**University of British Columbia, Vancouver**
Department of Computer Science

# CPSC 304 Project Cover Page

Milestone #: 3

Date: October 30th, 2023

Group Number: 67

| Name | Student Number | CS Alias (Userid) | Preferred E-mail Address |
|---|---|---|---|
| Justin Prasad | 78028420 | c8s1o | justinm.prasad@gmail.com |
| Andrew Joji | 28440428 | x0m6d | andrewjoji71@gmail.com |
| Pedro de Sant’Anna Novais | 41950486 | g9u5j | psantnovais@gmail.com |

## ***Brief Description:***

Our project lies primarily in living quarter management (apartments, condos, etc). We are aiming to build a database that can mimic what a database would look like and entail in real life by tracking tenants, mail, staff, equipment available to be rented, etc.

Repo https://github.students.cs.ubc.ca/CPSC304-2023W-T1/project_c8s1o_g9u5j_x0m6d

## Tasks:

- Have milestones 1 and 2 ready to upload
- Structure Repo with Folders for Project Documents and Code
- Create README.md with project summary and usage instructions
- Make branches for each member
- Select adequate tech stack
- Implement Scope of Application (User Stories for features):
    - **All users** should be able to login as the proper user (login page)
    - An **admin** should be able to:
        - Assign staffs to a building
        - Assign rooms to a building
        - Assign tenants to rooms
        - Update data from any entity
    - A **tenant** should be able to:
        - see a table of items available to rent from the Building, and select and rent an available item (Rental Page)
        - return a rented item by selecting a dropdown option in the (Rental Page)
        - submit a Maintenance Request and see past requests they made (Requests Page, tenant side)
        - View the mail available for them
    - A **staff** should be able to:
        - see a list of requests by date and their status , and mark them as completed if they haven’t been already (Requests Page, staff side)
        - See a list of all vehicles, select a vehicle to use, and return (Vehicle page)
        - See a list of available equipment and borrow one from it (Rental Page)
        - Return a borrowed equipment (Rental Page)
        - Assign a mail to a tenant, and to a building (Mail Page)

## Timeline:

| Number of Task | Task | Team Member Assigned to Task | Prerequisite Task(s) |
|---|---|---|---|
| 1 | Have Milestones 1 and 2 ready to upload<br><br>Deadline: Oct 30th | Justin, Pedro, Andrew | None |
| 2 | Structure Repo with Folders for Project Documents and Code<br><br>Deadline: Oct 30th | Justin | None |
| 3 | Add Milestones in Markdown files<br><br>Deadline: Oct 30th | Justin, Pedro | 0.5 |
| 4 | Create README.md with project summary and usage instructions<br><br>Deadline: Oct 30th | Andrew | None |
| 5 | Make branches for each member | Justin, Pedro, Andrew | None |

| | | | |
|---|---|---|---|
| | Deadline: Nov 3rd | | |
| 6 | Select adequate tech stack<br><br>Deadline: Nov 3rd | Justin, Pedro, Andrew | None |
| 7 | Create Tables for Relations and Insert Default Data in Oracle DB<br><br>Deadline: Nov 10th | Andrew | 6 |
| 8 | Implement login/authentication functionality (login page)<br><br>Deadline: Nov 11th | Justin | 6 |
| 9 | Admin: Assign staffs to a building<br><br>Deadline: Nov 23 | Andrew | 6 |
| 10 | Admin: Assign rooms to a building<br><br>Deadline: Nov 23 | Andrew | 6, 7, 8, 9 |

| | | | |
|---|---|---|---|
| 11 | Admin: Assign tenants to rooms<br><br>Deadline: Nov 24 | Pedro | 6, 7, 8, 9, 10 |
| 12 | Admin: Update data from any entity<br><br>Deadline: Nov 24 | Pedro | 6, 7, 8, 9, 10, 11 |
| 13 | Tenant: Rental and Requests Pages functionality<br><br>Deadline: Nov 30 | Justin | 6 |
| 14 | Tenant: Mail page functionality. Tenants should see their mail<br><br>Deadline: Nov 30 | Justin | 6, 13 |
| 15 | Staff: Requests and Vehicle Pages functionality<br><br>Deadline: Nov 30 | Andrew | 6, 13, 14 |

| | | | |
|---|---|---|---|
| 16 | Staff: Rental and Mail Pages functionality<br><br>Deadline: Nov 30 | Pedro | 6, 13, 14, 15 |

## *Challenges:*

### *High Level:*

Technical Debt:

- **Challenge**: Rushing to meet deadlines may lead to suboptimal code practices, resulting in technical debt that can hinder future development.
- **Mitigation**: We will allocate time for refactoring and addressing technical debt regularly.

Dependency on External Services or Libraries:

- **Challenge**: The project may rely on external services or libraries that could become deprecated, unavailable, or have breaking changes.
- **Mitigation**: We will keep all dependencies up to date and use services with reliable support and documentation.

Change in Project Scope:

- **Challenge**: The project scope might change mid-development if we realise it is too much to achieve in the given time frame, leading to rework and delays.
- **Mitigation**: We defined the project scope before starting (with our current understanding and experience) and will discuss any scope adjustments to focus development on key features if needed.

Scalability:

- **Challenge**: Scalability wasn't something we accounted for in our initial design phase, so it could be a problem if we populate the database to a point that our implementation cannot handle.
- **Mitigation**: Current things to think about would include indexing, and query optimization to ensure no bottlenecks occur and faster data retrieval times. Additionally, since we anticipate data tables to get somewhat big, partitioning these tables may also increase performance by dividing tables into smaller pieces. This is something to be considered of course in conjunction with project requirements.

Security:

- **Challenge**: Our conceptual idea involves a database where different logins would be used to obtain different administrative privileges. Examples would be if a tenant logs in, they can't see things related to staff work or where certain staff members are located. Ideally, we want to display correct windows based on security level
- **Mitigation**: At the moment we plan to use Java/Oracle as our stack and OOP languages have good security mitigations by allowing the programmer to create data objects that only authenticated users can look at. Some more research needs to be done, but it seems Java is able to handle what we need at the moment.

## *Low Level:*

Error Handling:

- **Challenge**: As with any project, the possibility of errors coming up is inevitable. Our challenge will be how to handle these errors in a stack we haven't used recently (or even at all).
- **Mitigation**: At this point all we can do at the moment is discuss, and research ways to optimize this. Ideally we would implement a robust error-handling system (probably just a bunch of Java try-catches) to account for these.

Testing and Debugging:

- **Challenge**: As with the previous challenge, this is an inevitability. The challenge here is how to find ways to debug efficiently and make test cases (if needed).
- **Mitigation**: As implementation has not started yet, we have yet to really put anything into practice. A testing suite would be nice but it ultimately takes time to make and it's likely we don't have that so that will be reviewed when we actually implement the DBMS.