for databases and authentication is Firebase.

Firebase offers an emulator suite which allows for unit testing, as well as testing the database's security rules. We have little control over how we unit test Firebase, but we feel confident that the suite of testing tools provided by Firebase will be appropriate. In addition, Firebase allows us to use Jest to test our Firebase rules and setup as well, which means we can standardize our testing tools across front end, back-end, and database and authentication.

### Continuous integration and regression testing

We will use GitHub Actions to do continuous integration and regression testing. We will use this feature to do several things, some of which include:

- Automatic linting
- Running all unit tests (regression testing)

We might add more features (smoke testing, for instance) to this continuous integration workflow later, as needed.

For now, we have setup a GitHub Action, and we have a basic script that runs on every pull request to our develop and master branches. We have added an ESLint action in this script, but we will need to wait until we have actual code in our repository before we can add more useful actions and commands to this script.

---

## List of Questions and Answers

## Account Creation Questions:

### Why would the mentor complete the form before being verified? What is the benefit of storing information in a database that you may not use?

Crucial to the matching process is for users to fill in their information through the form so that an appropriate mentor/mentee match can be found. Not only this, but by providing user information, the administrators from each organization can verify that the user is legitimate, which is why we're gaining information from the user before verification. However, this could potentially result in information being stored in the database for an idle account, as the question addresses. To ensure only active users are considered, a timestamp could potentially be utilized. If a user doesn't verify their email within a set timeframe (say, for a week), and the user doesn't log onto the portal for an extended time (possibly a month), then their information could be removed from the database to reduce unnecessary storage. If the user was previously active, an email could be sent out before deletion to see if the user still wishes to remain in the system.

### Will the user be able to go back pages when signing up? I feel like you didn't express whether they can make changes to their account if they messed up or wished to change something?

Since the form will be split up into different sections, a "Back" button will be available so that users can alternate between the various sections to modify their information.

For the current scope of the project, we're hoping that what users enter after filling out the form will remain accurate for a period of time. However, we recognize that users make errors, and their skills/availability may change. If time permits, we could find a user-friendly way for users to be able to modify their information after the form has been submitted. A potential solution would be to include a link in the verification email of the filled in form to the user to ensure the information is accurate, and if not, they can make changes through that link.

### Does a mentor/mentee match require admin approval?

This is not something that our team has considered. Currently the way that the system is setup is that admins approve mentors, and once mentors have been approved, they have the ability to accept mentee requests. As the platform grows, admin approval of mentee matching might be added on.

## Matching Process/Prototype Questions:

### Can you explain how mentors and mentees will be matched? Will you use a specific algorithm or something more generic (match based on which attributes overlap the most)?

The process of how mentors and mentees will be matched will be decided by the skills and skill levels that both the mentor and mentee possess. The mentor will then be able to view all of the mentees that have matched based on their skills and their level in those skills. The ultimate decision will be up to the mentor to accept the mentee.

The system will also match the mentor and mentee based off of time zones, and availability of both users.

### Is the system limited to a specific kind of mentoring, and will the mentors have to pay to be on this service?

The system is not limited to a specific kind of mentoring. The mentee will have the ability to write down and select any skill(s) that they have. If there is a mentor who has the same type of skill(s) as the mentee, the mentee and mentor will be matched.

The mentors/mentees do not need to pay for this service.

### What software did you use to make the wireframing? How will you convert your design sketches into an actual design for the website?

The software we used to create the wireframe is called Balsamiq.

We will convert the wireframe to the actual design by using HTML and CSS to model and improve our initial wireframe.

## Security Questions:

### Does safety in non-functional requirements mean security? Did the client ask for security to be non-functional?

Yes, safety means system security. From our meetings with the client, security is not specifically required, but it was mentioned for some security purposes such as Account registration of administrators.

### How will security be approached when it comes to registration? (for example, to protect against SQL injection attacks, data encryption, etc.)

To protect collected data from users, we will make security measures for data within our reach.

### Do you guys have any system in place to deal with abuse, such as mentees constantly denying mentorship etc?

We have not consider this part yet. However, if we enough time, we can discuss about it but not for sure because this is a human factor and should not be included in system requirement.

## Testing/Tech Stack Questions:

### Do you have a plan on which browsers you would like to support and how that will be incorporated into your testing process? Will the project be limited to desktops, or will it be usable on mobile?

As per this source, the worldwide market share for browsers is as follows: Chrome 66.3%, Safari 16.76%, Firefox 4.08%, and other browsers. Therefore, those 3 browsers will be our main focus, and we will manually test the products on these browsers to ensure it works well across them.

Our primary focus will be on desktop development, with full mobile compatibility being more of a nice-to-have. However, we are hoping that the design libraries we use will help us with mobile compatibility, since many of them include solutions to this issue out of the box.

### When the Mentor and Mentee both import information into the database at the same time, how will the system handle multiple queries and results at the same time?

This problem will be solved by using the dedicated Firebase solution, which is called Transactions. From the linked document: "Cloud Firestore supports atomic operations for reading and writing data".

### Will there be a manual confirmation for admin and mentor signups? If not, what is the verification process going to look like other than the age verification?

Regarding mentors, it is the admins who will have to validate their registration. For admins, we will have a fourth type of user, which we call the "superuser", who will be responsible for validating admin registration. Indeed, this will be a manual verification process, where there will be a UI for each type of user, with requests coming in, and options to accept or reject them.