Connectd CRM

Development Team | Project Overview | Documentation Shortcuts | External Links | Recently updated

👥 Development Team

Name	Role	Contact
@ David Fletcher	Product Owner	dfletcher@student.unimelb.edu.au
@ Alexander Cain	Scrum Master	acain1@student.unimelb.edu.au
@ Han Liu	Frontend Lead	liuh8@student.unimelb.edu.au
@ Jackson Hu	Backend Lead	renweih@student.unimelb.edu.au
@ Tymara Metcalf	UI / UX Designer	tmetcalf@student.unimelb.edu.au

💻 Project Overview

Customer Relationship Management systems (CRM) are usually used to keep track of relationships with clients. ***Connectd***, as a personal CRM will serve the purpose of keeping track of personal relationships and maintaining a good relationship network. ***Connectd*** allows you to store important information about your friends, colleagues and clients while also keeping notes about them to help you remember their preferences or avoid forgetting important dates. ***Connectd*** can be utilised as a handy to-do list to manage your tasks as well, with a clear view of their progress.

🔗 Documentation Shortcuts

Name	Link
Client Requirements	Requirements
Meetings	Meetings
Design & Architecture	Design & Architecture
Sprints Documentation	Sprints

🛠️ External Links

Name	Link
Frontend GitHub Repository	https://github.com/Andrew-Liu-mel/COMP30022-FrontEnd
Backend GitHub Repository	https://github.com/Andrew-Liu-mel/COMP30022
Heroku Deployment	Frontend - https://connectd-front.herokuapp.com Backend - https://connectdcrm.herokuapp.com
Trello Workspace	https://trello.com/personalcrmteam79

Figma Interactive Prototype	https://www.figma.com/proto/3kh2FcVcRMxlPcicvkyS4M/PCRM?node-id=21%3A23&scaling=min-zoom&page-id=0%3A1&starting-point-node-id=21%3A23

Recently updated

You'll see the 5 most recently updated pages that you and your team create.

REST API Routes
Nov 12, 2021 • contributed by Jackson Hu

Sprint 3 Trello
Nov 12, 2021 • contributed by Jackson Hu

Sprint 3 Planning
Nov 08, 2021 • contributed by Jackson Hu

Client Handover
Nov 05, 2021 • contributed by Jackson Hu

5) 2021-11-05 Client - Handover
Nov 05, 2021 • contributed by Tymara Metcalf

Client Handover

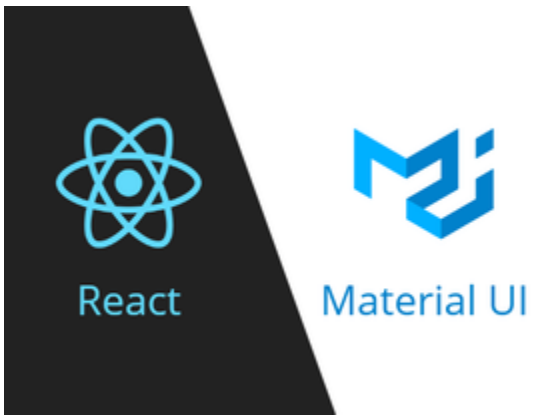
- Deployment Documentation
- User Manual

Deployment Documentation

The product separates its frontend and backend with a GitHub repository for each. This documentation will introduce how to configure locally for testing and deploy to cloud hosting platform (Heroku) for production.

- Frontend repository is located at: **Frontend Repository**
- Backend repository is located at: **Backend Repository**
- Frontend hosted on: **Heroku - connectd-front**
- Backend hosted on: **Heroku - connectdcrm**

Frontend



The frontend application is a **single page web application** written by `ReactJS` & `ma terial UI` framework. It aims to help the user manage their connections & tasks.

1. Local Configuration

Pre-requirement:

1. `Node` version of at least 10
2. A package manager, `npm` or `yarn`
3. A code editor (optional)
4. Modern browser

Steps:

1. Pull the backend repository from GitHub

    - You may use command `git clone <repo>` if you haven't cloned the repository before
    - Or simply run `git pull` to fetch the latest commit
2. Make sure you've already installed `yarn`
3. Open a terminal window at the folder root directory
    - Check current path using `pwd` command
    - You should see a directory path like `.../COMP30022-Frontend`
4. Run command `yarn install` to install all dependencies required for frontend
5. Run command `yarn start` to run yarn frontend server,
    - By default, server will run on port `3000`
6. The application will start running, and a web page will be shown in your default browser.
7. Open the project in code editor, make any changes you want. (Optional)

2. Deploy to Cloud

Pre-requirement:

1. A Heroku account
2. Install the Heroku CLI
3. React build pack
4. Git

Steps:

1. Download the project source files from Github repository provided above.
2. Log into your Heroku account & create a new application.
3. Navigate to Settings of the new app, add buildpack link to `Buildpacks section` which locates in the middle of the page.
4. Open terminal, then navigate to project folder. Run the follow commands:
    - `heroku login` (login your Heroku account).
    - `git init` (make the project folder a git repository).
    - `heroku git:remote -a {new_app_name}` (link local repository to your Heroku app).
    - `git add .` (stage project files).
    - `git commit -am "new application deployment"` (commit project files).
    - `git push heroku master` (push your project to Heroku, and build the application).
5. If the building process succeed, then you have successfully deployed the application to cloud.
6. You can find the URL of your application in setting section of your new app.
    - Normally in the form of: `https://{new_app_name}.herokuapp.com`

Backend

The backend server of our CRM is built based on `Django` and `Django REST frameworks` using `Python` language. The backend provides list of API endpoints to allow data exchange with frontend application, including **retrieve**, **create**, **update** and **delete** user's **connections**, **tasks**, **gr oups**.

1. Local Configuration

Pre-requirement:

1. Python 3
2. `pip` as the package manager
3. Terminal
4. Git
5. A code editor (optional)

Steps:

1. Pull the backend repository from GitHub
    - You may use command `git clone <repo>` if you haven't cloned the repository before
    - Or simply run `git pull` to fetch the latest commit
2. Make sure you've already installed `Python` and `pipenv`
3. Open a terminal window at the folder root directory
    - Check current path using `pwd` command
    - You should see a directory path like `.../COMP30022`
4. Run command `pipenv sync` to install all dependencies required for backend
5. Activate virtual environment, either by:

- Type in `pipenv shell` in terminal
- Open in VS Code and open a new terminal prompt (will automatically switch to virtual environment)

6. Run command `Python manage.py runserver` to run Django backend server,
   - By default, Django server will run on port `8000`
   - Can change the server's port by `python manage.py runserver <port number>`
   - Correctly running server will look like:

```
Performing system checks...

System check identified no issues (0 silenced).
November 02, 2021 - 22:31:45
Django version 3.2.8, using settings 'backend.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Can skip steps 1-4 after setting up the environment for Django

2. Deploy to Cloud

Pre-requirement:

1. A Heroku account
2. Install the Heroku CLI
3. Python build pack
4. A `Procfile` required by Heroku
5. Git

Steps:

1. Download the project source files from Github repository provided above.
2. Ensure the `Procfile` is located in the root of the directory.
3. Run command `pip list` to check `gunicorn` and `django_heroku` are already installed.
   - Be sure to add `django-heroku` to your `requirements.txt` file as well.
4. Log into your Heroku account & create a new application.
5. Navigate to Settings of the new app, add buildpack link to `Buildpacks section` which locates in the middle of the page.
6. Open terminal, then navigate to project folder. Run the follow commands:
   - `heroku login` (login your Heroku account).
   - `git init` (make the project folder a git repository).
   - `heroku git:remote -a {new_app_name}` (link local repository to your Heroku app).
   - `git add .` (stage project files).
   - `git commit -am "new application deployment"` (commit project files).
   - `git push heroku master` (push your project to Heroku, and build the application).
7. If the building process succeed, then you have successfully deployed the application to cloud.
8. You can find the URL of your application in setting section of your new app.
   - Normally in the form of: `https://{new_app_name}.herokuapp.com`

3. API Endpoints

Swagger has been implemented as the API documentation tool which is also hosted on Heroku. Swagger is an UI friendly documentation tool for APIs and allows to visualise and interact with the API's resources directly. It lists all the API endpoints showed below and can be accessed through the main page of backend server **here**.

Route	Endpoint	HTTP Verb	Description
Authentication	auth/register/	POST	Register a new user with email and name

	auth/login/	POST	Validate and log in the user
	auth/logout/	POST	Log out user with verification
Connections	api/connections/	POST	Create a new connection
	api/connections/	GET	Get a list of all connections
	api/connections/:id	GET	Retrieve a specific connection
	api/connections/?userId=	GET	Get a user's all connections
	api/connections/:id	PATCH	Update a specific connection's information
	api/connections/:id	DELETE	Remove a specific connection
	api/connections/?userId=	DELETE	Remove all connections belong to this user
Task	api/tasks/	POST	Create a new task
	api/tasks/	GET	Get a list of all tasks
	api/tasks/?userId=	GET	Get a user's all tasks
	api/tasks/:id	GET	Retrieve a specific task
	api/tasks/:connectionId=	GET	Gets list of all connection’s tasks.
	api/tasks/:id	PATCH	Update a specific task's information
	api/tasks/:id	DELETE	Remove a specific task
	api/tasks/?userId=	DELETE	Remove all tasks belong to this user
Groups	api/groups/	POST	Create a new group
	api/groups/	GET	Get a list of all groups
	api/groups/:id	GET	Gets specific group. (includes list of all attached tasks/connections)
	api/groups/?userId=	GET	Get’s all groups created by a user.
	api/groups/:id	PATCH	Update a specific groups info, could include information about the group or objects attached to group (add/remove tasks/connections). (includes JSON as body)
	api/groups/:id	DELETE	Remove a specific group
	api/groups/?userId=	DELETE	Remove all groups belong to this user

4. Authentication System

Despite Django has its built-in module for authentication using forms, we construct the authentication system for our CRM using `Knox` `authentication` frameworks for its simplicity and higher compatibility. All the requests related to account are handled by `Knox` `authentication` including **user registration**, **login** and **logout**.

`Knox` is a token based authentication system, which means operations are validated with a `token`. A `token`will be provided when the user successfully logged in. All the requests through the given API endpoints need to include this `token` as the value in HTTP `Authorization` head er.

- `token` value is followed by a `Token` prefix such as `Token 60221cf9735769d9b1066b6d32...`

While the backend returns a `token`, it also comes with an `expiry` timestamp. Requests are not only validated with a `token` value, but it will also bee verified whether the `token` has expired according to the `expiry`timestamp stored on the server side.

The following requests will be identified as **invalid** and therefore **rejected**:

- A request with missing `Authorization` header.
- A request with invalid / expired `token` value.

For more information about the authentication system, please refer to our *Authentication API Specification.*

Database Configuration

We chose **MongoDB** as our primary database to store the data records of user profiles, connections, tasks created by the user of our ***Connectd*** CRM. This second will guide you to create a MongoDB hosted online with MongoDB Atlas and link your database to Django backend server.

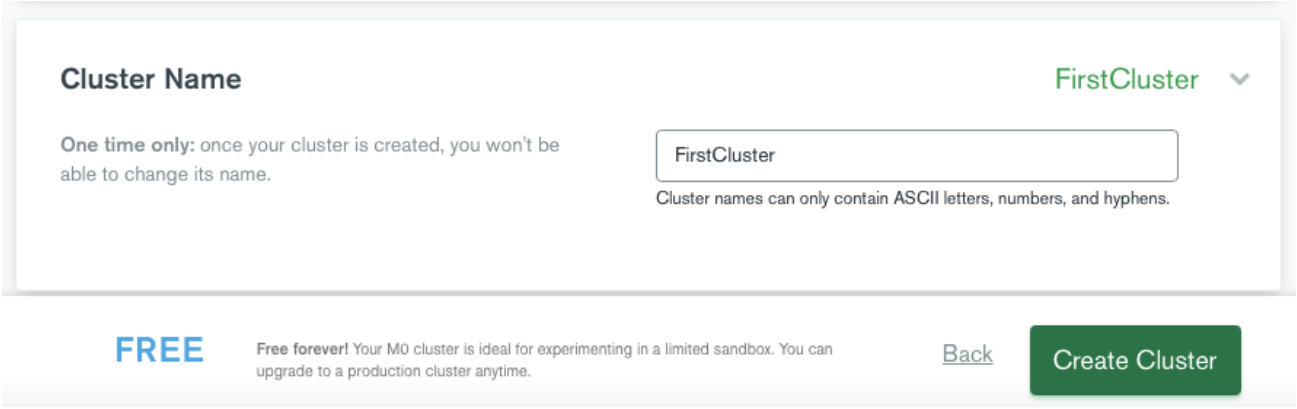
What you need:

1. A MongoDB Atlas account
2. Backend files pulled from repository

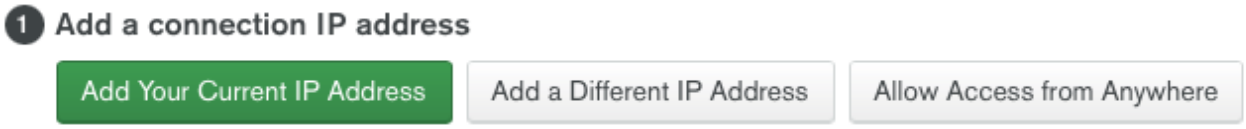
1. Create a MongoDB database

Steps:

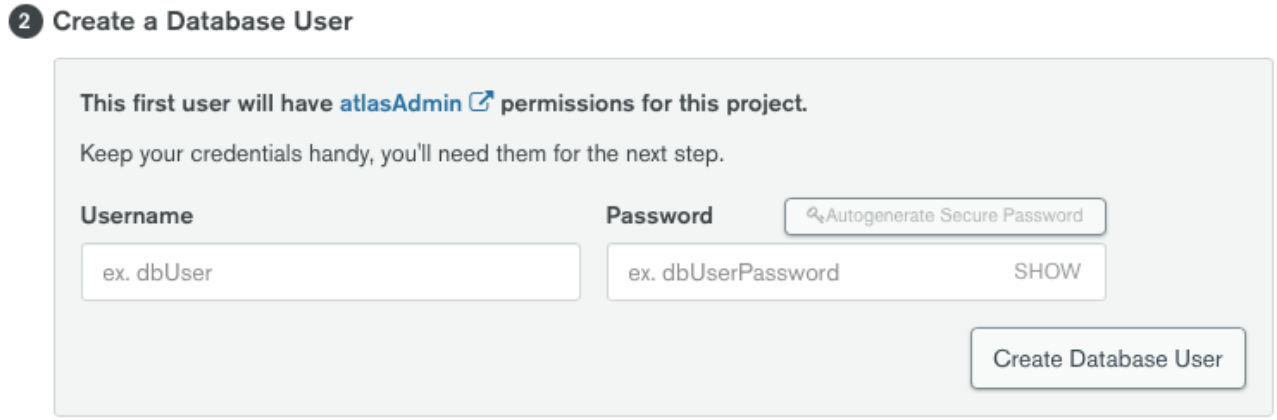
1. Once you logged in your account, you can choose which type of database to deploy.
    - You can start with a shared database for free.
2. Create a cluster with a custom cluster name to get started.



3. Now you can click `Connect` to set up your connection.
4. You will be asked to add trusted IP addresses into connection white list.



    - You can add your current IP directly without checking your IP address.
    - Even though it's possible to allow connection from any IP addresses, it's not encouraged for security concerns.
5. Also create a database account with `Username` and `Password`.

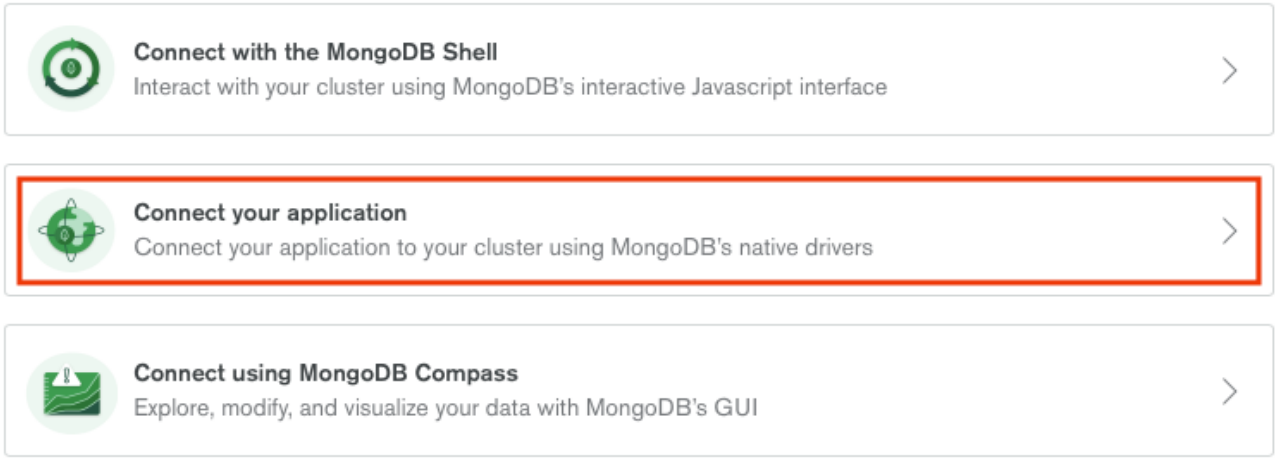


    - This is the account which will be used later in Django to validate database connection.

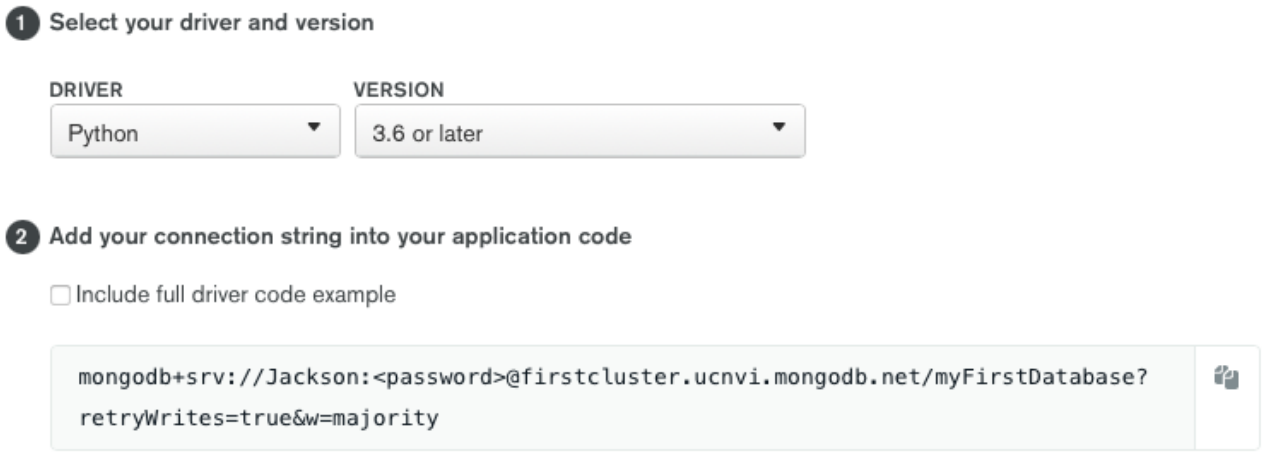
2. Configure database in Django

Steps:

1. As you can see there are three ways to connect to the MongoDB database you just created.



2. Since MongoDB is a NoSQL database so we choose to use `djongo` as the SQL query transpiler which can help us translate SQL queries to MongoDB queries, like an adapter.
    - `djongo` has already be listed in `Pipfile` and `requirements.txt` so you don't need to worry about installing it if you followed the instructions to successfully run Django server.
3. Thus we choose the second option here to proceed.
4. Choose `Python` as the driver language with version `> 3.6`.



5. You should see the provided connection string here as well.
6. Now open the `settings.py` file under `/backend` in backend directory, this is the place to configure the database used by the backend server.
7. Find a chunk of code started with DATABASES like this:

```
DATABASES = {
    'default': {
        'ENGINE': 'djongo',
        'NAME': 'Connectd',
        'ENFORCE_SCHEMA': False,
        'CLIENT': {
            'host': '',
            'authMechanism': 'SCRAM-SHA-1'
        }
    }
}
```

8. Copy the connection string given before as the value of `host` like:

```
'CLIENT': {
            'host': 'mongodb+srv://<username>:<password>@<cluster>.
ucnvi.mongodb.net/<database>?retryWrites=true&w=majority',
            'authMechanism': 'SCRAM-SHA-1'
        }
```

9. Replace the following values in connection string:
    - `<username>` - the database username.
    - `<password>` - the password for this database user.
    - `<cluster>` - the cluster name set earlier.
    - `<database>` - the name of the database that connections will use by default.
10. Open terminal at backend root directory and run command `python manage.py makemigrations` followed by `python manage.py migrate` to initialise the database with the data models pre-defined.
11. Now the database has been successfully configured and linked with Django backend ready to use, congratulations!

User Manual

Welcome to the Connectd CRM User Manual

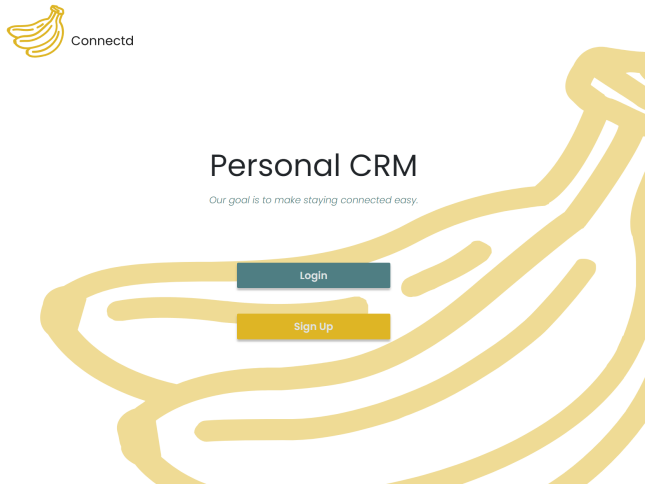
How to connect to your own database

Please see: https://comp30022-079.atlassian.net/wiki/spaces/CRM/pages/39682070/Deployment+Documentation#Database-Configuration

---

Setting up your user profile

If you are new to using the ***Connectd*** CRM, we recommend you create your profile by registering an account.



1. Click on the **Sign Up** button, the second button in the middle of the screen to register a new account.
2. Complete the information to proceed.

User information	Instructions
First Name	Write your first name in the textbox.

Last Name	Write your last name in the textbox.
Email	Write your complete email in the textbox. This will serve as your username.
Password	Write a new password in the textbox.
Confirm Password	Write the same new password in the textbox.
Accept terms and conditions	Check box to accept company terms and conditions.

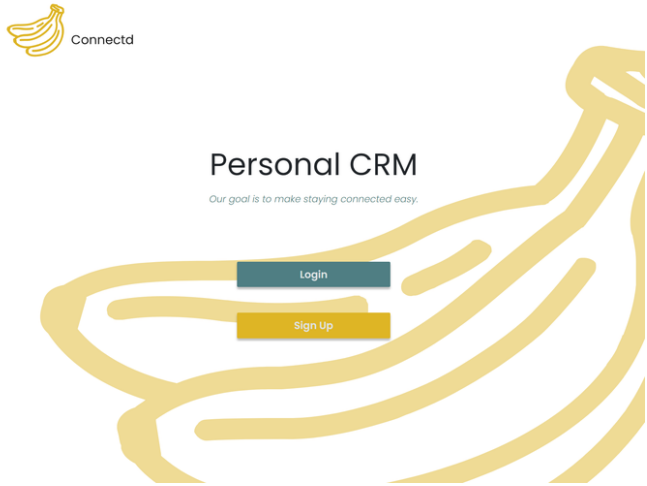
3. Click on the **Register** button.

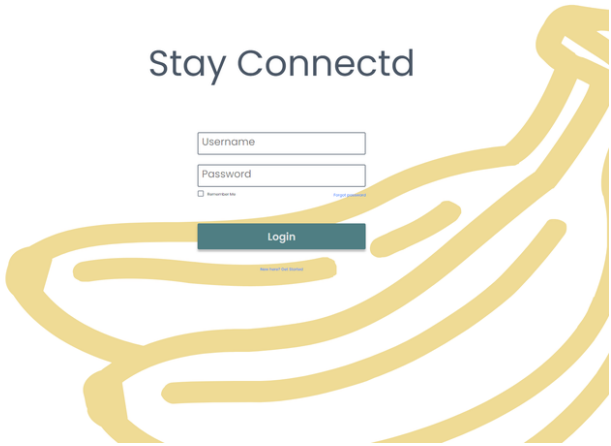
4. You will be redirected to the Login Page.

Congratulations on being a new ***Connectd*** user! You're now ready to discover all the features and options that ***Connectd*** has to offer!

Log in to your account

If you are an existing user, navigate to the Login page.





To access your account:

1. Click on the **Login** button, the first button in the middle of the screen to log in to your existing account.
2. Enter your credentials to log in.

User information	Instructions
Email	Write your complete email in the textbox.
Password	Write your password in the textbox.
Remember me	Check box to remember your information for next time. (optional)

3. Click on the **Login** button. You will be redirected to your Dashboard.

Navigation

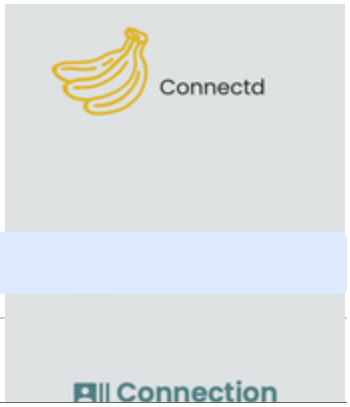
Connectd is divided into 3 main collections, all displayed in the **sidebar:**

Dashboard, Connections, and Tasks. Each category contains the available information for that content type.

Within these 3 collections, you can create, manage, and publish content.

The **Navigation Bar** consists of:

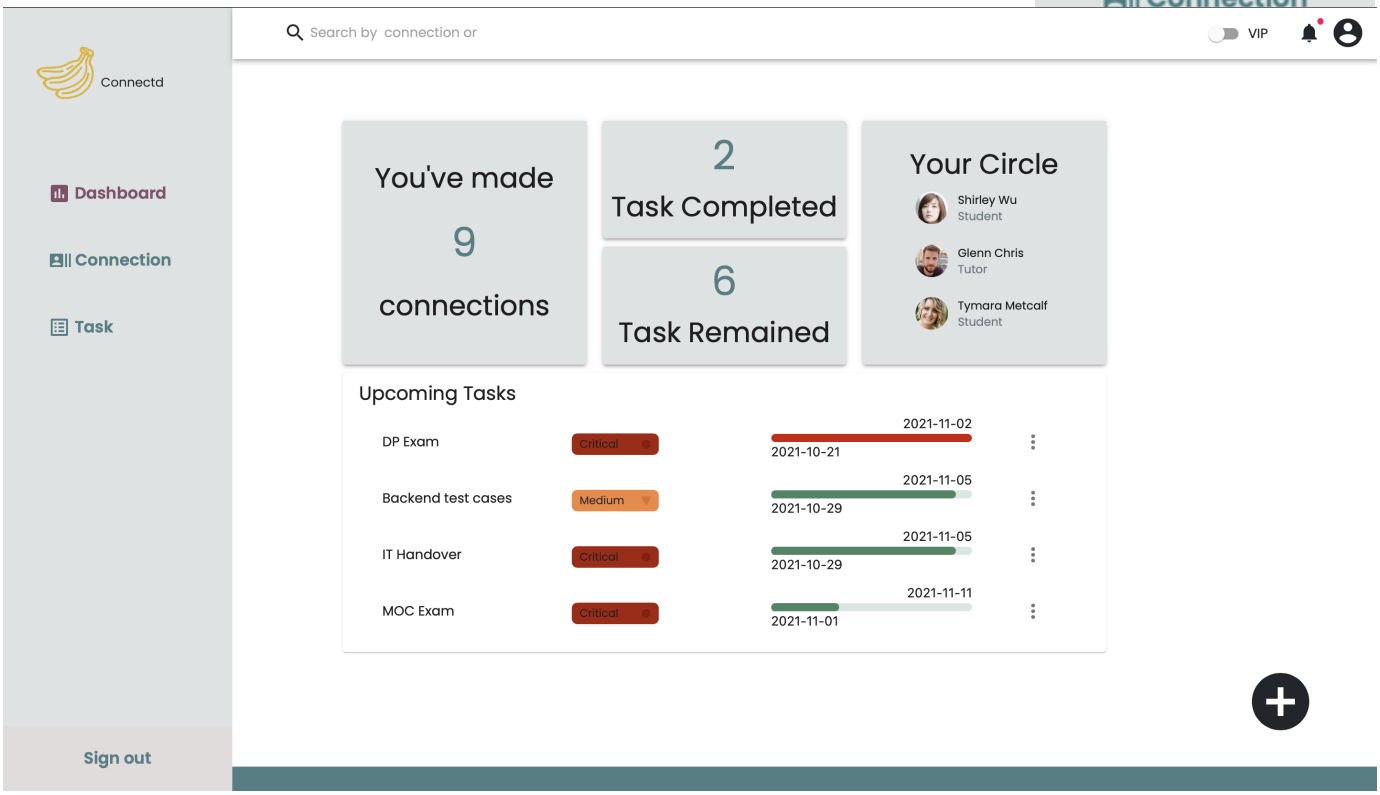
1. Page-specific search bar
2. Page-specific filter - It allows setting a condition-based filter, which works along with your search field.
3. Notification bell - A dot appears when an upcoming task is due within 2 days. Click to view task name.
4. User Profile - Click to open edit view of your profile



Sign out of your account using the **Sign out** button in the bottom left-hand corner.

TIP: Clicking the banana logo will bring you back to the Dashboard.

Dashboard



The Dashboard is your main hub of information. It consists of 5 main metric tables.

It will give you a quick overview of:

1. How many connections you've made.
2. How many tasks are complete.
3. What number of tasks are to be completed.
4. Keep track of your important connections.
5. Upcoming tasks.

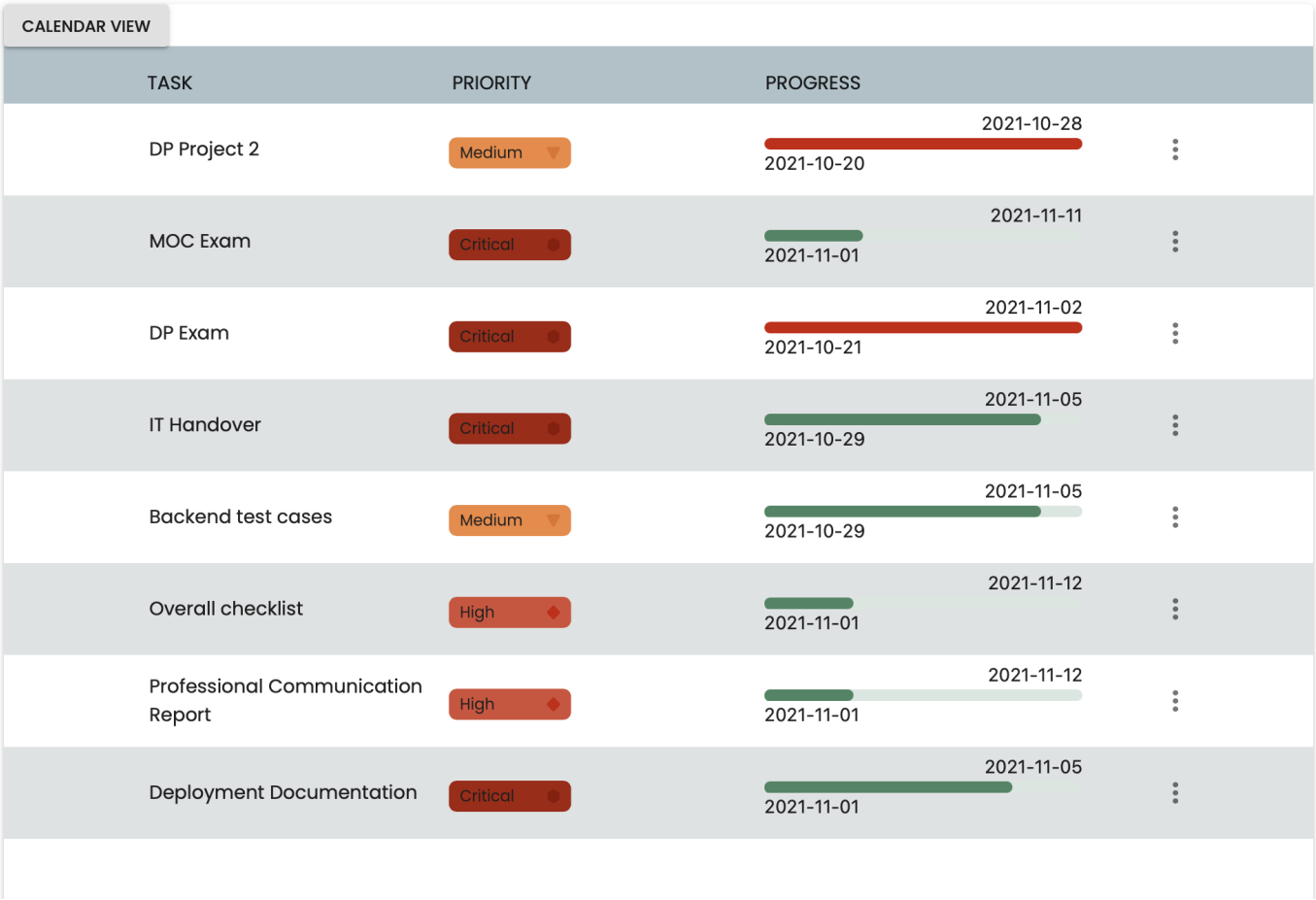
Connections

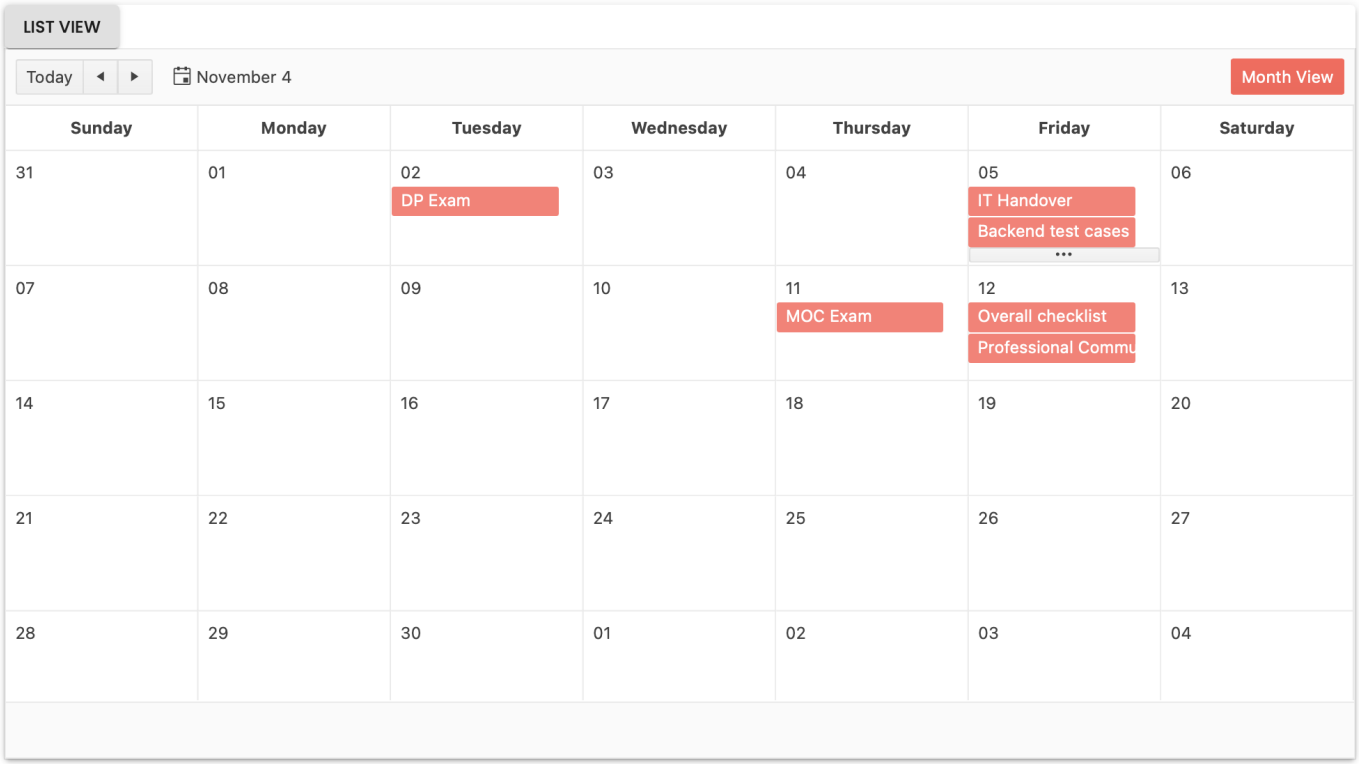
A table view of your connections is available here.

1. You can view a summary of each connection's vital information:
    a. Name, Company, Phone Number, and VIP status.
2. Access more details or delete a connection via the **3 dot menu** button on the far right of each corresponding row.
3. Add Connections with the **speed dial** button in the bottom right corner.

TIP: Click the search bar in the main navigation to use a text search and find one of your connections more quickly! You can also toggle VIPs during your search.

Tasks

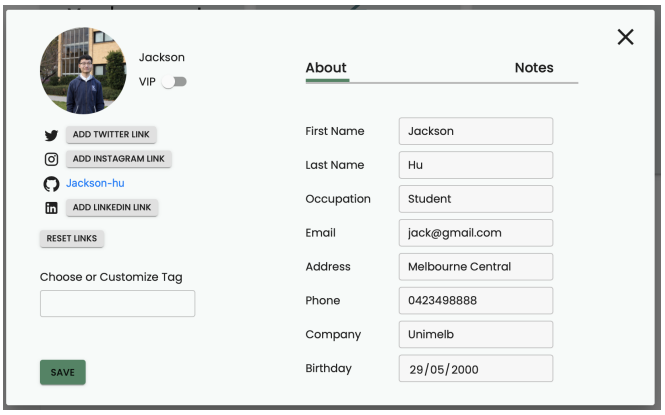


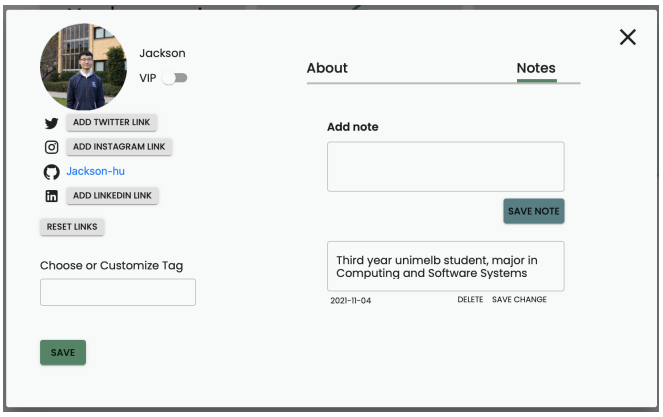


A **table** and **calendar view** is available for your current tasks is created and displayed here.

TIP: Click the search bar in the main navigation to use a text search and find one of your tasks more quickly! You can also filter your results based on priority.

Profile





To modify your user information:

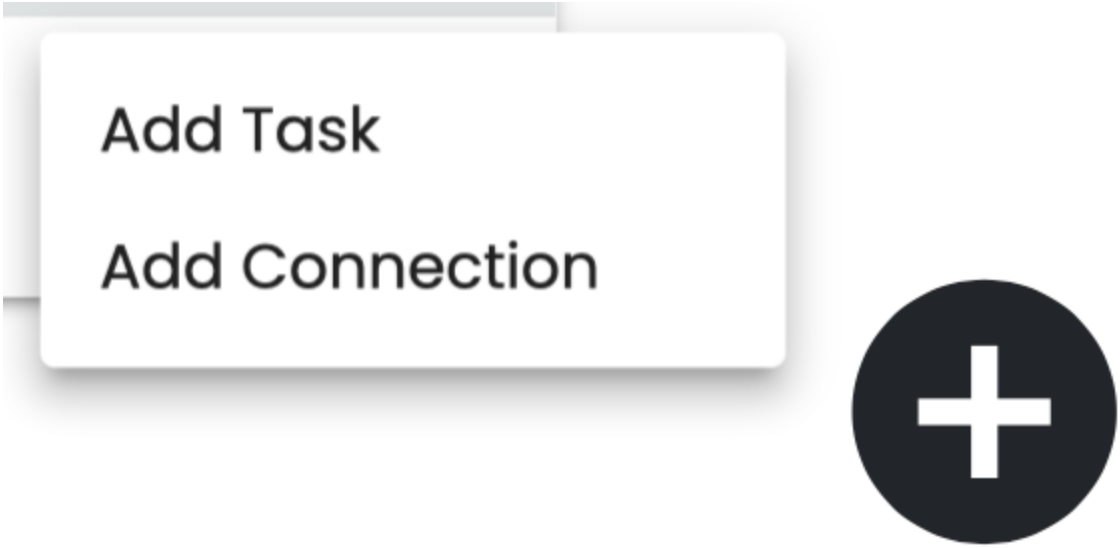
1. Click on your user avatar in the top right-hand corner.
2. Modify the information of your choice
3. Click the **Save** button.

Create, Manage, and Publish Content

Creating a new entry

On the bottom right corner of the table view interface, an **Add New [collection type name] speed dial** button is displayed. It allows you to create a new entry for your collection type.

Clicking on the new entry button will redirect you to the edit view, where you will be able to write the content of the new entry.

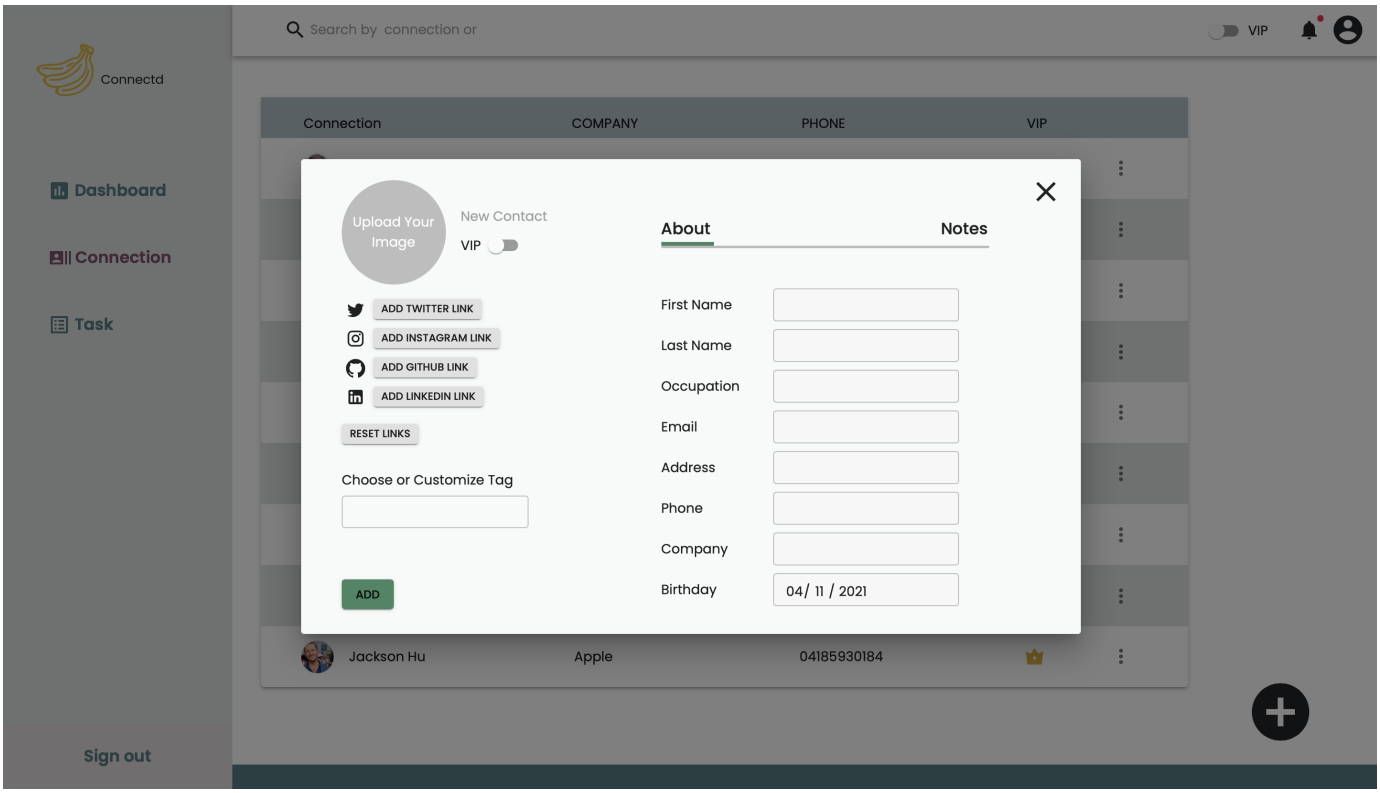


Filling entry fields

To write or edit content:

1. Access the edit view of your collection type.
2. Write your content, following the available field schema. You can refer to the table below for more information and instructions on how to fill up each field type.

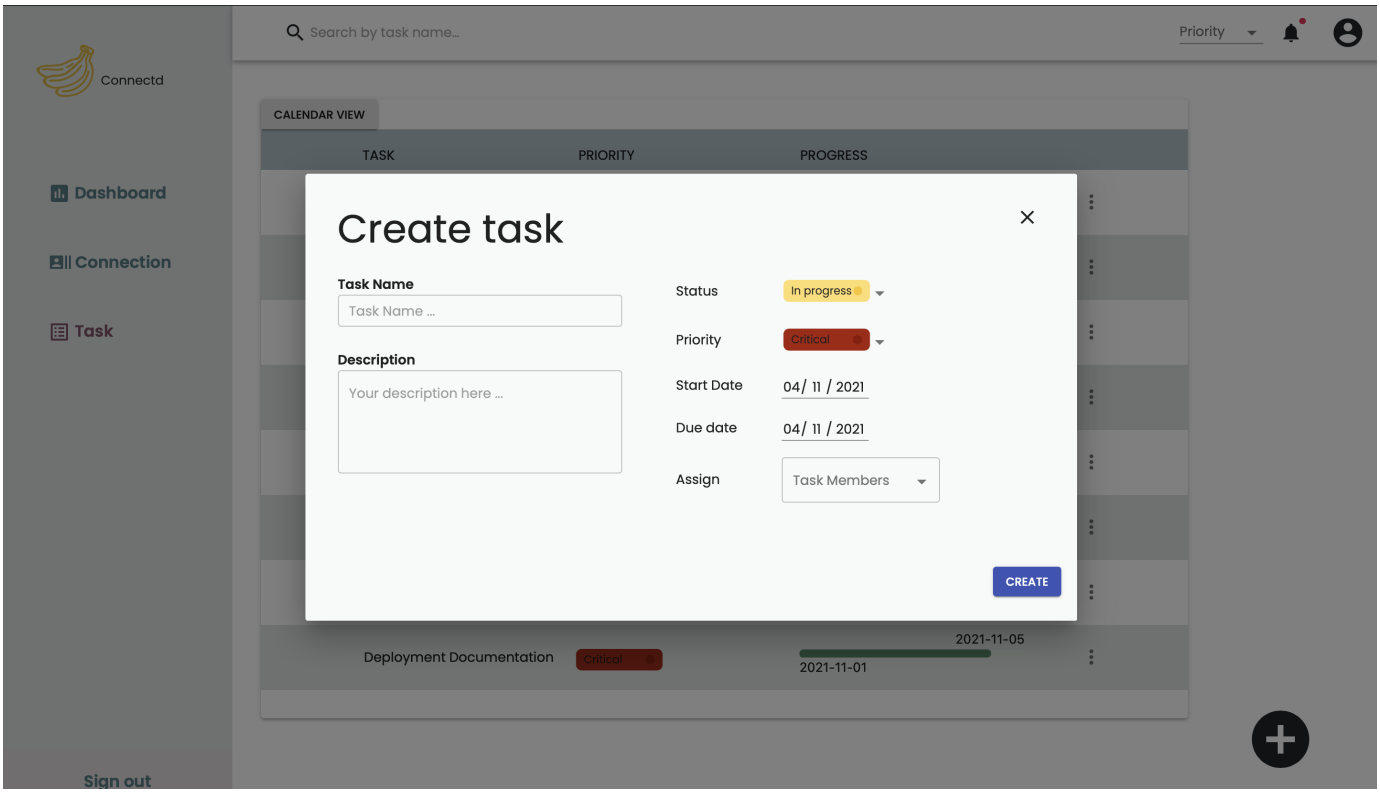
1. Connections



Field name	Instructions
First Name	Write connection’s first name in the textbox.
Last Name	Write connection’s last name in the textbox.
Occupation	Write connection’s occupation in the textbox.
Email	Write connection’s email in the textbox.
Address	Write connection’s address in the textbox.

Phone	Write connection’s phone number in the textbox.
Company	Write connection’s company in the textbox.
Birthday	Calendar select or write connection’s birthday in the textbox.
Add Twitter Link	Write connection’s Twitter username in the textbox.
Add Instagram Link	Write connection’s Instagram username in the textbox.
Add Github Link	Write connection’s Github username in the textbox.
Add Linkedin Link	Write connection’s Linkedin username in the textbox.
Reset Links	Clear all connection’s social media links.
Upload Your Image	Click to upload a photo for connection.
New Contact	Alternative area to write connection’s first name.
VIP	Toggle switch to mark connection as a very important person.
Choose or Customize Tag	Write a custom tag/category for your connection or choose an existing tag from dropdown.
Add Note	Write note about connection in textbox.

2. Tasks



Field name	Instructions
Task Name	Write your task name in the textbox.
Description	Write a description of task in the textbox.
Status	Choose `In progress`, `Review`, or `Complete` from the dropdown.
Priority	Choose `Critical`, `High`, `Medium`, `Low`, or `Unknown` from the dropdown.
Start Date	Calendar select or write date for start date of task.

Due Date	Calendar select or write date for due date of task.
Assign	Select Connections assigned to task from dropdown or write in textbox.

Managing entries

1. Connections

1. Click the **3 dot menu** button on the far right of each corresponding row to edit or delete a connection.
2. Clicking on the **Edit** button will redirect you to the edit view, where you will be able to write the content of the existing entry.
3. Clicking on the **Notes** tab will allow you to manage existing notes. You can edit by writing in the textbox or deleting the note.
4. Click the **Save** button when finished modifying content.
5. You will be redirected to your Connections page and the table view refreshed.

2. Tasks

1. Click the **3 dot menu** button on the far right of each corresponding row to edit or delete a task.
2. Clicking on the **Edit** button will redirect you to the edit view, where you will be able to write the content of the existing entry.
3. Click the **Save** button when finished modifying content.
4. You will be redirected to your Tasks page and the table view refreshed.

Timeline

Deadlines

- ~~Week 6 “Inception” checklist due Sept 3 @ 23:59~~
- ~~Peer feedback due Sept 19 @ 23:59~~
- ~~Week 9 checklist due Oct 1 @ 23:59~~
- ~~Week 12 checklist due Oct 22 @ 23:59~~
- ~~Team presentation on Oct 19 @ 11:30am, due Oct 24 @ 23:59~~
- Product Checklist due Nov 12 @ 23:59
- Overall Checklist due Nov 12 @ 23:59
- Individual Contribution statement due Nov 12 @ 23:59
- Professional Communication Report due Nov 12 @ 23:59

Lecture Schedule

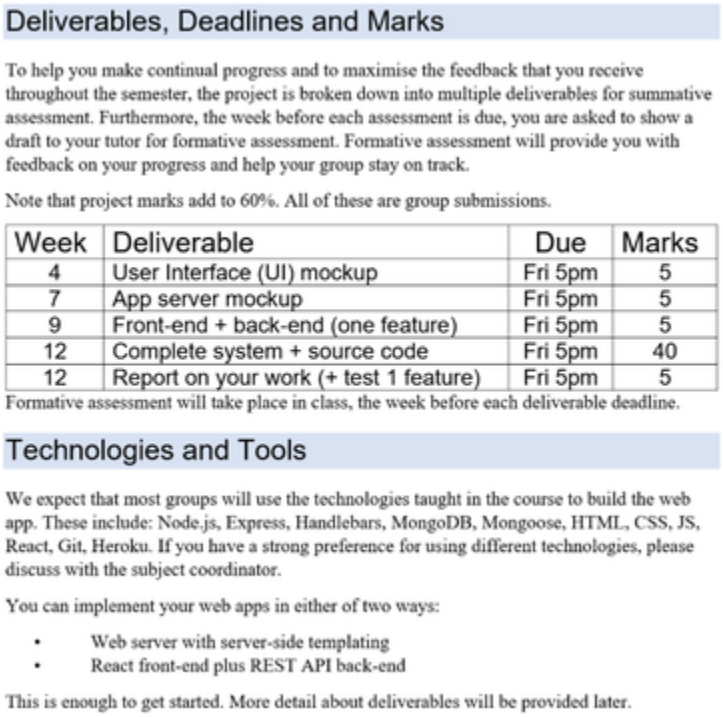
- ~~Lecture 1 – Subject and project overview – expectations~~
- ~~Lecture 2 – *Practical things you need to know before it is too late*~~
- ~~Lecture 3 – Requirements – user stories and other artefacts~~
- ~~Lecture 4 – Agile software development and communication tools~~
- ~~Lecture 5 – Architecture, Coding, Pair programming~~
- ~~Lecture 6 – Testing and Deployment~~
- ~~Lecture 7 – Feedback so far, Professional communication, Ethics~~
- ~~Lecture 8 – Transitioning from university to the workforce (Recent student perspective)~~
- ~~Lecture 9 – Transitioning from university to the workforce (Employer perspective)~~
- ~~Lecture 10 – Preparing for a career in IT~~
- ~~Lecture 11 – Subject review – project handover~~
- ~~Lecture week 12 Presentations~~

Workshop Schedule

- ~~Workshop 1 – Introductions and forming teams~~
- ~~Workshop 2 – Setting up tools~~
- ~~Workshop 3 – Requirements~~
- ~~Workshop 4 – Communication tools – estimation, Trello, stand-ups~~
- ~~Workshop 5 – Architecture and coding standards~~
- ~~Workshop 6 – Testing plan~~
- ~~Workshop 7 – Sprint reviews~~
- ~~Workshop 8 – Pipelines and Deployment~~
- ~~Workshop 9 – Professional communication, peer feedback~~
- ~~Workshop 10 – Ethics~~

- ~~Workshop 11 – Revisiting clients, handover and presentations~~
- ~~Workshop week 12 Presentations~~

A guide from Web IT:



Deliverables, Deadlines and Marks

To help you make continual progress and to maximise the feedback that you receive throughout the semester, the project is broken down into multiple deliverables for summative assessment. Furthermore, the week before each assessment is due, you are asked to show a draft to your tutor for formative assessment. Formative assessment will provide you with feedback on your progress and help your group stay on track.

Note that project marks add to 60%. All of these are group submissions.

Week	Deliverable	Due	Marks
4	User Interface (UI) mockup	Fri 5pm	5
7	App server mockup	Fri 5pm	5
9	Front-end + back-end (one feature)	Fri 5pm	5
12	Complete system + source code	Fri 5pm	40
12	Report on your work (+ test 1 feature)	Fri 5pm	5

Formative assessment will take place in class, the week before each deliverable deadline.

Technologies and Tools

We expect that most groups will use the technologies taught in the course to build the web app. These include: Node.js, Express, Handlebars, MongoDB, Mongoose, HTML, CSS, JS, React, Git, Heroku. If you have a strong preference for using different technologies, please discuss with the subject coordinator.

You can implement your web apps in either of two ways:

- Web server with server-side templating
- React front-end plus REST API back-end

This is enough to get started. More detail about deliverables will be provided later.

Meetings

- Team Meetings
- Supervisor Meetings
- Client Meetings

Team Meetings

Error rendering macro 'create-from-template' : Failed to find net.sf.hibernate.Session from the current thread

Incomplete tasks from meetings

Description	Due date	Assignee	Task appears on
☐ List packages and file structure for Django			6) 2021-09-28 Team

Decisions from meetings

Page Title	Decisions

No decisions found

All meeting notes

Title	Creator	Modified
4) 2021-10-29 Client	Tymara Metcalf	Nov 05, 2021
7) 2021-10-26 Team	Tymara Metcalf	Oct 26, 2021
8) 2021-10-22 Supervisor	Tymara Metcalf	Oct 22, 2021

7) 2021-10-08 Supervisor	Tymara Metcalf	Oct 22, 2021
6) 2021-09-28 Team	Tymara Metcalf	Oct 13, 2021
3) 2021-10-8 Client	Tymara Metcalf	Oct 08, 2021
6) 2021-10-01 Supervisor	Tymara Metcalf	Oct 01, 2021
1) 2021-08-09 Team	Jackson Hu	Oct 01, 2021
3) 2021-09-03 Supervisor	Jackson Hu	Oct 01, 2021
3) 2021-08-31 Team	Tymara Metcalf	Oct 01, 2021
2) 2021-08-20 Supervisor	Tymara Metcalf	Oct 01, 2021
4) 2021-09-10 Supervisor	Tymara Metcalf	Oct 01, 2021
5) 2021-09-22 Team	Tymara Metcalf	Sep 23, 2021
5) 2021-09-17 Supervisor	Tymara Metcalf	Sep 17, 2021
2) 2021-09-7 Client	Jackson Hu	Sep 08, 2021
1) 2021-08-13 Client	Tymara Metcalf	Sep 08, 2021
4) 2021-09-02 Team	Jackson Hu	Sep 06, 2021
1) 2021-08-13 Supervisor	Tymara Metcalf	Aug 31, 2021
2) 2021-08-12 Team	Jackson Hu	Aug 31, 2021

1) 2021-08-09 Team

Date

09 Aug 2021

Participants

- @ Jackson Hu
- @ Han Liu (Unlicensed)
- @ Alexander Cain
- @ David Fletcher

Goals

- Finalise a team name
- Set preference of what technologies we're going to use

Discussion topics

Item	Notes
Team Name	"Connectd"
Project Type	Web-based Application

Tools	Notes
Front End	React Framework
Back End	Django Framework (may use other databases rather than built-in SQL lite)
Testing Framework	Github Actions / Jest

Deployment	Heroku
API Documentation	Swagger

Checklist

- [x] Decide on a technology stack
    - [x] Front-end framework
    - [x] Back-end framework
    - [x] Testing framework
    - [x] Deployment tool
    - [x] API documentation

To do next

- [x] Question list for client during first meeting
- [x] Learning the technologies
- [x] Try some of the existing CRMs know the functionalities

2) 2021-08-12 Team

Date

12 Aug 2021

Participants

- @ Jackson Hu
- @ Han Liu (Unlicensed)
- @ Alexander Cain
- @ David Fletcher
- @ Tymara Metcalf

Goals

- Go through the things prepared for the meeting
- Finalise the client meeting agenda
- Discuss how to guide the client for requirements
- Discuss whether we need to do the Do/Be/Feel list together
- Welcome new team member - Tymara

Discussion topics

Topic	Duration	Speaker	Show anything?
Greeting	3 mins	David	Agenda on slide
General Questions	10 mins	David	Questions on slide
Do/Be/Feel	5 mins	Together	List Template
User Stories	15 mins	Together	List Template
Specific Questions (clarify)	5 mins	Alex	
Mood Board	5 mins	Jackson	Mood Board
Organise next meeting	3 mins	David	

Checklist

- [x] Go through things we've done
- [x]

- [ ] Finalise meeting agenda
- [x] Discuss what to discuss with client and how
- [x] Assign the roles during the meeting (e.g. host)

To do next

- [x] Create a simple slide with guiding questions
- [x] Prepare the list template on Notion
- [x] Notify the client about the upcoming meeting time with zoom link sent in Slack

3) 2021-08-31 Team

Date

31 Aug 2021

Participants

- @ Jackson Hu
- @ Han Liu
- @ David Fletcher
- @ Tymara Metcalf

Goals

- Prepare final to-dos for sprint 1 end

To do next

- [x] Link Heroku to Github repo(s) @ Han Liu let me know if you have any issues with assess or connection
- [x] Clean up Documentation for Review
  - [x] Add Backend and Frontend architecture/diagrams to Design Folder
  - [x] Justify stack (tool) choices
  - [x] Trello Board
- [x] Schedule sprint 1 Retrospective
- [x] Schedule sprint 1 Client review
- [x] Schedule sprint 2 Planning session
- [x] Submit the Week 6 Deliverable by this Friday (there is a one week grace period if you have any difficulties but please get in touch with me if this is the case)

4) 2021-09-02 Team

Date

02 Sep 2021

Participants

- @ Jackson Hu
- @ Han Liu
- @ David Fletcher
- @ Tymara Metcalf
- @ Alexander Cain

Goals

- Organise retrospective for sprint 1
- Start working on the inception list

- Update current progress with deployment on Heroku and authentication API
- Review diagrams from other groups and discuss the artefacts we better add in
- Roughly assigned task we need to do by next Tuesday
- Discuss the sprint 2 planning

Summary

1. Both repositories successfully linked to Heroku but currently not running properly
2. Decide to choose other authentication framework or packages to develop account related API since the previous version was not compatible with React frontend
3. Need to delay the time to finish sprint 1 retrospective and submit inception list because some artefacts are considered to be added later during weekend
4. Confirm the time for client meeting on 07 Sep 2021 at 1pm
5. Alex will probably be absent for this Friday's stand-up meeting

Task	Sub-task	Participant
Artefact	Motivational model	@ Tymara Metcalf
	Deployment diagram	@ Tymara Metcalf
	User stories	@ Alexander Cain
	Database diagram	@ Tymara Metcalf @ David Fletcher
	Other diagrams	@ David Fletcher
Stack Choice	Frontend justification	@ Han Liu
	Backend justification	@ Jackson Hu
Revised Authentication API		@ Jackson Hu
Inception List	Draft on Google Doc	@All

To do next

- [x] Complete the tasks on to-do list from last team meeting
- [x] Finish and submit inception list by Tuesday next week
- [x] Start to organise what we've done in sprint 1 and prepare for second client meeting on Tuesday

5) 2021-09-22 Team

Date

22 Sep 2021

Participants

- @ Tymara Metcalf
- @ Han Liu
- @ Jackson Hu
- @ Alexander Cain

Goals

- Check-in, team stand-up, address any issues.

Discussion topics

Time	Item	Presenter	Notes
4:05	General	@ Alexander Cain	• Sprint 2 ending this Friday, how can we all better use our time during this break and going into next week?
4:10	BE	@ Jackson Hu	• Work has been done getting Login and Logout to POST, using postman. • View Demo of Postman for login and logout. • It would be good to use Swagger for documentation purposes. • Username will be email in database and Auth API. Change Frontend to reflect this @ Han Liu @ Tymara Metcalf . • Register will still take First name and last name.
4:15		@ Alexander Cain @ Han Liu	• Can we get some response when user logs out?
4:15	BE, Testing	@ Jackson Hu	• Will work on logout response. • Need to get BE deployed and then setup CORS. • In meantime, can @ Alexander Cain to TEST login/logout and document?
4:17	Deployment	All	• To initiate deployment we need to get backend API and database sorted and merged.. then push file structure to be used by all to main. @ David Fletcher @ Jackson Hu will settle this week. • Then @ David Fletcher @ Han Liu will merge their frontend, and prepare for push to main(Frontend), basic prototype that can be navigated at minimum.
4:30	FE	@ Han Liu @ Tymara Metcalf	• View Demo of Frontend navigation • Change password type to “password” to help browser censor information. • Create more icons for tasks priority and status.
4:40	Requirements, optional tasks	All	• No need for public/private toggle for tasks. • Tasks will allow for dropdown categories, no need for tags per client. But this can be implemented in Sprint 3 (optional). • Google/Facebook API (optional) save for Sprint 3 consideration. • Contacts will have tags and categories. • Tags are custom generated by user, while categories are preset in dropdown selection (ie. ENUM). • Tag and categories are something to keep in mind but will be done later (sprint 3 /filtering/optional).
4:50	Prioritizing HIGH requirements, our focus this sprint.	All	1. Deployment 2. Login/Logout 3. Create/edit/delete tasks (basic) 4. Create/edit/delete contacts (basic) 5. Search for contact by task (basic) 6. Search for task by contact (basic) 7. VIP tag for contact General idea is to create simple design and output to start then FE/design will tidy up.
4:50	Medium	All	1. Edit user profile
4:55	Trello	@ Tymara Metcalf	• Assign tasks on trello, organise. • about 15 tasks were completed Sprint 1, with 3 uncompleted. We should aim for ~15 this sprint. Currently at 11 tasks in progress/completed, with 10 pending assignment. • User profile, Login, Mainpage skeleton brought over to sprint 2 from 1st sprint.

6) 2021-09-28 Team

Date

28 Sep 2021

Participants

- @ Tymara Metcalf
- @ Han Liu
- @ Jackson Hu
- @ Alexander Cain
- @ David Fletcher

Goals

- Sprint 2 Checklist run through

Discussion topics

Time	Item	Presenter	Notes
11:09	Sprint Checklist	@ Alexander Cain	• Processes • Artefacts • Testing
11:10		Glenn	• A level of fidelity is expected, start with high what types of testing will be done, manual, unit testing, or integration, • Then lower level: What are you going to run these tests on? • Then how you are going to test? Create a table for a certain feature. Screenshots on the side Google some structures for this table. • Discuss what the team knows about testing. • Decide on how team will move forward with testing • Get code coverage up. Find faults in the code. Do not need to check every line of code. • Never 100% certainty that code is bug free • Code standards: write up doc for this • Deployment pipeline: discuss branching structure, if you create branches, etc. Talk about when you deploy, when you commit, push, what is automatic, or manual.
11:15		@ David Fletcher	Are integrations are a necessicity?
11:15		Glenn	• Focus on high quality tests. Security is more important than rendering a page. • Define a reasonable scope. That is the key to this subject. Be honest with yourself, your team, and your client.
11:20		@ Alexander Cain	Run through checklist. So we can setup goals for end of sprint.
11:22		@ Alexander Cain	Requirements need updating? No
		@ Tymara Metcalf	Create a V2 for client requirements, if any changes have been made.
11:33		@ Jackson Hu	Pull up page version history in parent to see version changes.
11:36		@ Tymara Metcalf	Have design artefacts been modified?
11:36		@ Jackson Hu	Add API
11:37		@ Alexander Cain	Edit update database model? and explain how Django file structure works.
11:40		@ Tymara Metcalf	Use Trello for tasks completed this sprint.
11:41		@ Jackson Hu	I'll write up some coding standards.
11:42		@ Han Liu	Need 2 days to finalise some frontend for unit tests
11:45		@ Alexander Cain	I can work on some testing plans. Will work with @ Tymara Metcalf for functional tests.
11:50		All	Scope for this sprint, we won't be able to cover deployment pipeline.

Action items

- [x] Create a V2 for client requirements
- [x] Edit update database model, V3
- [ ] List packages and file structure for Django
- [x] Testing plans, HI level: What types of tests? LO level: What will you run these tests on, How will you test?

7) 2021-10-26 Team

Date

26 Oct 2021

Participants

- @ Tymara Metcalf
- @ Han Liu
- @ Jackson Hu
- @ Alexander Cain
- @ David Fletcher

Goals

- Go over what needs to be done before handoff to Client.

Discussion topics

Time	Item	Presenter	Notes
2:07	Sprint 3 Review	@ Tymara Metcalf	• Setup meeting with Client for Sprint review, pre-handover, and actual handover. • @ David Fletcher will arrange. • @ Jackson Hu set up a week 13+ for Trello • @ Alexander Cain will work on some responsive design • @ Han Liu @ Tymara Metcalf will work on last dashboard final touches
2:12	Handover	@ Jackson Hu	• Readme file for both repos. ◦ Briefly mention how to setup environment • Prepare a comprehensive PDF for product. • Handover Next week 5/11 Friday 4:30pm

2:14	Overall checklist	@ Alexander Cain	• 1st part: State how and when the handover happened
			  ◦ You need to say when the handover meeting happened, if and where the project is deployed, and what was handed over,
			
			  for example a release from Github, an export from Confluence or something else.
			• 2nd part: Boasting page (List of our Team Accomplishments)
			  ◦ which aspects of your project the team is most proud of
			• Overall submission should be at most two pages.

2:19	Product checklist	@ Tymara Metcalf	• Has the project been deployed? If so, please provide link
			• Has the project been handed over to the client? Yes/no
			  ◦ If yes, please specify what was handed over.
			  ◦ Possibilities include a Github release, an exported Confluence site, or just a link to a repository,
			• If something else, please briefly specify.
			• Your submission should be no more than 1 page.

2:20	Sprint 3 Retro	All	https://comp30022-079.atlassian.net/wiki/pages/resumedraft.action?draftId=35061895
2:24	Client Reivew	@ Alexander Cain	• Go over user requirements and acceptance criteria with client. • Then prepare that for the final handover. • This week 29/10 Friday 4:30pm

Supervisor Meetings

1) 2021-08-13 Supervisor

Date

13 Aug 2021

Participants

- @ Jackson Hu
- @ Han Liu (Unlicensed)
- @ Alexander Cain
- @ David Fletcher
- @ Tymara Metcalf
- Glenn Phillips (Supervisor)

Goals

Discuss progress from client meeting

Discussion topics

- The format of workshops will change to a more flexible style in the coming weeks
- Teams have the ability to sit in on other stand-ups if desired.
- Limit standup time to 10-15 minutes, as there is another team allocated at 2:15pm
- Avoid meeting fatigue, consider pair programming or smaller groups

To do next

- [x] Glenn to join all conversation channels by next week
- [ ]

2) 2021-08-20 Supervisor

Date

20 Aug 2021

Participants

- @ Jackson Hu
- @ Han Liu
- @ Alexander Cain
- @ David Fletcher
- @ Tymara Metcalf
- Glenn Phillips (Supervisor)

Discussion topics

Notes
• Maintain 5/5 attendance and communication
• Bring what you are working on, have a little to say or questions prepared.
• Understanding Django, unique ID clash with MongoDB
• List database entities for David
• Think about how sprint 1 will be compiled.

Action items

- [x] Get Deployment going on Heroku
- [x] Start Inception Checklist
- [x] Setup routes for backend API
- [x] Link Github to Heroku
- [x] Start compiling documents for Sprint 1

Decisions

3) 2021-09-03 Supervisor

Date

03 Sep 2021

Participants

- @ Jackson Hu
- @ Han Liu
- @ David Fletcher
- Glenn Phillips (Supervisor)

Discussion topics

Notes
• Agree to delay the submission of inception list until next Tuesday **before workshop**
• Better to submit by Monday night
• The only thing to submit is the **inception list**, with links to specific documentation or diagrams
• Need to finish the **team retrospective** by then, but flexible for review with the client
• Show the updated draft of Google doc
• It's okay to not have the deployment done, but it's a must for sprint 2
• Reminder of the planning of sprint 2 starting next week

Action items

- [x] Finish retrospective
- [x] Complete the inception list on time
- [x] Submit
- [x] Finish review during client meeting

Decisions

4) 2021-09-10 Supervisor

Date

10 Sep 2021

Participants

- @ Alexander Cain
- @ David Fletcher
- @ Han Liu
- @ Tymara Metcalf
- Glenn Phillips (Supervisor)

Goals

- Short meeting to touch base after Sprint 1's end
- Receive feedback going into Sprint 2

Feedback

- Set focus on deployment, key for end of sprint 2.
- Keep documentation flowing, good models

Action items

- [x] Organise Confluence pages for ease of navigation
- [x] Consider some testing methodology and coding standards

Links to consider

https://medium.com/@psengayire/the-importance-of-coding-standards-and-conventions-in-the-software-development-team-how-they-can-5d252556a05

https://codebeautify.org/

https://www.geeksforgeeks.org/coding-standards-and-guidelines/

https://medium.com/zenkit/agile-methodology-an-overview-7c7e3b398c3d

https://www.atlassian.com/agile/project-management/estimation

https://www.pmi.org/learning/library/agile-project-estimation-techniques-6110

5) 2021-09-17 Supervisor

Date

17 Sep 2021

Participants

- @ David Fletcher
- @ Han Liu
- @ Jackson Hu
- @ Tymara Metcalf
- Glenn Phillips (Supervisor)

Goals

- Discuss 2nd week of Sprint 2 in Stand-up

Discussion topics

Time	Item	Presenter	Notes
2:00	Overview	@ Tymara Metcalf	Quick touch on trello, daily/weekly tasks, keeping to-do's in one place
2:02	FE	@ Han Liu	Working on task page, to continue through weekend
2:04	BE	@ David Fletcher	Finishing Connections, to work through the weekend
2:04	BE	@ Jackson Hu	Auth/Routing, plans to Pair program with @ Han Liu
2:04	Time management	Glenn Phillips	• Note: we are half way through the project, start looking at what is going to be feasible • Prioritise what you want to do, have a chat as a team. • Dont feel the need to work through the break • Try to get something concrete, polished, it doesnt have to be the whole app • Dont take on too much. Complete some features. • Dont stress too much, the course is about the development and feeling of completing something as a team. • Any roadblocks? "Nope."
2:09		Glenn Phillips	• Create a burn down chart. • Figure out how many story points you did in S1 and then compare to S2 that equals your Sprint velocity.
2:09	Requirements	@ Jackson Hu	Will not meeting clients requirements affect grading?
2:10		Glenn Phillips	• Adjust for it and make a plan. It's all about Time management. • Big reason requirements change is TIME, money, resources. • Tailor project requirements to the team. • Make sure to update the requirements as such. • Then Create a v2 of product backlog or product model. • It's on you to define the requirements as a team, AND make sure the client is onboard as well.

6) 2021-10-01 Supervisor

Date

01 Oct 2021

Participants

- @ David Fletcher
- @ Han Liu
- @ Jackson Hu
- @ Tymara Metcalf
- @ Alexander Cain
- Glenn Phillips (Supervisor)

Discussion topics

Time	Item	Presenter	Notes
2:00	Presentation	Glenn	Week 12: 11:30 slot, 12-15 mins.
2:02	Testing	@ Alexander Cain	Looking into testing plans.
2:02		@ Tymara Metcalf	Showing Confluence.
2:08	Connection	@ David Fletcher	Posting a Database model.
2:08	BE	@ Jackson Hu	Finalise backend API routes, Heroku deployment, coding standards.
2:10	FE	@ Han Liu	Testing POST.

2:11	Admin	@ Tymara Metcalf	Write up Testing approach, instructions.
2:12	Checklist	Glenn	Fair to submit after the weekend, before Tuesday morning.

7) 2021-10-08 Supervisor

Date

08 Oct 2021

Participants

- @ David Fletcher
- @ Han Liu
- @ Jackson Hu
- @ Tymara Metcalf
- @ Alexander Cain
- Glenn Phillips (Supervisor)

Discussion topics

Time	Item	Presenter	Notes
2:00	Work balance	Glenn	Alex and Tymara could have contributed a bit more on the coding side of things. Could have been why something did not get done. Try to find a balance on coding, and collaborate in execution.
2:02	Testing	Glenn	-marks, because I didn't see any automated tests or pipelines.
2:04	Documentation	Glenn	-marks, documentation lacking.
2:08	Mock Presentation	Glenn	• We'll run through presentation Tuesday next week, good opportunity to get feedback. • Try to have slides, if no time use Confluence. • General slide format: Overview + example.

8) 2021-10-22 Supervisor

Date

22 Oct 2021

Participants

- @ David Fletcher
- @ Han Liu
- @ Jackson Hu
- @ Tymara Metcalf
- @ Alexander Cain
- Glenn Phillips (Supervisor)

Goals

- Go through the remaining assignments
- Discuss about the handover

Discussion topics

Time	Item	Presenter	Notes
2:00	Overview	All	• Quick overview from all on what's been going on. (Code, testing, Checklist, etc.)
2:09	Overall Checklist		• First page (5 parts)

		Glenn	• Second page (boasting page)
2:13	Product Checklist	Glenn	• Similar to Sprint checklists • GitHub release
2:15	Professional Comm Report	Glenn	• Individual • Reflective writing
2:16	Individual contribution Statement	Glenn	Optional, your chance to say what you've done.
	Product handover	Glenn	• GitHub release • README file including instructions of how to set up dev environment • Prepare a bundle (documentation + code) • Prefer formal meeting rather than email

Action items

- [ ]

Decisions

Client Meetings

Client info:

- Name: Siyuan Wu
- Student ID: 1110062
- Email: sw4@student.unimelb.edu.au

Meeting 1 Aug 13, 2021 01:00 PM Australia/Melbourne zoom_0.mp4	https://unimelb.zoom.us/j/89109138812?pwd=ZGt1V2hCR2VHTDZCNmlQbFhzR2dxUT09 Password: 055286 Meeting ID: 891 0913 8812
Meeting 2 Sept 7, 2021 01:00 PM Australia/Melbourne	https://unimelb.zoom.us/j/86919971820?pwd=UXdyVitCclFJUlZWNXc5WE5NREJJZz09
Meeting 3 Oct 8, 2021 04:30 PM Australia/Melbourne zoom_0.mp4	
Meeting 4 Oct 29, 2021 04:30 PM Australia/Melbourne	

Question list prepared for client

General

1. Mobile application or web application? (platform)
2. Main purpose of the product?
3. Main target users?

Functionality

1. What are the main functionalities and features?
    a. Contact management (basic information?)
    b. Interaction tracking
2. Common features (CRM product on market)
    a. Attach events to contacts
    b. Set reminders about important dates
    c. Push notifications
    d. Email integration
3. Optional features

    a. Reporting / analytics (e.g. monthly summary)
    b. Tags / groups
    c. Track money owed
    d. Scan business card

Development

1. Preference of front / back end framework?
2. Performance requirement?
3. Load capacity requirement?
4. Security concerns?
5. Task priority?
6. Due date for specific module?
7. Early deployment for demonstration (for client feedback)?

UI/UX Design

1. Overall style?
2. Colour theme preference? (show mood board)
3. Design language
    a. Flat design (iOS 7)
    b. Neumorphism (macOS 11 Big Sur)
    c. Skeuomorphic design (iOS 6)
    d. Material design (Google)

1) 2021-08-13 Client

Date

13 Aug 2021

Participants

- @ Jackson Hu
- @ Han Liu (Unlicensed)
- @ Alexander Cain
- @ David Fletcher
- @ Tymara Metcalf
- Siyuan Wu (Client)

Slides & Mood Boards

Goals

- Meet Client
- Go over Scope, DoBeFeel, User stories, UX/UI design

Discussion topics

Topic	Duration	Speaker	Show anything?
Greeting	3 mins	David	Agenda on slide
General Questions	10 mins	David	Questions on slide
Do/Be/Feel	5 mins	Together	List Template
User Stories	15 mins	Together	List Template
Specific Questions (clarify)	5 mins	Alex	
Mood Board	5 mins	Jackson	Mood Board
Organise next meeting	3 mins	David	

Details

Scope:

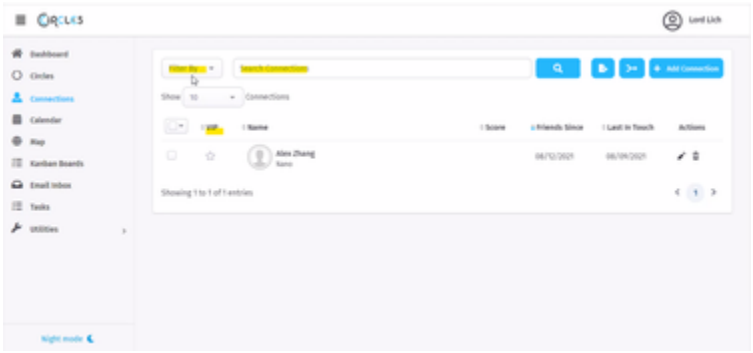
Why are we doing this?
To assist in connection management. Keep track of **connections** - of which members, companies, **tasks** each person is responsible for. So the client can easily find and doesn't forget who has which member or which companies/tasks to be done.

Who needs to be involved?
The client for personal use. But can be used freely by anyone. Remain easily accessible.

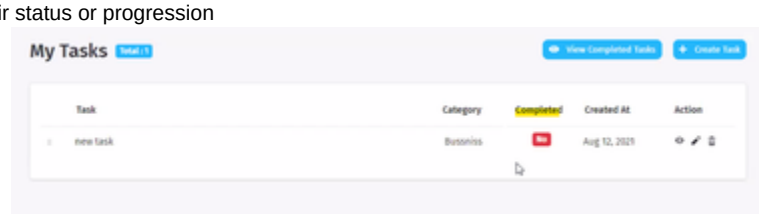
What exactly are we going to do?

Features:

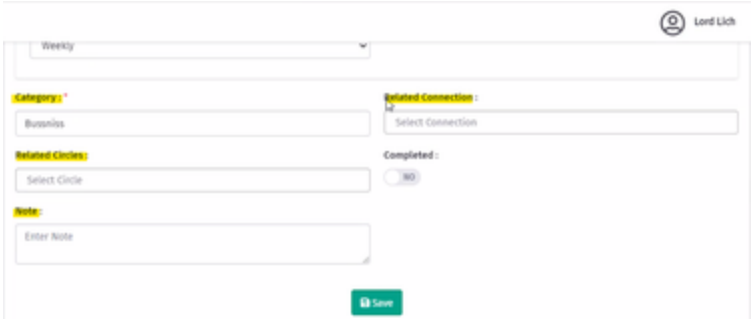
- **Search** by name, related connections, tags, business, VIP(on/off)



- **Add/update Tasks** and their status or progression



- Contact details: name, address, email, company, etc.
- Profile customisation, upload picture, edit personal details.
- **Categories**
- Notes



- Track a connection (optional)
- Filter by <keyword> drop down (optional)

See https://comp30022-079.atlassian.net/wiki/spaces/CRM/pages/33175/Requirements#User-Stories *for more information.*

When are we going to do it?
The client has a flexible schedule for completion. Aim for weeks 11-12.

What is the goal of the project?
Create a program that can record **connections** (name, company, team members, location, etc.), **tasks**, company team members, eg. Monica, Circles, and **search** functionality.

Design:

- The client prefers #2, 4, 10 colours from mood board. Prefers a not too vivid theme (simple, elegant). Highlight only necessary information.
- Deployed Web application, with the potential to work on mobile platforms, consider responsive design.
- Clear, simple, easy to use.
- Function over form.
- No app name preference
- Similar to an advance database/spreadsheet that is searchable and filterable.
- Make it easy to find a connection.

To do next

- [x] Finalise Roles
- [x] Break tasks into sprints (every 3 weeks: 3, 6, 9, 12)
- [x] Assess tasks to be done and migrate to Trello
- [x] Mockup design and review

2) 2021-09-7 Client

Date

07 Sep 2021

Participants

- @ Jackson Hu
- @ Han Liu
- @ Alexander Cain
- @ David Fletcher
- @ Tymara Metcalf
- Siyuan Wu (Client)

Slides

Goals

- Present the Figma prototype
- Ask for further clarification of some requirements
- Get feedback from the prototype
- Complete sprint 1 review

Discussion topics

Topic	Duration	Presenter	Notes
Greeting	2 mins	Alex	
UI Prototype Demo (Figma)	10 mins	Tymara	
Design Review	10 mins	Tymara + Jackson	
User Requirement Clarifications	10 mins	Everyone	
Final Thoughts	5 mins	Everyone	

Further Requirements Clarifications

Feature	Clarifications
Search function	• Filter by name, company or group • Search on key fields (name, company, date, task, connections, notes) - with checkbox to indicate which fields • Search function only needed on tasks and connections page • Ability to search with a filter applied
Dashboard and Metrics	• Count of finished and unfinished tasks (added) • Show most important tasks first (priority and due date) • VIP and most frequent connections
Connections	• Include phone number in quick view (main connection page) • Any additional information can be put in notes
Tasks	• Ability to add start and end date (both optional) • Don’t need way to make public/private tasks (tasks are only seen by user)
Tags/Categories	• Combine the two features • Needs to be customizable

Sprint 1 Review

See Sprint 1 Review *for more information.*

3) 2021-10-8 Client

Date

08 Oct 2021

Participants

- @ Jackson Hu
- @ Han Liu
- @ Alexander Cain
- @ David Fletcher
- @ Tymara Metcalf
- Siyuan Wu (Client)

Slides

Goals

- Present the Heroku prototype
- Ask for further clarification of some requirements
- Get feedback from the prototype
- Complete sprint 2 review

Discussion topics

Topic	Duration	Presenter	Notes
Greeting	2 mins	Alex	
UI Prototype Demo	10 mins	Han	Frontend prototype deployed on Heroku
Design Review	10 mins	Tymara	
User Requirement Clarifications	10 mins	Everyone	Covered part of negotiation of priority for features
Time Management	5 mins	Tymara Jackson	Client agreed to slightly adjust some requirements and sacrifice certain non-core features if needed
Handover	5 mins	Everyone	Inform the client about approximate handover date

Further Requirements Clarifications

Feature	Clarifications
Search function	• Filter by name, company or group

	• Search on key fields (name, company, date, task, connections, notes) - with checkbox to indicate which fields • Search function only needed on tasks and connections page • Ability to search with a filter applied (lower priority)
Dashboard and Metrics	• Count of finished and unfinished tasks (added) • Show most important tasks first (priority and due date) • VIP and most frequent connections
Connections	• Include phone number in quick view (main connection page) • Any additional information can be put in notes
Tasks	• Ability to add start and end date (both optional)
Categories	• Define some preset categories, but allow client to add additional categories
Tags	• Can be anything, no need to be associated with any group. e.g. special event
Assign Members to Task	• Make drop down
Task Notes and Associated Tasks	• Need to be implemented

Time Management

> *Considering the current development progress and the time left until the final handover of program, we updated priority status for certain features together with the client. Since the core features are nearly completed, our client agreed to* ***sacrifice some non-core features*** *if have to.*

Feature	Adjustment
Categorise and prioritise tasks	Client suggested that the ability to categorise connections is still important but it's not a must have feature for tasks, so the priority for it has been decreased from **High** **Medium**
Ability to search with a filter applied	Search by key fields of connection information remains high priority, but the importance of including a filter for search has been decreased from **High** **Medium**
Ability to add start and due date for tasks	Initially the client only asked to have due date, since the client really loves our design of progress bar thus the priority of this feature has been increased from **Optional** **Medium**
Note for connections and tasks	Client reconsidered this feature and categorised this as high priority. **Medium** **High**

Handover

We informed the client about approximate handover date (around the first week of November), client suggested that he would be available to arrange our final meeting for handover before 10 Nov 2021.

Sprint 2 Review

See Sprint 2 Review (draft) *for more information.*

4) 2021-10-29 Client

Date

29 Oct 2021

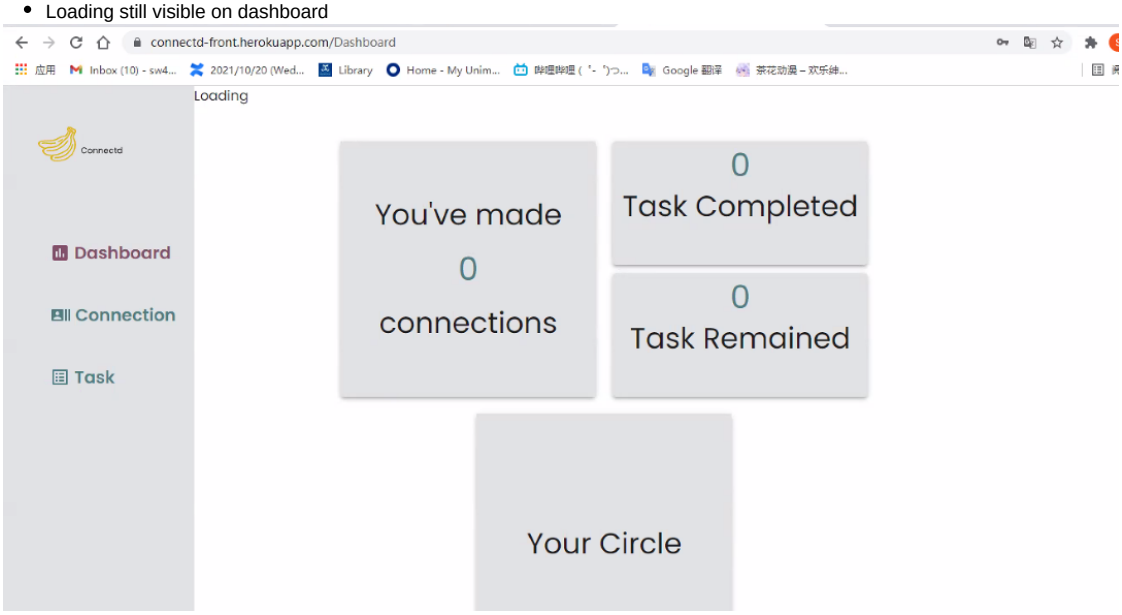
Participants

- @ Jackson Hu
- @ Han Liu
- @ Alexander Cain
- @ David Fletcher
- @ Tymara Metcalf
- Siyuan Wu (Client)

Goals

- Present the Heroku prototype
- Ask for further clarification of some requirements
- Get feedback from the prototype
- Complete sprint 3 review

Discussion topics

Topic	Duration	Presenter	Notes
Live Demo	15 mins	@ David Fletcher @ Tymara Metcalf @ Alexander Cain	n/a
Handover	5 mins	@ Tymara Metcalf @ David Fletcher @ Alexander Cain Siyuan	• Github repository (x2), Heroku Link, MongoDB acecess • Doc: deployment, use (how-to), i.e. how to connect to your own database and how to deploy. • Handover on Friday
Feedback	5 mins	Siyuan	• Refresh page automatically (not required) • Add function to **prevent** register to same email multiple times or submitting blank register form. • Add tool tips to login/register form • Some more responsive design (not required) • Loading still visible on dashboard  • Will send message on Slack if he has any other bugs or comments. • “Besides the form check when registering, creating tasks and connections, I think it would be better if there can have tips to tell user their operations are successful or failed. When I tried to create a task, the dialog was stuck for a while after clicked the cre button, so I clicked multiple times. Then I found it created many same tasks after refreshing page. So I think it would be good to have some tip to tell user if their operation are successful, or something they filled need to be changed.”

Sprint 3 Review

See https://comp30022-079.atlassian.net/wiki/pages/resumedraft.action?draftId=39419960 *for more information.*

5) 2021-11-05 Client - Handover

Date

05 Nov 2021

Participants

- @ Jackson Hu
- @ Han Liu
- @ Alexander Cain
- @ David Fletcher
- @ Tymara Metcalf
- Siyuan Wu (Client)

Slides

Goals

- Handover source code
- Run through acceptance criteria
- Handover deployment documentation and user manual

Discussion topics

Topic	Duration	Presenter	Notes
Acceptance Criteria	20 min	@ Jackson Hu	• Start and End date issue, i.e. dont allow user to have a due date before start date. • Some helper text for required or optional fields. • Fix some of alert's spelling/grammer • It is fine if works well on iPad or Desktop, but not mobile due to nature of tables and information collections. • Product acceptable
Repository Access	10 mins	@ Han Liu	• Access given to both repositories • Explained README
Deployment Documentation	4 mins	@ Jackson Hu	
Authentication API	3 mins	@ Jackson Hu	
User Manual	3 mins	@ Jackson Hu	
Thank you	5 mins	all	• Release tags are essential for this type of project. • Acceptance critiera: add paths for failures as well of successes.

Final Handover Documents

See Client Handover *for more information.*

Tools and Standards

Tools	Notes
Repository	Github: https://github.com/Andrew-Liu-mel/COMP30022 https://github.com/Andrew-Liu-mel/COMP30022-FrontEnd
Front End	React Framework
Back End	Django Framework (may use other databases rather than built-in SQL lite)
Database	MongoDB
Deployment	Heroku: https://connectdcrm.herokuapp.com/
API Documentation	Swagger
Testing Framework	Github Actions / Jest
Pipelines	
Project Communication	Confluence
Task Management	Trello
General Communication	Slack / Zoom https://app.slack.com/client/T029F2JEJ04/C029W5F1K5K

Coding Standards and Guidelines

Importance

> *Code is read much more often than it is written. The guidelines are intended to improve the **readability** of code and make it **consistent** across the whole project.*

- A coding standard gives a uniform appearance to the codes written by different engineers.
- It improves readability, and maintainability of the code and it reduces complexity also.
- It helps in code reuse and helps to detect error easily.
- It promotes sound programming practices and increases efficiency of the programmers.

Languages Used

Django (Backend)

1. Python

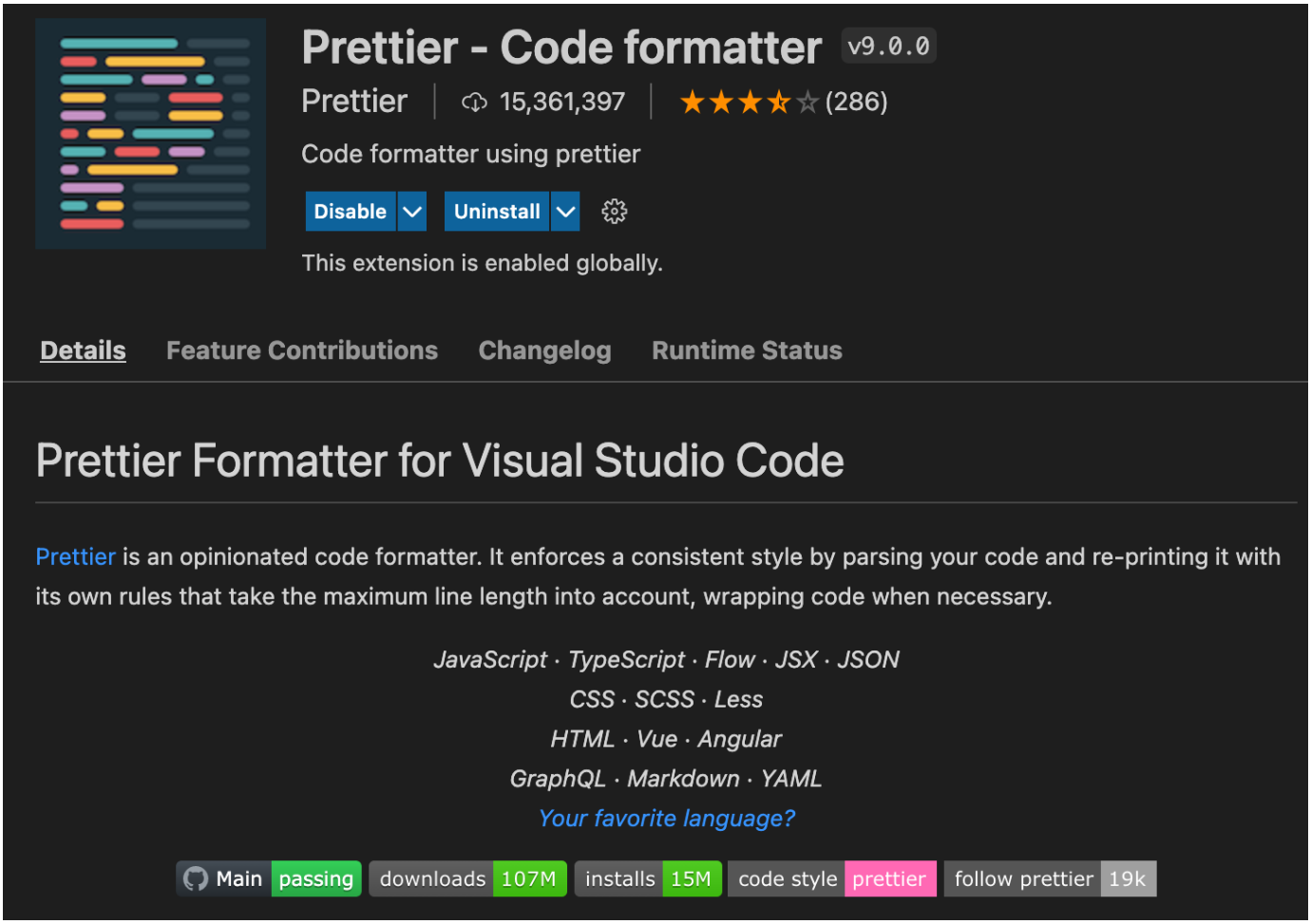
React (Frontend)

1. JavaScript
2. JSX
3. CSS
4. HTML

Formatter Tool

1. Prettier (VSCode Plugin)

Since we all use VSCode as our IDE, we decided to use a VSCode formatter plugin to minimise the work for a consistent coding style across the project.



2. Code Beautify (Online Tool)

Free Online Tools For Developers - codebeautify.org

Popular Functionality

JSON Beautifier	Image to Base64	Base64 to Image	Source Code Viewer
Binary to Text	JSON Viewer	JSON Validator	Base64 Decode
Hex to Decimal	XML Viewer	XML to JSON	HTML Viewer
Encryption-Decryption	Excel to HTML	CSS Validator	XML Validator
JavaScript Validator	CSS Beautifier	ONLINE JSON EDITOR	Decimal to Hex
Binary to Decimal	MD5 Hash Generator	Random IP Address	

Django Coding Standards

https://docs.djangoproject.com/en/dev/internals/contributing/writing-code/coding-style/#imports

https://www.python.org/dev/peps/pep-0008/#naming-conventions

Unless otherwise specified, follow **PEP 8**

[Python Naming Convention] [Imports] [Template Style] [View Style] [Model Style]

Python Naming Convention

- Module Names
  - Short, lowercase names, without underscores.
  - e.g. `myfile.py`
- Class Names
  - CapWords convention.
  - e.g. `MyClass`
- Variable and Function Names
  - Use lowercase with words separated by underscores.
  - e.g. `my_variable`
- Method Names and Instance Variables
  - Same as function naming rules
  - Use one leading underscore only for non-public methods and instance variables.
  - e.g. `_myProtectedVar`
  - Double leading underscores should be used only to avoid name conflicts with attributes in classes designed to be subclassed.

Imports

- Put imports in these groups:
  - Future
  - Standard library
  - Third-party libraries
  - Other Django components
  - Local Django component
  - Try/excepts
- Sort lines in each group alphabetically by the full module name.
- Place all `import module` statements before `from module import objects` in each section.

```
# future
from __future__ import unicode_literals

# standard library
import json
from itertools import chain

# third-party
import bcrypt

# Django
from django.http import Http404
from django.http.response import (
    Http404, HttpResponse, HttpResponseNotAllowed,
StreamingHttpResponse,
    cookie,
)
```

```
# local Django
from .models import LogEntry

# try/except
try:
    import yaml
except ImportError:
    yaml = None

CONSTANT = 'foo'


class Example:
    # ...
```

Template Style

- Put one (and only one) space between the curly brackets and the tag contents.
    - Use `{{ foo }}` instead of `{{foo}}`

View Style

- The first parameter in a view function should be called `request`.

```
def my_view(request, foo):
    # ...
```

Model Style

- Field names should be all lowercase.
- Use underscores instead of camelCase.

```
class Person(models.Model):
    first_name = models.CharField(max_length=20)
    last_name = models.CharField(max_length=40)
```

- The `class Meta` should appear *after* the fields are defined, with a single blank line separating the fields and the class definition.

```
class Person(models.Model):
    first_name = models.CharField(max_length=20)
    last_name = models.CharField(max_length=40)

    class Meta:
        verbose_name_plural = 'people'
```

- If `choices` is defined for a given model field, define each choice as a list of tuples, with an all-uppercase name as a class attribute on the model.

```
class MyModel(models.Model):
    DIRECTION_UP = 'U'
    DIRECTION_DOWN = 'D'
    DIRECTION_CHOICES = [
        (DIRECTION_UP, 'Up'),
        (DIRECTION_DOWN, 'Down'),
    ]
```

React Coding Standards

http://es6-features.org/#Constants

https://en.wikipedia.org/wiki/ECMAScript#6th_Edition_%E2%80%93_ECMAScript_2015

https://www.w3schools.com/react/react_es6.asp

https://www.jondjones.com/frontend/react/react-tutorials/react-coding-standards-and-practices-to-level-up-your-code/

Generally follow ECMAScript 6 standards for JavaScript

[Naming Convention] [Bug Avoidance] [Architecture & Clean Code] [CSS]

Naming Convention

- Component's Names
  - Should be written using pascal case.
  - e.g. `Header.js` `HeroBanner.js` `CookieBanner.js`
- Non-components Names
  - Should be written using camelCase.
  - e.g. `myUtilityFile.js` `cookieHelper.js` `fetchApi.js`
- Unit Test Files
  - Use the same name as its corresponding file.
  - e.g. `CookieBanner.js` `CookieBanner.test.js`
- Attribute Names
  - Should be written using camelCase.
  - e.g. `onClick`
- Inline Styles
  - Should be written using camelCase.
  - e.g. `<div style={{fontSize:'1rem'}}></div>`
- Variable Names
  - Should be written using camelCase.
  - Can contain number and special characters.
  - e.g. `const variable = 'test'; let variableBoolean = true;`
- CSS Files
  - Should be named the same as the component.
  - e.g. `CookieBanner.css`
- Use `.jsx` or `.tsx` extension a for React components

Bug Avoidance

- Use optional chaining if things can be null
- Use the guard pattern/prop types/typescript to ensure your passed in parameters are valid
- Create PURE functions and avoid side-effects
- Avoid mutating state when working with arrays
- Remove all `console.log()`
- Treat props as read-only. Do not try to modify them

Architecture & Clean Code

- No DRY violations. Create utility files to avoid duplicate code
- Follow the component/presentation pattern where appropriate. Components should follow the single responsibility principle

- Use Higher Order Components where appropriate
- Split code into respective files, JavaScript, test, and CSS
- Create a `index.js` within each folder for exporting. This will reduce repeating names on the imports

```
import {Nav} from './Nav.js';
import {CookieBanner} from './CookieBanner.js';

export {Nav, CookieBanner}
```

- Only include one React component per file
- Favour functionless components
- Do not use mixins
- No unneeded comments
- Methods that are longer than the screen should be refactored into smaller units
- Commented out code should be deleted, not committed

CSS

- Avoid Inline CSS
- A naming convention is defined and followed (BEM, SUIT, etc..)

Frontend

🛠️ Stack Choice

Component	Chosen Tool	Justification	Alternatives Considered
Frontend Framework	React JS	➕ Light weight library ➕ Previous knowledge of React & Smooth learning curve for new learners (i.e. based on JavaScript) ➕ Rich documentation, tools/packages & community support ➕ suitable for light applications and fast frontend development	Angular ➖ Steep learning curve (i.e. based on typescript) ➖ Heavy weight framework ➖ Suitable for large-scale application and mature IT team
UI Framework	Material UI	➕ Fast and easy UI development ➕ Follows Material Design principles ➕ Works closely with React ➕ Easy optimization for mobile	Bootstrap & Tailwind ➖ Relatively steep learning curve of tailwind (i.e. require profound understanding of CSS) ➖ Heavy weight and dependence of Bootstrap

📦 Frontend Packages (Up to Now)

Package	Purpose
react, react-dom & react-scripts	Enable and use React
react-router-dom	Set routes & redirection of components
@material-ui/*	Enable material UI
Axios	Fetch data from backend APP using AJAX
react-window	rendering part of a large data set, improves performance

Backend

🛠️ Stack Choice

Component	Chosen Tool	Justification	Alternatives Considered
Backend Framework	Django	➕ Mature web frameworks for Python ➕ Relatively easy to learn ➕ Lots of built-in modules ready to use ➕ Easy to extend and scale	ExpressJS ➖ Low-scope framework ➖ Doesn't handle well with `async/await`
API Documentation	Swagger	➕ Documentation generation ➕ Can be directly embedded and hosted with project ➕ UI friendly for client	Postman ➖ Not ideal for documentation ➖ Limited testing area
Database	MongoDB	➕ NoSQL database ➕ JSON file with structured key-value pair ➕ Work well with no clear schema definitions ➕ Suitable for storing connections and networks ➖ No clear diagram for database models	mySQL ➖ Less flexibility ➖ Requires strict relational schema
Deployment	Heroku	➕ Offers free host for cloud application ➕ Flexible and easy to deploy ➕ Can be configured and linked to GitHub ➖ Limited hosting regions	Elastic Beanstalk (AWS) ➖ Steepy learning curve ➖ Too comprehensive for small projects ➖ Weak application performance monitoring
Testing Framework	Github Actions	➕ Automatic CI/CD testing ➕ Integrated within GitHub ➕ Visualise the workflow	

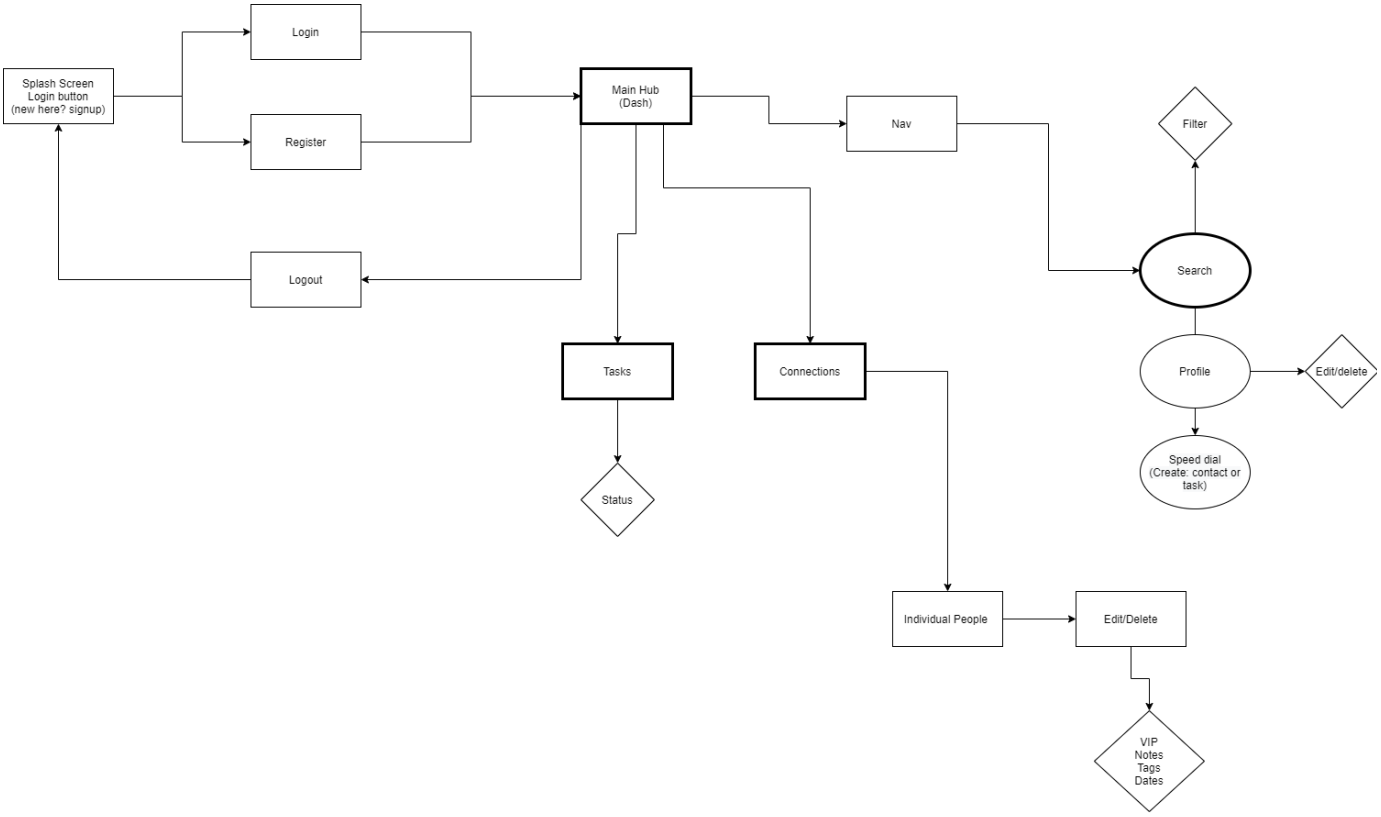
📦 Django Packages

Package	Purpose
Django REST Framework	Powerful and flexible toolkit for building Web APIs
Djongo	Extension of Django for easier database configuration and querying
Django-cors-headers	Handles server headers required for Cross-Origin Resource Sharing (CORS)
Django-Rest-Knox	A token based authentication system which provides easy to use authentication for Django REST Framework

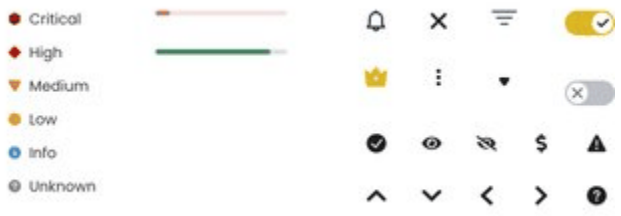
Design & Architecture

- Flowchart
- Style Guide
- Figma Prototype v.1
- Deployment Diagram
- Database Model Diagram
- REST API Routes
- Authentication API Specification

Flowchart



Style Guide



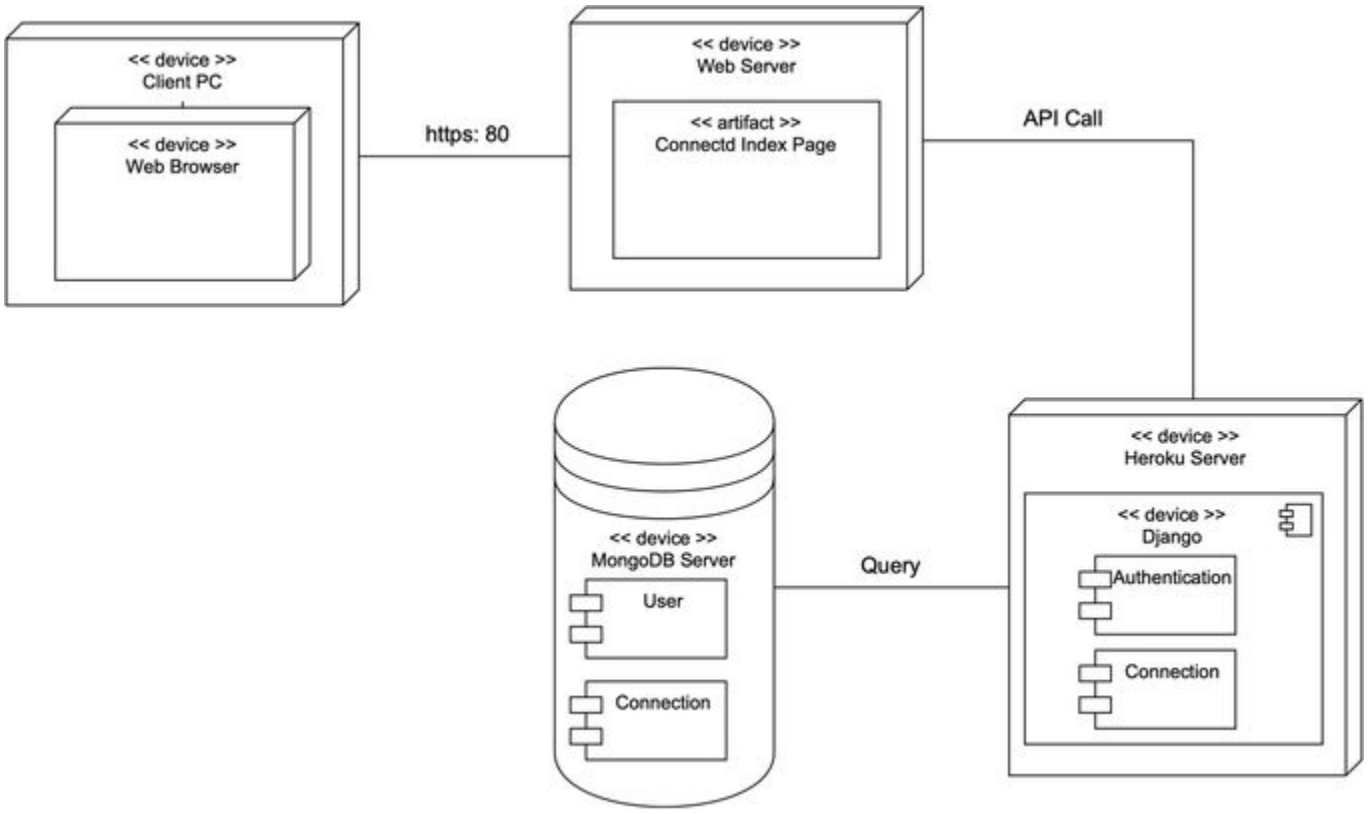
Figma Prototype v.1

PCRM-annotated.pdf

Figma Prototype annotated

https://www.figma.com/file/3kh2FcVcRMxlPcicvkyS4M/PCRM?node-id=0%3A1 (read-only link)

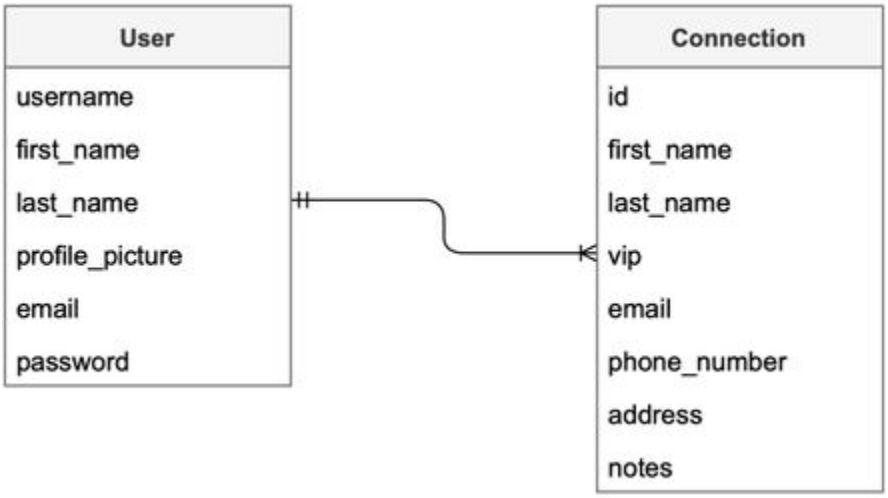
Deployment Diagram

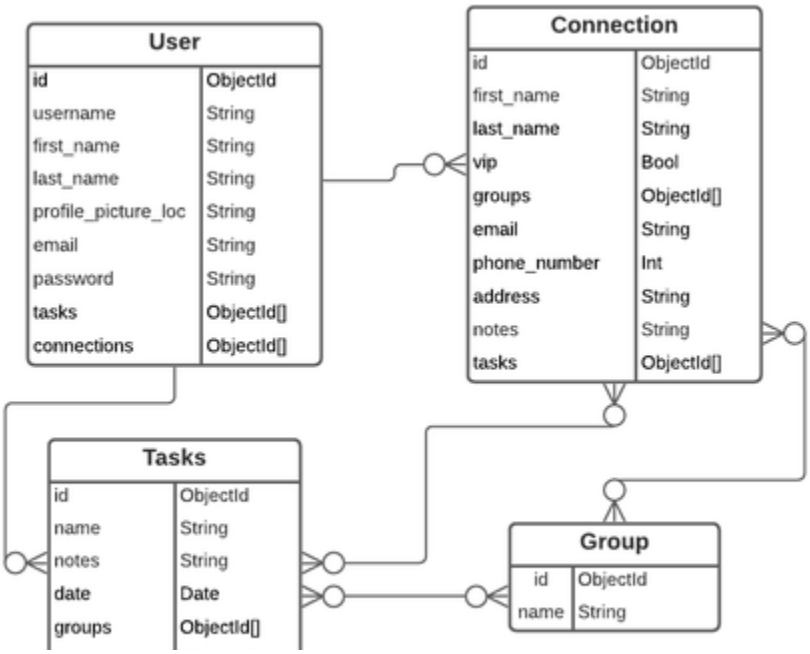


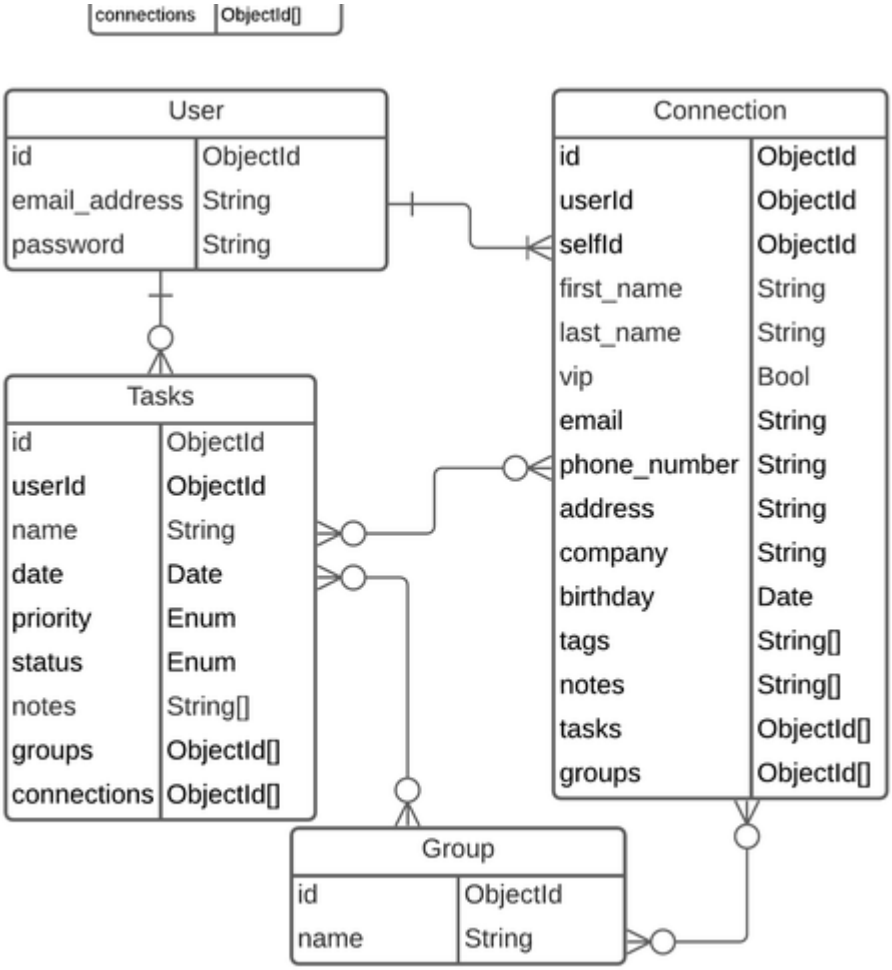
Deployment Diagram.drawio

.drawio file available for later edit

Database Model Diagram







⚠ MongoDB is a NoSQL database which doesn't have a schema like other relational databases.

The diagrams above are just a high level representation of data model relationship.

https://lucid.app/lucidchart/invitations/accept/inv_59a1ddf6-ff06-4133-bb79-282cb411390a?viewport_loc=-486%2C-165%2C2140%2C947%2C0_0

ℹ .drawio file available for later edit

REST API Routes

This is to act as a rough guide for development of the backend API, and to be updated as developed and act as documentation of endpoints for front end to reference.

All endpoints preceded by: BACKEND_URL/api/

Route	Endpoint	HTTP Verb	Description
Connections	connections/	POST	Posts new connection with relevant JSON as body.
	connections/	GET	Gets lists of all connections.
	connections/:id	GET	Returns specific connection.
	connections/?userId=	GET	Get's list of connections based on what user they belong to.
	connections/:id	PATCH	Updates info of specific connection. (includes JSON)
	connections/:id	DELETE	Removes specific connection.
	connections/?userId=	DELETE	Removes all connections belong to this user
Task	tasks/	POST	Posts new task with relevant JSON as body.
	tasks/	GET	Gets list of all tasks.
	tasks/?userId=	GET	Gets list of all user's tasks.
	tasks/:id	GET	Gets specific task.
	tasks/:connectionId=	GET	Gets list of all connection's tasks.
	tasks/:id	PATCH	Updates info of specific task. (includes JSON as body)
	tasks/:id	DELETE	Removes specific task.
	tasks/?userId=	DELETE	Deletes all of a user's tasks.
Groups	groups/	POST	Creates a new group with relevant JSON as body.
	groups/	GET	Gets list of all groups.
	groups/:id	GET	Gets specific group. (includes list of all attached tasks/connections)
	groups/?userId=	GET	Get's all groups created by a user.
	groups/:id	PATCH	Update a specific groups info, could include information about the group or objects attached to group (add/remove tasks/connections). (includes JSON as body)
	groups/:id	DELETE	Deletes a specific group.
	groups/?userId=	DELETE	Deletes all of a user's groups.

Authentication API Specification

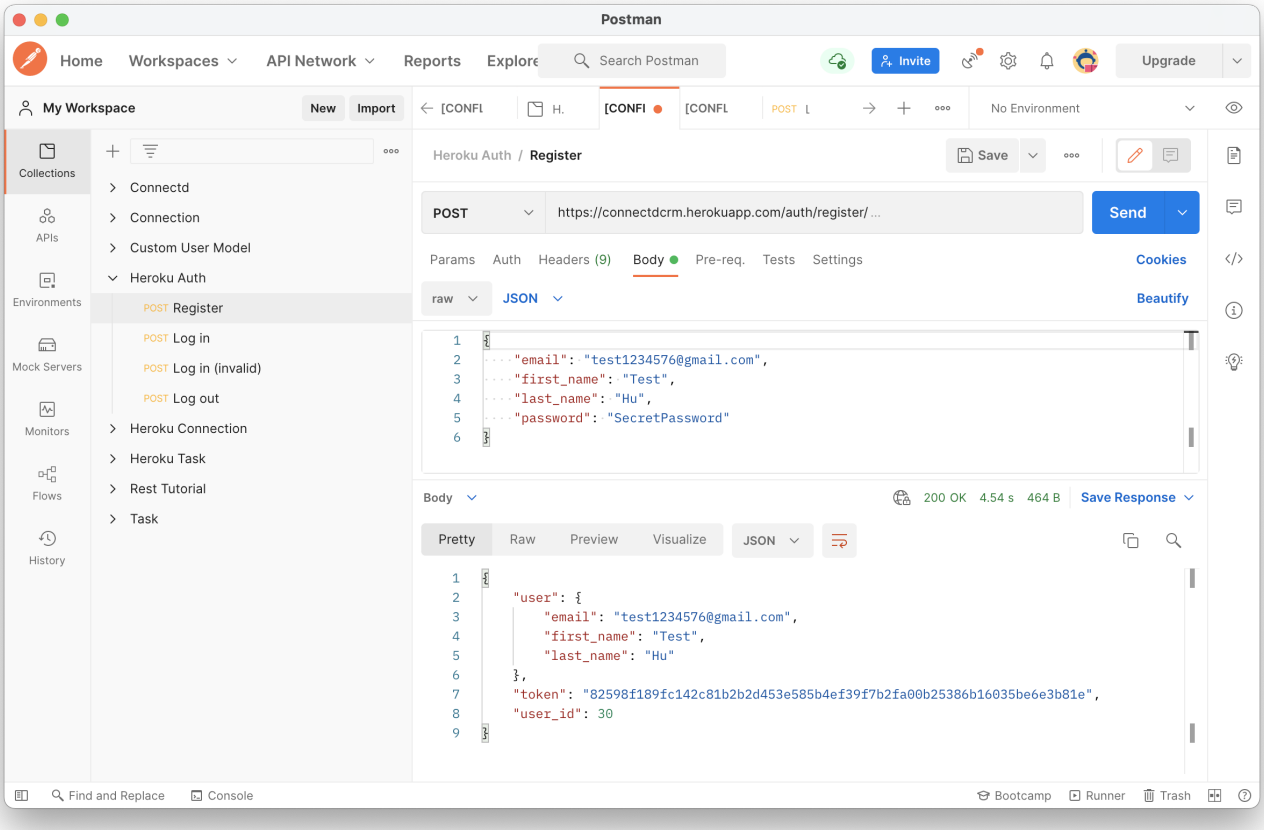
Implement API for account authentication based on ***Django REST*** and ***Knox authentication*** frameworks.

- We initially built an authentication system based on Django built-in auth package using some handy forms provided by Django
- Those built-in forms did well in terms of simplicity and automatic input validation, but without API endpoints we then realised that wasn't compatible with React.
- Then we decided to switch the package and start from scratch again.
- After exploring several alternatives, ***Knox authentication*** was chosen as the new authentication package since it's simple and easy to use while also suitable for our small project.

Endpoint	HTTP Verb	Description
auth/register/	POST	Register a new user with email and name
auth/login/	POST	Validate and log in the user
auth/logout/	POST	Log out user with verification

Register

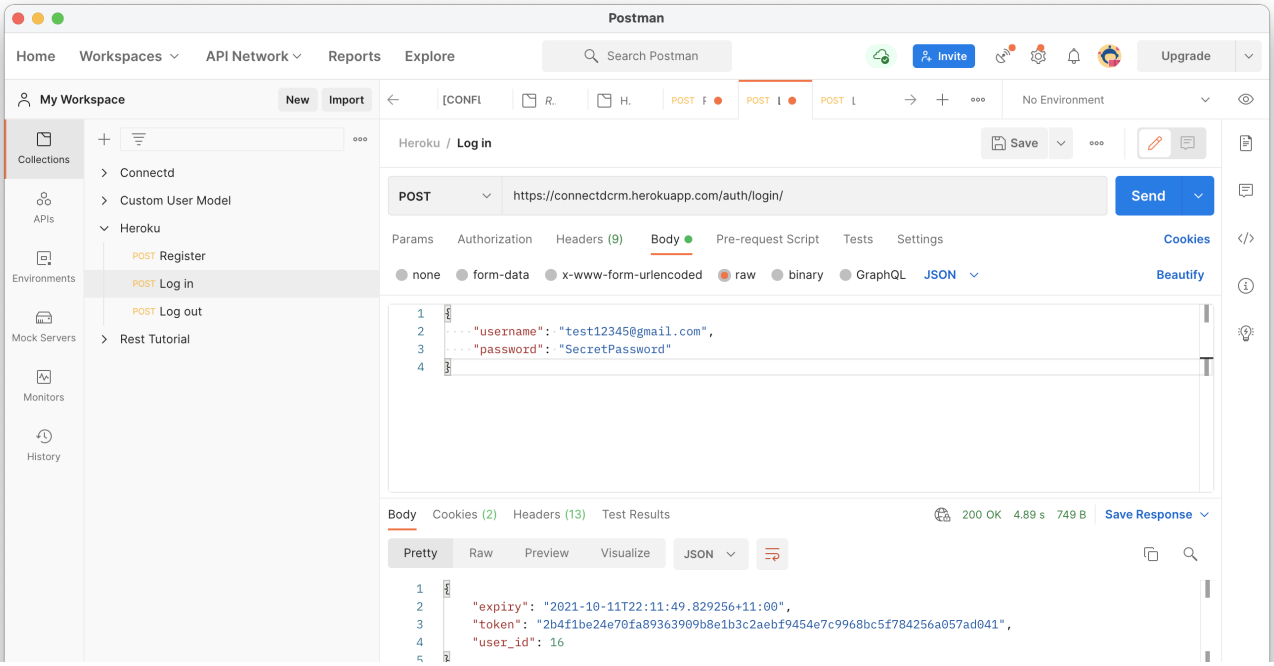
- Request contains `email` (used as unique username), `first_name`, `last_name`, `password`
- Successful user registration will respond with provided user info (including `user_id`) and a valid `token`

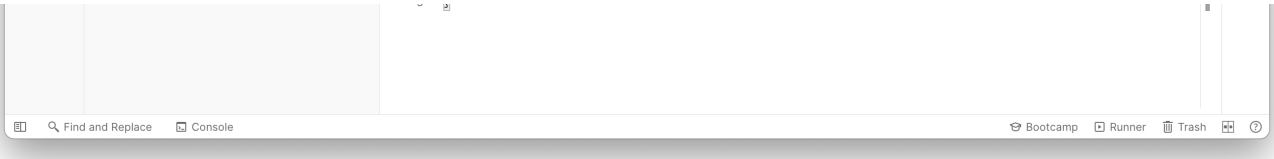


Log in

- Request contains `username` and `password` trying to verify the user
- Successful login will return an `expiry` timestamp, a valid `token` and corresponding `user_id`

⚠️ Notice that `username` is being used here rather than `email` as specified by *Knox*





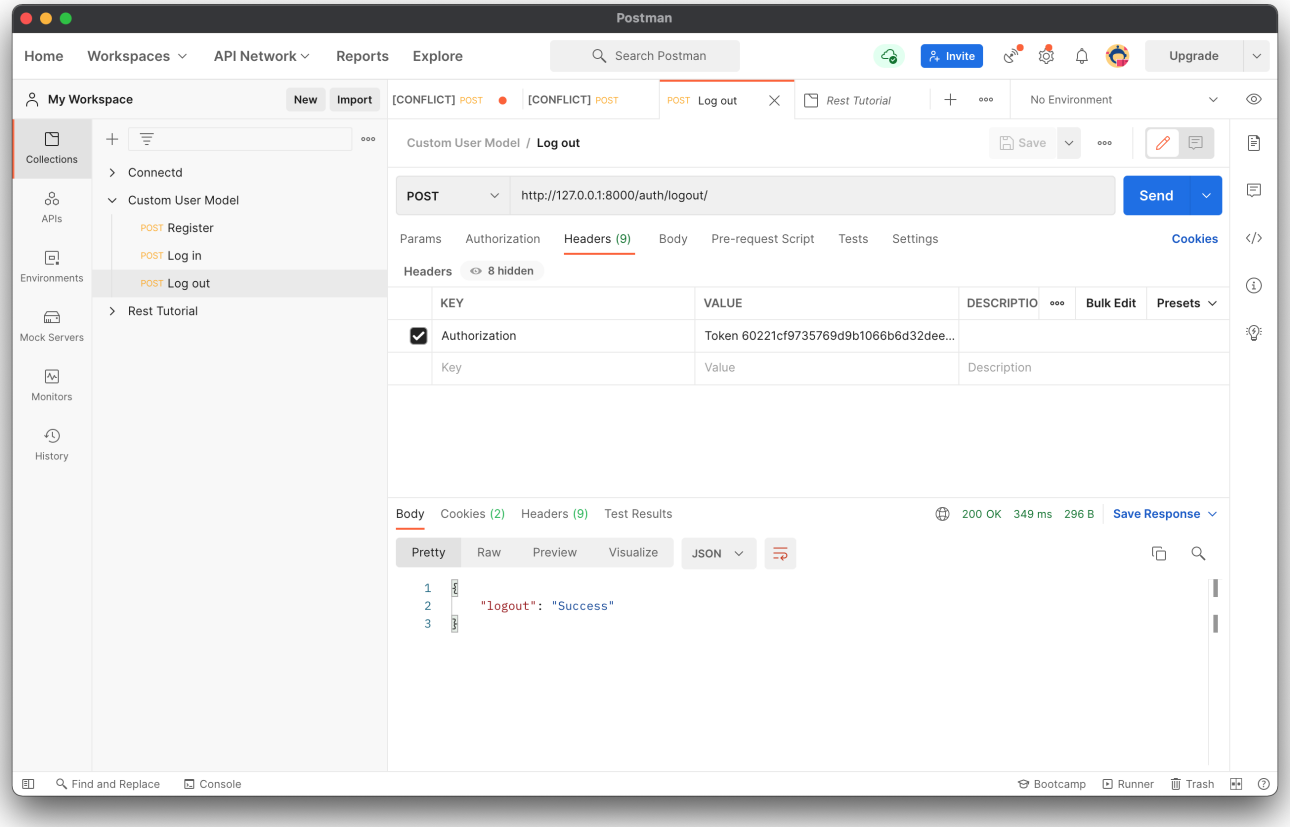
Log out

- Request should have a `Authorization` header with a valid `token` as value, which is obtained after successful login
- Token value is followed by a `Token` prefix such as `Token 60221cf9735769d9b1066b6d32...`
- Successful logout will get a confirmation message like `"logout": "Success"`

⚠ Notice the HTTP `Authorization` header & `Token` prefix

⚠ Any API requests initiated by the user are required to include this HTTP `Authorization` header with valid `token` to retrieve / add / delete data records

✅ Has been advised to add a descriptive response message after successfully log out



Django Admin Panel

- Used a custom user admin to adapt the custom user model
- The custom user model now use `email` as the unique user identifier (previously it's `username`)
- Also includes user's `first_name` and `last_name`
- Admin Panel now works well like the original built-in user management
- Can *add*, *view*, *delete* particular user using Admin Panel features

EMAIL ADDRESS	FIRST NAME	LAST NAME	STAFF STATUS
jackson@gmail.com	Jackson	Hu	✗
renweih@student.unimelb.edu.au	Jackson	Hu	✓
test213124@gmail.com	Jackson	Test	✗
test@gmail.com	Test	Last	✗

4 users

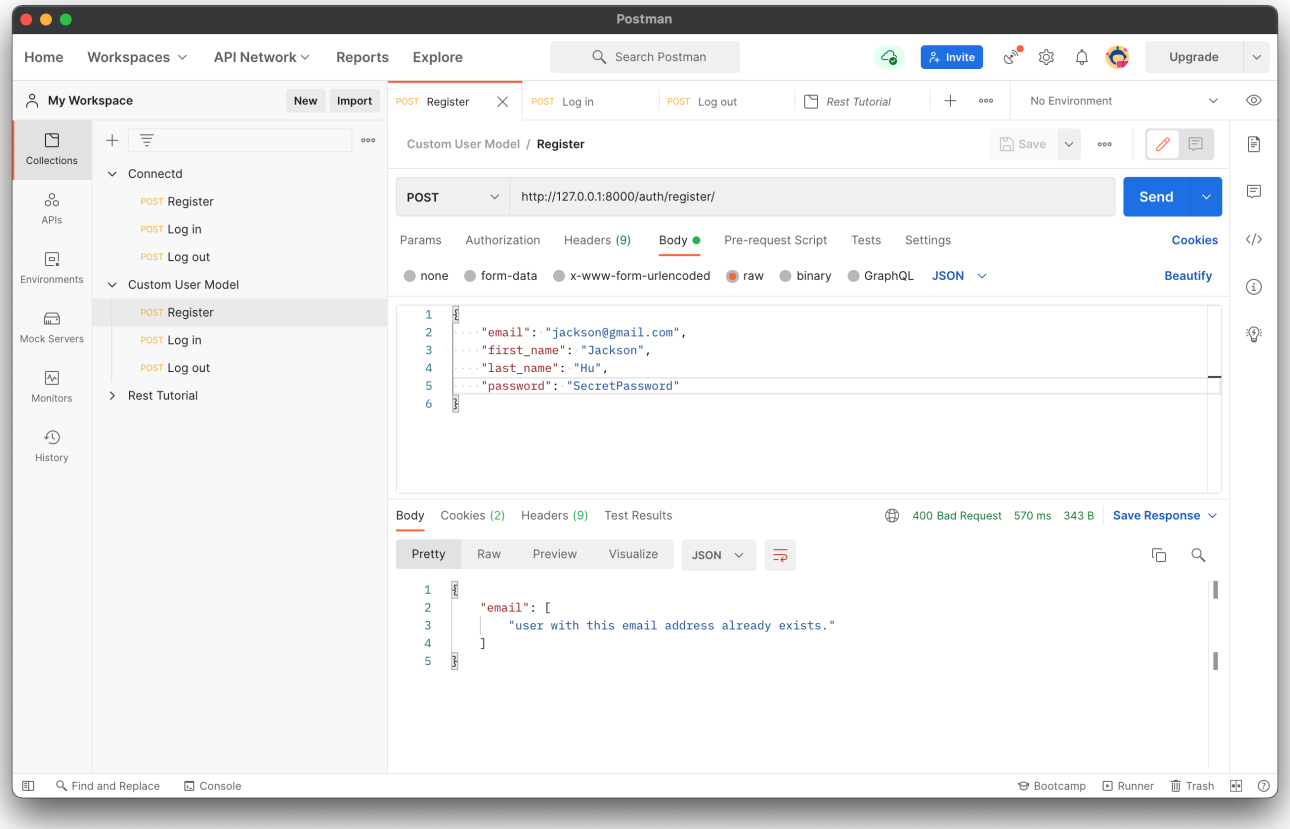
Notice

- All request and response are in the form of **JSON** file, except logout request
- Login request use `username` rather than `email`, even though `email` is used as unique identifier
- Logout and other API requests related to users' data records are required to add an HTTP `Authorization` header with a valid `token`
- Logout token value should start with a `Token` prefix

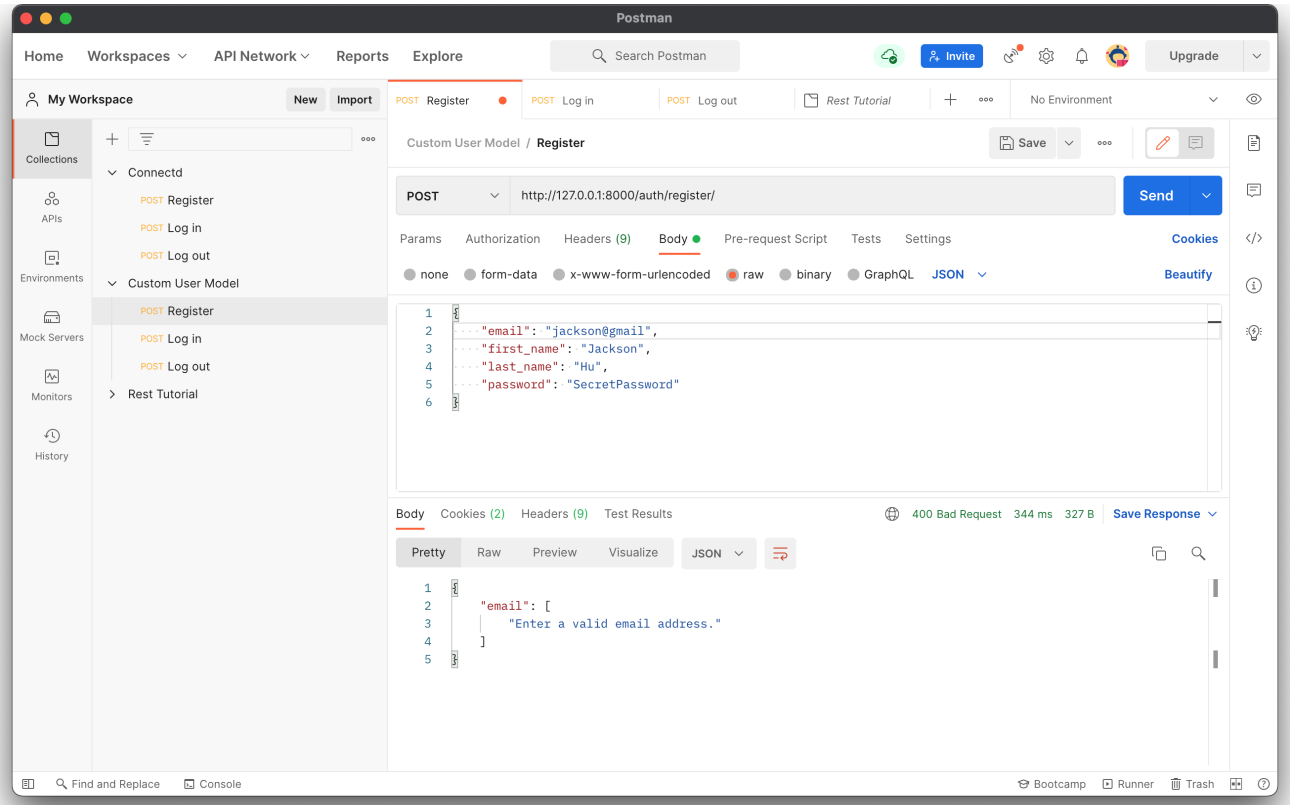
Invalid Request Examples

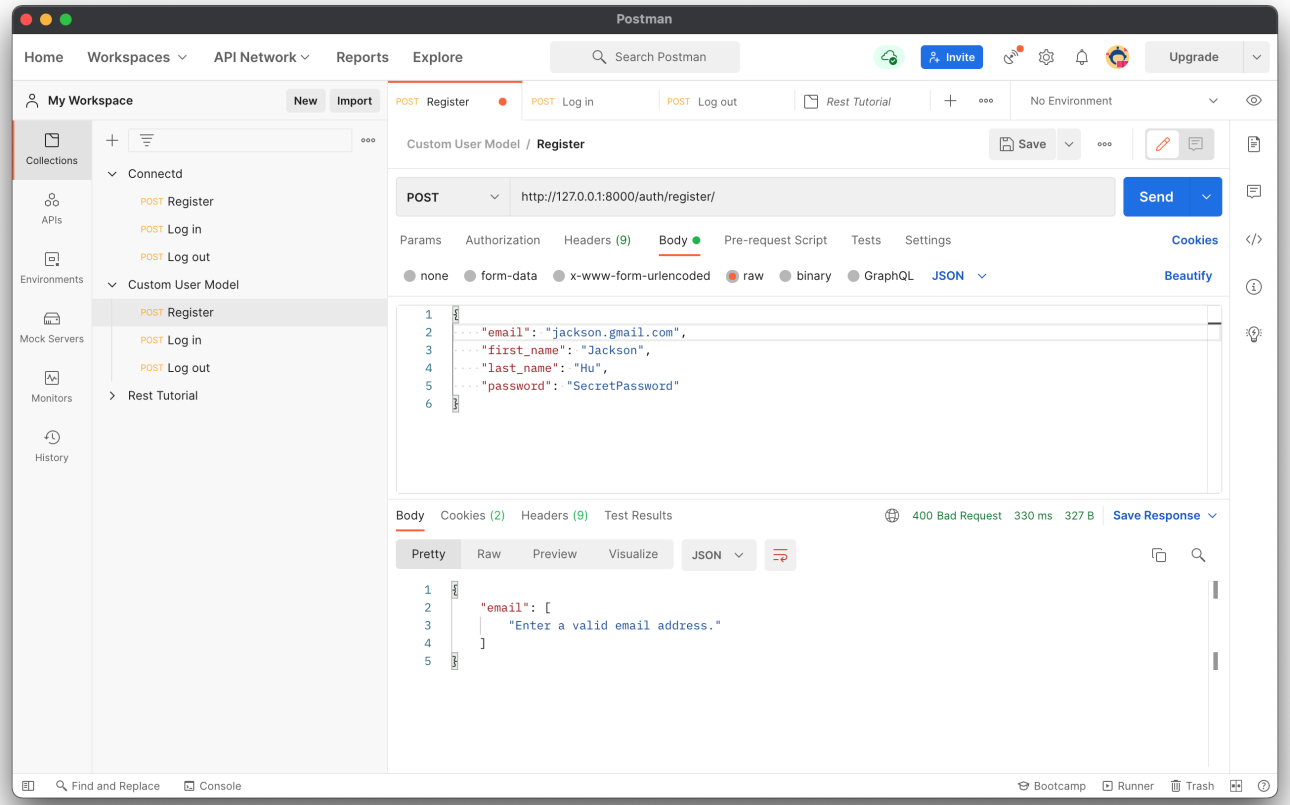
⚠ Invalid requests without specified attributes or using wrong formats will get responses including an error message

1. Email already exist

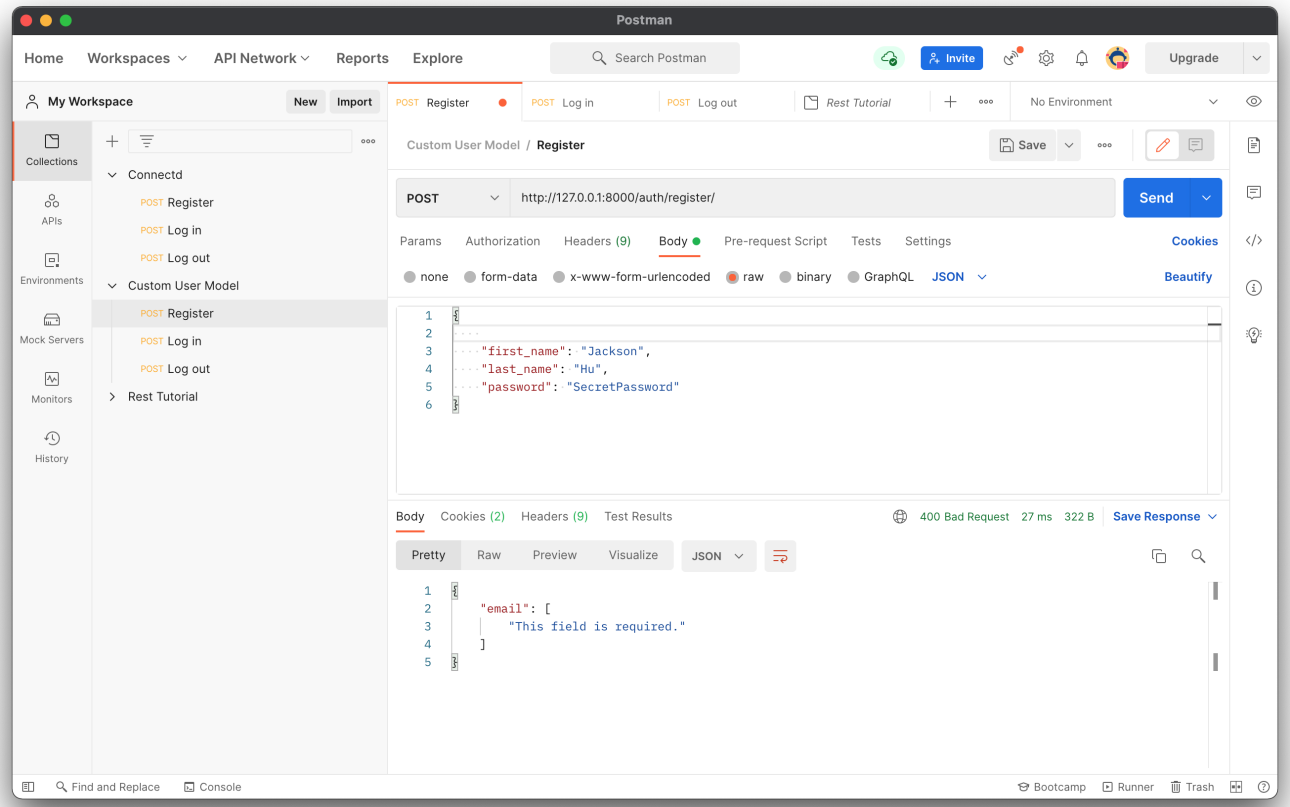


2. Email format not valid

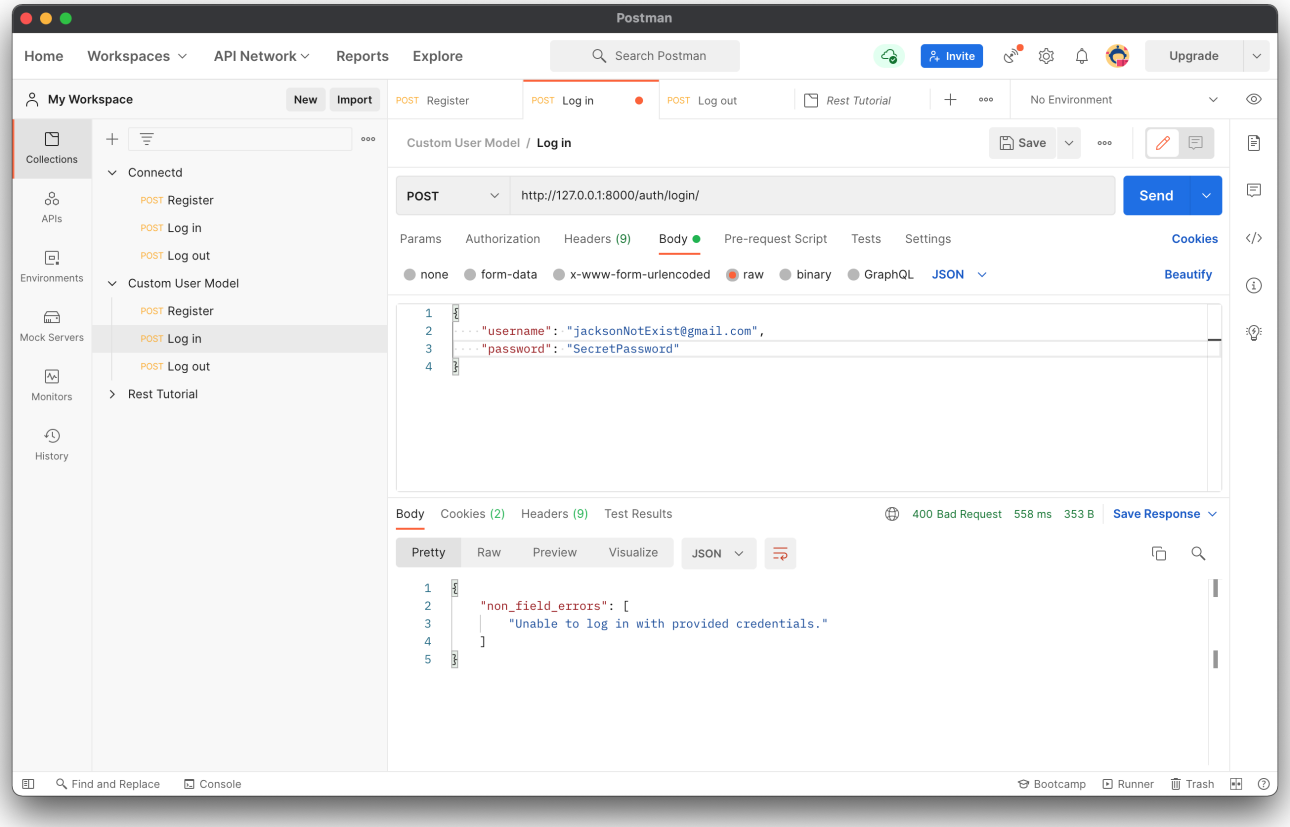




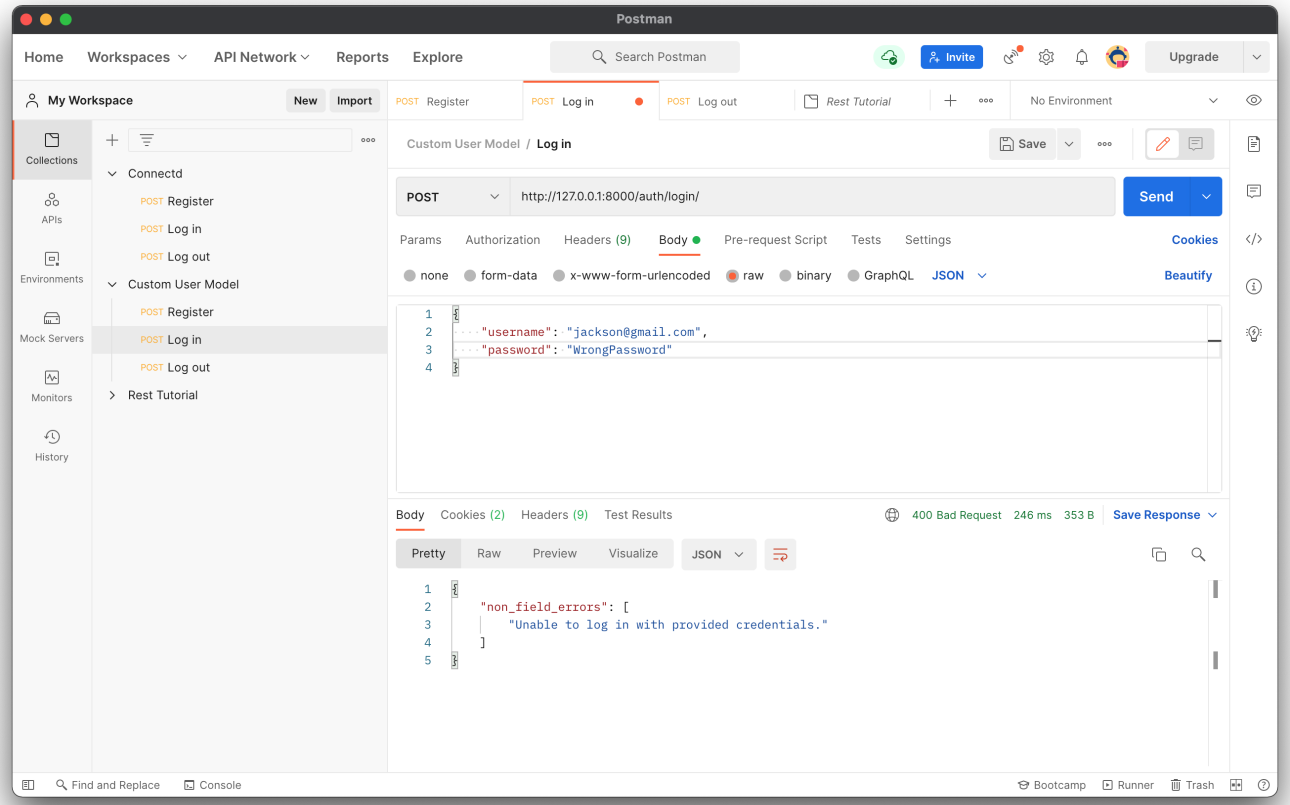
3. Register with missing email field



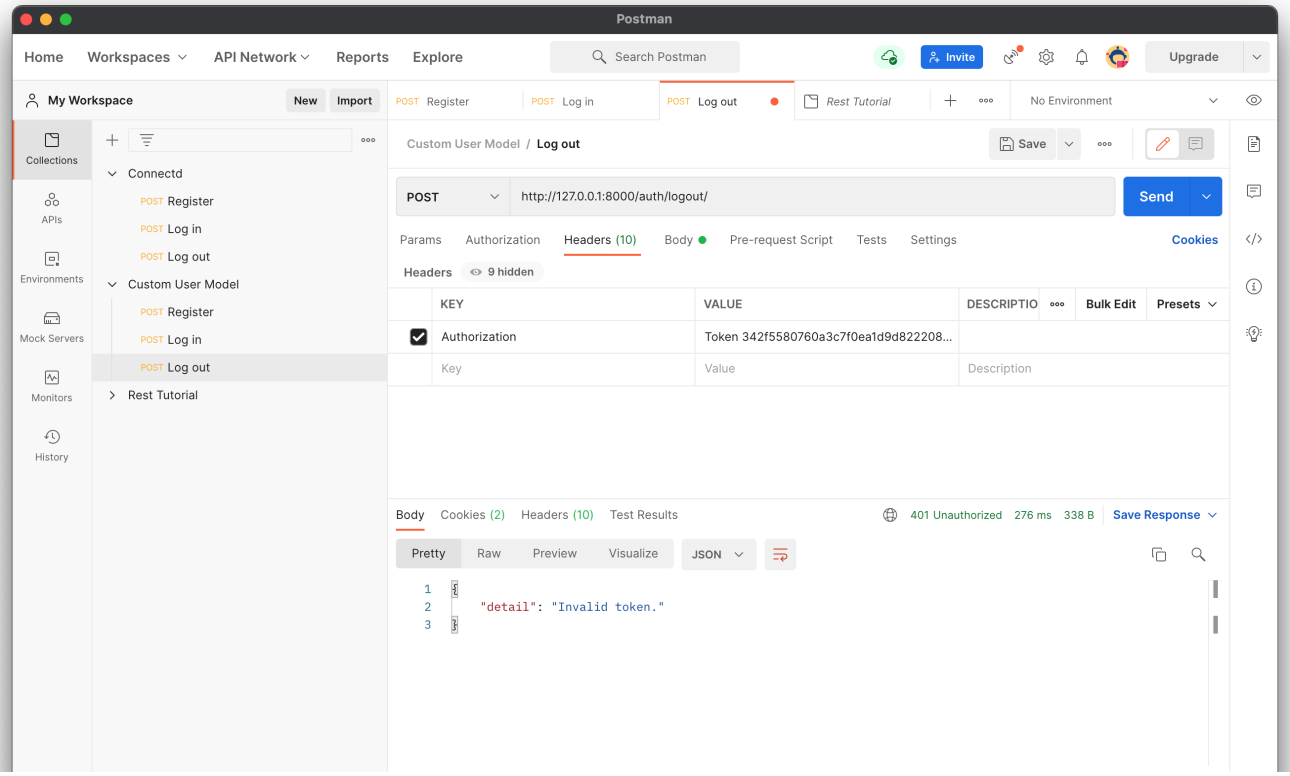
4. Log in with email not exist



5. Log in with wrong password



6. Log out with invalid token



Sprints

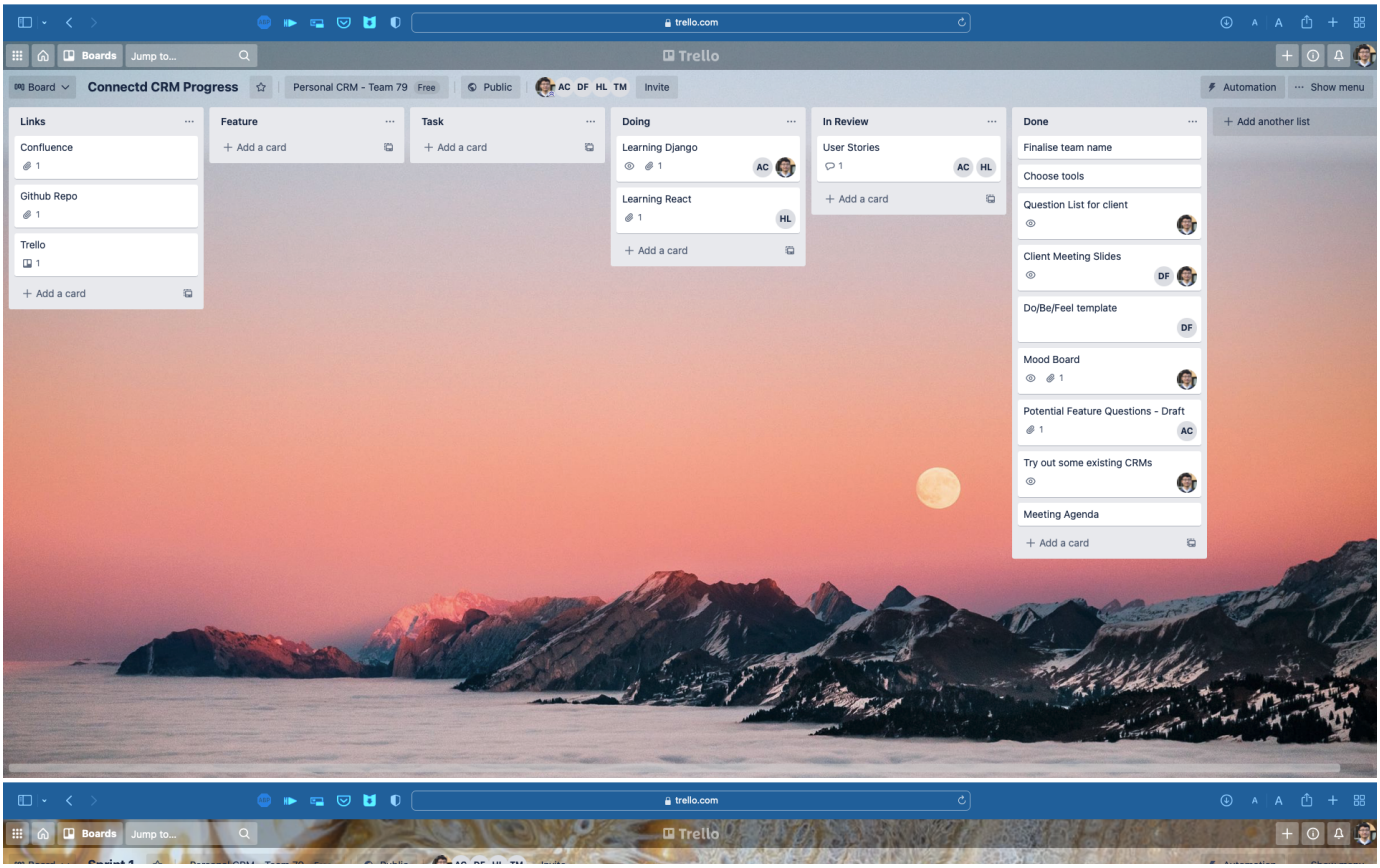
- Sprint 1
- Sprint 2
- Sprint 3

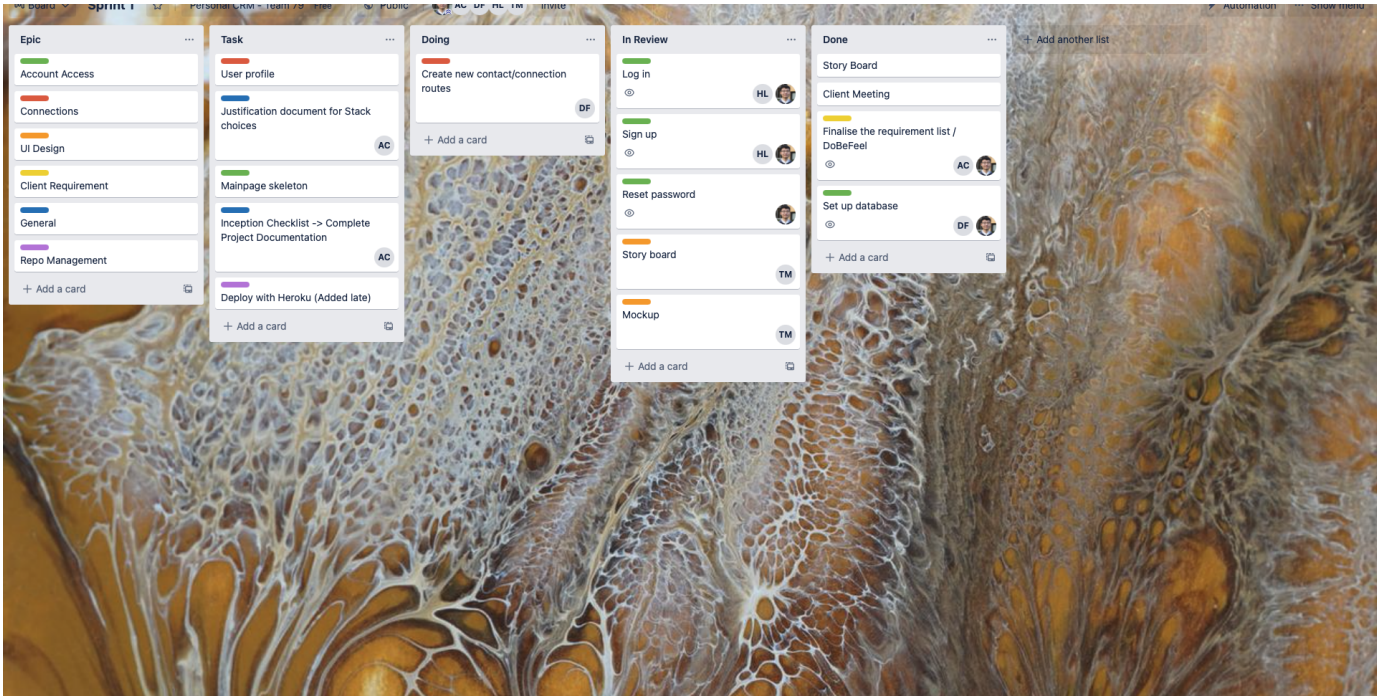
Sprint 1

Checklist

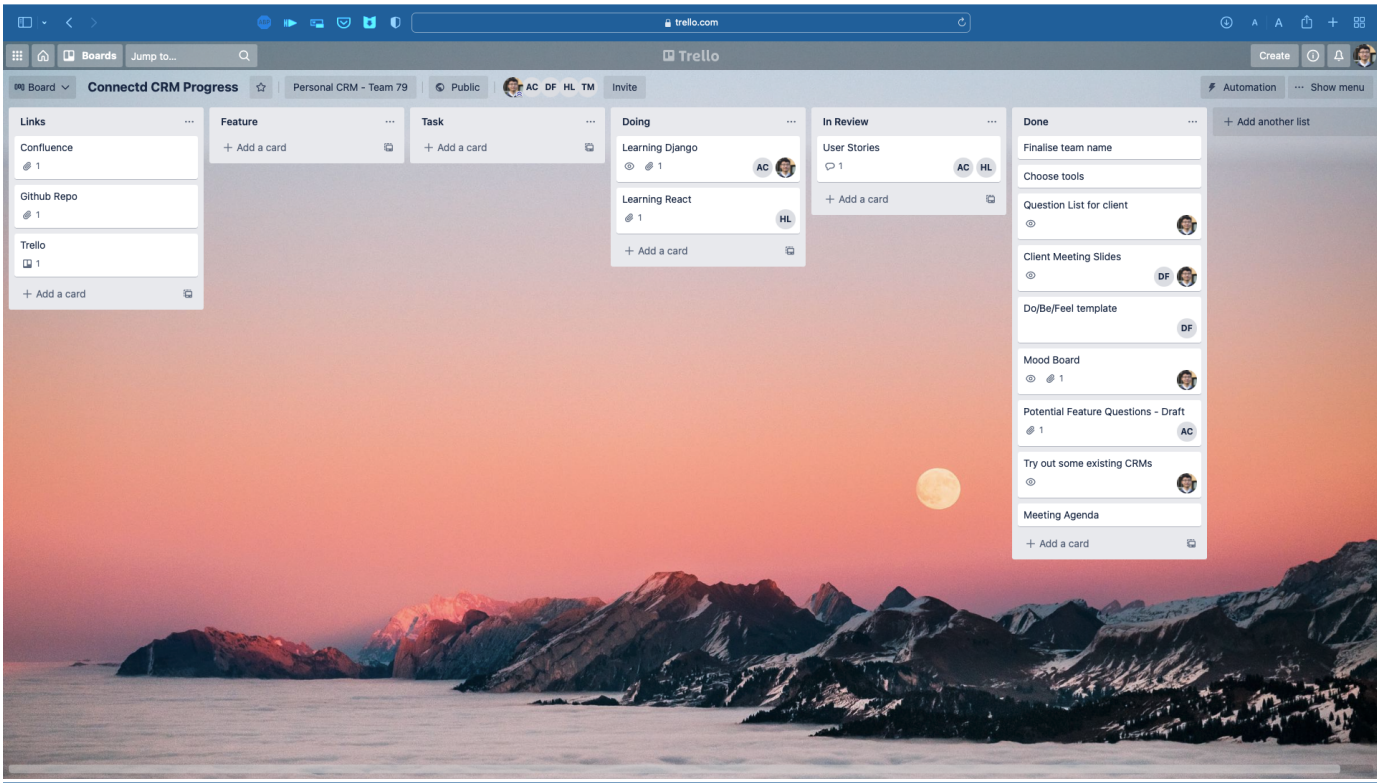
Sprint 1 Trello

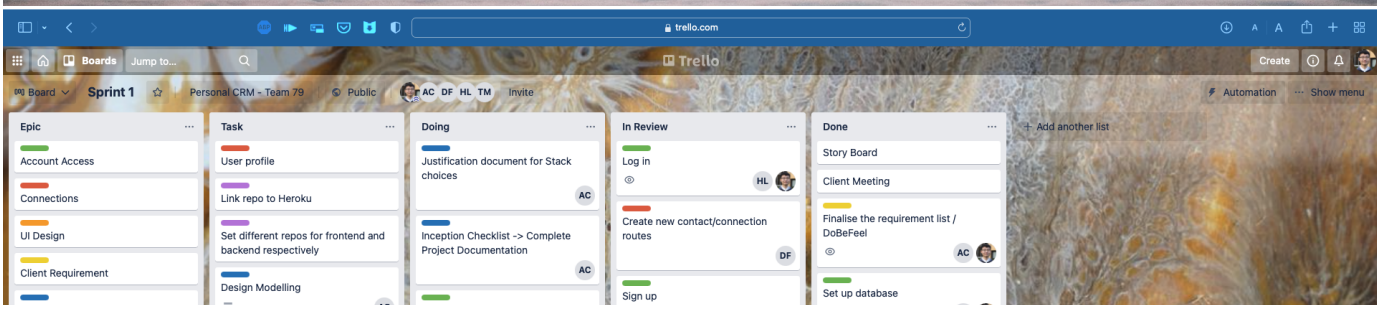
Week 4 Trello Status

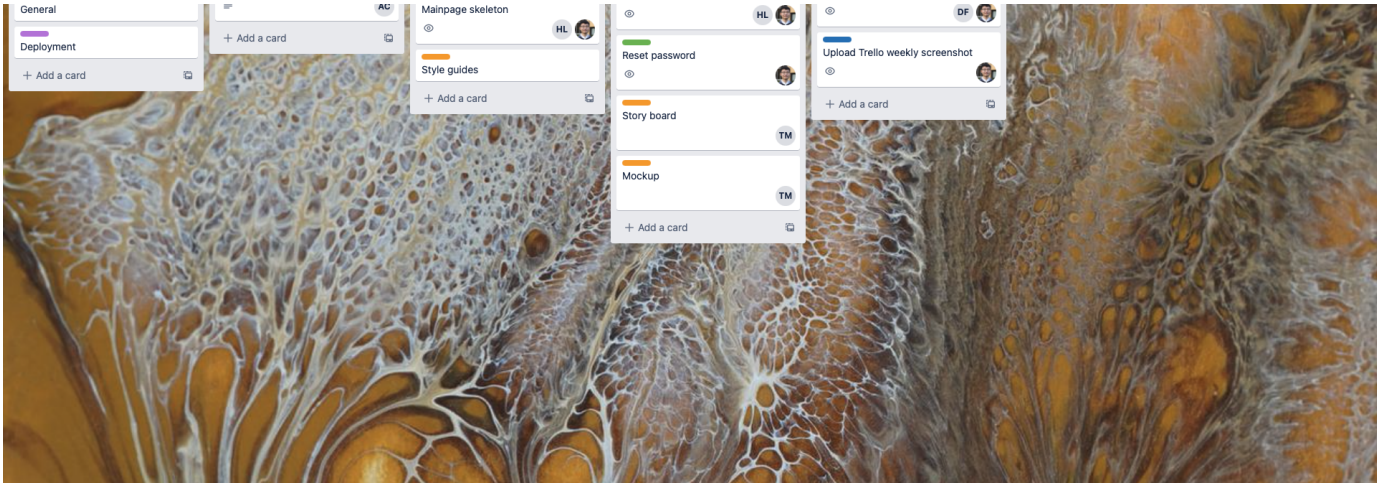




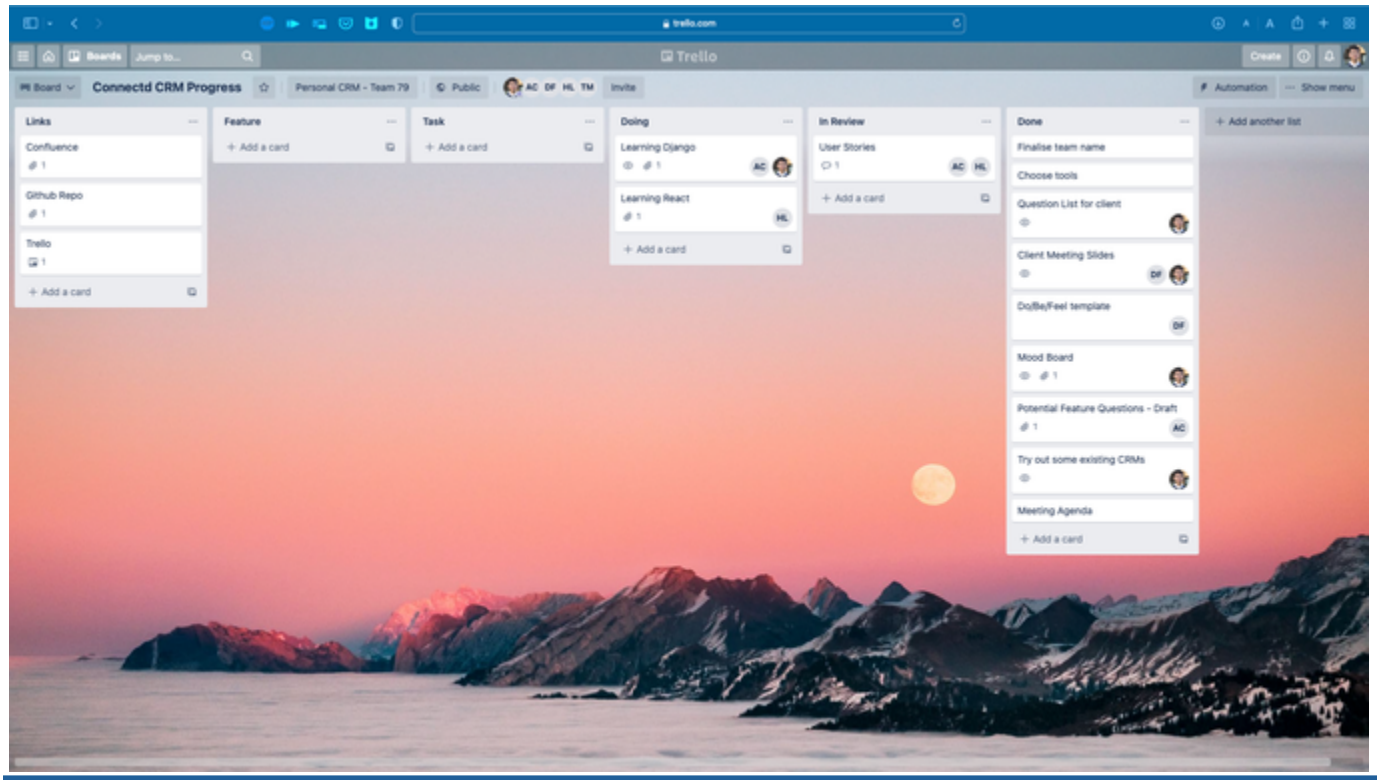
Week 5 Trello Status

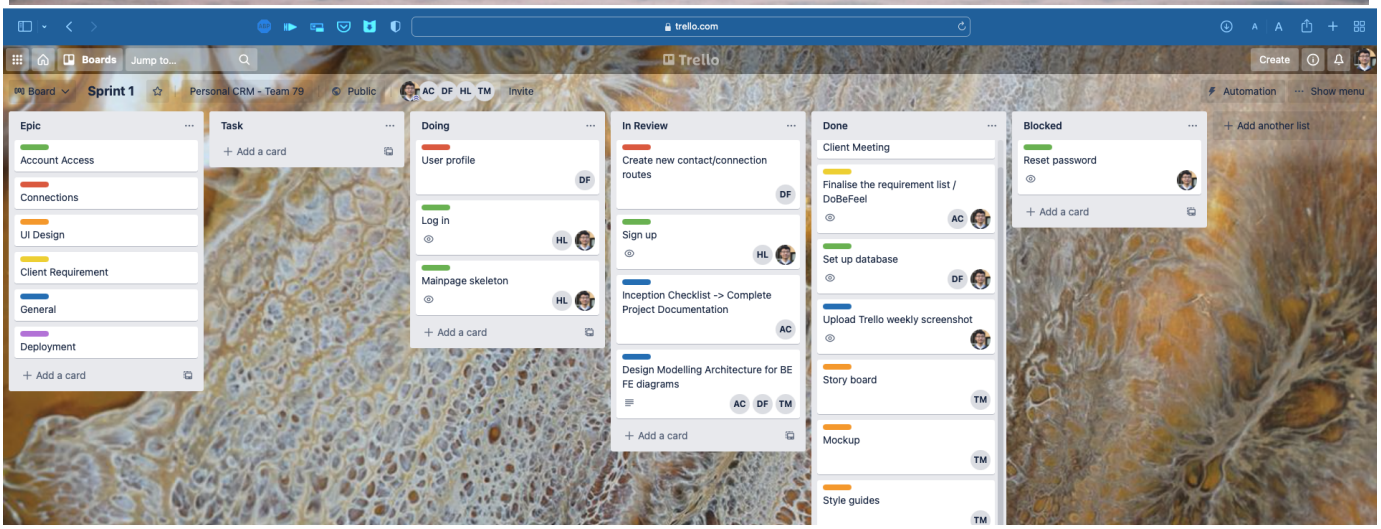






Week 6 Trello Status





Sprint 1 Planning

✅ Sprint planning checklist

Preparation	Meeting	Follow up
☑ Organise the backlog ☑ Understand the client requirements ☑ Try some existing CRMs An Example CRM - Circles (by ZooWho)	☑ Assign the team roles ☑ Confirm the epics	☑ Update Trello board ☑ Document progress on Confluence

👥 Sprint team members

Name	Role
@ David Fletcher	Product Owner
@ Alexander Cain	Scrum Master
@ Han Liu	Frontend Lead
@ Jackson Hu	Backend Lead
@ Tymara Metcalf	UI / UX Designer

✏️ Objectives

Overall objectives supposed to be achieved by the end of sprint 1

1. Get used to Agile software development approach
2. Come up with a skeleton prototype
3. Set up tools and development environments
4. Complete basic features required by the client
    a. Account access
    b. Connection management
5. Confirm UI design and present mockups

📄 Sprint 1 Backlog

Epic	Task	Sub Task	Story Point (1-10)	Participant	Priority	Status
General	Documentation Update	Justification for Stack choices	2 pt	@ Jackson Hu @ Han Liu	LOW	DONE
		Weekly Trello screenshot	1 pt	@ Jackson Hu	LOW	DONE
Client Requirement	Finalise Requirement	Do/Be/Feel list	2 pt	@ Jackson Hu @ Alexander Cain	HIGH	DONE
		User stories	3 pt	@ Jackson Hu @ Alexander Cain	HIGH	DONE
		Motivational Model	2 pt	@ Tymara Metcalf	MEDIUM	DONE
Account Access	Sign up	Sign up page	5 pt	@ Han Liu @ Jackson Hu	HIGH	IN PROGRESS
		Sign up API	4 pt	@ Jackson Hu	HIGH	DONE
	Log in	Log in page	5 pt	@ Han Liu @ Jackson Hu	HIGH	IN PROGRESS
		Log in API	4 pt	@ Jackson Hu	HIGH	DONE
	Reset password	Reset page	6 pt	@ Jackson Hu	OPTIONAL	
		Reset API	5 pt	@ Jackson Hu	OPTIONAL	DONE
		AWS email service	7 pt	@ Jackson Hu	OPTIONAL	
	Database model	Set up the Mongodb	2 pt	@ Jackson Hu @ David Fletcher	HIGH	DONE
		Link to Mongodb	1 pt	@ David Fletcher @ Jackson Hu	HIGH	DONE
Deployment	Set up Heroku	Initiate Heroku app	2 pt	@ Tymara Metcalf	HIGH	DONE
		Set different repositories	4 pt	@ Han Liu	HIGH	DONE
	Deploy on Heroku	Link to Heroku	6 pt		HIGH	
Connections	Create new contacts			@ David Fletcher	HIGH	IN PROGRESS
	User profile				MEDIUM	
UI / UX Design	Mockup		2 pt	@ Tymara Metcalf	HIGH	DONE
	Storyboard	Flow chart	2 pt	@ Tymara Metcalf	HIGH	DONE
	Figma Prototype		7 pt	@ Tymara Metcalf	HIGH	DONE

Potential risks

Risk	Mitigation
Not familiar with development tools yet	Find tutorials and help each other when problems raised
No idea how to integrate backend with frontend	Keep in touch and try to find tutorials as a guidance

Sprint planning resources

Sprint boards and retrospectives

Team resources and definitions

Sprint 1 Retrospective

An internal reflection regarding the processes that the team has followed over the past sprint.

Note: a process is how you did something; not what you did.

E.g. The team created meeting agendas before each meeting.
The team split into subgroups for pair programming.

Sample Questions:

- How did we manage meeting minutes? How did we prepare for client, supervisor, and team meetings
  - Positives:
    1. We prepared an agenda before meetings.
    2. We record minutes during meetings.
    3. We keep meeting productive and don't meet unnecessarily.
  - Negatives:
    1. It would help to have defined roles during meetings. e.g. Minute taking.
  - Improve on:
    1. Have a leader, assistant, and minute-taker defined for meetings beforehand.
    2. Rotate roles for each sprint (or check-in with each member and see if they'd like to add something different that day)
- How did we keep track of tasks between members? How did we meet deadlines?
  - Positives:
    1. We have a detailed sprint backlog (Trello), where we record each task for the sprint.
    2. During our meetings we cataloged to-dos for the week via confluence.
    3. During our weekly stand-ups we could talk about how each member's progress was going, and if anyone needed help in certain places.
    4. We keep Trello tasks labeled so we know which category each task belongs to.
  - Negatives:
    1. Keeping track of what was actually done.
    2. Too many places where tasks are listed for check-off.
    3. Trello was not always updated so it was hard to tell where tasks were in the kanban (doing, in review, done, etc.)
  - Improve on:
    1. Each weekly meeting review our tasks, map to-do list to Trello.
    2. View Trello as sprint backlog, Meeting log as daily to-do, and Confluence Sprint backlog as high-level tasks.
- How did we decide on which tools to use? How did we adapt to new technologies?
  - Positives:
    1. Some members had past experience with certain tools, so ease of use was put as a priority.
    2. We did research on other tools that we weren't familiar with to help decide on a suitable stack.
    3. Getting familiar with tools by watching YouTube videos, going through online tutorials.
  - Negatives:
    1. Didn't separate repositories for frontend and backend respectively at the start so that later deployment would be easier.
    2. Took some time to configure the repositories and link to Heroku.
    3. Spent longer time than expected to learn new tools in terms of backend framework.
  - Improve on:
    1. Communicate with other team members to ensure modules are compatible.
    2. Do more research on the packages / modules we're going to use for a better understanding.
    3. Finalising tech stack before development (testing framework, Material UI/Bootstrap)

- How did we adapt to Agile Process? How did we keep organised and develop a workflow that worked for all members?
  - Positives:
    1. Organised well for sprint planning and weekly stand-up meetings.

    - Negatives:
        1. Maybe too ambitious about the goals for the first sprint.
        2. We didn't develop our documentation in parallel with development. (Design modeling before Code)
    - Improve on:
        1. Be more realistic and come up with a more detailed sprint planning.
        2. Develop a structure for GitHub folders

- How did the team recognise when others needed assistance? How did we check in with each other?
    - Positives:
        1. We would have meetings 2-3 times a week, and get an idea of how everyone was tracking.
        2. No one felt overwhelmed and seeking assistance was easy for everyone.
        3. We were able to just message casually on Slack whenever we had questions or had to organise something.
        4. We had pair programming sessions to work together.
    - Negatives:
        1. Didn't structure well in terms of framework directory structure.
    - Improve on:
        1. Share learning resources.
        2. Prepare for next sprints where people may need more assistance programming than this current sprint.

Sprint 1 Review

Sprint Review is expected to be an **external reflection** with the **client** regarding the **product** that we have created thus far

Product Presentation

We did a quick presentation about the things we've done for the sprint 1 which includes:

1. Figma interactive **prototype**
    a. Main pages of CRM
        i. Register / sign up / log in
        ii. Dashboard
        iii. Connections
        iv. Tasks
        v. Profile
    b. Actions corresponding to user stories based on requirements from client
    c. Design style and component layout
2. Some **artefacts** documented on Confluence
    a. Collected Requirements from first client meeting
        i. Do / Be / Feel List
        ii. User Stories
    b. Design & Architecture Diagrams
        i. Flowchart
        ii. Annotated Figma prototype
        iii. Style guide
        iv. Motivational Model
        v. Deployment Diagram
        vi. Database Model Diagram
        vii. REST API Routes

Reflection

Client considers our prototype as excellent overall and really can't point out things which are bad

Good	Bad	Areas of Improvement
Client really loves our Figma prototype and design style, saying it's excellent	Some categories of connection list may not be necessary (e.g. location)	Replace the location with specific address for connections
The progress bar reflecting the due date on task page looks fancy		Would be better to have little icons on the navigation bar
Dashboard captures all the important information		Dropbox from the search to select category
Client accepts our logo		Can also set the start date for the task

Feedback 1

Home/Overview page is missing on your confluence. I have not deducted marks for this but please add it.

Goal Setting (1/1): Team has clear goals set out in their sprint planning

Group Decision Making (1/1): Decisions clearly documented

Distribution of work (1/1): Work is being clearly distributed on Trello

Collaboration (1/1): Team is collaborating well

Team Interaction (1/1): Team has not had any issues regarding communication. Please ensure that if a team member is absent from a team meeting that everyone knows.

Managing Conflict (1/1): Team has handled the addition of a new team member well and appears on track.

Requirements (1/1): Your requirements artefacts were somewhat difficult to find, please clearly label and place each of these within individual Requirements sub-pages. Despite this, requirements were very well documented.

Design (1/1): Very well done and well documented.

Additional Accomplishments (0.5/2): Architecture documentation looks great. Some consideration of testing, clearer confluence layout and a clear link to deployment would have resulted in additional marks.

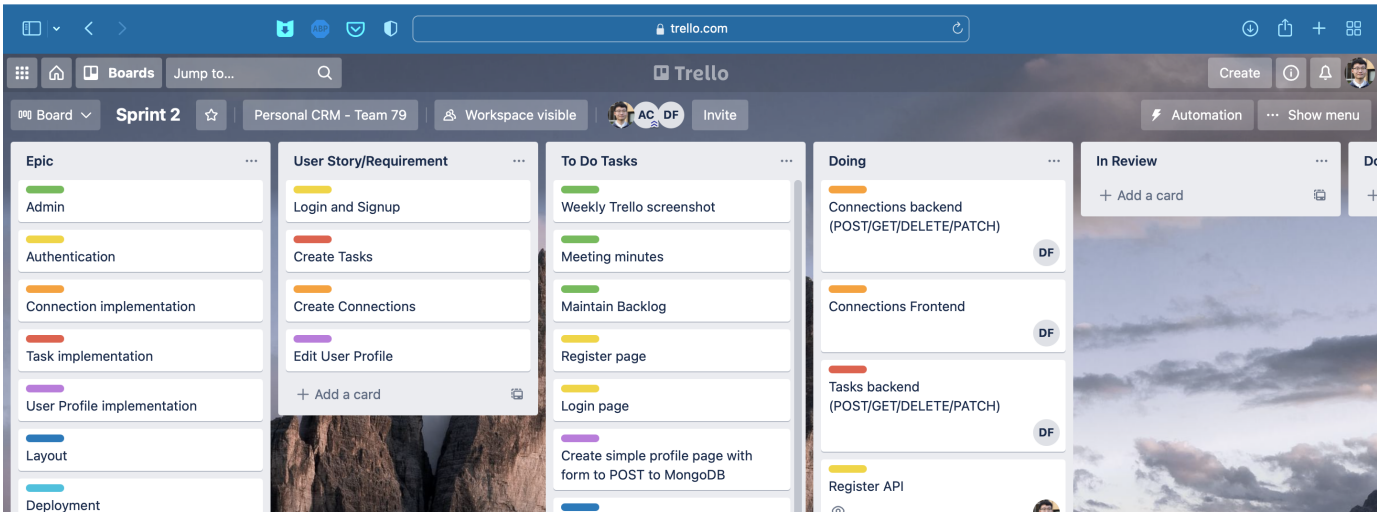
Overall (8.5/10): Awesome work!

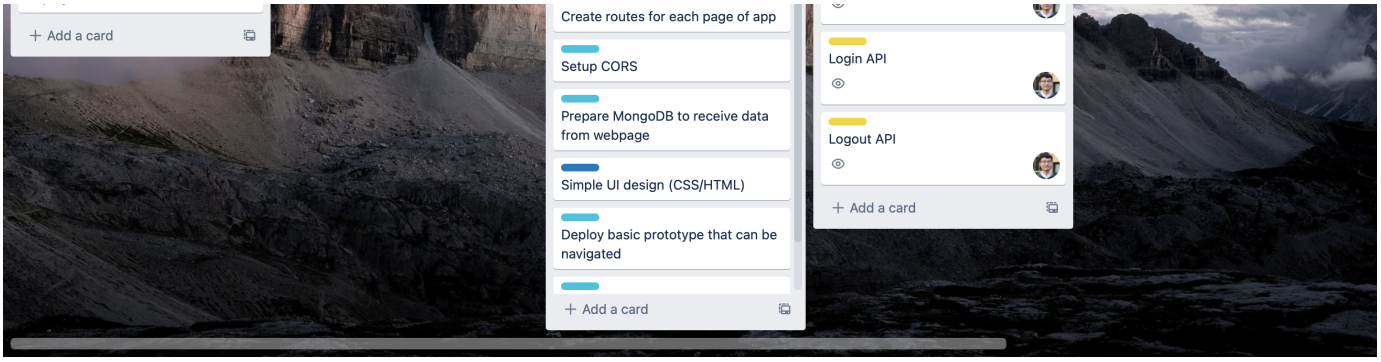
Sprint 2

Sprint 2 Checklist

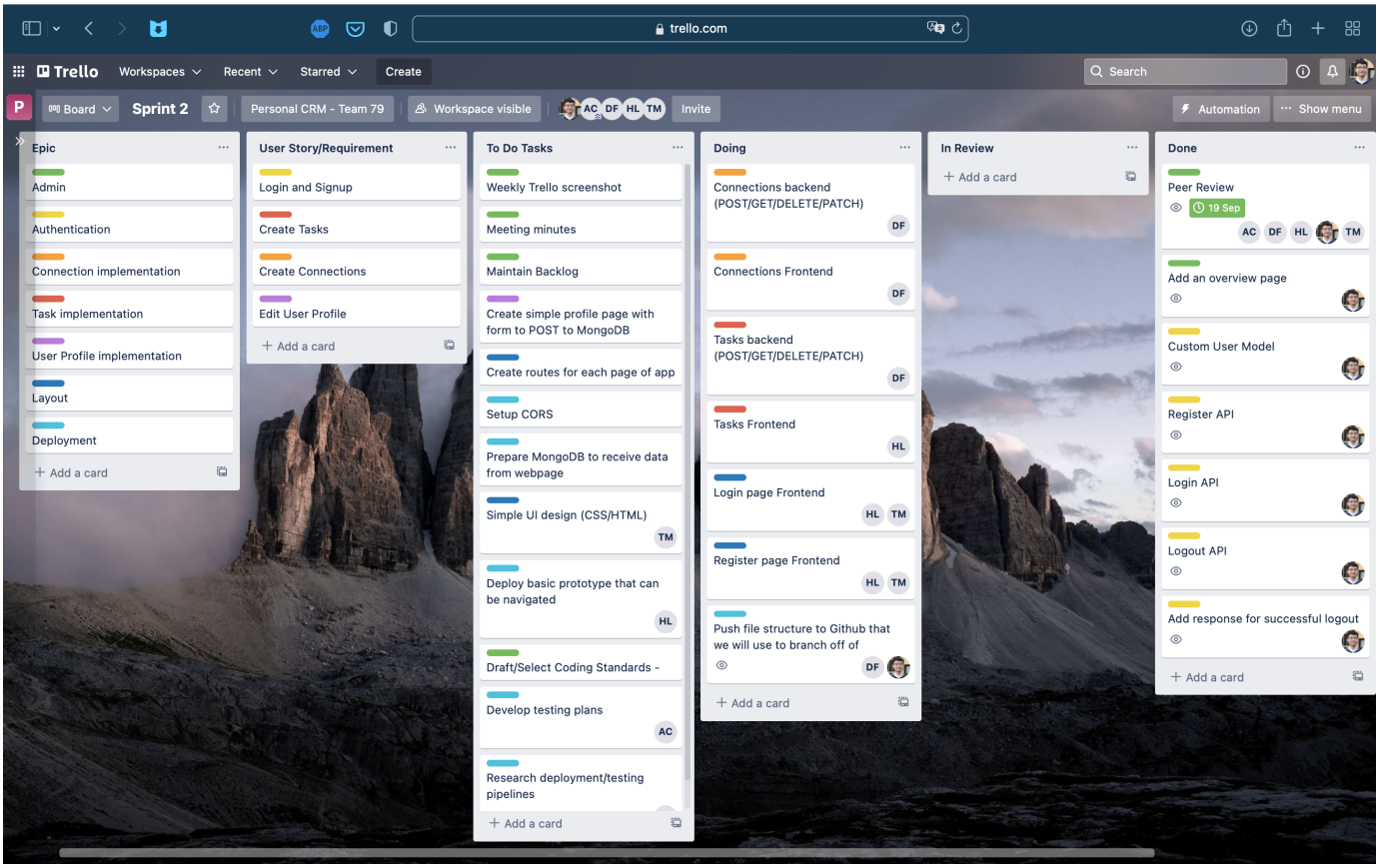
Sprint 2 Trello

Week 7 Trello Status

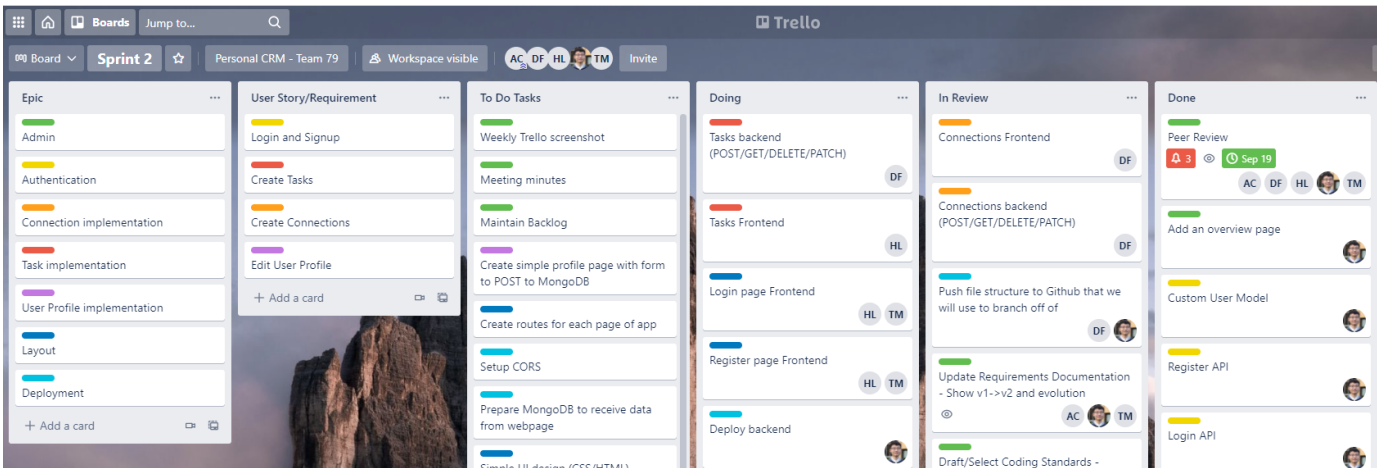


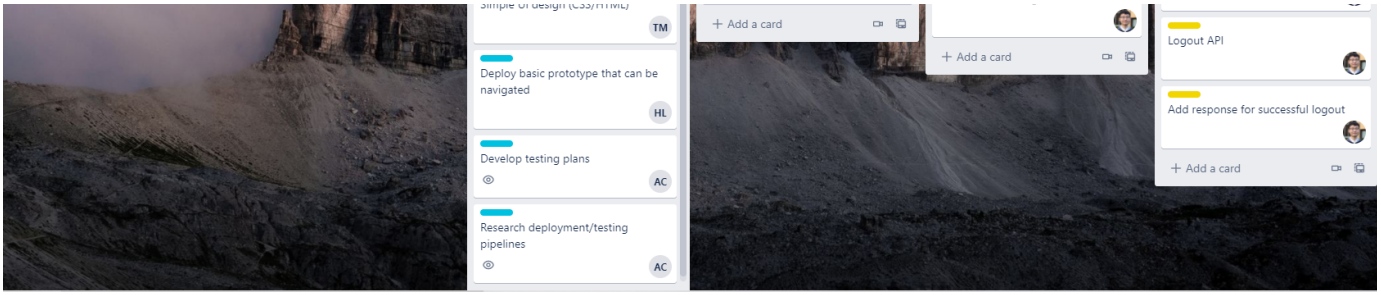


Week 8 Trello Status

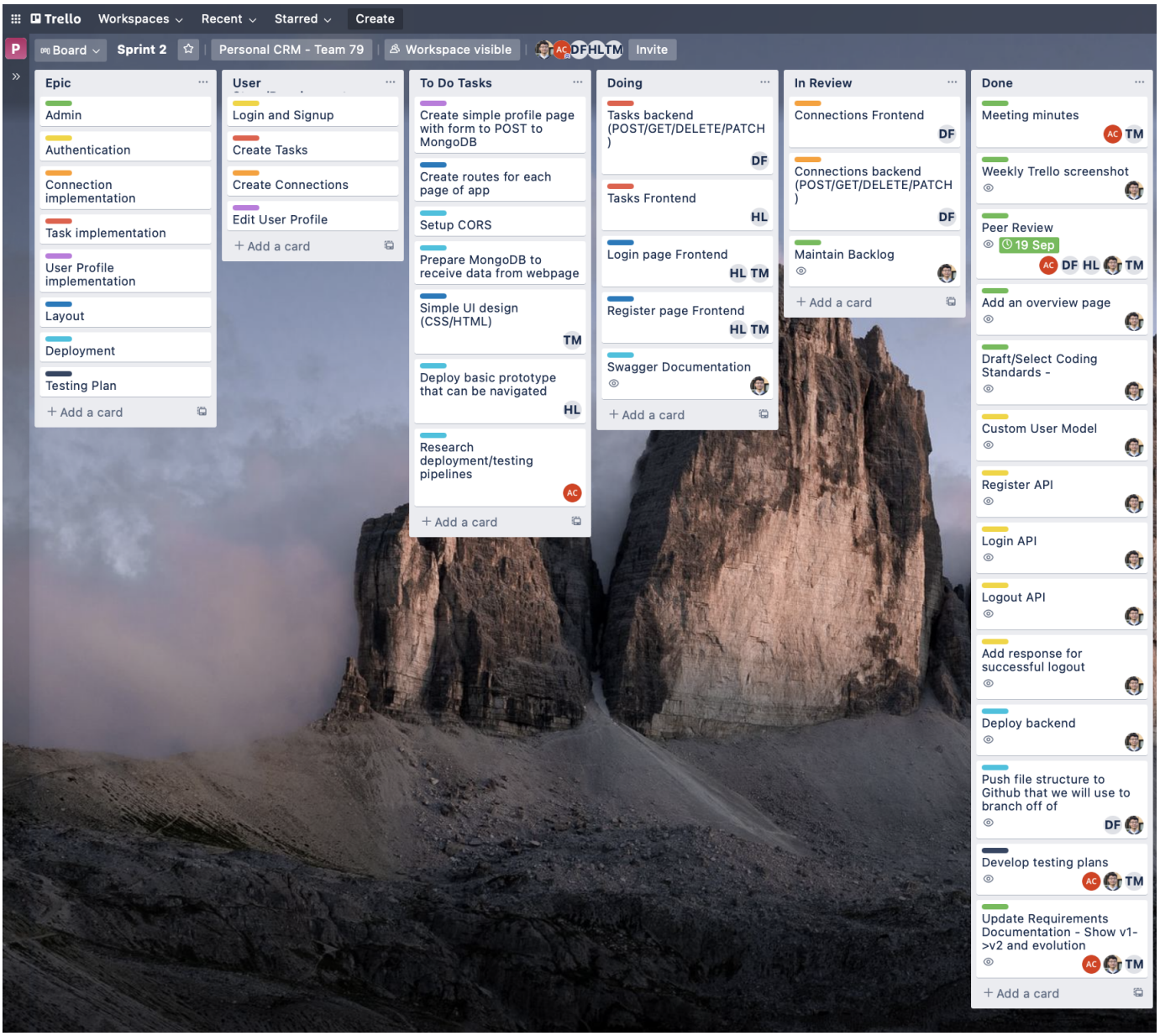


Week 9 Trello Status





Final Trello Update for Sprint 2



Sprint 2 Planning

- Sprint planning checklist
- Sprint team members
- Objectives
- Sprint 2 Backlog
- Potential risks
- Sprint planning resources

- Sprint boards and retrospectives
- Team resources and definitions

Sprint planning checklist

Preparation	Meeting	Follow up
☑ Organise the backlog ☐	☑ Assign the team roles ☑ Confirm the epics	☑ Update Trello board ☑ Document progress on Confluence

Sprint team members

ℹ We keep the same role as last sprint

Name	Role
@ David Fletcher	Product Owner
@ Alexander Cain	Scrum Master
@ Han Liu	Frontend Lead
@ Jackson Hu	Backend Lead
@ Tymara Metcalf	UI / UX Designer

Objectives

Overall objectives supposed to be achieved by the end of sprint 2

1. Begin focus on Login, Task, Connection Pages
2. Design basic navigation
3. Consolidate database models on MongoDB (Someone logging in, creating task, or adding connection)
4. Simple UI design (CSS/HTML)
5. At end of sprint 2 have a deployed prototype

Sprint 2 Backlog

Epic	Task	Sub Task	Story Point (1-10)	Participant	Priority	Status
Admin	Documentation	Weekly Trello screenshot	1 pt	@ Alexander Cain @ Jackson Hu	HIGH	DONE
		Meeting Minutes	2 pt	@ Alexander Cain @ Tymara Metcalf	HIGH	DONE
		Maintain Trello, backlog	2 pt	@ Alexander Cain @ Jackson Hu	HIGH	DONE
		Sprint checklist	4 pt	@ Alexander Cain @ David Fletcher	HIGH	DONE

				@ Han Liu @ Jackson Hu @ Tymara Metcalf		
		Coding Standards	2 pt	@ Jackson Hu	MEDIUM	DONE
	Design Documentation	Update design diagrams	2 pt	@ David Fletcher	LOW	DONE
	Update Requirements	Add further clarifications	3 pt	@ Alexander Cain @ Tymara Metcalf @ Jackson Hu	MEDIUM	DONE
		Update document structure	2 pt	@ Tymara Metcalf @ Jackson Hu	MEDIUM	DONE
		Maintain a change log	4 pt	@ Tymara Metcalf @ Jackson Hu	MEDIUM	DONE
Authentication	Auth Package	Configure REST auth package	2 pt	@ Jackson Hu	HIGH	DONE
	User model	Custom user model	3 pt	@ Jackson Hu	HIGH	DONE
		Update database	2 pt	@ Jackson Hu	HIGH	DONE
	Compatible with access permission		5 pt	@ Jackson Hu	MEDIUM	IN PROGRESS
	Register	Register Page		@ Han Liu	HIGH	DONE
		Register API	4 pt	@ Jackson Hu	HIGH	DONE
	Login	Login Page		@ Han Liu	HIGH	DONE
		Login API	4 pt	@ Jackson Hu	HIGH	DONE
	Logout	Logout Page		@ Han Liu	HIGH	DONE
		Logout API	4 pt	@ Jackson Hu	HIGH	DONE
Connection	Create connection	Update connection model		@ David Fletcher	HIGH	
		Link to user model		@ David Fletcher @ Jackson Hu	HIGH	IN PROGRESS
		POST dummy data		@ David Fletcher	LOW	IN PROGRESS
	Edit connection				MEDIUM	
	Delete connection				MEDIUM	
Task	Create task	Task model		@ David Fletcher	HIGH	DONE
		Task status label	2 pt	@ David Fletcher	HIGH	DONE
	Edit task				MEDIUM	
	Delete task				MEDIUM	
User Profile	View profile				LOW	

	Edit profile				LOW	
Layout	Navigation	Routes (FE)		@ Han Liu @ Tymara Metcalf	HIGH	DONE
Deployment	Link frontend and backend		6 pt	@ Han Liu @ Jackson Hu	HIGH	IN PROGRESS
	Deploy to Heroku	API request	4 pt	@ Jackson Hu	HIGH	DONE
		Set up CORS				
	Swagger set up		4 pt	@ Jackson Hu	MEDIUM	IN PROGRESS
Testing Plan	Test approach overview	Test strategy research	2 pt	@ Tymara Metcalf	HIGH	DONE
		Write an overview	4 pt	@ Tymara Metcalf @ Jackson Hu	HIGH	DONE
	Test instructions		4 pt	@ Tymara Metcalf	HIGH	DONE
	Test cases	Test case template	5 pt	@ Tymara Metcalf	HIGH	DONE
		Organise categories	3 pt	@ Tymara Metcalf	HIGH	DONE
		Test case log	4 pt	@ Tymara Metcalf	MEDIUM	DONE
	Acceptance criteria		3 pt	@ Alexander Cain @ Tymara Metcalf	HIGH	DONE
	Set up testing pipeline		5 pt	@ Alexander Cain	HIGH	IN PROGRESS

⚠️ **Potential risks**

Risk	Mitigation
Prefer to use email as the only username for account access	Switch to a custom user model instead from Django's built in user model
Can't remotely type commands for Heroku	• Install the Heroku CLI • Read Heroku CLI documentation • Set up local environment
Encounter `Heroku error code=14` when deploying backend	• Go through some posts on StackOverflow • Ensure local server is running correctly • Check MongoDB access settings • Install additional packages such like `django-heroku` • Include a `Procfile`
Django backend deployed on Heroku can't communicate with MongoDB	• Check MongoDB access settings • Run `python manage.py migrate --run-syncdb` using Heroku CLI
Need to update data model to put user's own information as a record in connections	Decide to add a `selfId` in connections schema as attribute to identify the information for user itself
Didn't come up with a testing plan	• Did some research on Agile testing approach • Draft test instructions • Create testing template for test cases • Add acceptance criteria • Add a test case log

Hard to schedule a meeting for everyone to complete the sprint checklist	Decide to use the template for sprint 1 and we just add contents on a shared Google doc

Sprint planning resources

Sprint boards and retrospectives

-

Team resources and definitions

- Coding Standards and Guidelines

Sprint 2 Retrospective

An internal reflection regarding the processes that the team has followed over the past sprint.

Note: a process is how you did something; not what you did.

E.g. The team created meeting agendas before each meeting.
The team split into subgroups for pair programming.

Sample Questions:

- **How did we manage meeting minutes? How did we prepare for client, supervisor, and team meetings?**
  - Positives:
    1. We prepared an agenda before client meetings.
    2. We record minutes during meetings.
    3. A detailed meeting record with main talking points will be published after each meeting, thanks to @ Tymara Metcalf
    4. We keep meeting productive and don't meet unnecessarily.
  - Negatives:
    1. We should be more active in terms of reviewing and adding information on meeting minutes.
  - Improve on:
    1. While we have one person responsible for taking the minutes, we may have another one help with taking extra notes or polish the page on Confluence.
    2. Each person keeps tracking of key points.
    3. We should prepare agendas for all meetings.
- **How did we keep track of tasks between members? How did we meet deadlines?**
  - Positives:
    1. We inherited the template from sprint 1 and created a detailed sprint backlog on Confluence, where we record each task for this sprint.
    2. We also utilised Trello board as the place where we can frequently update and manage current ongoing tasks.
    3. Trello cards were labeled with epic and members assigned for those tasks.
    4. We also list some to-dos during the meetings as part of our meeting minutes.
    5. During our weekly stand-ups we could catch up with latest progress for everyone, and if anyone needed help in certain places.
    6. We did improve the way to record and update upcoming tasks based on experience from sprint 1.
  - Negatives:
    1. Part of the sprint backlog hasn't been updated while we were adding new epics, tasks.
    2. Members forgot to assign story points for their own tasks.
    3. Trello was not updated frequently (although better than sprint 1).
  - Improve on:
    1. Be more active on Trello board to update current progress.
    2. Can use Trello board to mark some comments on cards.
    3. Can add a due date for each task on Trello.
    4. Begin submission documents earlier.
- **How did we decide on which tools to use? How did we adapt to new technologies?**
  - Positives:
    1. Nearly all tool stacks were finalised with clear listed tools and packages used documented on Confluence.
    2. A coding standard for our Django and React has now published on Confluence.
    3. We agreed to use Prettier as our vscode formatter plugin.
    4. We continued to help each other to overcome any problems encountered during development.
  - Negatives:
    1. Didn't configure Heroku properly at the beginning, spent a fair bit of time deploying our backend.
    2. Need to link our frontend and backend ASAP.
  - Improve on:
    1. Schedule more pair programming sessions to help each other or tackle issue together.
    2. Greater communication on issues with deployment between team members.
- **How did we adapt to Agile Process? How did we keep organised and develop a workflow that worked for all members?**

    - Positives:
        1. Currently get used to Agile Process and familiar with works for each sprint.
        2. Organised well for sprint planning and weekly stand-up meetings.
    - Negatives:
        1. Continuously update design models while developing.
        2. A bit of rush at the end of each sprint.
    - Improve on:
        1. Be more realistic and come up with a more detailed sprint planning.
        2. Do our retrospective and review earlier for the next sprint.
- **Did we have a testing plan in place? How did we implement it throughout the project?**
    - Positives:
        1. Detailed testing plan in place on confluence.
        2. A testing overview and instructions are available for ease of use.
        3. Also a template is available for standardised tests to be implemented by other users.
    - Negatives:
        1. Initially we did not have a testing plan in parallel with our code rollout.
        2. We didn't have an automated testing pipeline set up yet.
    - Improve on:
        1. We could add instructions for how to configure the testing environment for both frontend and backend.
- **How did the team recognise when others needed assistance? How did we check in with each other?**
    - Positives:
        1. We would have meetings 2-3 times a week, and get an idea of how everyone was tracking.
        2. No one felt overwhelmed and seeking assistance was easy for everyone.
        3. We were able to just message casually on Slack whenever we had questions or had to organise something.
        4. We had pair programming sessions to work together on programming, testing plans and other documentation.
    - Negatives:
        1. Members working with different tasks may need more communication and cooperate.
    - Improve on:
        1. Share learning resources.
        2. Should be more active to schedule meetings.
        3. Update team on Slack when you are working on something.

Sprint 2 Review

> Sprint Review is expected to be an **external reflection** with the **client** regarding the **product** that we have created thus far

Product Presentation

We did a quick presentation about the things we've done for the sprint 1 which includes:

1. Heroku interactive **prototype**
    a. Main pages of CRM
        i. Register / sign up / log in
        ii. Dashboard
        iii. Connections
        iv. Tasks
        v. Profile
    b. Actions corresponding to user stories based on requirements from client
    c. Design style and component layout
2. Clarification and Feedback on **prototype**
    a. Missing features
    b. Features requiring modification
    c. Non-functional changes
3. Update client requirements
    a. Final iteration of user requirements
    b. Set minimum deliverables
    c. Discuss handover dates and meeting times

Reflection

> Client considered our prototype deployment as strong overall just missing the functionality from connecting backend with frontend.

Good	Bad	Areas of Improvement
Client really likes the look of the prototype and design style, saying it's excellent	Layout, responsiveness is a little unpredictable.	The search feature is critical.

The progress bar reflecting the due date on task page looks fancy		Tags/categories are very similar. What is difference and how should they be used?
The flow of pages is really good.		Unclear whether user can access tasks associated with a connection from their page.
Lots of features have been delivered, not just the basics.		

Feedback 2

Teamwork (2/2): Great job working together this sprint. All team members have contributed equally to the success of the project both on github and within the documentation. Note: The team currently has very clear roles and is only working in these roles. I would strongly recommend either changing these roles or ensure that all team members have a chance to work on everything, including the code.

Consistency (1/1): Team has maintained a consistent amount of effort throughout the project to date.

Requirements (1/1): Requirements are well documented and changes have been added, along with a description of these changes. Note: Your links on the submission document to requirements were broken, however I was still able to find this section.

Design (1/1): Design is well documented and changes have been added, along with a description of these changes.

Coding (1/1): A clear set of coding standards has been documented and these standards have been followed.

Testing (1/1): A clear test plan has been produced and the team has made an effort to act on this.

Deployment (0.5/1): Deployment only appears to have occured for the backend, at least from the links I can find. Some attempt has been made at documenting this but a pipeline is lacking.

Additional Accomplishments (0.5/2): The team's confluence space goes above and beyond what is expected at this point, really great work here. The team has very comprehsensive testing. documentation, however it would have been nice to see some automated testing. The team's commit history on github is a bit lackluster, and somewhat concerning.

Overall (8.5/10): Great work. I feel like with some small tweaks to the teams' roles this team would have had more time to work on additional things such as testing and pipelines. Please ensure that ALL team members have contributed to the coding as well as the documentation by the end of this project.

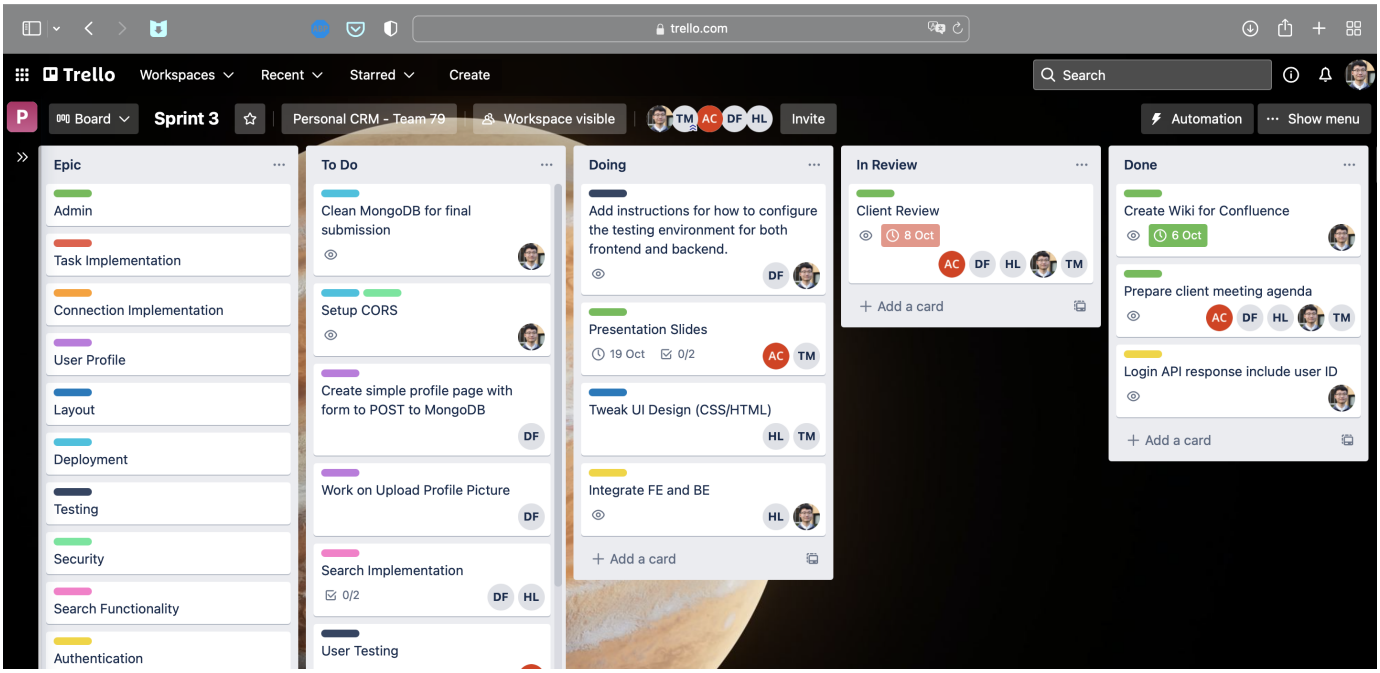
UPDATE: Additional Accomplishments should be (1/2). The total score of 8.5 is correct.

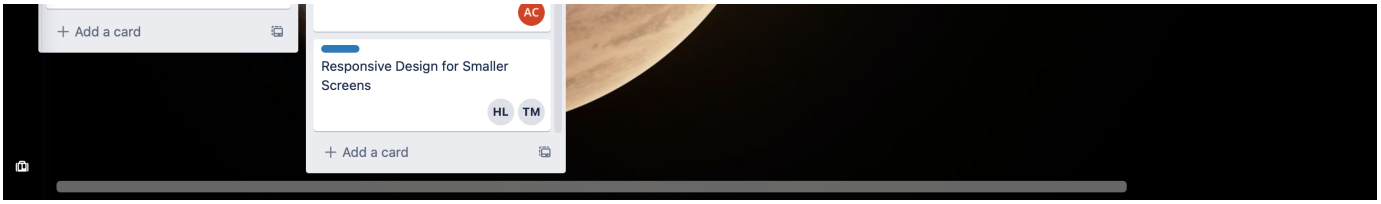
Sprint 3

Sprint 3 Checklist

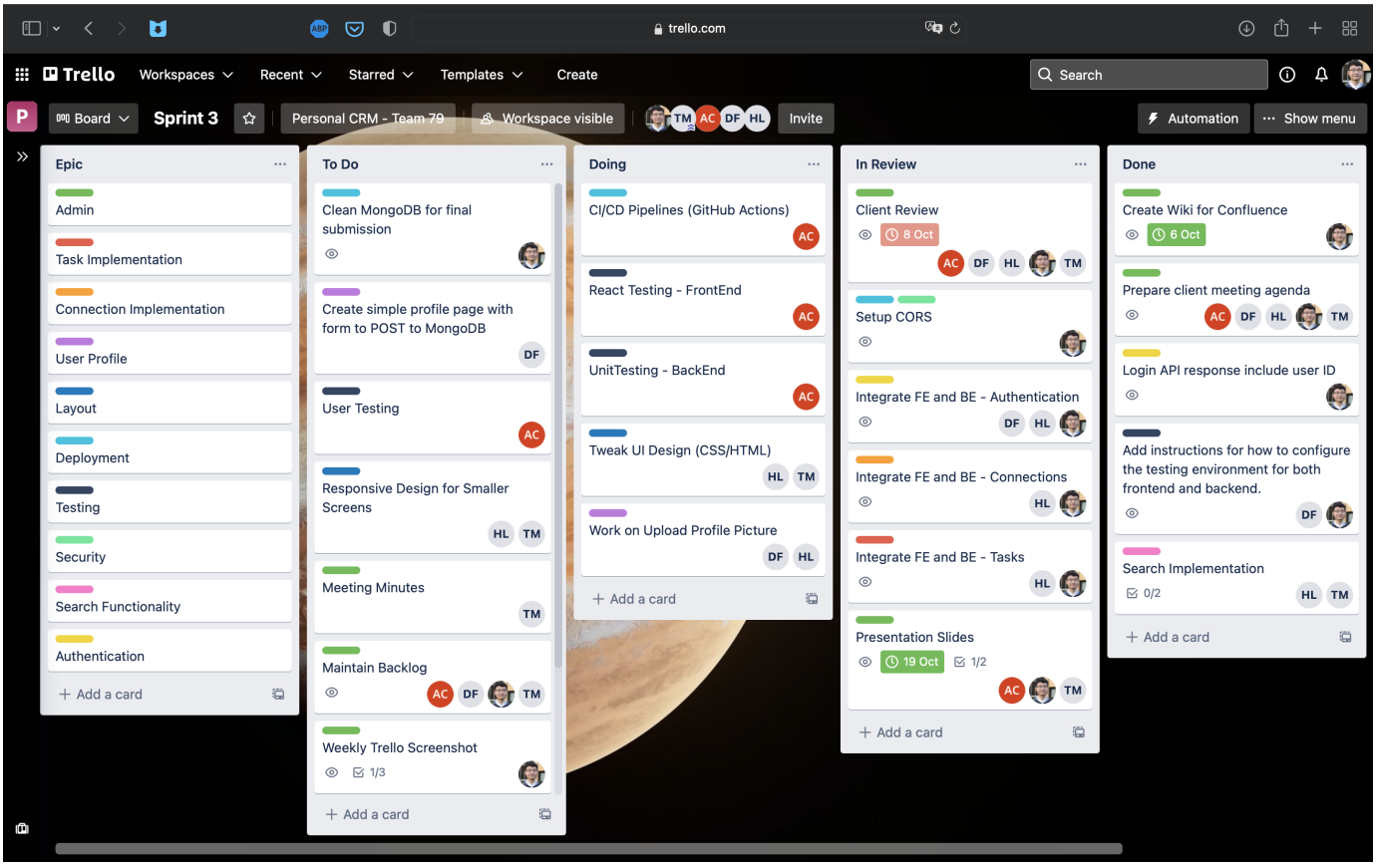
Sprint 3 Trello

Week 10 Trello Status

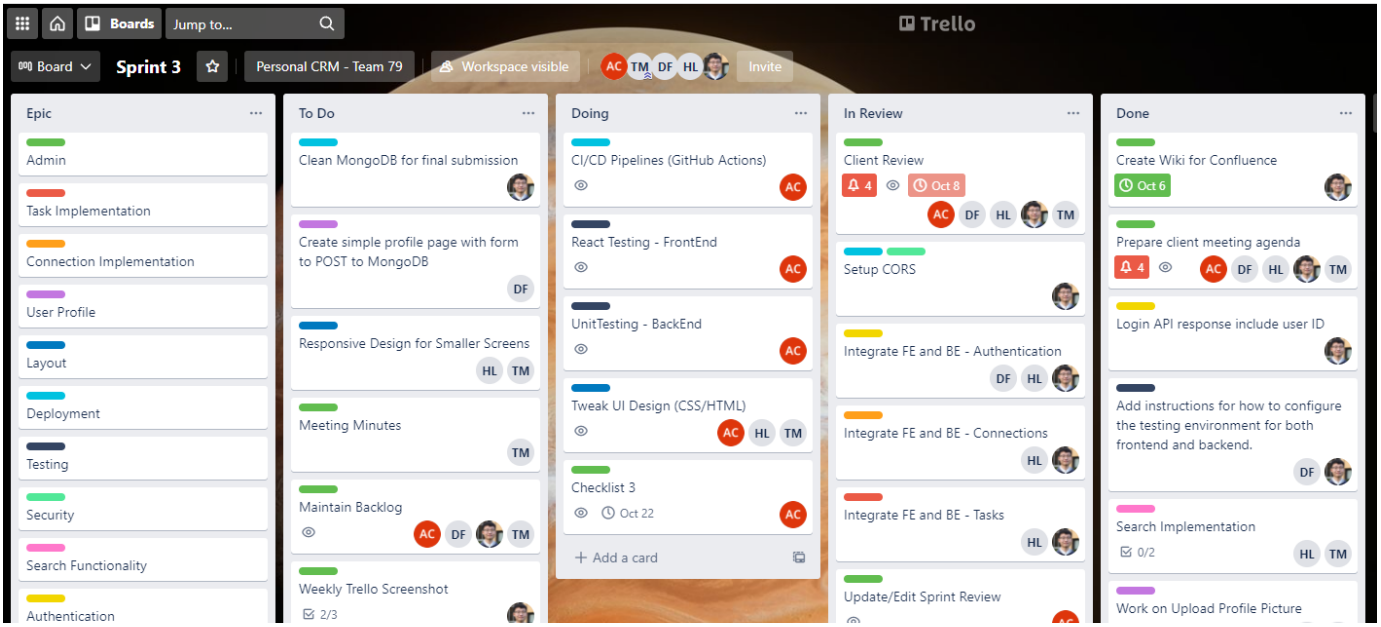




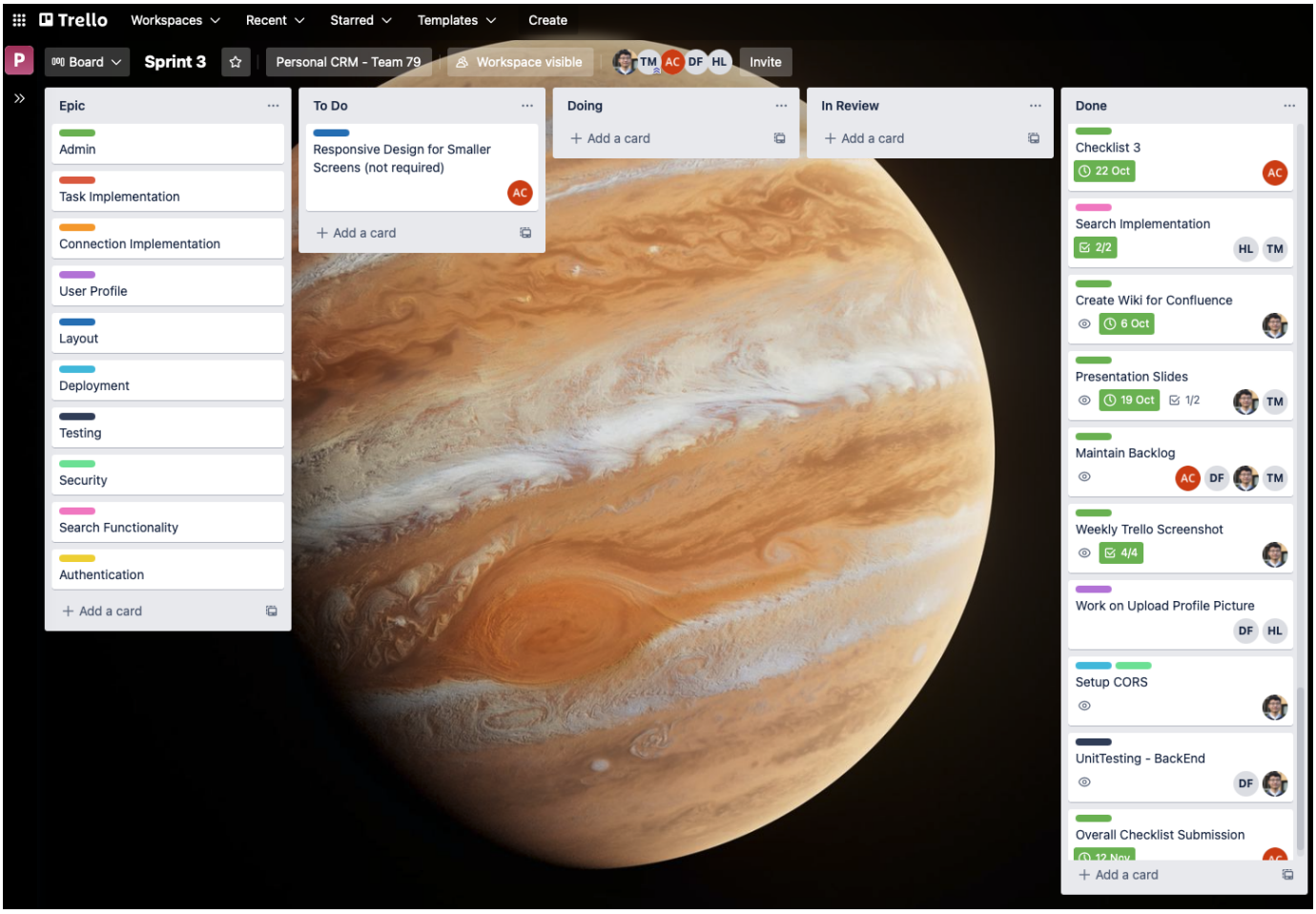
Week 11 Trello Status



Week 12 Trello Status



Final Trello Status



Sprint 3 Planning

✅ Sprint planning checklist

Preparation	Meeting	Follow up
☑ Organise the backlog	☑ Assign the team roles	☑ Update Trello board
☑ Collect unfinished tasks from Sprint 2	☑ Confirm the epics	☑ Document progress on Confluence

👥 Sprint team members

We keep the same role as last sprint

Name	Role
@ David Fletcher	Product Owner
@ Alexander Cain	Scrum Master
@ Han Liu	Frontend Lead
@ Jackson Hu	Backend Lead
@ Tymara Metcalf	UI / UX Designer

Objectives

Overall objectives supposed to be achieved by the end of sprint 3

1. Share task load
2. Frontend and backend integration
3. Parallel Code<>Testing
4. Prepare a working prototype / demo
5. Prepare for handover

Sprint 3 Backlog

Epic	Task	Sub Task	Story Point (1-10)	Participant	Priority	Status
Admin	Documentation	Weekly Trello screenshot	1 pt	@ Alexander Cain @ Jackson Hu	HIGH	DONE
		Meeting Minutes	2 pt	@ Alexander Cain @ Tymara Metcalf	HIGH	DONE
		Maintain Trello, backlog	2 pt	@ Alexander Cain @ Jackson Hu	HIGH	DONE
		Sprint checklist	4 pt	Everyone	HIGH	DONE
	Resources	Shared file list	1 pt	@ Jackson Hu	LOW	DONE
		Learning resource / Wiki	4 pt	@ Jackson Hu	LOW	DONE
	Presentation	Slides	4 pt	@ Tymara Metcalf @ Jackson Hu	HIGH	DONE
		Scenarios	2 pt	Everyone	HIGH	DONE
		Live demo	5 pt	Everyone	HIGH	DONE
Authentication	Integrate with React	Update login response to include user ID	2 pt	@ Jackson Hu @ Han Liu	HIGH	DONE
		Update register response to include user ID	2 pt	@ Jackson Hu	HIGH	DONE

Connection	Data model	Add selfId field	2 pt	@ David Fletcher	HIGH	DONE
		Update connection serializer	2 pt	@ Jackson Hu	HIGH	DONE
		Update JSONField	2 pt	@ David Fletcher @ Jackson Hu	HIGH	DONE
		Replace DataField CharField	1 pt	@ Han Liu @ Jackson Hu	MEDIUM	DONE
	Create connection	Add a new connection	4 pt	@ David Fletcher	HIGH	DONE
		Link to user model	3 pt	@ David Fletcher @ Jackson Hu	HIGH	DONE
		POST dummy data	2 pt	@ David Fletcher	LOW	DONE
	Retrieve connection	Get all connections in database	2 pt	@ David Fletcher	LOW	DONE
		Get a specific connection	2 pt	@ David Fletcher	HIGH	DONE
		Get all connections of a user	3 pt	@ David Fletcher	HIGH	DONE
	Edit connection	Update connection info	4 pt	@ David Fletcher	HIGH	DONE
	Delete connection	Delete a single connection	3 pt	@ David Fletcher	HIGH	DONE
		Delete all connections of a user	3 pt	@ David Fletcher @ Jackson Hu	MEDIUM	DONE
		Fix unable to delete a connection	2 pt	@ Jackson Hu	HIGH	DONE
	Connection page		8 pt	@ Han Liu	HIGH	DONE
Task	Data model	Update JSONField	2 pt	@ David Fletcher @ Jackson Hu	HIGH	DONE
		Replace DataField CharField	1 pt	@ Han Liu @ Jackson Hu	MEDIUM	DONE
	Create task	Add a new task	4 pt	@ David Fletcher	HIGH	DONE
	Retrieve task	Get all tasks in database	2 pt	@ David Fletcher	LOW	DONE
		Get a specific task	2 pt	@ David Fletcher	HIGH	DONE
		Get all tasks of a user	3 pt	@ David Fletcher	HIGH	DONE
	Edit task	Update task info	4 pt	@ David Fletcher	HIGH	DONE
	Delete task	Delete a single task	3 pt	@ David Fletcher	HIGH	DONE
		Delete all tasks of a user	3 pt	@ David Fletcher @ Jackson Hu	MEDIUM	DONE
	Task page		8 pt	@ Han Liu	HIGH	DONE

User Profile	Simple profile page		4 pt	@ Tymara Metcalf @ Han Liu	MEDIUM	DONE
	View profile		3 pt	@ Tymara Metcalf @ Han Liu	LOW	DONE
	Edit profile		3 pt	@ Tymara Metcalf @ Han Liu	LOW	DONE
Search Functionality	Search in connections		4 pt	@ Han Liu @ Tymara Metcalf	MEDIUM	DONE
	Search in tasks		4 pt	@ Han Liu @ Tymara Metcalf	MEDIUM	DONE
Layout	Responsive Design for Smaller Screens		3 pt	@ Han Liu @ Alexander Cain	MEDIUM	DONE
	Tweak UI Design (CSS/HTML)		3 pt	@ Han Liu @ Alexander Cain @ Tymara Metcalf	HIGH	DONE
Deployment	Link frontend and backend	Call corresponding API with correct format	6 pt	@ Han Liu @ Jackson Hu	HIGH	DONE
	Deploy to Heroku	Set up CORS	4 pt	@ Jackson Hu	MEDIUM	DONE
	Clean MongoDB for final submission		3 pt	@ Jackson Hu	MEDIUM	DONE
Testing	Set up testing pipeline		5 pt	@ Alexander Cain @ Jackson Hu	MEDIUM	DONE
	GitHub Actions build CI	For frontend	3 pt		HIGH	IN PROGRESS
		For backend	3 pt	@ David Fletcher @ Jackson Hu	HIGH	DONE
	Instructions for environment configuration	For frontend	2 pt	@ David Fletcher	HIGH	DONE
		For backend	2 pt	@ Jackson Hu	HIGH	DONE
	Test case records		4 pt	@ Jackson Hu	HIGH	DONE

⚠️ **Potential risks**

Risk	Mitigation
Improper navigation	System and User testing
Noticed our app deployed on Heroku isn't using MongoDB as configured	• Found that Heroku automatically set a `DATABASE_URL config var` so our database setting is overridden

	• It's caused by a postgresql add-on which somehow Heroku automatically added • Have to delete that add-on and set a `MONGODB_URI config var` to let it connect to MongoDB again
Time management for features required by the client	Negotiate with the client and update priority for some requirements
Didn't integrate frontend and backend until the last minute	• In-person programming session to speed up the pace • More efficient communication about issues found • Concurrent workflow for different features (e.g. function & UI)

 **Sprint planning resources**

Sprint boards and retrospectives

Team resources and definitions

- Learning Resources

Sprint 3 Retrospective

An internal reflection regarding the processes that the team has followed over the past sprint.

Note: a process is how you did something; not what you did.

E.g. The team created meeting agendas before each meeting.
The team split into subgroups for pair programming.

Sample Questions:

- **How did we manage meeting minutes? How did we prepare for client, supervisor, and team meetings?**
  - Positives:
    1. We've gotten pretty good at organising ourselves and tracking meeting items through Confluence. By preparing drafts and templates that can be re-used by any member of the team.
    2.
    3.
  - Negatives:
    1. Great improvements over time have left little to speak on as far as negatives.
    2.
    3.
  - Improve on:
    1. Similar to the above, there are no comments from the team on how to improve at this time.
    2.
    3.
- **How did we keep track of tasks between members? How did we meet deadlines?**
  - Positives:
    1. We have been much better at utilizing both Trello and the Sprint Backlog in adding tasks and assigning team members.
    2. We did better this time to meet deadlines such as presentation, frontend and backend integration, etc.
    3.
  - Negatives:
    1. Distribution of tasks and workload wasn't always apparent in Trello + Backlog as different tasks and user stories had higher time requirements.
    2.
    3.
  - Improve on:
    1. Continue to set deadlines and assign a team member to tasks for any task, even if it isn't being started just yet. Can help with building a timeline of expectations and keep the project moving.
    2. Could use a priority system to ensure core tasks done first.
    3.
- **How did we adapt to Agile Process? How did we keep organized and develop a workflow that worked for all members?**
  - Positives:
    1. We communicated a lot more this sprint. There was a lot to get done and we were able to keep track of where we were at with Trello.
    2. We planned out early via Trello and team meetings, on how we wanted to attack this sprint. Which features we really were going to be able to complete. And then started the design and building process. Our design early on carried us through what and how we would implement the product. That in line with what the client requested in the requirements/user stories helped up build/test/review and repeat easily.
    3. Our building process included a lot of peer-programming this sprint. We had a lot to do as a team, i.e. deployment of our 2 repositories and frontend design. We were able to come together and assist each other very well this way. No one was left in the dark.

- Negatives:
  1. Leaving deployment and development features to this last sprint was not in line with an ideal agile process.
  2.
  3.
- Improve on:
  1. We could schedule and finish the sprint retrospective and review earlier rather than in the middle of the next sprint.
  2.
  3.

- **How did the team recognise when others needed assistance? How did we check in with each other?**
  - Positives:
    1. We had lots of pair programming sessions during this sprint, working for frontend features, integration of frontend and backend, deployment.
    2. Didn't have miscommunication this time like the mistake we had during sprint 2.
    3.
  - Negatives:
    1.
    2.
    3.
  - Improve on:
    1. Keep all members updated with the work you're doing.
    2.
    3.

- **Is the testing plan up to date? How's the testing going for the project? Did the team have automated testing?**
  - Positives:
    1. GitHub Actions was set up for backend repository and running.
    2. Some Django automated test cases have been prepared.
    3. Have test case records well-documented on Confluence with a log.
  - Negatives:
    1. Still working on the GitHub Actions for frontend repository.
    2.
    3.
  - Improve on:
    1. Should include more high level testing, not only unit tests.
    2. May be better to have more automated test cases.
    3. Set up automated testing pipeline for frontend too.

Sprint 3 Review (draft)

Sprint Review is expected to be an **external reflection** with the **client** regarding the **product** that we have created thus far

Product Presentation

We did a quick presentation about the things we've done for sprint 3 which includes:

1. Heroku Live Demo
   a. Features / Requirements / User Stories check off ✅
      i. Register / sign up / log in
      ii. Dashboard
      iii. Connections & VIP (Create/View/Edit/Delete)
      iv. Tasks (Create/View/Edit/Delete)
      v. User Profile
      vi. Search/Filter
      vii. Notes
      viii. Reminders
   b. Walk through with client of site layout and general use
2. Feedback of **Live Demo**
   a. Allow client to use product
   b. Improvements to User experience and interface
   c. Minor bugs on Dashboard
   d. Add tooltips to forms and submissions
3. Update client requirements
   a. Discuss required items for handover
   b. Discuss handover dates and meeting times
   c. No further changes to client requirements
   d. Will prepare a Acceptance Criteria prior to handover to assure client is happy with product.

Reflection

Client is very happy with product and is ready to receive on coming Friday 5/11/21

Good	Bad	Areas of Improvement
Client is still happy with the look and design of final product.	Appears to allow multiple registrations of the same email and/or empty forms. Functionality is there but not apparent to user.	Add tool-tips to the UI/UX, to tell the user their operations are successful or failed.
All features requested have been implemented. Impressed that his requirements have been met fully.	Bug: “Loading…” still visible on dashboard	Form checks on register/login/creating tasks and connections. So user knows what fields are required to proceed.
Happy with implementation of optional features.	Tasks/Connections do not auto-refresh for user.	Site is semi-responsive on different screens, would love to see more implementations, but not a required feature.
		Tasks/Connection pages refresh automatically when changes (adding/editing /deleting) occur.

Requirements

Changelog

User Stories

Date	Change
23 Oct 2021	• **Revise requirement for Dashboard Metrics:** Show VIP Connections ~~and most frequent connections~~ Quickly find VIP connections ~~and close connections~~
01 Oct 2021	• **Formatting** change • **Grouping** of similar features • **Additions** from previous client meeting (https://comp30022-079.atlassian.net/wiki/spaces/CRM/pages/20480001/2+2021-09-7+Client#Further-Requirements-Clarifications) • **Clarify** the search functionality
17 Sep 2021	• **Update** Requirements file structure organization • Separate User stories, Do / Be / Feel List, Motivational Model
05 Sep 2021	• **Add** “Edit Tasks: Update information associated to task” • **Add** “Delete Tasks: Remove expired or unnecessary tasks” • **Add** Filter feature for categorizing tasks • Notes: **added** “Store additional information about ~~them~~tasks and people” • Connections of stored contacts: **added** “See relationship between connections”
17 Aug 2021	• **Add** “So that…” Column

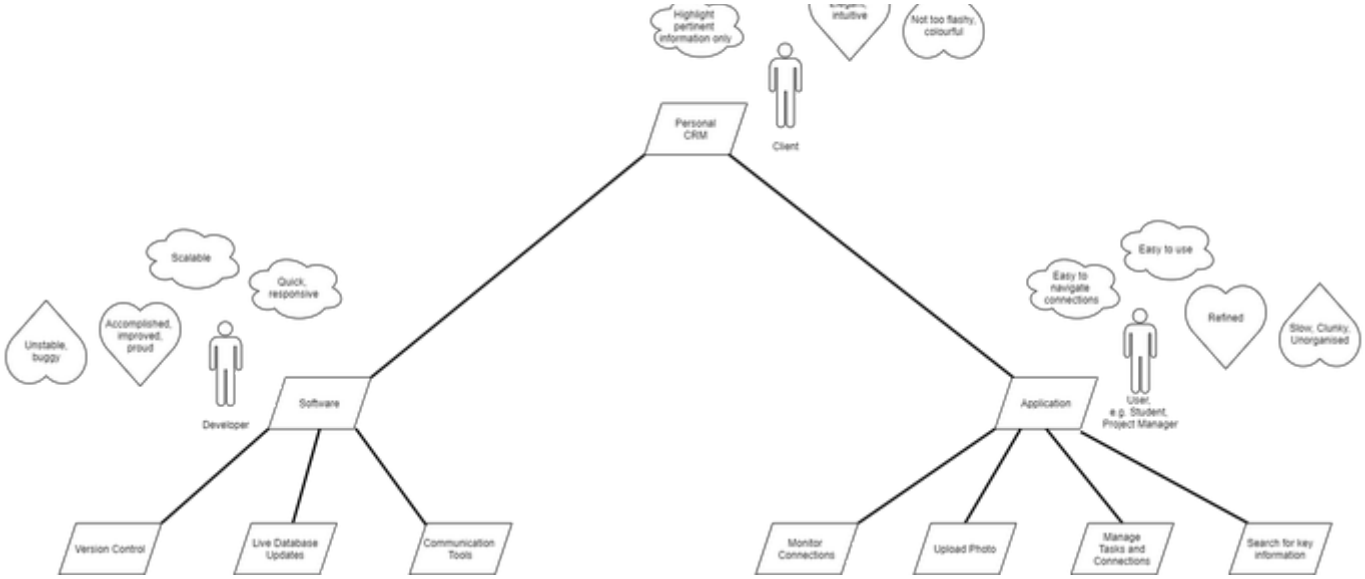
Do / Be / Feel List

Who	Do	Be	Feel
User	Manage contacts	Clear and simple	Engaged
	Track connections	Organised	Satisfied
	Search for the key information	Quick and efficient	Delighted
	Check the tasks / events	Easy to change the status	Disciplined
	Check the related connections (connection groups)	Easy to add or remove connections in groups	Comfortable

User Stories

As a ...	I want to ...	So that ...	Priority	Done
User	Log in and sign up	Can access own account	HIGH	
	Tasks: • Create tasks (filter e.g. due dates)	Keep track of tasks	HIGH	
	• Edit tasks	Update information associated to task	HIGH	
	• Delete tasks	Remove expired or unnecessary tasks	HIGH	
	• Calendar view for the tasks	Check due dates for tasks clearly	OPTIONAL	
	• Ability to add start and end date	Know the due date and current progress	MEDIUM	
	Connections: • Create new connections	Manage new person I met	HIGH	
	• Save contact information (e.g. email, phone ...)	Know how to contact them	HIGH	
	• Delete connections	Remove the person unrelated anymore	HIGH	
	• Edit connections	Update their information	HIGH	
	• VIP tag	Know they're important	HIGH	
	• Include phone number in quick view (main connection page)	See relationship between connections	HIGH	
	• Categorise and prioritise my tasks	Know whether a task if important	MEDIUM	
	• Customise connection tags	Add customised tags for different connections	MEDIUM	
	Search: • Filter search on key fields (checkbox) (name, company, date, task, connections, notes)	Quickly find the person with some attributes	HIGH	
	• Ability to search with a filter applied	Find the common attributes	MEDIUM	
	• Search function only needed on tasks and connections page	Search on connections and tasks	HIGH	
	Notes: • Any additional information can be put in notes	Store additional information about tasks and people	HIGH	
	Dashboard: • Show most important tasks first (priority and due date)	Know which tasks are the most important with highest priority	LOW	
	• Count of finished and unfinished tasks (added)	Have a better understanding of current task process	HIGH	
	• Show VIP Connections	Quickly find VIP connections	HIGH	
	Visual summary for existing connections	Get a sense of overall relationship	LOW	
	Edit user profile (user name, password, profile photo)	To be more personalised	MEDIUM	
	Optimise for mobile platforms (Responsive Design)	Easy to use everywhere (Mobile)	OPTIONAL	
	Reminder for important dates	Won't forget to celebrate their birthdays etc.	OPTIONAL	

Motivational Model



Motivational Model.drawio

.io file available for edit

Acceptance Criteria

User Story ID	User Story	Given	When	Then
1.1	As a user I can sign up to the application to create my account and start tracking my network.	My email isn't associated with an existing account.	I provide a valid email and password.	An account is created for me and I am logged in to my blank personal dashboard.
1.2	As a user I can login in to my existing personal CRM and retrieve my stored information.	I have an existing account.	I provide my associated email and password.	My login details are authenticated and I access my personal dashboard.
2.1	As a user I can access all of my stored tasks and view detailed information for any task.	I have added at least one task.	I navigate to the tasks page and select a specific task with its three dot button.	I am shown the associated details with the selected task (title, date added, due date, associated connections, notes, location etc.).
2.2	As a user I can add new tasks to my personal CRM to track.	I am logged in and on the dashboard or tasks page.	I click on the new tasks button (speed dial) and input relevant information including at least a name and description.	My new task is created and will be shown in the list of tasks on the tasks page.
2.3	As a user I can edit an existing task to update any information.	I am on a specific tasks information page.	I click on the manage tasks button (three dots) and then the	

			edit option to update relevant information including at least a name and description.	My task is updated and modified as shown in the list of tasks on the tasks page.
2.4	As a user I can remove a task from my personal CRM I no longer need to track	I am on the tasks page and have found a task to be deleted.	I click on the unwanted tasks manage tasks button (three dots) and then the delete option that appears.	My task is removed from the list of tasks on the tasks page. My task is also unassociated with any connections and won' t appear on their information page.
3.1	As a user I can access all of my stored connections and view detailed information for any connection.	I have added at least one connection.	I navigate to the connections page and select a specific connection with its three dot button.	I am shown the associated details with the selected connection (name, job, location, contact details, etc.).
3.2	As a user I can add new connections to my personal CRM to track.	I am logged in and on the dashboard or connections page.	I click on the new connections button (speed dial) and input relevant information including at least a name.	My new connection is created and will be shown in the list of connections on the connections page.
3.3	As a user I can edit an existing connection to update any information.	I am on a specific connections information page.	I click on the manage connections button (three dots) and then the edit option to update relevant information including at least a name.	My connection is updated and modified as shown in the list of connections on the connections page.
3.4	As a user I can remove a connection from my personal CRM I no longer need to track	I am on the connections page and have found an unneeded connection.	I click on the unwanted connections manage button (three dots) and then the delete option that appears.	My connection is removed from the list of connections on the connections page. My connection is also unassociated with any task and won't appear on their information page.
3.5	As a user I can mark a connection as a VIP to note they are higher priority than other connections.	I am on the chosen connections information page.	I click on the VIP toggle button.	Their status on their information page is updated to VIP. This change is also shown in the connections and dashboard pages where they are shown as higher priority.
4.1	As a user I can search for connections by different search criteria.	I have added a number of connections with detailed information.	I click on the search bar and enter a string.	A search results pop out appears displaying relevant connections in a table.
4.2	As a user I can search for tasks by different search criteria.	I have added a number of tasks with detailed information.	I click on the search bar and enter a string.	A search results pop out appears displaying relevant tasks in a table.
5.1	As a user I can get a summary of important information about my connections.	I have added a number of connections with detailed information.	I navigate to the main dashboard.	I am displayed my most frequent connections, VIP connections and analytics about connections.
5.2	As a user I can get a summary of important information about my tasks.	I have added a number of tasks with detailed information.	I navigate to the main dashboard.	I am displayed my highest priority tasks, tasks that are due soon and analytics about tasks.
6.1	As a user I can filter connections by VIP status.	I have added a number of connections with at least one with VIP status.	I click on the VIP filter next to the search bar.	Only connections marked with VIP status are retained in display.
6.2	As a user I can filter tasks by priority.	I have added a number of tasks with at least one of the priority being filtered on.	I click on the priority filter next to the search bar and choose a priority level.	Only tasks marked with the designated priority are retained in display.
7	As a user I am reminded of important upcoming dates that are associated with a task or connection.	I have an upcoming important date.	That upcoming date is less than a week away.	The notification bell is alerted and the number of alerts is increased by one. An associated alert is given about the important date and the task /connection it is attached to.
8	As a user I can view tasks in a calendar view to visualize my upcoming tasks.	I have at least one task stored in the personal CRM.	I navigate to the tasks page and click on the calendar view button.	A calendar view page opens displaying a navigable calendar with tasks, important dates and birthdays displayed.
9	As a user I can easily use this service on either a mobile or desktop device.	I am on a mobile platform.	I try to complete any of the other user stories.	I receive the same outcome as expected for a desktop user.

Testing

Test Approach Overview

Overview

This testing strategy determines our project's approach to testing. It is intended for the developer team, designers, testers, and any other members that may be involved in the system tests. Members of the team will take their roles in this testing procedure as listed below.

Key objectives are:

- Determine the types of tests required for each test case.
- Determining the significance or critical nature of issues within the project to the stakeholder.
- Ensure the reliability of core functionalities and components of the software which guarantees the quality of the software.
- Identify areas that do not meet acceptance criteria.

Benefits are:

- Faster development and deliverables.
- Earlier identification of defects

Testing procedures are:

- Tests will be conducted using the template provided in Instructions
- Copy the template for new test cases
- Fill in the basic information about the test case
    - Test type
    - Test case name
    - Brief description about the test case
    - Members conducted the test
    - Test result
- Elaborate on the steps involved for the test case
- Attach any evidence of testing e.g. screenshots

Roles and Responsibilities

Name	Role	Contact
@ Tymara Metcalf	Project Manager	tmetcalf@student.unimelb.edu.au
@ Alexander Cain	Test Leader	acain1@student.unimelb.edu.au
@ David Fletcher	Individual Tester	dfletcher@student.unimelb.edu.au
@ Han Liu	Individual Tester	liuh8@student.unimelb.edu.au
	Individual Tester	renweih@student.unimelb.edu.au

@ Jackson Hu		

Scope

TC 1: Registration Functionality

> 🛠️ *Tier 1 TC will be performed to evaluate the system's ability to process user's data communicate with the database and create a new record in database.*

TC 2: Login Functionality

> 🛠️ *Tier 2 TC will evaluate the system's ability to maintain user's information throughout the session.*

TC 3: Password Authentication Functionality

> 🛠️ *Tier 3 TC will assert that unauthorised access is denied and user protection is properly in place.*

TC 4: Connections Testing

- TC 4.1 Adding Contact Functionality
- TC 4.2 Edit Contact Functionality
- TC 4.3 Delete Contact Functionality
- TC 4.4 VIP Toggle
- TC 4.5 Notes Management

> 🛠️ *Tier 4 TC ensures the user can easily create connections as well as test backend functionality.*

TC 5: Tasks Testing

- TC 5.1 Adding Task Functionality
- TC 5.2 Edit Task Functionality
- TC 5.3 Delete Task Functionality

> 🛠️ *Tier 5 TC ensures the user can easily manage tasks as well as test backend functionality.*

TC 6: Account Profile Functionality

- TC 6.1 Edit Contact Information
- TC 6.2 Upload Photo

> 🛠️ *Tier 6 TC lets the user to successfully customise their profile throughout the whole system.*

TC 7: Dashboard Functionality

- TC 7.1 Summary of Tasks
- TC 7.2 Summary of Connections
- TC 7.3 Connection Metrics

> 🛠️ *Tier 7 TC ensures the dashboard captures all the necessary information from different components to be displayed for user.*

TC 8: System Testing

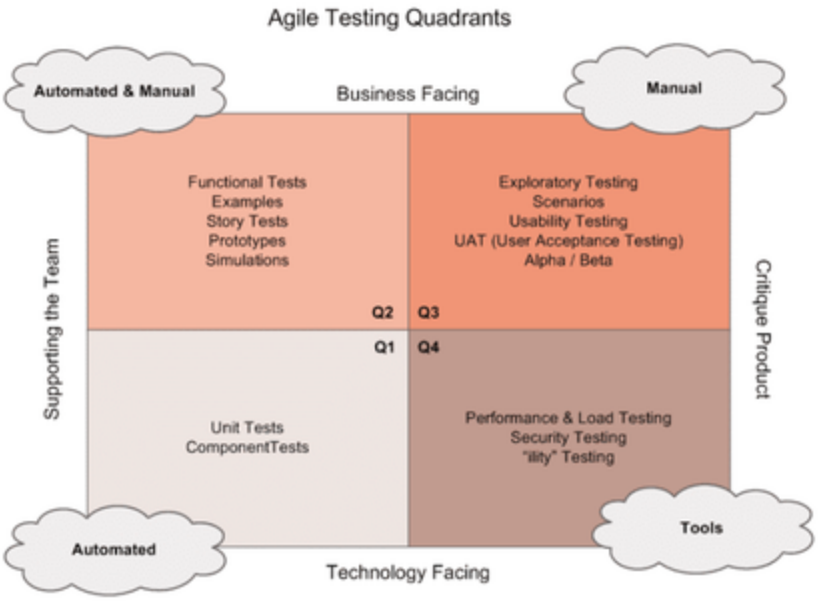
- TC 8.1 Create Account/Login
- TC 8.2 Navigation

> 🛠️ *Tier 8 TC catch-all testing that ensures system is working as a whole.*

Test Levels

1. Unit Testing
2. API Testing
3. Integration Testing
4. System Testing

5. Security Testing
6. User Acceptance Testing
7. Smoke Pack (<15mins): test high-level functionality.
8. Regression Pack (<60mins): quick feedback on a larger set of combined tests.



Test Types

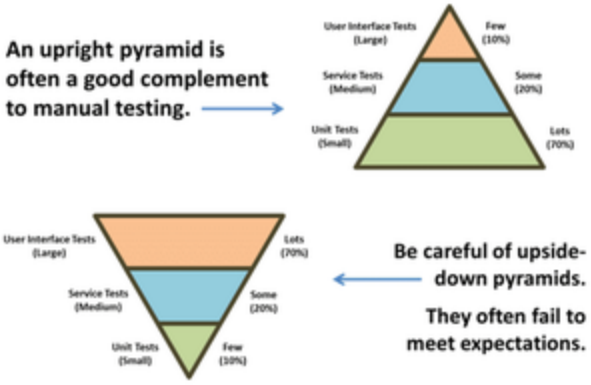
1. Manual
2. Automated testing tools
3. Automated testing scripts

Environment Requirements

- Unix / Windows Subsystem for Linux
- Correctly configured virtual environment for Django backend
- Packages needed:
  - Python
  - Pip
  - Pipenv
  - Other packages should be included in virtual environment
- Or visit backend endpoints deployed on Heroku

Testing Tools

1. Navigate through the software and test functionalities manually
2. Backend
   a. Postman for API testing
   b. REST Framework testing interface
3. Frontend
   a. React Testing Library



Instructions

Use Tables below:

- Open this page in edit view and copy these tables into specific test page.
- Label page #tc (bottom right corner).
- Page Properties table Macro will update Parent page with relevant information.
- Select which test type and how you are executing said test.
- Test Priority is based on Client Requirement priority.
- Test Case ID will be the TEST CASE Parent + Child + any additional tests done on the same child in 0-9 ascending order:
  - e.g. a second test on TC 1 Registration Functionality would be = 1.0.1
  - e.g. a forth test on TC 4.2 Edit Contact Functionality would be = 4.2.3
- Add TC Steps Numbers as needed. Some tests will required more or less steps to complete.

Test/Execution Type	**<Unit, Functional/Manual, Auto>**	**Designed By**	
Test Case Name	<FILL>	**Designed Date**	
Test Case ID	<FILL>	**Last Modified By**	
Test Priority	<Optional, Low, Medium, High>	**Last Modified Date**	
Description	<FILL>	**Executed By**	<FILL>
Final Results	PENDING FAIL PASS	**Execution Date**	<FILL>

Pre-Condition	<FILL>				
TC Step No.	**Setup** <list the pre-conditions to carry thru this TC>	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	<FILL>	<FILL>	<FILL>	☐	☐

2	<FILL>	<FILL>	<FILL>	☐	☐
Post-Condition	<FILL>				
Verified By	<FILL>				
Comments	<FILL>				
Screenshots	<FILL>				
Time Constraint	<FILL>				

Environment Configuration

Backend

Dependencies	Will use `pipenv` to install all dependencies required under virtual environment
Tools	*Postman* or other API testing tools
Type	Support both locally and online

Run Django server

Steps:

1. Pull the backend repository from GitHub
   a. You may use command `git clone <repo>` if you haven't cloned the repository before
   b. Or simply run `git pull` to fetch the latest commit
2. Make sure you've already installed `Python` and `pipenv`
3. Open a terminal window at the folder root directory
   a. Check current path using `pwd` command
   b. You should see a directory path like `.../COMP30022`
4. Run command `pipenv sync` to install all dependencies required for backend
5. Activate virtual environment, either by:
   a. Type in `pipenv shell` in terminal
   b. Open in VS Code and open a new terminal prompt (will automatically switch to virtual environment)
6. Run command `Python manage.py runserver` to run Django backend server,
   a. By default, Django server will run on port `8000`
   b. Can change the server's port by `python manage.py runserver <port number>`
   c. Correctly running server will look like:

```
Performing system checks...

System check identified no issues (0 silenced).
October 06, 2021 - 02:53:23
Django version 3.2.7, using settings 'backend.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Can skip steps 1-4 after setting up the environment for Django

Frontend

Dependencies	Will use `yarn` to install all packages and run server
Type	Support both locally and online

Run frontend server

Steps:

1. Pull the backend repository from GitHub
    a. You may use command `git clone <repo>` if you haven't cloned the repository before
    b. Or simply run `git pull` to fetch the latest commit
2. Make sure you've already installed `yarn`
3. Open a terminal window at the folder root directory
    a. Check current path using `pwd` command
    b. You should see a directory path like `.../COMP30022-Frontend`
4. Run command `yarn install` to install all dependencies required for frontend
5. Run command `yarn start` to run yarn frontend server,
    a. By default, server will run on port `3000`
    b. Correctly running server will look like should automatically open your browser.

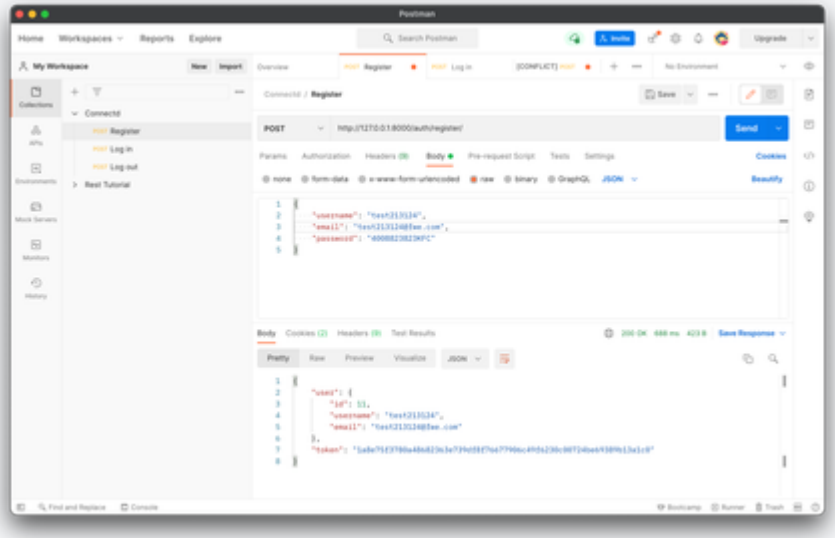
Test Cases

TC 1 Registration Functionality

TC 1.1 Django Registration Request

Test/Execution Type	**Manual**	**Designed By**	@ **Jackson Hu**
Test Case Name	Registration Request	**Designed Date**	15 Sep 2021
Test Case ID	1.1	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	04 Oct 2021
Description	Locally send a POST request to Django backend server to see if can successfully register new user in system	**Executed By**	@ Jackson Hu
Final Results	**PASS**	**Execution Date**	15 Sep 2021

Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Prepare a JSON file with fields:{"username", "email", "password"}			☑	☐
2	Fill in those fields with a username not pre-exist			☑	☐

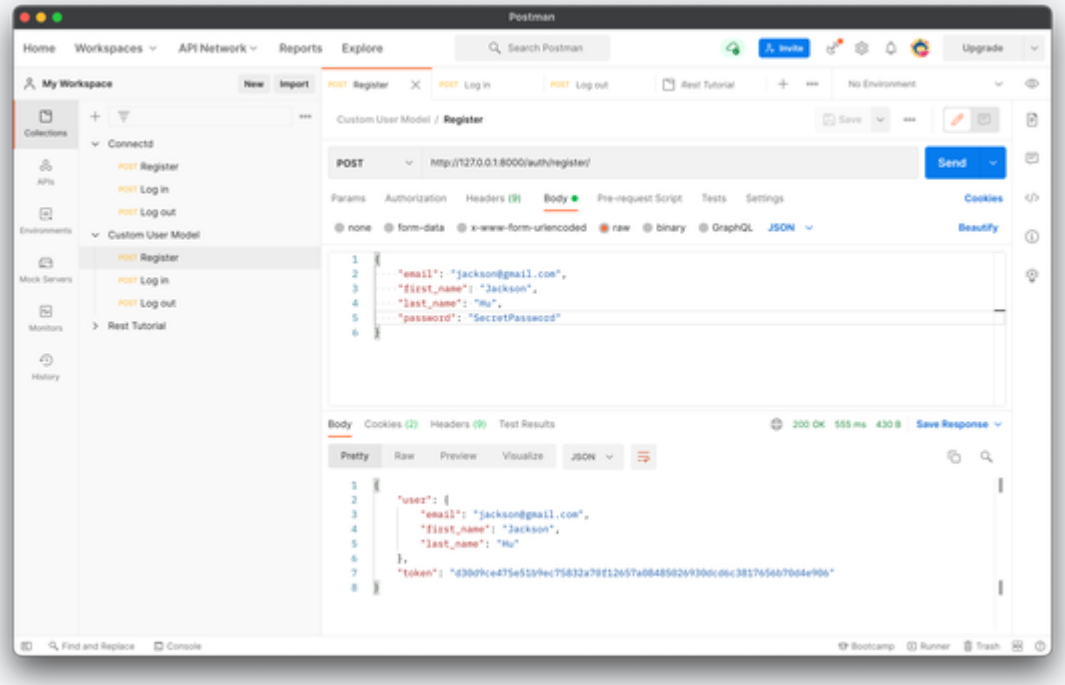
3	Post to “http://127.0.0.1:8000/auth/register/“	Return an HTTP response as JSON file with serialised user info and token	Expected JSON response received	☑	☐
Post-Conditon	Successfully register a new user with user info stored in database, corresponding token returned to indicate registration is successful				
Verified By	@ Jackson Hu				
Comments	Handles the POST request to register new user as expected				
Screenshots					
Time Constraint	< 1s				

TC 1.2 Django Registration Request

Test/Execution Type	**Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Registration Request	**Designed Date**	20 Sep 2021
Test Case ID	1.2	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	04 Oct 2021
Description	Locally send a POST request to Django backend server to see if can successfully register new user in system, now with our custom user model	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	20 Sep 2021

Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Prepare a JSON file with fields:{“email”, “first_name”, “last_name“, “password“}			☑	☐
2	Fill in those fields with a username not pre-exist			☑	☐

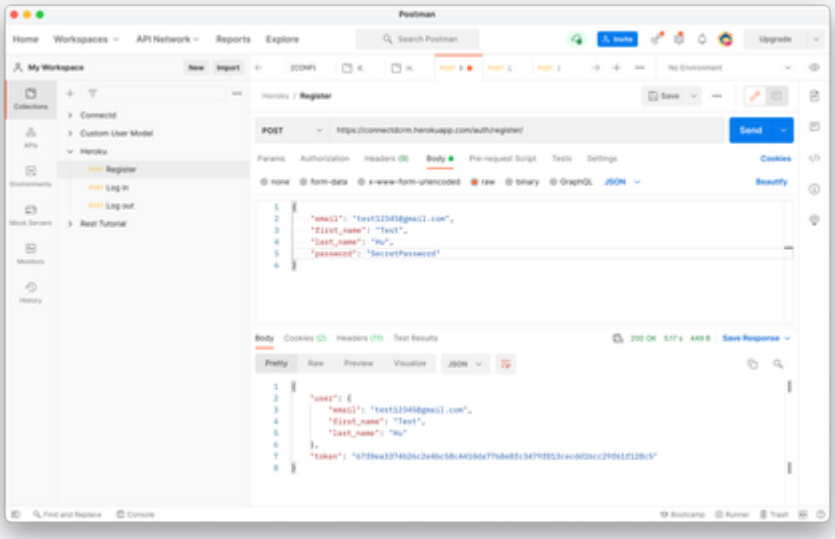
3	Post to “http://127.0.0.1:8000/auth/register/“	Return an HTTP response as JSON file with serialised user info and token	Expected JSON response received	☑	☐

Post-Condition	Successfully register a new user with user info stored in database, corresponding token returned to indicate registration is successful
Verified By	@ Jackson Hu
Comments	Now the registration works with email as unique user identifier
Screenshots	
Time Constraint	< 1s

TC 1.3 Django Registration Request

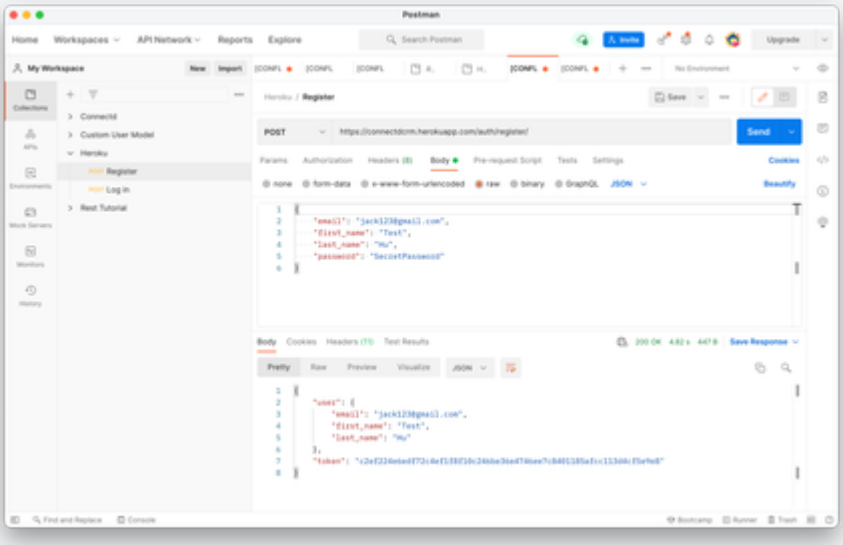
Test/Execution Type	**Manual**	**Designed By**	**@ Jackson Hu**
Test Case Name	Registration Request	**Designed Date**	07 Oct 2021
Test Case ID	1.3	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	08 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to see if can successfully register new user in system, now with our custom user model	**Executed By**	@ Jackson Hu
Final Results	**PASS**	**Execution Date**	07 Oct 2021

Pre-Condition	Django backend server deployed on Heroku

TC Step No.	Setup Ensure Django backend server is running on Heroku	Expected Results	Actual Results	Results	
				As Expected	Not As Expected
1	Create a POST request in Postman			☑	☐
2	Prepare a JSON file as the body of request with fields:{“email”, “first_name”, “last_name“, “password“}			☑	☐
2	Fill in those fields with a username not pre-exist			☑	☐
3	Post to “https://connectdcrm.herokuapp.com /auth/register/“	Return an HTTP response as JSON file with serialised user info and token	Expected JSON response received	☑	☐
Post-Conditon	Successfully register a new user with user info stored in database, corresponding token returned to indicate registration is successful				
Verified By	@ Jackson Hu				
Comments	Now the registration works on Django backend server deployed on Heroku				
Screenshots					
Time Constraint	< 1s				

TC 1.4 Django Registration Request

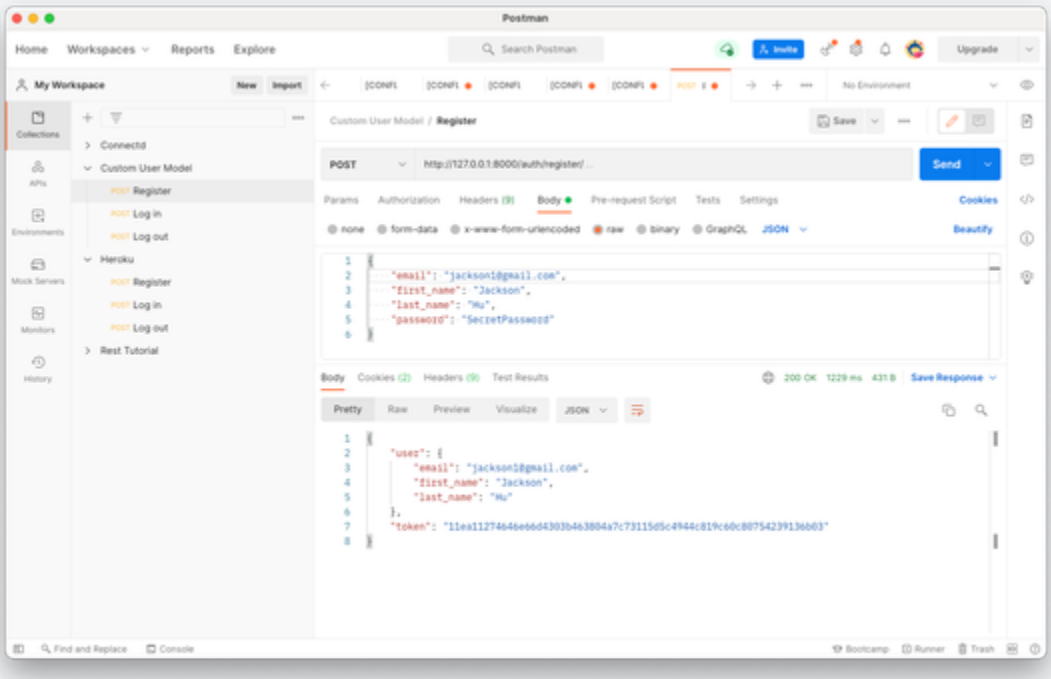
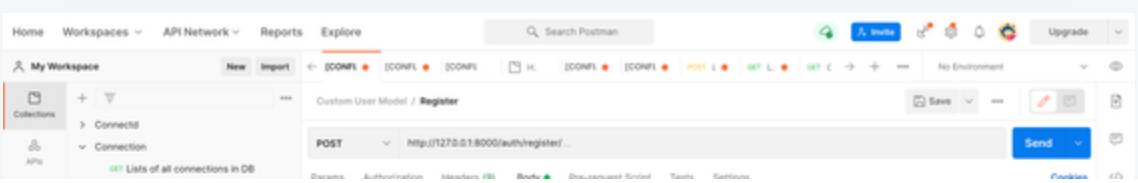
Test/Execution Type	Manual	Designed By	@ Jackson Hu
Test Case Name	Registration Request	**Designed Date**	11 Oct 2021
Test Case ID	1.4	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	11 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to see if can successfully register new user in system. Test if the issue of linking to MongoDB has been fixed.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	11 Oct 2021

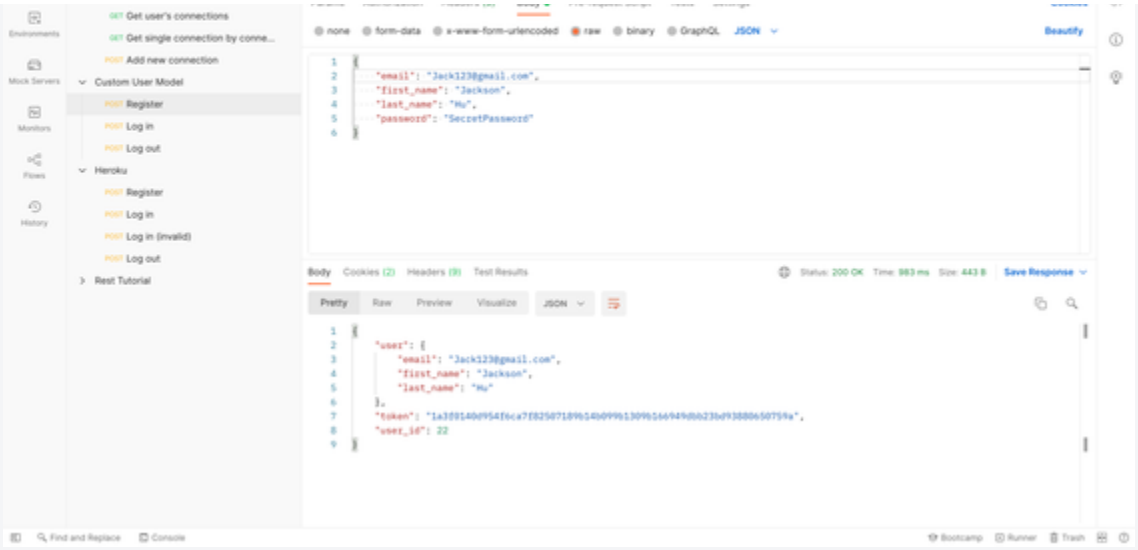
Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a POST request in Postman			☑	☐
2	Prepare a JSON file as the body of request with fields:{“email”, “first_name”, “last_name“, “password“}			☑	☐
2	Fill in those fields with a username not pre-exist			☑	☐
3	Post to “https://connectdcrm.herokuapp.com/auth/register/“	Return an HTTP response as JSON file with serialised user info and token	Expected JSON response received	☑	☐
Post-Conditon	Successfully register a new user with user info stored in database, corresponding token returned to indicate registration is successful				
Verified By	@ Jackson Hu				
Comments	Now the registration works on Django backend server deployed on Heroku. The problem with connecting to MongoDB has been fixed after modifying `config var` on Heroku				
Screenshots					
Time Constraint	< 1s				

TC 1.5 Django Registration Request

Test/Execution Type	**Manual**	**Designed By**	**@ Jackson Hu**
Test Case Name	Registration Request	**Designed Date**	14 Oct 2021
Test Case ID	1.5	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	14 Oct 2021
Description		**Executed By**	@ Jackson Hu

	Response of successful user registration now includes `user_id` as well.		
Final Results	**PASS**	**Execution Date**	14 Oct 2021

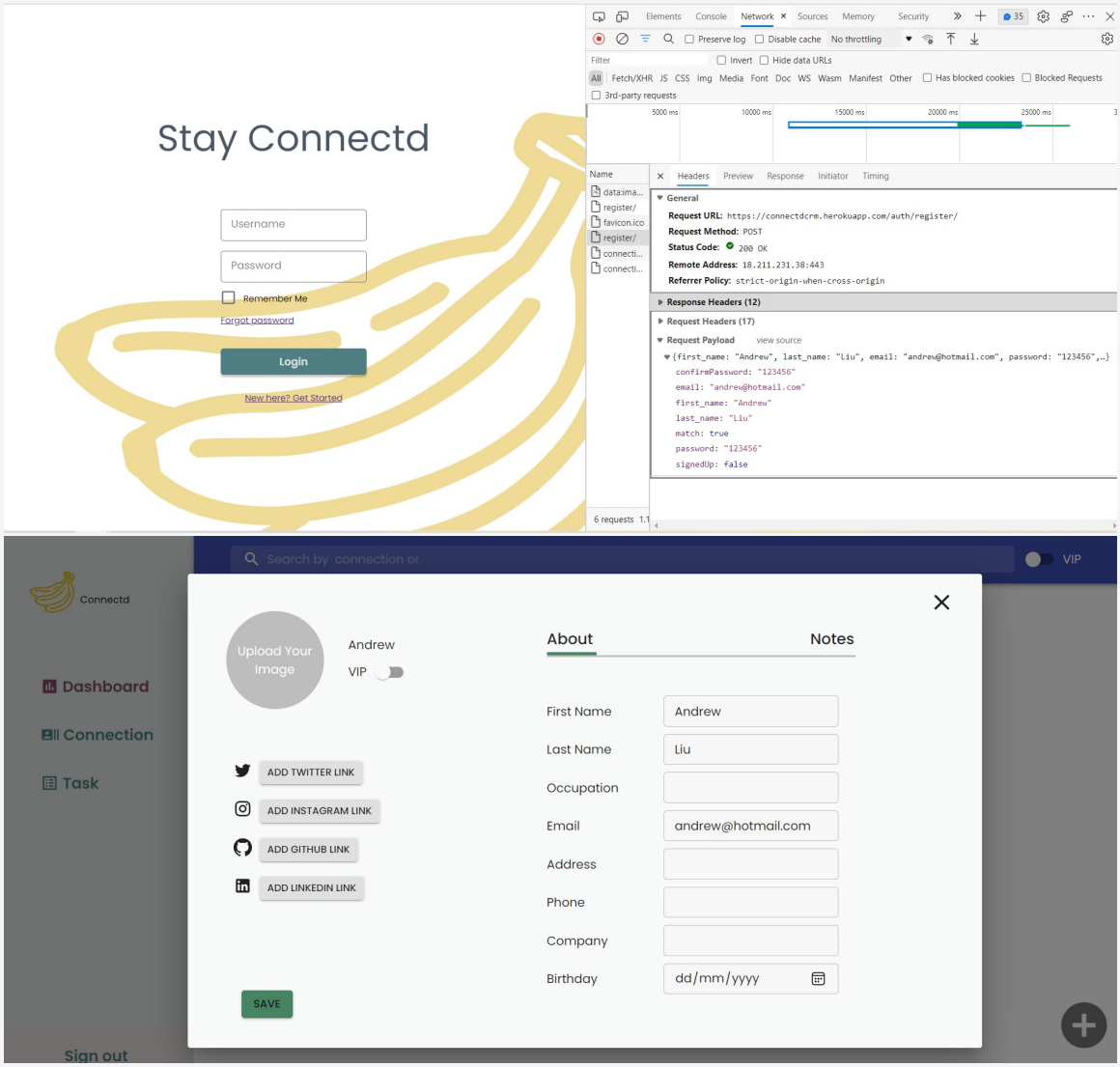
Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a POST request in Postman			☑	☐
2	Prepare a JSON file as the body of request with fields:{“email”, “first_name”, “last_name“, “password“}			☑	☐
2	Fill in those fields with a username not pre-exist			☑	☐
3	Post to “http://127.0.0.1:8000/auth/register/“	Return an HTTP response as JSON file with serialised user info, user token and corresponding user id	Expected JSON response received	☑	☐
Post-Condition	Successfully register a new user with user info stored in database, corresponding token returned to indicate registration is successful				
Verified By	@ Jackson Hu				
Comments	Now successful login will return a response containing `user` info, `token` and `user_id` for current user.				
Screenshots	  				

	
Time Constraint	< 1s

TC 1.6 User Registration

Test/Execution Type	**Manual**	**Designed By**	@ Han Liu
Test Case Name	New user registration	**Designed Date**	22/10/2021
Test Case ID	1.6	**Last Modified By**	@ Han Liu
Test Priority	High	**Last Modified Date**	22/10/2021
Description	1). Capability of record and send new user's information to backend. 2). A new user account has been created	**Executed By**	@ Han Liu
Final Results	PASS	**Execution Date**	22/10/2021

Pre-Condition	N/A				
TC Step No.	**Setup** N/A, only navigate to https://connectd-front.herokuapp.com/	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Fill in detail		<	☑	☐
2	Send detail to backend	correct data has been sent to backend	correct data has been sent to backend	☑	☐
3	Verify if user account have been created	a new user account has been created	a new user account has been created	☑	☐
Post-Condition	Successfully send new user detail to backend and create an account				
Verified By	@ Han Liu				
	Client can create account to use the application				

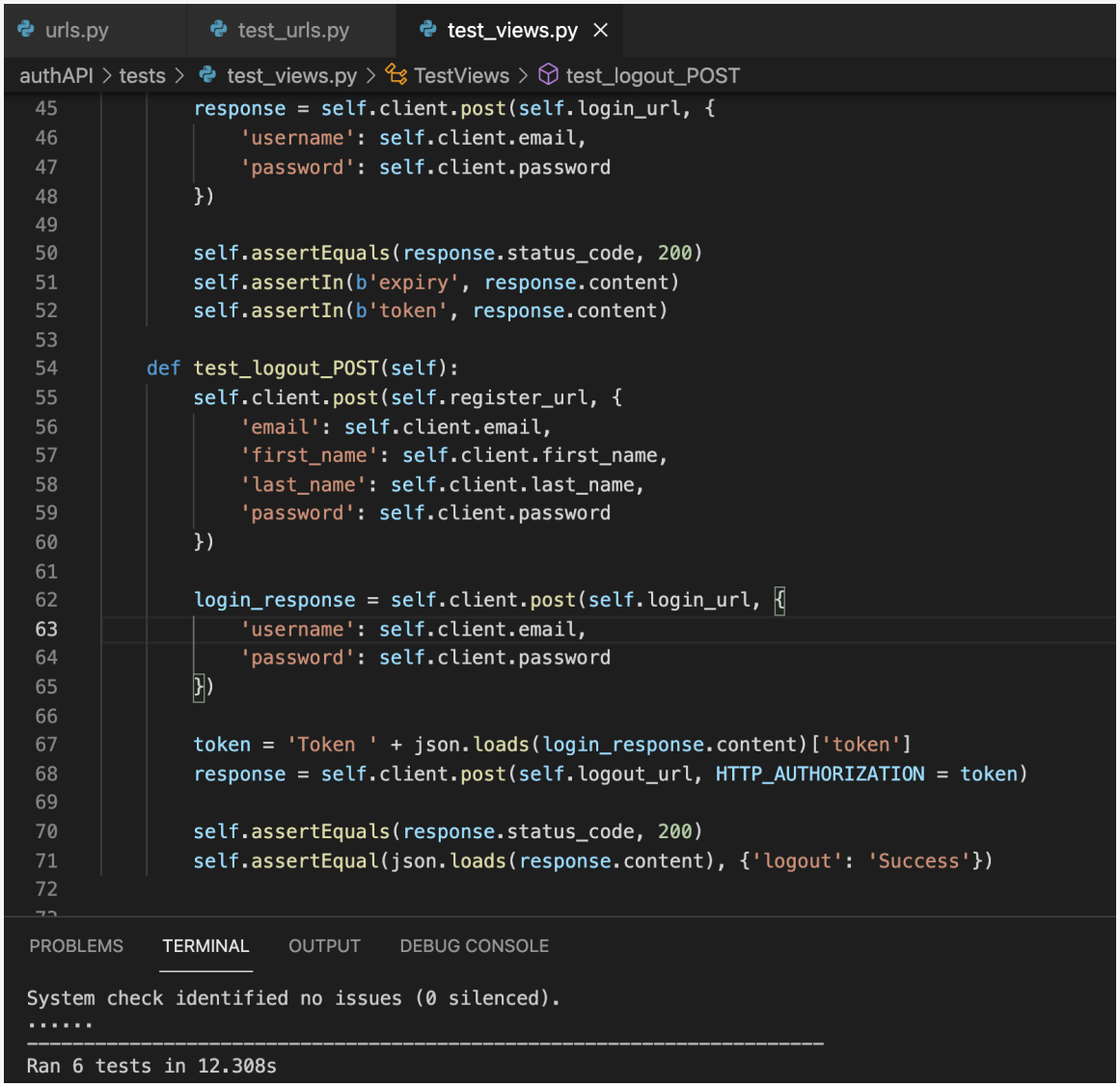
Comments	
Screenshots	
Time Constraint	N/A

TC 1.7 Django Registration Request (Script)

Test/Execution Type	**Unit / Automatic**	**Designed By**	@ Jackson Hu
Test Case Name	Django test script for registration	**Designed Date**	26 Oct 2021
Test Case ID	1.7	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	26 Oct 2021
Description	Run Django test script to automatically test user registration	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	26 Oct 2021

Pre-Condition	Ensure `pipenv` is correctly configured				
TC Step No.	**Setup** Activate virtual environment for backend	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Run command `python manage.py test authAPI` to execute test cases written for authAPI module			☑	☐

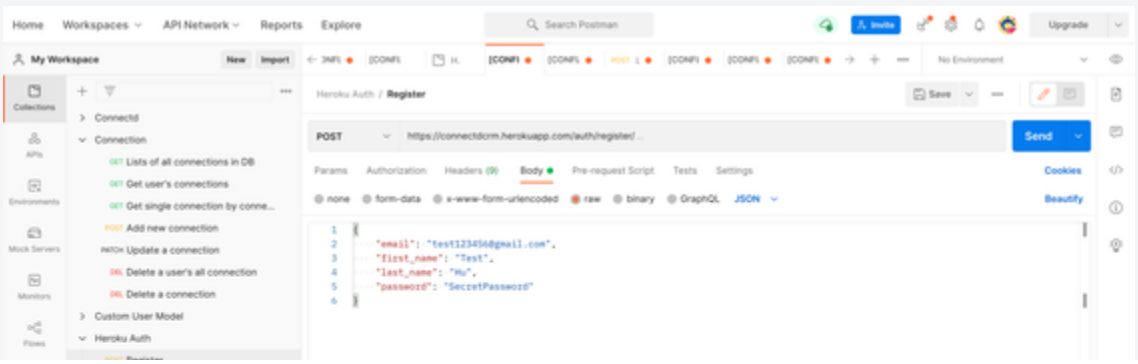
2	Check if all test cases are passed	Not raise any errors or exceptions	Successfully passed all test cases with message:
			`Ran <x> tests in <seconds>`
			`OK`
				☑	☐

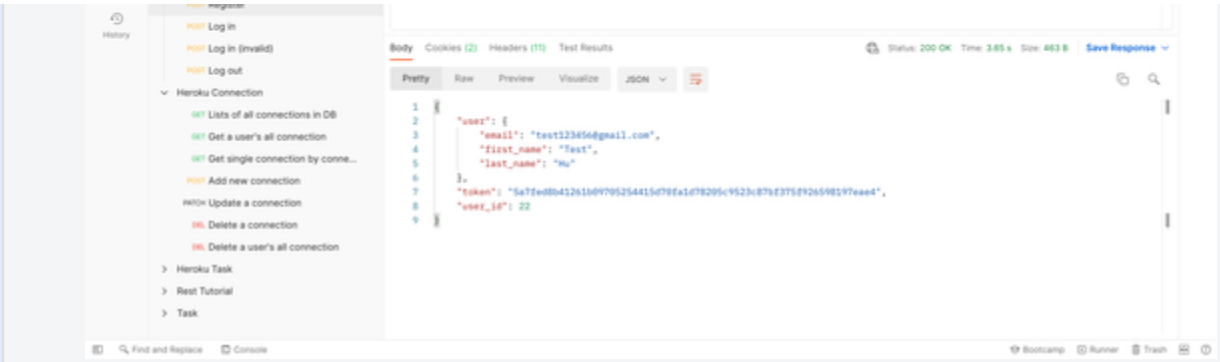
Post-Conditon	Successfully run all the test cases against `authAPI` module, temporary testing database destroyed.				
Verified By	@ Jackson Hu				
Comments	Now can easily run the test cases written for backend automatically.				
Screenshots					

	OK Destroying test database for alias 'default'... (COMP30022) jackson@Jacksons-MBP COMP30022 %
Time Constraint	< 20s

TC 1.8 Django Registration Request

Test/Execution Type	**Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Registration Request	**Designed Date**	28 Oct 2021
Test Case ID	1.8	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	28 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to see if can successfully register new user in system.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	28 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a POST request in Postman			☑	☐
2	Prepare a JSON file as the body of request with fields:{“email”, “first_name”, “last_name“, “password“}			☑	☐
2	Fill in those fields with a username not pre-exist			☑	☐
3	Post to “https://connectdcrm.herokuapp.com /auth/register/“	Return an HTTP response as JSON file with serialised user info and token	Expected JSON response received	☑	☐
Post-Conditon	Successfully register a new user with user info stored in database, corresponding token returned to indicate registration is successful				
Verified By	@ Jackson Hu				
Comments	Now the registration works on Django backend server deployed on Heroku. CORS has been activated.				
Screenshots					

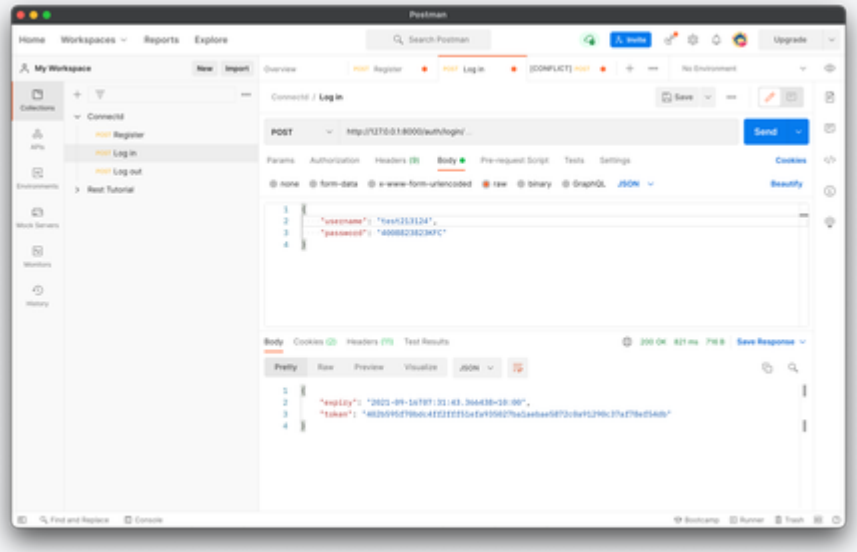
	
Time Constraint	< 1s

TC 2 Login Functionality

TC 2.1 Django Login Request

Test/Execution Type	**Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Login Request	**Designed Date**	15 Sep 2021
Test Case ID	2.1	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	04 Oct 2021
Description	Locally send a POST request to Django backend server to validate user login	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	15 Sep 2021

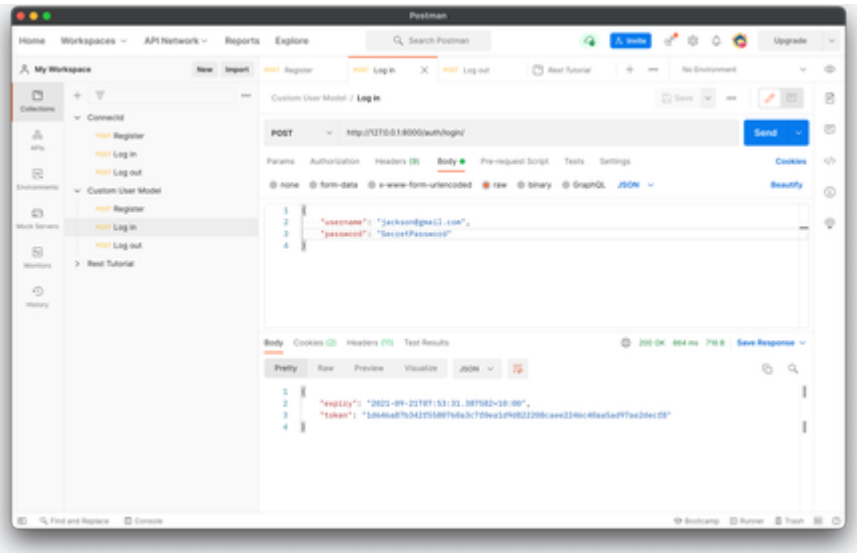
Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Prepare a JSON file with fields:{“username”, “password“}			☑	☐
2	Fill in those fields with an existing username and password			☑	☐
3	Post to “http://127.0.0.1:8000/auth/login/“	Return an HTTP response as JSON file with expiry time and user token	Expected JSON response received	☑	☐
Post-Conditon	Successfully log in an existing user with given password being validated against record stored in database				
Verified By	@ Jackson Hu				
Comments	Handles the POST request to validate and log in existing user				

Screenshots	
Time Constraint	< 1s

TC 2.2 Django Login Request

Test/Execution Type	**Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Login Request	**Designed Date**	20 Sep 2021
Test Case ID	2.2	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	04 Oct 2021
Description	Locally send a POST request to Django backend server to validate user login, now with our custom user model	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	20 Sep 2021

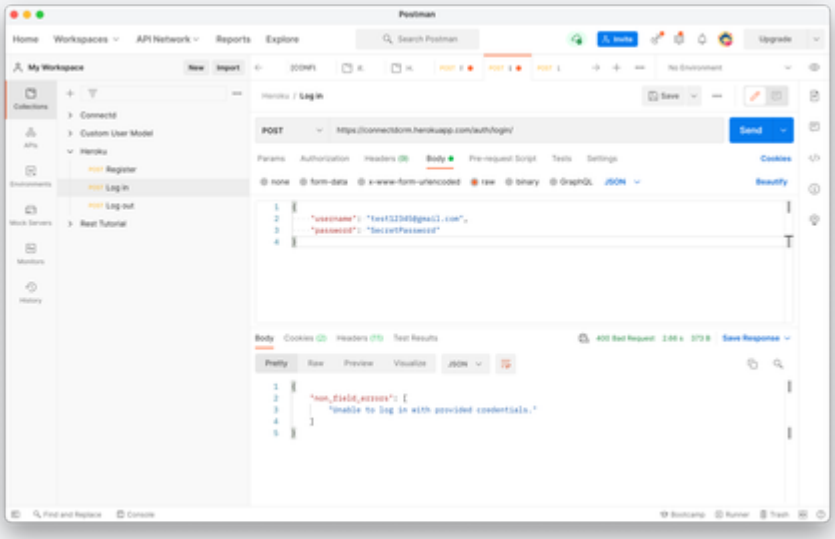
Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Prepare a JSON file with fields:{“username”, “password“}			☑	☐
2	Fill in those fields with an existing username and password			☑	☐
3	Post to “http://127.0.0.1:8000/auth/login/“	Return an HTTP response as JSON file with expiry time and user token	Expected JSON response received	☑	☐
Post-Conditon	Successfully log in an existing user with given password being validated against record stored in database				
Verified By	@ Jackson Hu				
Comments	Updated user model now use “email“ as the unique identifier, but here uses “username“ field as specified by Knox framework				

Screenshots	
Time Constraint	< 1s

TC 2.3 Django Login Request

Test/Execution Type	**Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Login Request	**Designed Date**	07 Oct 2021
Test Case ID	2.3	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	08 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to validate user login, now with our custom user model	**Executed By**	@ Jackson Hu
Final Results	FAIL	**Execution Date**	07 Oct 2021

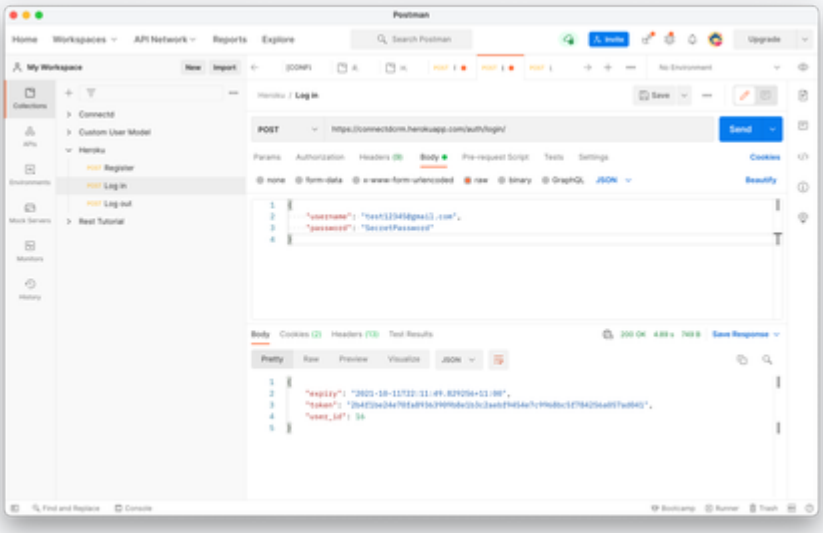
Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a POST request in Postman			☑	☐
2	Prepare a JSON file as the body of request with fields:{“username”, “password“}			☑	☐
3	Fill in those fields with an existing username and password			☑	☐
4	Post to “https://connectdcrm.herokuapp.com/auth/login/“	Return an HTTP response as JSON file with expiry time and user token	Expected JSON response received	☐	☑
Post-Conditon	Failed to log in an existing user with `"non_field_errors": ["Unable to log in with provided credentials."]`				
Verified By	@ Jackson Hu				
	First time tested with backend server deployed on Heroku, but seems like the database used by the server didn't sync with MongoDB.				

Comments	
Screenshots	
Time Constraint	< 1s

TC 2.4 Django Login Request

Test/Execution Type	**Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Login Request	**Designed Date**	09 Oct 2021
Test Case ID	2.4	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	10 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to validate user login. Now returned `user_id` in response as well.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	09 Oct 2021

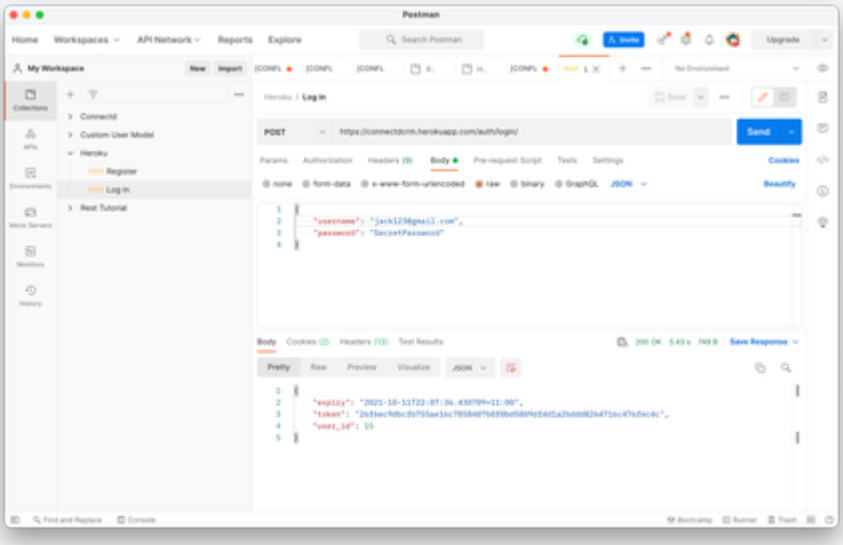
Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a POST request in Postman			☑	☐
2	Prepare a JSON file as the body of request with fields:{“username”, “password“}			☑	☐
3	Fill in those fields with an existing username and password			☑	☐
4	Post to “https://connectdcrm.herokuapp.com/auth/login/“	Return an HTTP response as JSON file with expiry time, user token and corresponding user id	Expected JSON response received	☑	☐
	Successfully log in an existing user with given password being validated against record stored in database.				

Post-Conditon	Now returned `user_id` for current user as @ Han Liu suggested.
Verified By	@ Jackson Hu
Comments	Solved the problem of Heroku not syncing with MongoDB by: 1. Remove the `postgresql` add on which Heroku automatically added. 2. Replace `DATABASE_URL` in `config var` with `MONGODB_URI` to configure our MongoDB as primary database.
Screenshots	 
Time Constraint	< 1s

TC 2.5 Django Login Request

Test/Execution Type	**Manual**	**Designed By**	**@ Jackson Hu**
Test Case Name	Login Request	**Designed Date**	11 Oct 2021
Test Case ID	2.5	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	11 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to validate user login. Now returned `user_id` in response as well.	**Executed By**	@ Jackson Hu
Final Results	**PASS**	**Execution Date**	11 Oct 2021

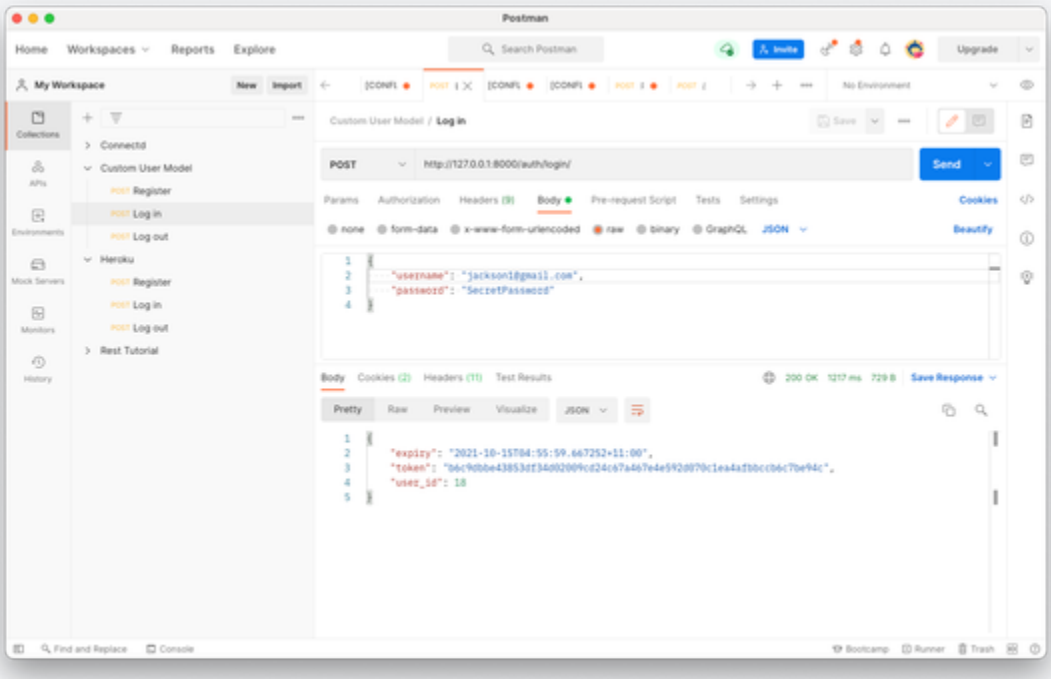
Pre-Condition	Django backend server deployed on Heroku			
	Setup	**Expected Results**	**Actual Results**	**Results**

TC Step No.	Ensure Django backend server is running on Heroku			As Expected	Not As Expected
1	Create a POST request in Postman			☑	☐
2	Prepare a JSON file as the body of request with fields:{“username”, “password“}			☑	☐
3	Fill in those fields with an existing username and password			☑	☐
4	Post to “https://connectdcrm.herokuapp.com /auth/login/“	Return an HTTP response as JSON file with expiry time, user token and corresponding user id	Expected JSON response received	☑	☐
Post-Conditon	Successfully log in an existing user with given password being validated against record stored in database.				
Verified By	@ Jackson Hu				
Comments	Now successful login will return a response containing `expiry`, `token` and `user_id` for current user.				
Screenshots					
Time Constraint	< 1s				

TC 2.6 Django Login Request

Test/Execution Type	Manual	Designed By	@ Jackson Hu
Test Case Name	Login Request	**Designed Date**	14 Oct 2021
Test Case ID	2.6	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	14 Oct 2021
Description	Locally send a POST request to Django backend server to validate user login.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	14 Oct 2021

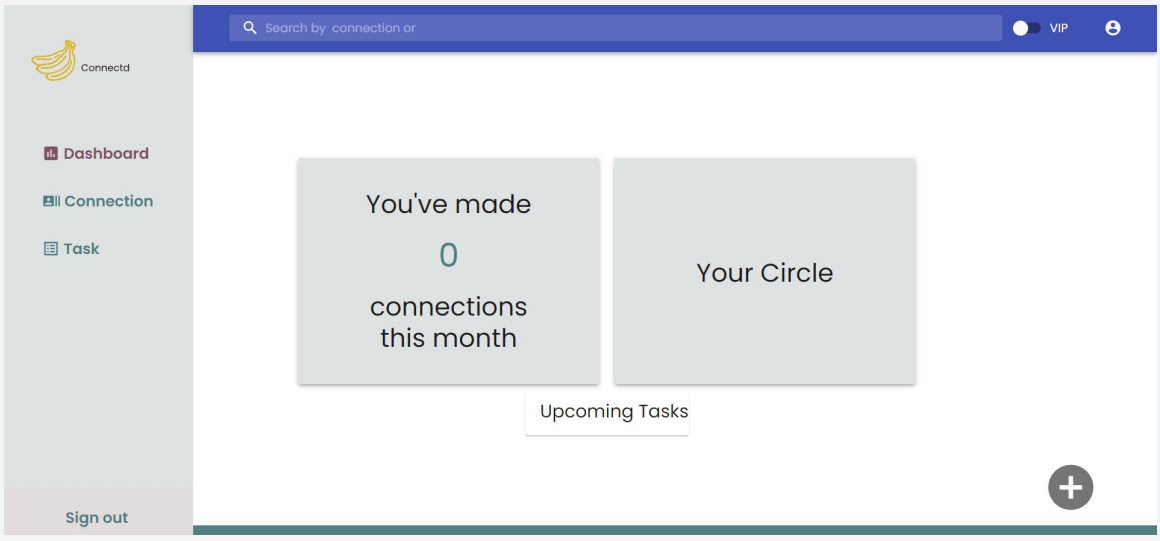
	Locally running Django backend with port:8000

Pre-Condition					
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a POST request in Postman			☑	☐
2	Prepare a JSON file as the body of request with fields:{"`username`", "`password`"}			☑	☐
3	Fill in those fields with an existing `username` and `password`			☑	☐
4	Post to "http://127.0.0.1:8000/auth/login/"	Return an HTTP response as JSON file with expiry time, user token and corresponding user id	Expected JSON response received	☑	☐
Post-Condition	Successfully log in an existing user with given password being validated against record stored in database.				
Verified By	@ Jackson Hu				
Comments	Routine test, successful login will return a response containing `expiry`, `token` and `user_id` for current user.				
Screenshots					
Time Constraint	< 1s				

TC 2.7 User Login

Test/Execution Type	**Manual**	**Designed By**	@ Han Liu
Test Case Name	User Login	**Designed Date**	22/10/2021

Test Case ID	2.7	**Last Modified By**	@ Han Liu
Test Priority	High	**Last Modified Date**	22/10/2021
Description	Testing client can log in the application with their account	**Executed By**	@ Han Liu
Final Results	PASS	**Execution Date**	22/10/2021

Pre-Condition	N/A				
TC Step No.	**Setup** N/A, only navigate to https://connectd-front.herokuapp.com/	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Fill in log in detail			☑	☐
2	Check if user logged into their account	User is directed to Dashboard page	User is directed to Dashboard page	☑	☐
Post-Condition	N/A				
Verified By	@ Han Liu				
Comments	User successfully logged into dashboard page				
Screenshots					
Time Constraint	N/A				

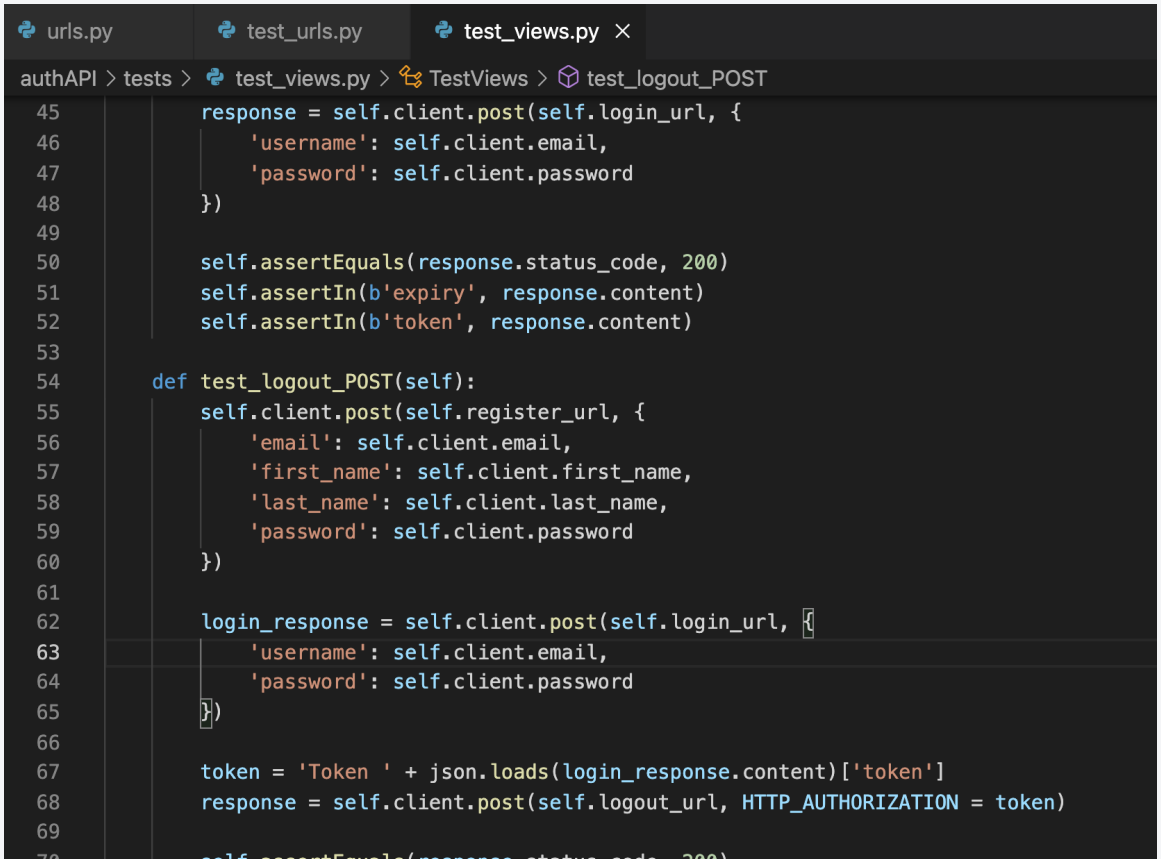
TC 2.8 Django Login Request (Script)

Test/Execution Type	**Unit / Automatic**	**Designed By**	@ Jackson Hu
Test Case Name	Django test script for login	**Designed Date**	26 Oct 2021

Test Case ID	2.8	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	26 Oct 2021
Description	Run Django test script to automatically test login	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	26 Oct 2021

Pre-Condition	Ensure `pipenv` is correctly configured				
TC Step No.	**Setup** Activate virtual environment for backend	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Run command `python manage.py test authAPI` to execute test cases written for authAPI module			☑	☐

2	Check if all test cases are passed	Not raise any errors or exceptions	Successfully passed all test cases with message:
			`Ran <x> tests in <seconds>`
			`OK`
				☑	☐

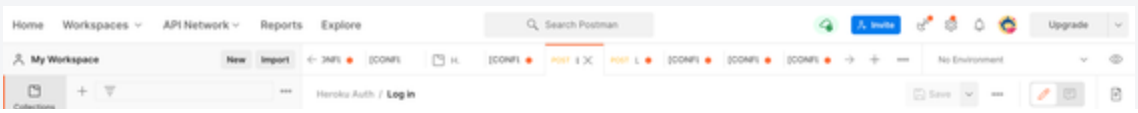
Post-Conditon	Successfully run all the test cases against `authAPI` module, temporary testing database destroyed.				
Verified By	@ Jackson Hu				
Comments	Now can easily run the test cases written for backend automatically.				
Screenshots					

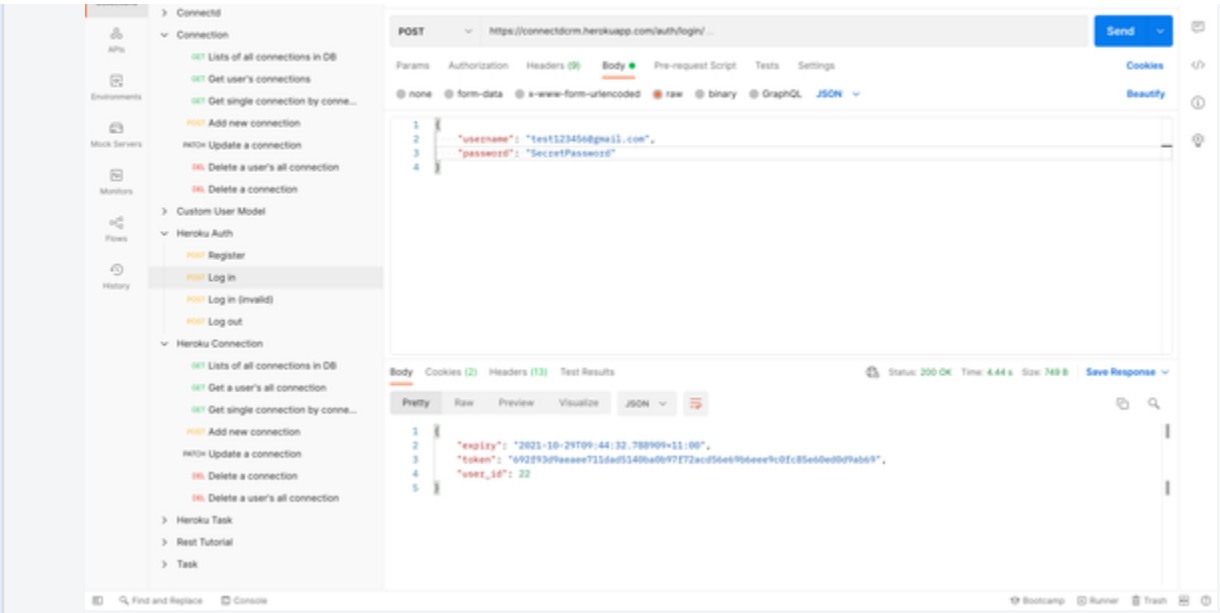
```
        self.assertEqual(response.status_code, 200)
        self.assertEqual(json.loads(response.content), {'logout': 'Success'})

PROBLEMS  TERMINAL  OUTPUT  DEBUG CONSOLE

System check identified no issues (0 silenced).
......
----------------------------------------------------------------------
Ran 6 tests in 12.308s

OK
Destroying test database for alias 'default'...
(COMP30022) jackson@Jacksons-MBP COMP30022 %
```

Time Constraint	< 20s

TC 2.9 Django Login Request

Test/Execution Type	**Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Login Request	**Designed Date**	28 Oct 2021
Test Case ID	2.9	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	28 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to validate user login.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	28 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a POST request in Postman			☑	☐
2	Prepare a JSON file as the body of request with fields:{“username”, “password“}			☑	☐
3	Fill in those fields with an existing username and password			☑	☐
4	Post to “https://connectdcrm.herokuapp.com /auth/login/“	Return an HTTP response as JSON file with expiry time, user token and corresponding user id	Expected JSON response received	☑	☐
Post-Condition	Successfully log in an existing user with given password being validated against record stored in database.				
Verified By	@ Jackson Hu				
Comments	Now successful login will return a response containing `expiry`, `token` and `user_id` for current user. CORS has been activated.				
Screenshots					

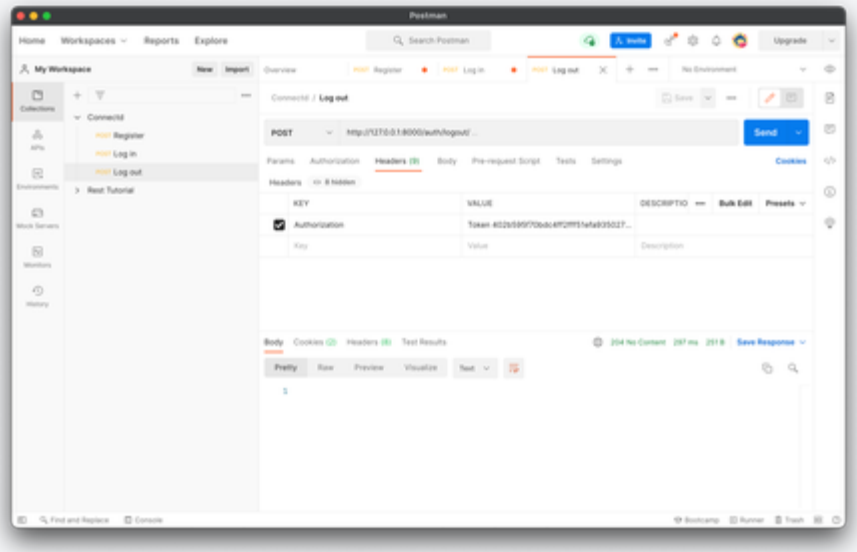
	
Time Constraint	< 5s

TC 3 Logout Functionality

TC 3.1 Django Logout Request

Test/Execution Type	**Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Logout Request	**Designed Date**	15 Sep 2021
Test Case ID	3.1	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	04 Oct 2021
Description	Locally send a POST request to Django backend server to verify user login status and logout	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	15 Sep 2021

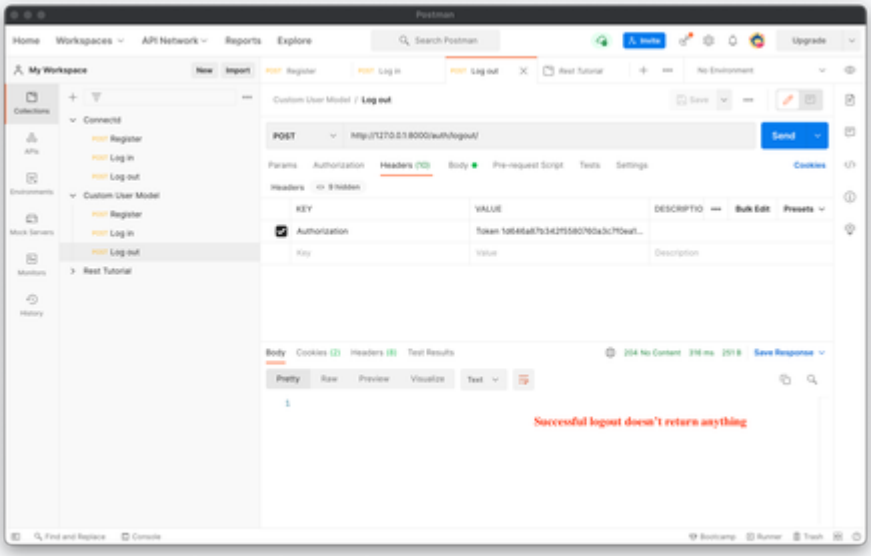
Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Add a “Authorization“ key in HTTP header			☑	☐
2	Put corresponding user token as value which start with a “Token“ prefix			☑	☐
3	Post to “http://127.0.0.1:8000/auth/logout/“	Return empty response to indicate successful logout	Expected empty response received	☑	☐

Post-Conditon	Successfully log out a user which currently logged in
Verified By	@ Jackson Hu
Comments	Handles the POST request to log out
Screenshots	
Time Constraint	< 1s

TC 3.2 Django Logout Request

Test/Execution Type	**Manual**	**Designed By**	**@ Jackson Hu**
Test Case Name	Logout Request	**Designed Date**	20 Sep 2021
Test Case ID	3.2	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	04 Oct 2021
Description	Locally send a POST request to Django backend server to verify user login status and logout, now with our custom user model	**Executed By**	@ Jackson Hu
Final Results	**PASS**	**Execution Date**	20 Sep 2021

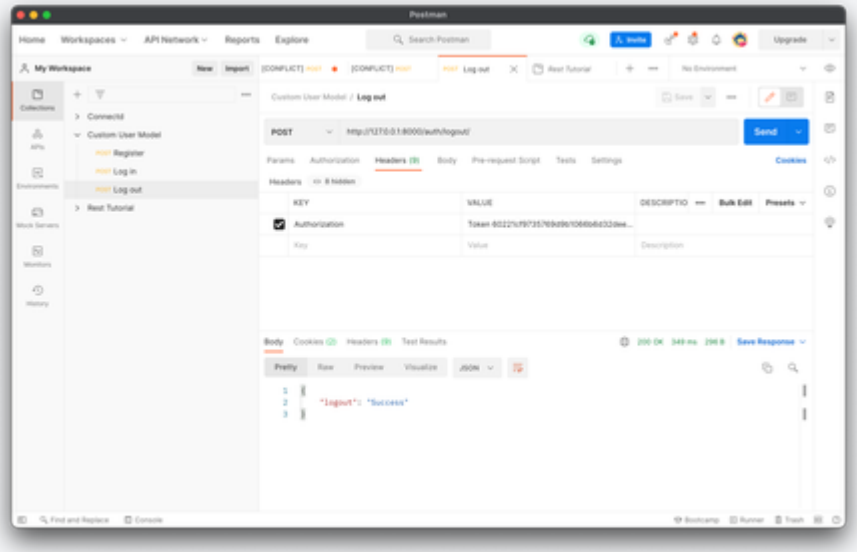
Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Add a “Authorization“ key in HTTP header			☑	☐
2	Put corresponding user token as value which start with a “Token“ prefix			☑	☐
3	Post to “http://127.0.0.1:8000/auth/logout/“	Return empty response to indicate successful logout	Expected empty response received	☑	☐
	Successfully log out a user which currently logged in				

Post-Conditon	
Verified By	@ Jackson Hu
Comments	Handles the POST request to log out
Screenshots	
Time Constraint	< 1s

TC 3.3 Django Logout Request

Test/Execution Type	**Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Logout Request	**Designed Date**	23 Sep 2021
Test Case ID	3.3	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	04 Oct 2021
Description	Locally send a POST request to Django backend server to verify user login status and logout, now with a descriptive response message	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	23 Sep 2021

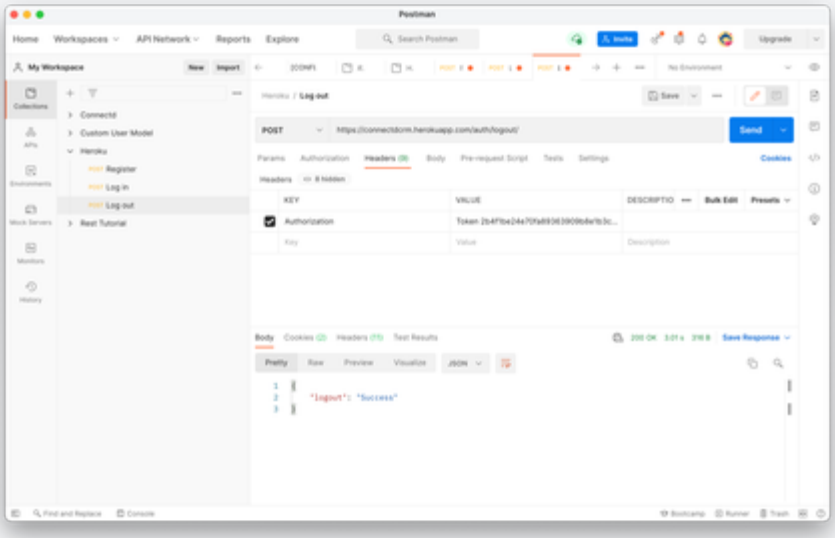
Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Add a “Authorization“ key in HTTP header			☑	☐
2	Put corresponding user token as value which start with a “Token“ prefix			☑	☐
3	Post to “http://127.0.0.1:8000/auth/logout/“	Return JSON format response like “logout“: “success“	{“logout“: “success“}	☑	☐

Post-Conditon	Successfully log out a user which currently logged in
Verified By	@ Jackson Hu
Comments	As suggested by @ Han Liu , logout request now will send back a descriptive response message indicating whether operation is successful or not
Screenshots	
Time Constraint	< 1s

TC 3.4 Django Logout Request

Test/Execution Type	**Manual**	**Designed By**	**@ Jackson Hu**
Test Case Name	Logout Request	**Designed Date**	09 Oct 2021
Test Case ID	3.4	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	10 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to verify user login status and logout. Test if the issue of linking to MongoDB has been fixed.	**Executed By**	@ Jackson Hu
Final Results	**PASS**	**Execution Date**	09 Oct 2021

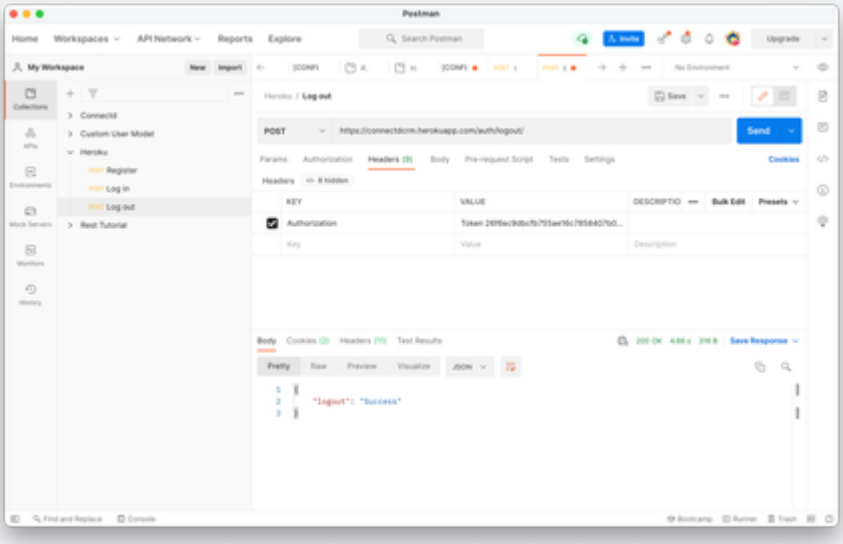
Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a POST request in Postman			☑	☐
2	Add a “Authorization“ key in HTTP header			☑	☐
3				☑	☐

	Put corresponding user token as value which start with a “Token“ prefix				
4	Post to “https://connectdcrm.herokuapp.com /auth/logout/“	Return JSON format response like “logout“: “success“	{“logout“: “success“}	☑	☐
Post-Conditon	Successfully log out a user which currently logged in				
Verified By	@ Jackson Hu				
Comments	Now the logout works on Django backend server deployed on Heroku. The problem with connecting to MongoDB has been fixed after modifying `config var` on Heroku				
Screenshots	 				
Time Constraint	< 1s				

TC 3.5 Django Logout Request

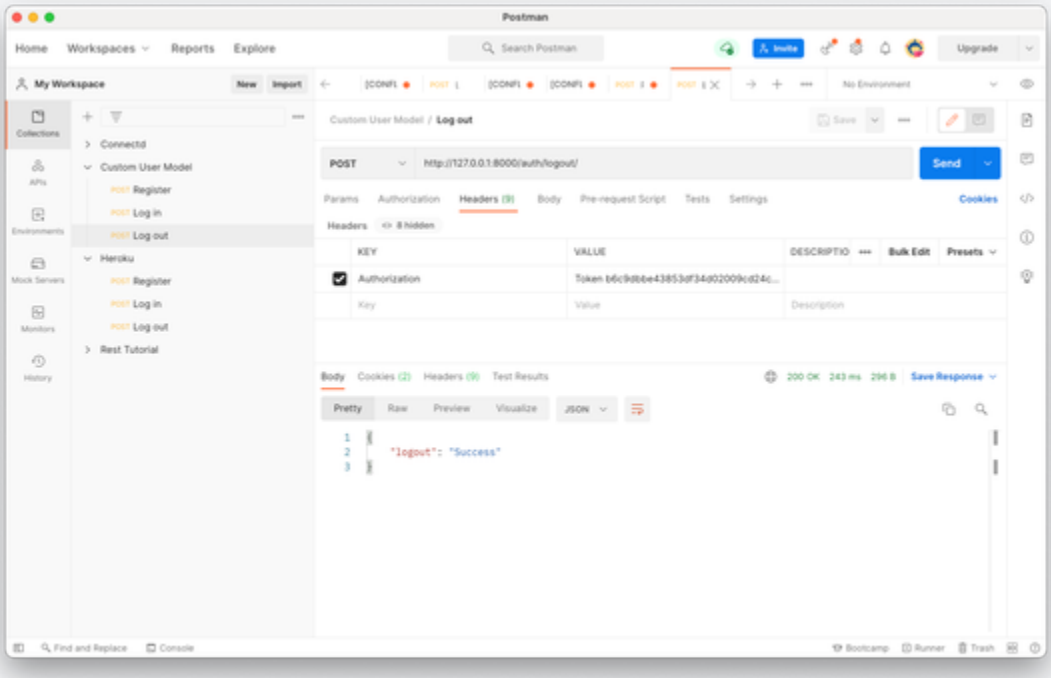
Test/Execution Type	**Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Logout Request	**Designed Date**	11 Oct 2021
Test Case ID	3.5	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	11 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to verify user login status and logout.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	11 Oct 2021

	Django backend server deployed on Heroku

Pre-Condition					
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a POST request in Postman			☑	☐
2	Add a “Authorization“ key in HTTP header			☑	☐
3	Put corresponding user token as value which start with a “Token“ prefix			☑	☐
4	Post to “https://connectdcrm.herokuapp.com /auth/logout/“	Return JSON format response like “logout“: “success“	{“logout“: “success“}	☑	☐
Post-Conditon	Successfully log out a user which currently logged in				
Verified By	@ Jackson Hu				
Comments	Now the logout works on Django backend server deployed on Heroku. Database correctly configured to sync with MongoDB after last test.				
Screenshots					
Time Constraint	< 1s				

TC 3.6 Django Logout Request

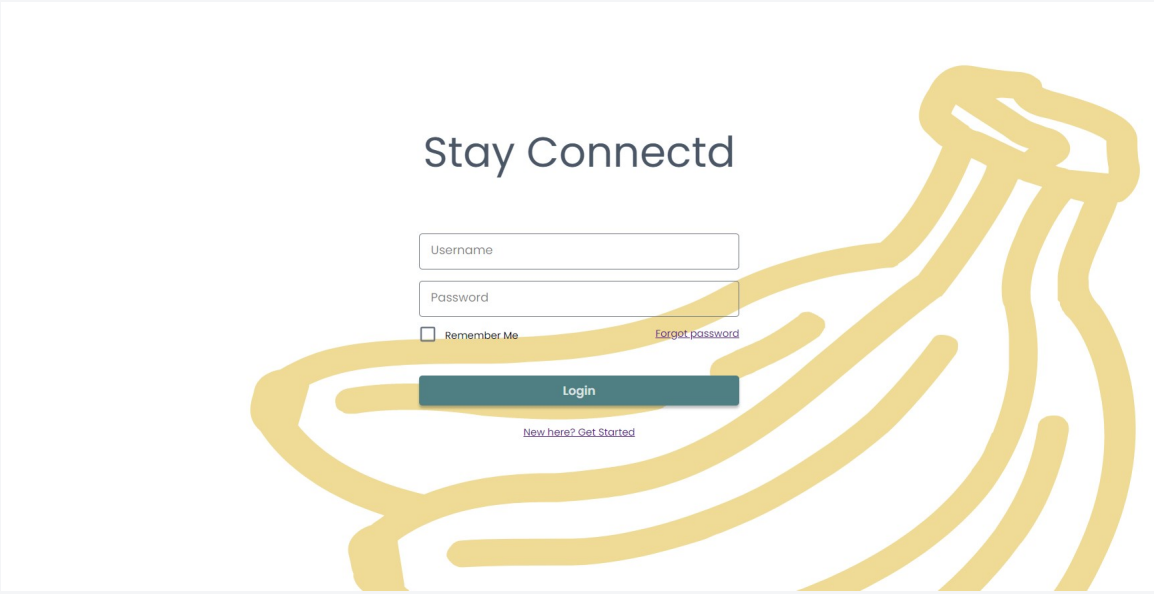
Test/Execution Type	**Manual**	**Designed By**	**@ Jackson Hu**
Test Case Name	Logout Request	**Designed Date**	14 Oct 2021
Test Case ID	3.5	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	14 Oct 2021
Description	Locally send a POST request to Django backend server to verify user login status and logout.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	14 Oct 2021

Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a POST request in Postman			☑	☐
2	Add a "`Authorization`" key in HTTP header			☑	☐
3	Put corresponding user token as value which start with a "`Token`" prefix			☑	☐
4	Post to "http://127.0.0.1:8000/auth/logout/"	Return JSON format response like "`logout`": "`success`"	`{"logout": "success"}`	☑	☐
Post-Condition	Successfully log out a user which currently logged in				
Verified By	@ Jackson Hu				
Comments	Routine test.				
Screenshots					
Time Constraint	< 1s				

TC 3.7 User Logout

Test/Execution Type	**Manual**	**Designed By**	@ Han Liu
Test Case Name	User logout	**Designed Date**	22/10/2021
Test Case ID	TC 3.7	**Last Modified By**	

			@ Han Liu
Test Priority	High	**Last Modified Date**	22/10/2021
Description	<FILL>	**Executed By**	@ Han Liu
Final Results	PASS	**Execution Date**	22/10/2021

Pre-Condition	User already logged in				
TC Step No.	**Setup** navigate to https://connectd-front.herokuapp.com/ and logged into account	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Click <Sign out> button to log out	redirect to login page	redirect to login page	☑	☐
2	Try to visit other pages when logged out	Unable to visit any pages other than greeting, sign in & register pages	Unable to visit any pages other than greeting, sign in & register pages	☐	☐
Post-Condition	N/A				
Verified By	@ Han Liu				
Comments	once logged out, user is unable to visit confidential pages				
Screenshots	Attempts to visit other page will always redirect user to login page 				
Time Constraint	N/A				

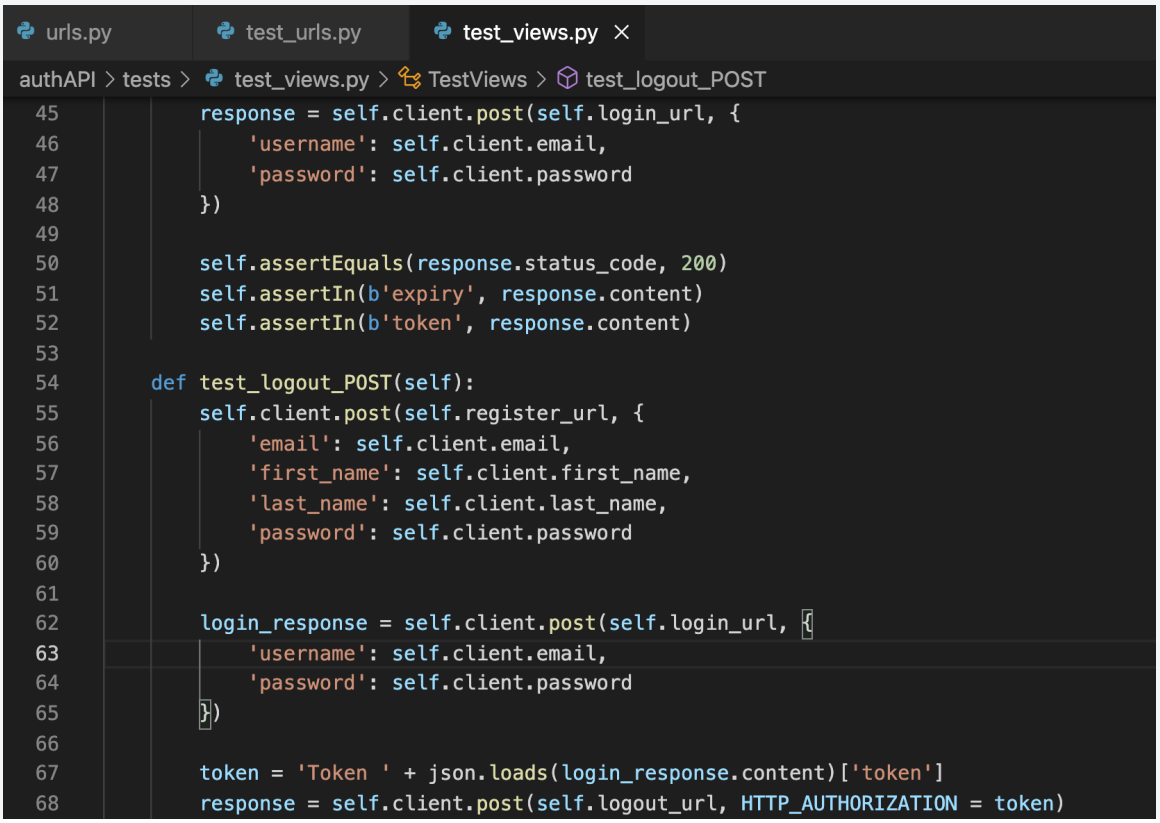
TC 3.8 Django Logout Request (Script)

Test/Execution Type	**Unit / Automatic**	**Designed By**	@ Jackson Hu

Test Case Name	Django test script for logout	**Designed Date**	26 Oct 2021
Test Case ID	3.8	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	26 Oct 2021
Description	Run Django test script to automatically test logout	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	26 Oct 2021

Pre-Condition	Ensure `pipenv` is correctly configured				
TC Step No.	**Setup** Activate virtual environment for backend	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Run command `python manage.py test authAPI` to execute test cases written for authAPI module			☑	☐

2	Check if all test cases are passed	Not raise any errors or exceptions	Successfully passed all test cases with message:
			`Ran <x> tests in <seconds>`
			`OK`
				☑	☐

Post-Conditon	Successfully run all the test cases against `authAPI` module, temporary testing database destroyed.				
Verified By	@ Jackson Hu				
Comments	Now can easily run the test cases written for backend automatically.				
Screenshots					

```
69
70          self.assertEquals(response.status_code, 200)
71          self.assertEqual(json.loads(response.content), {'logout': 'Success'})
72
```

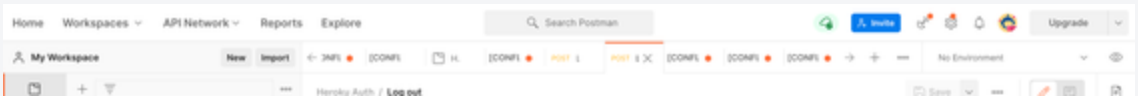
PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE

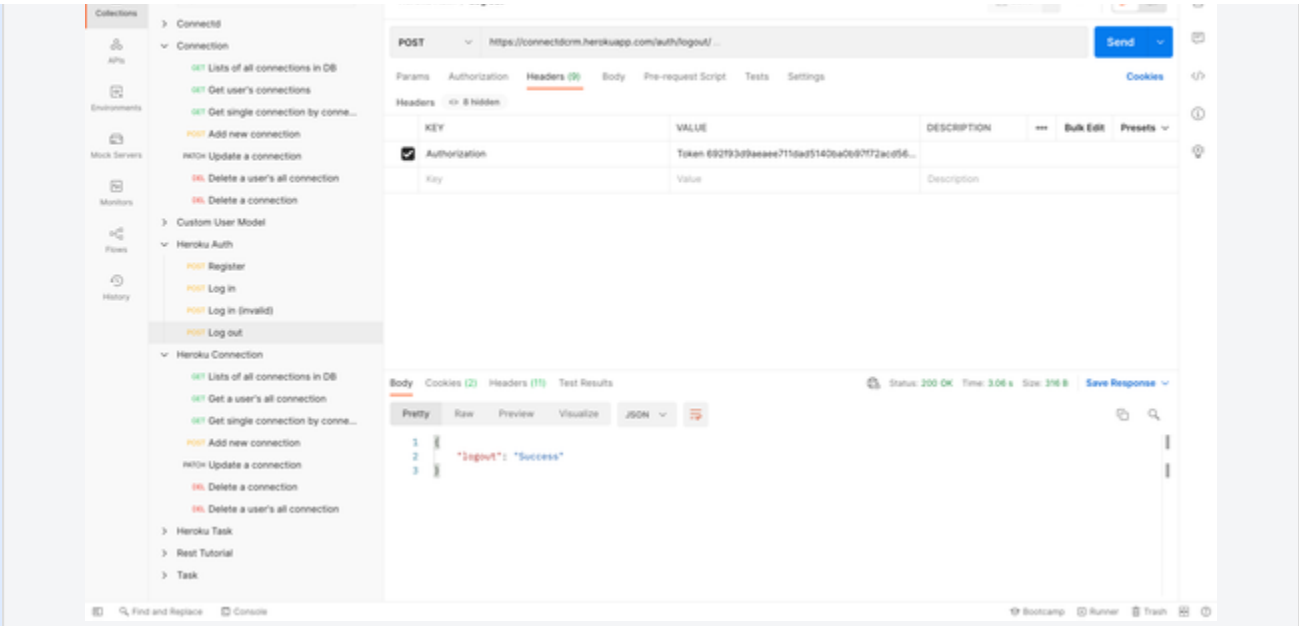
```
System check identified no issues (0 silenced).
......
----------------------------------------------------------------------
Ran 6 tests in 12.308s

OK
Destroying test database for alias 'default'...
(COMP30022) jackson@Jacksons-MBP COMP30022 %
```

Time Constraint	< 20s

TC 3.9 Django Logout Request

Test/Execution Type	**Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Logout Request	**Designed Date**	28 Oct 2021
Test Case ID	3.9	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	28 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to verify user login status and logout.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	28 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **POST** request in Postman			☑	☐
2	Add a "`Authorization`" key in HTTP header			☑	☐
3	Put corresponding user token as value which start with a "`Token`" prefix			☑	☐
4	Post to "https://connectdcrm.herokuapp.com/auth/logout/"	Return JSON format response like "`logout`": "`success`"	`{"logout": "success"}`	☑	☐
Post-Condition	Successfully log out a user which currently logged in				
Verified By	@ Jackson Hu				
Comments	Routine test. CORS has been activated.				
Screenshots					

	
Time Constraint	< 3s

TC 4 Connections Testing

TC 4.1 Connection Model

TC 4.1.1 Update Model - Include selfId

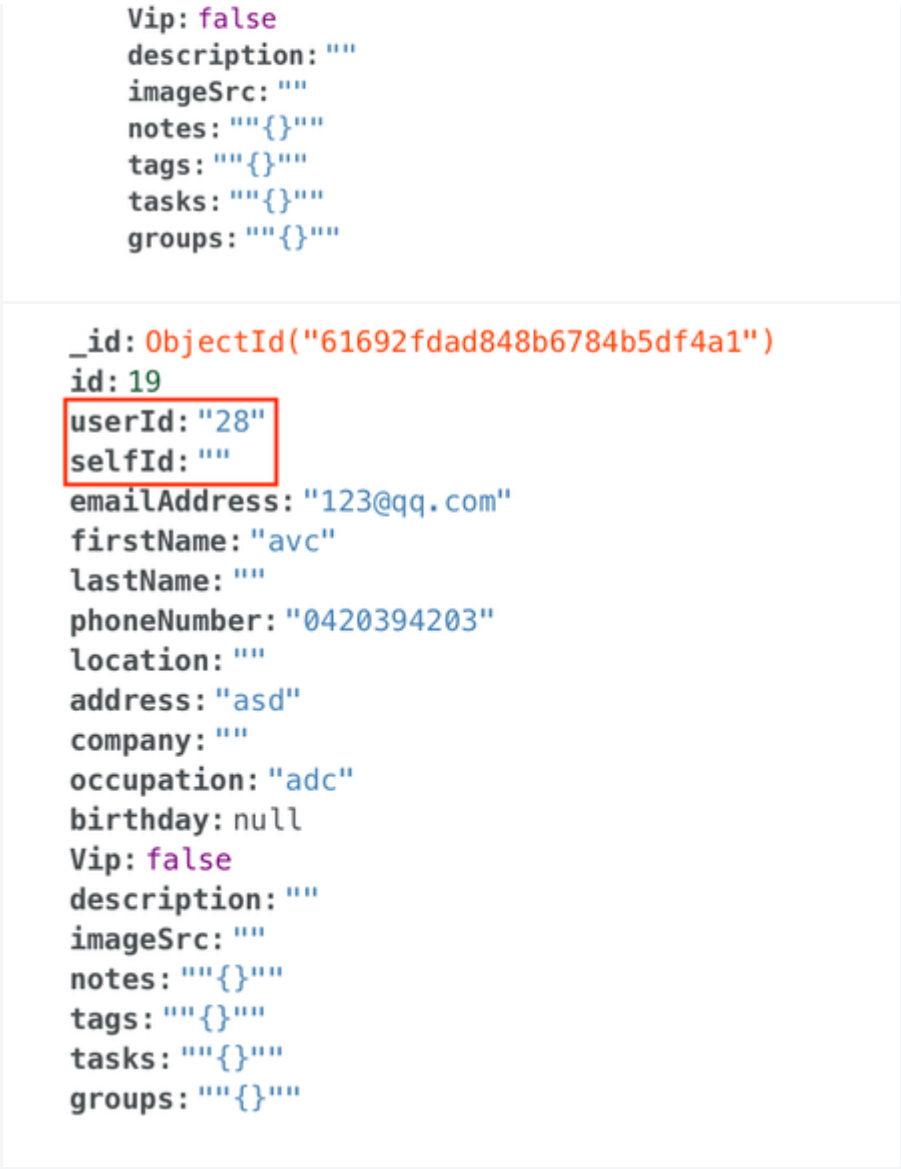
Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Update Model - Include `selfId`	**Designed Date**	15 Oct 2021
Test Case ID	4.1.1	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	15 Oct 2021
Description	Add `selfId` field in connection data model and serializer	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	15 Oct 2021

Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a POST request in Postman			☑	☐
2	Try to add a new connection to validate update data model			☑	☐

3	Prepare a JSON file as the body of request with fields showed on the screenshot attached below			☑	☐
4	Fill in those fields with connection information as value. `userId` and `firstName` are compulsory, can drop other fields or leave the values as empty.			☑	☐
5	Post to "http://127.0.0.1:8000/api/connections/"	Return an HTTP response as JSON file now includes a `selfId` field	Expected JSON response received with `selfId` field	☑	☐
Post-Conditon	New added connection in database now has a `selfId` field.				
Verified By	@ Jackson Hu				
Comments	The connection model has been updated to include this `selfId` in order to distinguish the information for user and other connections belong to the user. The record with `selfId` field fulfilled is the user's own information, the `userId` should match `selfId`. Otherwise the connection belongs to the user who matches the `userId`.				
Screenshots					

```python
class Connection(models.Model):
    userId = models.CharField(max_length=70, blank=False, default='')
    selfId = models.CharField(max_length=70, blank=True, default='')
    emailAddress = models.CharField(max_length=100, blank=True, default='')
    firstName = models.CharField(max_length=70, blank=False, default='')
    lastName = models.CharField(max_length=70, blank=True, default='')
    phoneNumber = models.CharField(max_length=15, blank=True, default='')
    location = models.CharField(max_length=50, blank=True, default='')
    address = models.CharField(max_length=100, blank=True, default='')
    company = models.CharField(max_length=70, blank=True, default='')
    occupation = models.CharField(max_length=70, blank=True, default='')
    birthday = models.DateField(auto_now=False, blank=True)
    Vip = models.BooleanField(default=False)
    description = models.CharField(max_length=300, blank=True, default='')
    imageSrc = models.CharField(max_length=200, blank=True, default='')
    notes = JSONField(default='{}')
    tags = JSONField(default='{}')
    tasks = JSONField(default='{}')
    groups = JSONField(default='{}')
```

```
_id: ObjectId("61692bce5609071c3b63174e")
id: 18
userId: "28"
selfId: "28"
emailAddress: "123987@qq.com"
firstName: "qwe"
lastName: "ewq"
phoneNumber: ""
location: ""
address: ""
company: ""
occupation: ""
birthday: null
```

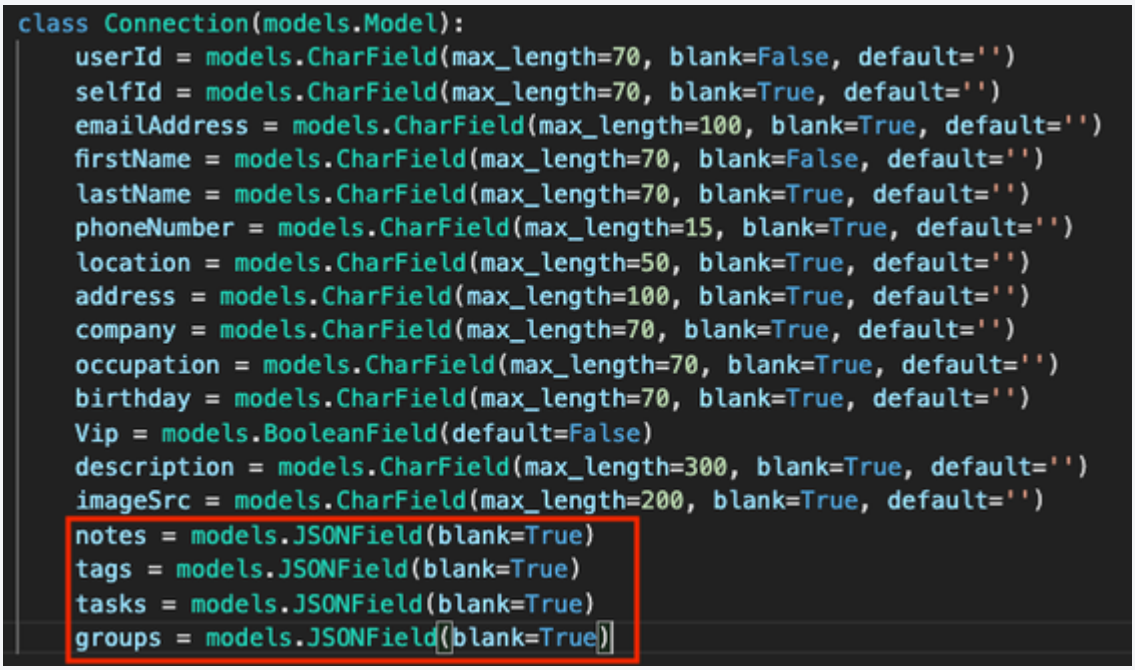
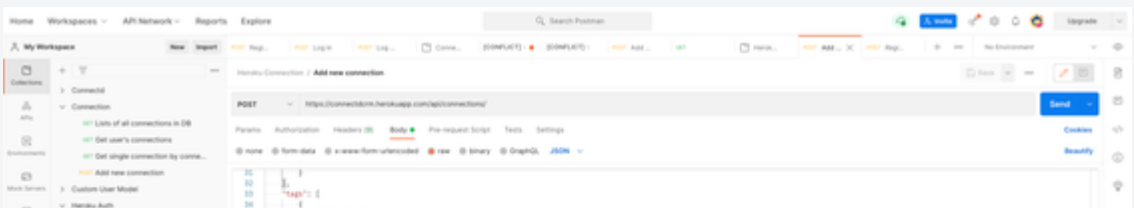


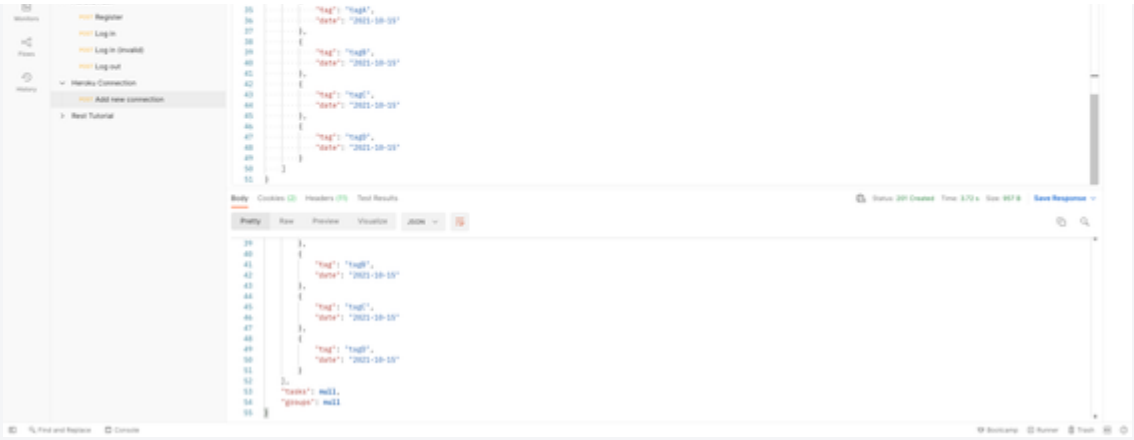
Time Constraint	< 1s

TC 4.1.2 Update Model - Update JSONField

Test/Execution Type	Unit / Manual	Designed By	@ Jackson Hu
Test Case Name	Update Model - Update `JSONField`	**Designed Date**	16 Oct 2021
Test Case ID	4.1.2	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	16 Oct 2021
Description	Update `JSONField` to store list of JSON objects for `notes` and `tags`	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	16 Oct 2021

	Locally running Django backend with port:8000

Pre-Condition					
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a POST request in Postman			☑	☐
2	Try to add a new connection to validate updated data model			☑	☐
3	Prepare a JSON file as the body of request with fields showed on the screenshot attached below			☑	☐
4	Fill in those fields with connection information as value. `userId` and `firstName` are compulsory, can drop other fields or leave the values as empty.			☑	☐
5	Include a list of JSON objects as value for both n`otes` and `tags`			☑	☐
6	Post to "http://127.0.0.1:8000/api/connections/"	Return an HTTP response as JSON file which contains list of `notes` and `tags`	Expected JSON response received with list of `notes` and `tags`	☑	☐
Post-Conditon	New added connection in database can now store multiple `notes` and `tags`.				
Verified By	@ Jackson Hu				
Comments	The connection model has been updated to use `django.db.models.JSONField` as the field type for both `notes` and `tags`. Connections can now store multiple `notes` and `tags` as list of JSON objects.				
Screenshots	 				



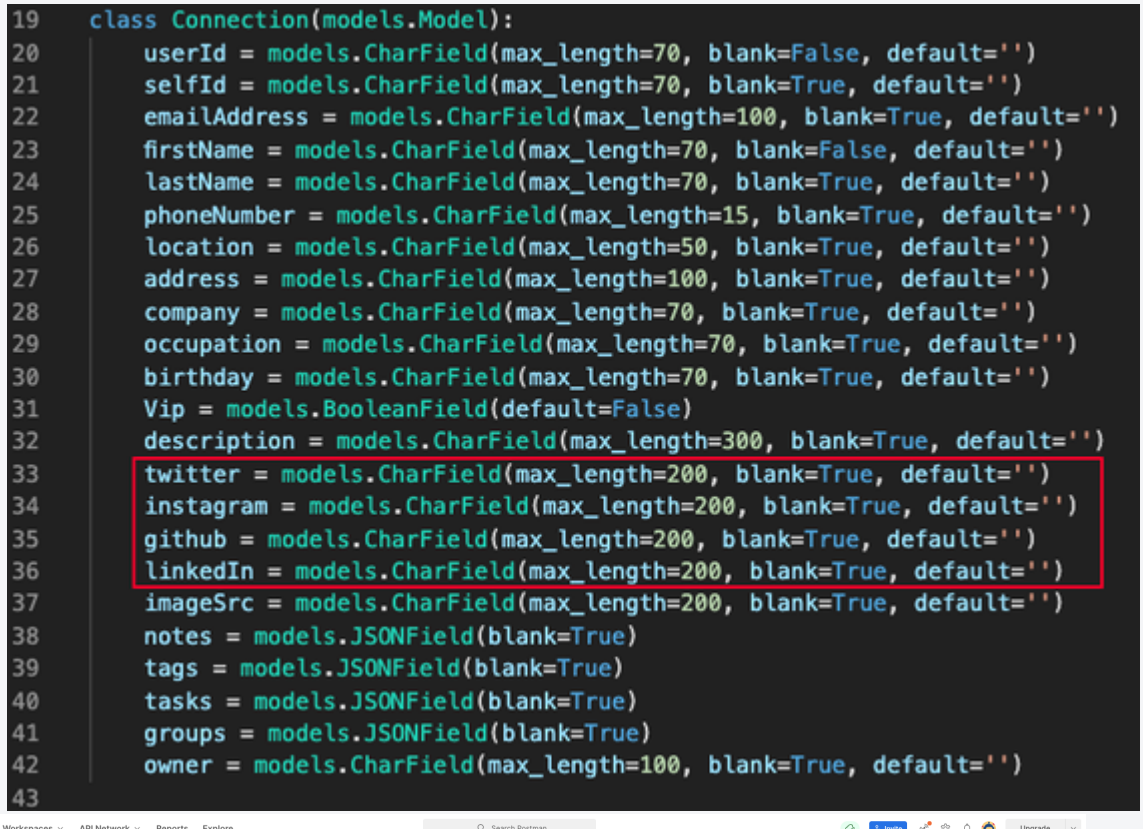

```
_id: ObjectId("61699994ff4a88c47ccd1b9e")
id: 31
userId: "29"
selfId: ""
emailAddress: "my email"
firstName: "my name"
lastName: ""
phoneNumber: "043032378"
location: ""
address: "my addr"
company: "Google.inc"
occupation: "my job"
birthday: "2021-11-18"
Vip: false
description: ""
imageSrc: ""
notes: "[{"note": "This is note A", "date": "2021-10-15"}, {"note": "Second no..."
tags: "[{"tag": "tagA", "date": "2021-10-15"}, {"tag": "tagB", "date": "2021-..."
tasks: null
groups: null
```


Time Constraint	< 1s

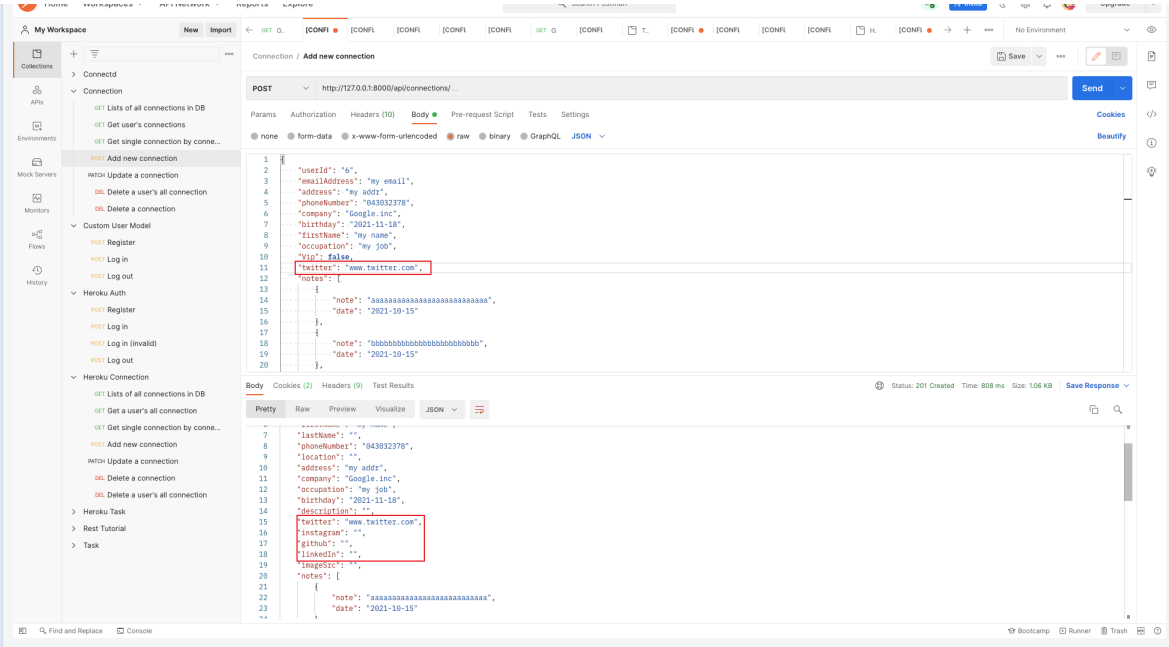
TC 4.1.3 Update Model - Add Social Media Links

Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Update Model - Add Social Media Links	**Designed Date**	29 Oct 2021
Test Case ID	4.1.3	**Last Modified By**	@ Jackson Hu
Test Priority	Medium	**Last Modified Date**	29 Oct 2021
Description	Add four social media links using `CharField`	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	29 Oct 2021

Pre-Condition	Locally running Django backend with port:8000			
	Setup	**Expected Results**	**Actual Results**	**Results**

TC Step No.	Activate virtual environment and run Django server locally			As Expected	Not As Expected
1	Create a **POST** request in Postman			☑	☐
2	Try to add a new connection to validate updated data model			☑	☐
3	Prepare a JSON file as the body of request with fields showed on the screenshot attached below			☑	☐
4	Fill in those fields with connection information as value. `userId` and `firstName` are compulsory, can drop other fields or leave the values as empty.			☑	☐
5	Include social media attributes like `twitter` and add user's profile page at twitter as the attribute value			☑	☐
6	Add a "`Authorization`" key in HTTP header			☑	☐
7	Copy the valid token of this user (provided when successfully logged in) as the value with prefix "`Token xxxxx`"			☑	☐
8	Post to "http://127.0.0.1:8000/api/connections/"	Return an HTTP response as JSON file which contains list social media links	Expected JSON response received with list social media links	☑	☐
Post-Conditon	New added connection in database can now store four social media links, including `twitter`, `instagram`, `github`, `linkedIn`.				
Verified By	@ Jackson Hu				
Comments	The connection model has been updated with extra attributes using `CharField`. Connections can now store four social media links, including `twitter`, `instagram`, `github`, `linkedIn`.				
Screenshots					

```python
class Connection(models.Model):
    userId = models.CharField(max_length=70, blank=False, default='')
    selfId = models.CharField(max_length=70, blank=True, default='')
    emailAddress = models.CharField(max_length=100, blank=True, default='')
    firstName = models.CharField(max_length=70, blank=False, default='')
    lastName = models.CharField(max_length=70, blank=True, default='')
    phoneNumber = models.CharField(max_length=15, blank=True, default='')
    location = models.CharField(max_length=50, blank=True, default='')
    address = models.CharField(max_length=100, blank=True, default='')
    company = models.CharField(max_length=70, blank=True, default='')
    occupation = models.CharField(max_length=70, blank=True, default='')
    birthday = models.CharField(max_length=70, blank=True, default='')
    Vip = models.BooleanField(default=False)
    description = models.CharField(max_length=300, blank=True, default='')
    twitter = models.CharField(max_length=200, blank=True, default='')
    instagram = models.CharField(max_length=200, blank=True, default='')
    github = models.CharField(max_length=200, blank=True, default='')
    linkedIn = models.CharField(max_length=200, blank=True, default='')
    imageSrc = models.CharField(max_length=200, blank=True, default='')
    notes = models.JSONField(blank=True)
    tags = models.JSONField(blank=True)
    tasks = models.JSONField(blank=True)
    groups = models.JSONField(blank=True)
    owner = models.CharField(max_length=100, blank=True, default='')

```



Time Constraint	< 1s

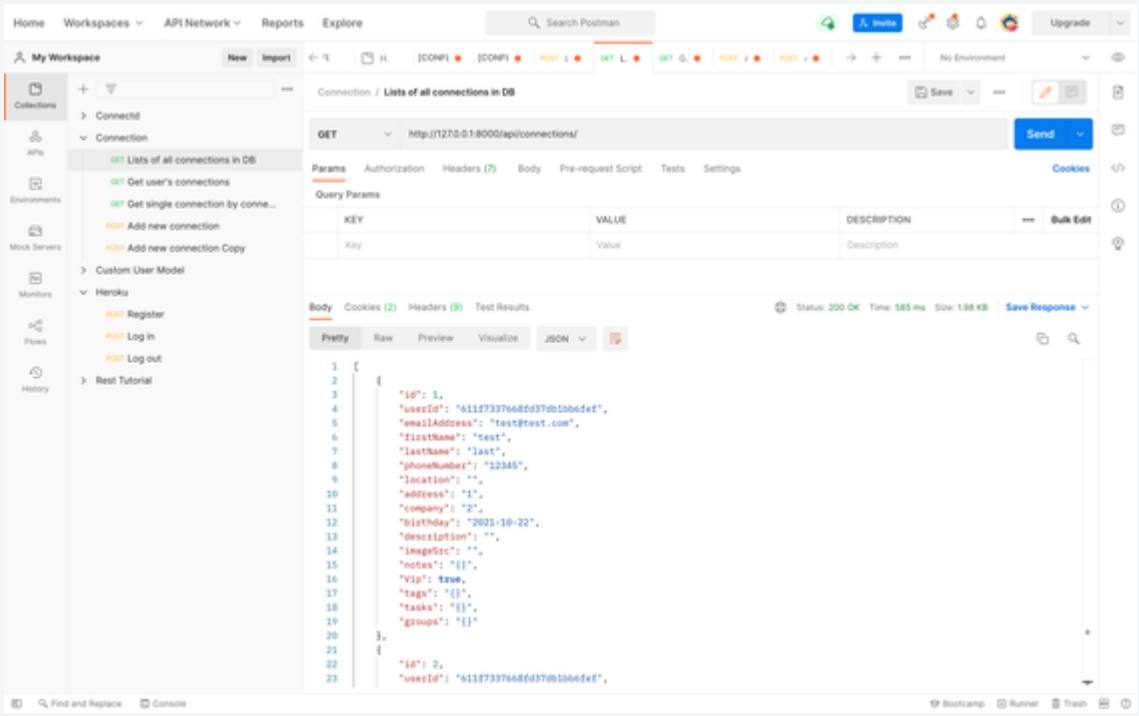
TC 4.2 Retrieve Connection Functionality

- TC 4.2.1 Django Request - Get All Connections
- TC 4.2.2 Django Request - Get a User's Connection
- TC 4.2.3 Django Request - Get a Single Connection
- TC 4.2.4 Django Request - Get a User's Connection
- TC 4.2.5 Front-End Retrieve & Display User's Connections
- TC 4.2.6 Django Request - Get a Single Connection
- TC 4.2.7 Django Request - Get a User's Connection
- TC 4.2.8 Django Request - Get a User's Connection

TC 4.2.1 Django Request - Get All Connections

Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Get all connections request	**Designed Date**	15 Oct 2021
Test Case ID	4.2.1	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	15 Oct 2021
Description	Locally send a POST request to Django backend server to retrieve all connections stored in database.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	15 Oct 2021

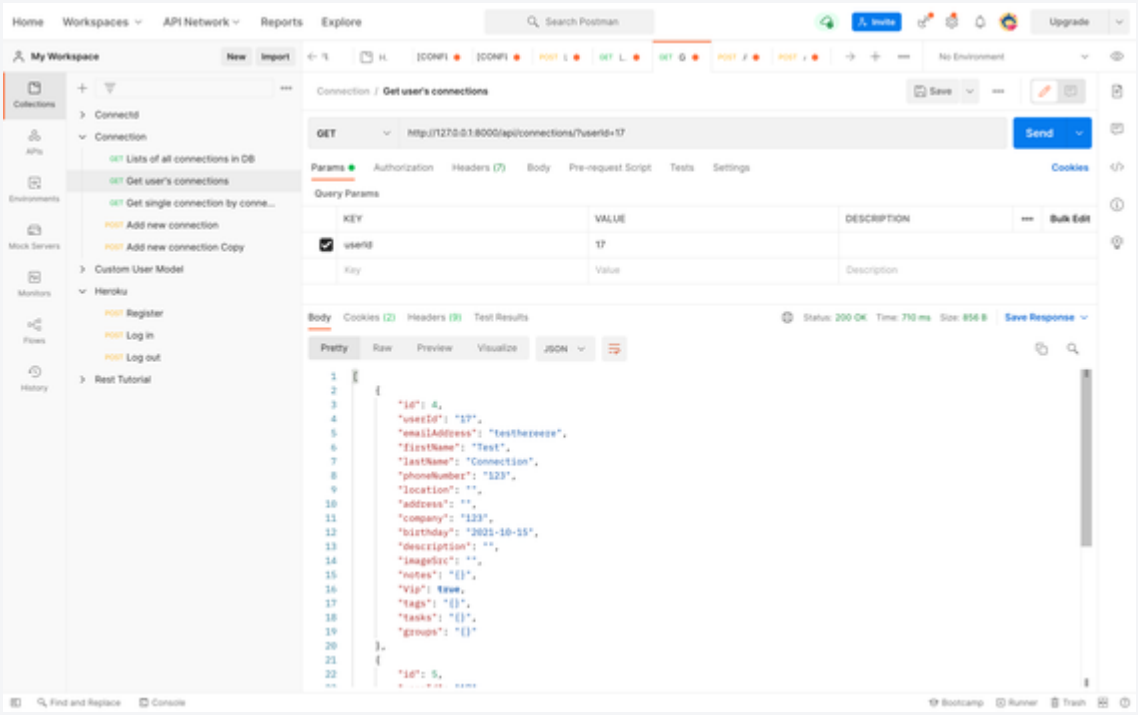
Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **GET** request in Postman			☑	☐
2	No need to include a JSON file as body			☑	☐

3	Post to “http://127.0.0.1:8000/api/connections/“	Return a JSON file with all connection records currently stored in database	Expected JSON response received which has list of JSON object for each connection record	☑	☐
Post-Condition	Retrieve all connection records stored in database.				
Verified By	@ Jackson Hu				
Comments	Get all connections belong to all users. Since we’ve set up the CORS so this API won’t be used by normal users / malicious users.				
Screenshots					
Time Constraint	< 1s				

TC 4.2.2 Django Request - Get a User's Connection

Test/Execution Type	**Unit / Manual**	**Designed By**	**@ Jackson Hu**
Test Case Name	Get a user's connection	**Designed Date**	15 Oct 2021
Test Case ID	4.2.2	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	15 Oct 2021
Description	Locally send a POST request to Django backend server to retrieve all connections belong to a given user.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	15 Oct 2021

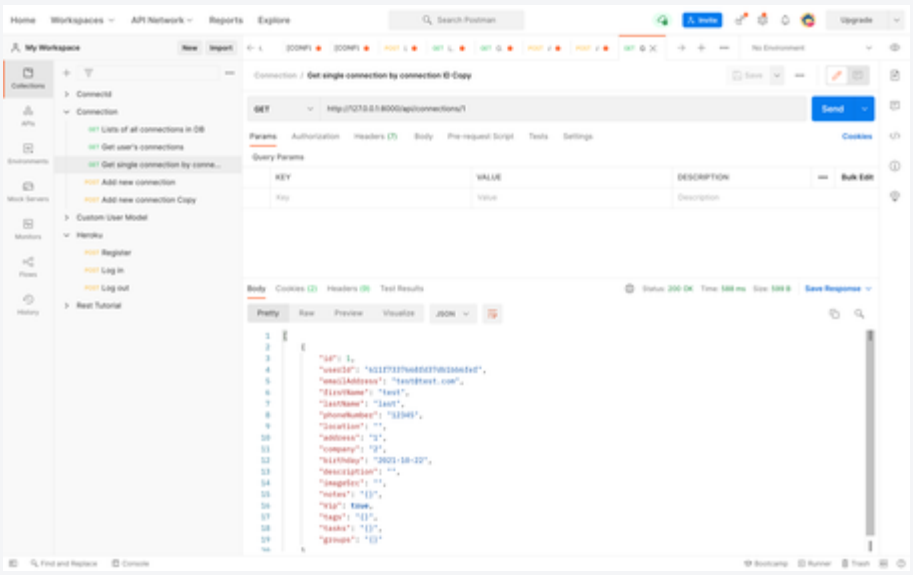
Pre-Condition	Locally running Django backend with port:8000			
	Setup	**Expected Results**	**Actual Results**	**Results**

TC Step No.	Activate virtual environment and run Django server locally			As Expected	Not As Expected
1	Create a **GET** request in Postman			☑	☐
2	Add a `Query Parameter` such that key is `userId` and value is the `userId` of a given user	Should add a suffix in url as well	Appears like `?userId = x`	☑	☐
3	For example, we want to get connections belong to a user whose `userId` is 17			☑	☐
4	Post to "http://127.0.0.1:8000/api/connections/?userId=17"	Return a JSON file with all connection related to the given user	Expected JSON response received which has list of JSON object for each connection record	☑	☐
Post-Condition	Retrieve all connections belong to a given user.				
Verified By	@ Jackson Hu				
Comments	Get all connections belong to a given user. Connections in database are filtered by its `userId` attribute.				
Screenshots					
Time Constraint	< 1s				

TC 4.2.3 Django Request - Get a Single Connection

Test/Execution Type	Unit / Manual	Designed By	@ Jackson Hu
Test Case Name	Get a single connection	**Designed Date**	15 Oct 2021
Test Case ID	4.2.3	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	15 Oct 2021
Description		**Executed By**	@ Jackson Hu

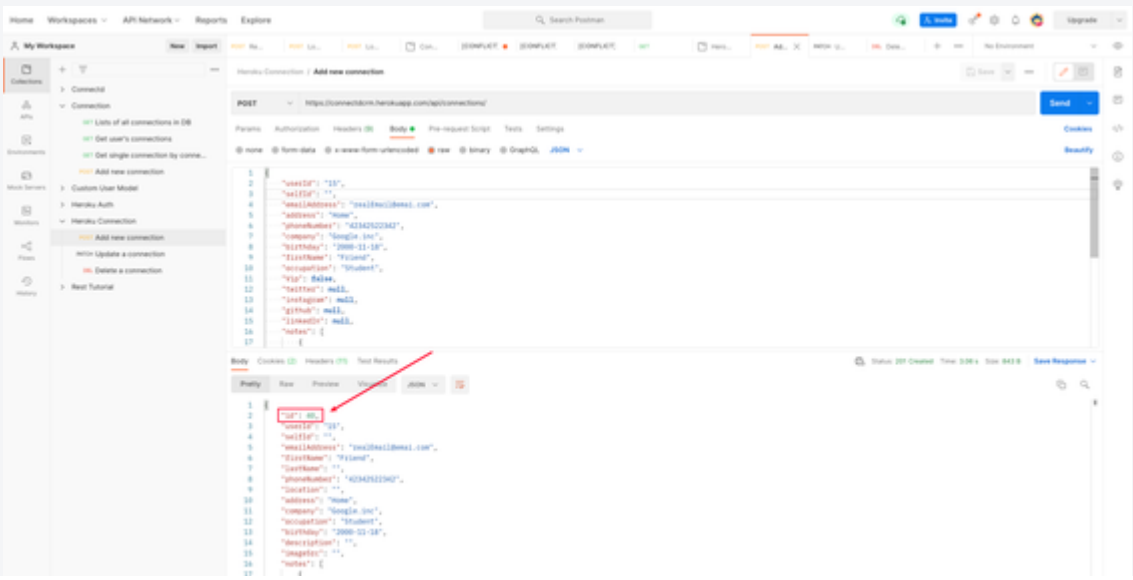
	Locally send a POST request to Django backend server to retrieve a single connection with its `id`.		
Final Results	PASS	**Execution Date**	15 Oct 2021

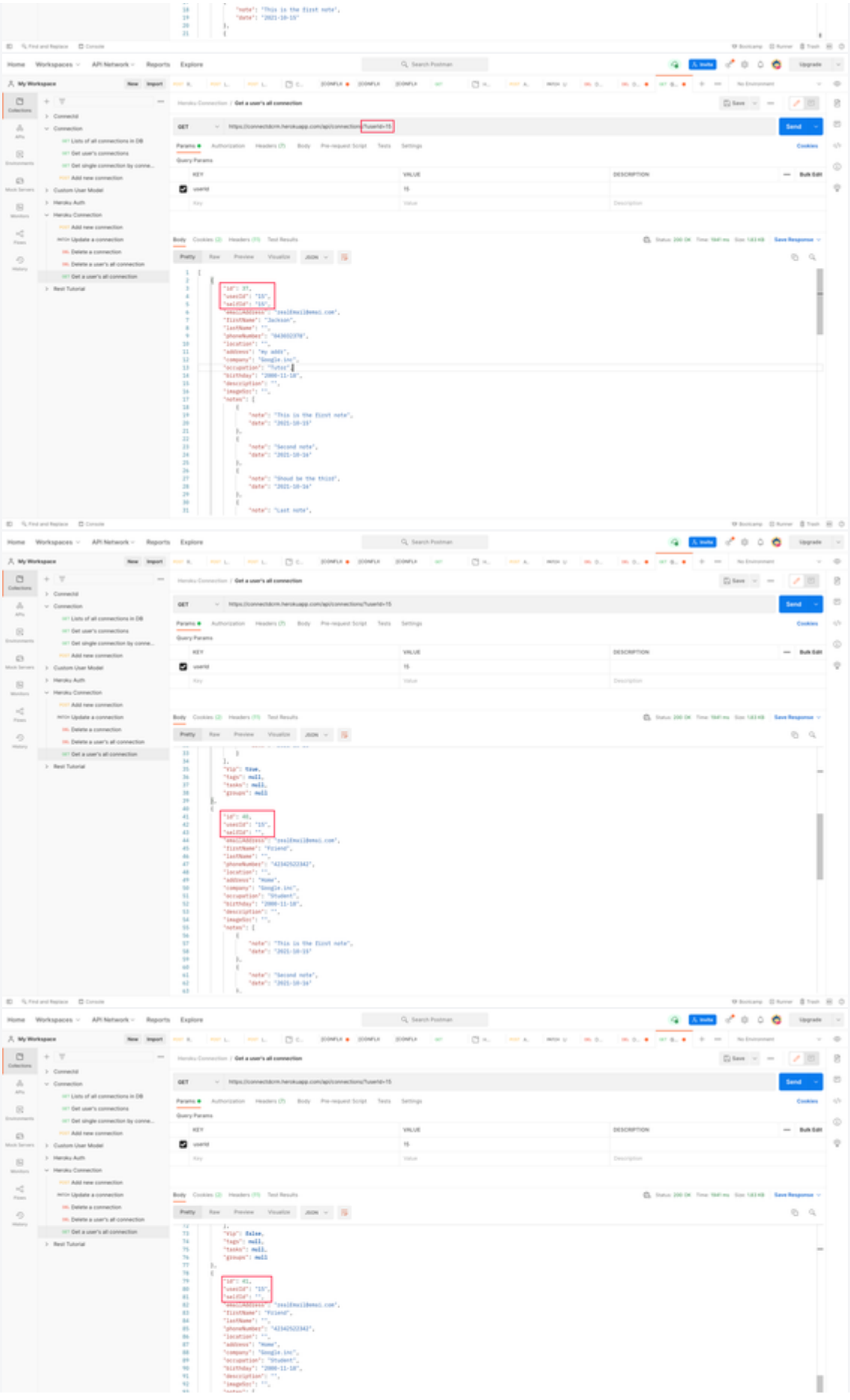
Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **GET** request in Postman			☑	☐
2	Add the value of connection `id` as the suffix in url		Appears like `.../connections/x/`	☑	☐
3	For example, we want to get a connection which the connection `id` is 2			☑	☐
4	Post to "https://connectdcrm.herokuapp.com/api/connections/2/"	Return a JSON file with the information of a connection given its `id`	Expected JSON response received with only a single connection with corresponding `id`	☑	☐
Post-Conditon	Retrieve the information of a single connection given its connection `id` already known				
Verified By	@ Jackson Hu				
Comments	The `id` attribute of a connection is generated automatically when the connection is created. Connections in database are filtered by its `id` attribute.				
Screenshots					
Time Constraint	< 1s				

TC 4.2.4 Django Request - Get a User's Connection

Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Get a user's connection	**Designed Date**	16 Oct 2021
Test Case ID	4.2.4	**Last Modified By**	@ Jackson Hu

Test Priority	High	**Last Modified Date**	16 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to retrieve all connections belong to a given user. Added `selfId` field to distinguish the information of user self and connections belong to user.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	16 Oct 2021

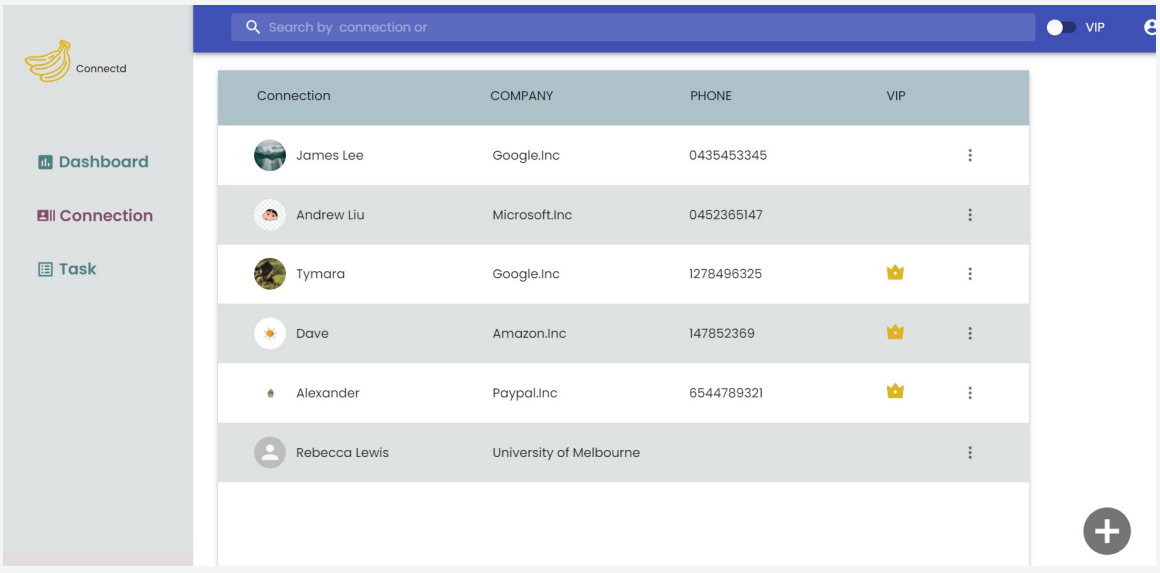
Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **GET** request in Postman			☑	☐
2	Add several new connections for user with `user Id` 15	Return JSON files with information of connections just created	Connections created successfully as expected	☑	☐
3	Add a `Query Parameter` such that key is `use rId` and value is the `userId` of a given user	Should add a suffix in url as well	Appears like `?userId = x`	☑	☐
4	For example, we want to get connections belong to a user whose `userId` is 15			☑	☐
5	Post to "https://connectdcrm.herokuapp.com/api/connections/?userId=15"	Return a JSON file with all connection related to the given user	Expected JSON response received which has list of JSON object for each connection record	☑	☐
Post-Condition	Retrieve all connections belong to a given user, including the connection which stores the user's own information.				
Verified By	@ Jackson Hu				
Comments	Get all connections belong to a given user. Connections in database are filtered by its `userId` attribute. Added `selfId` field to distinguish the information of user self and connections belong to user. • `selfId` == `userId` : this connection record contains the given user's own information. • `selfId` is empty : this connection record belongs to the user with `userId`.				
Screenshots					



	
Time Constraint	< 2s

TC 4.2.5 Front-End Retrieve & Display User's Connections

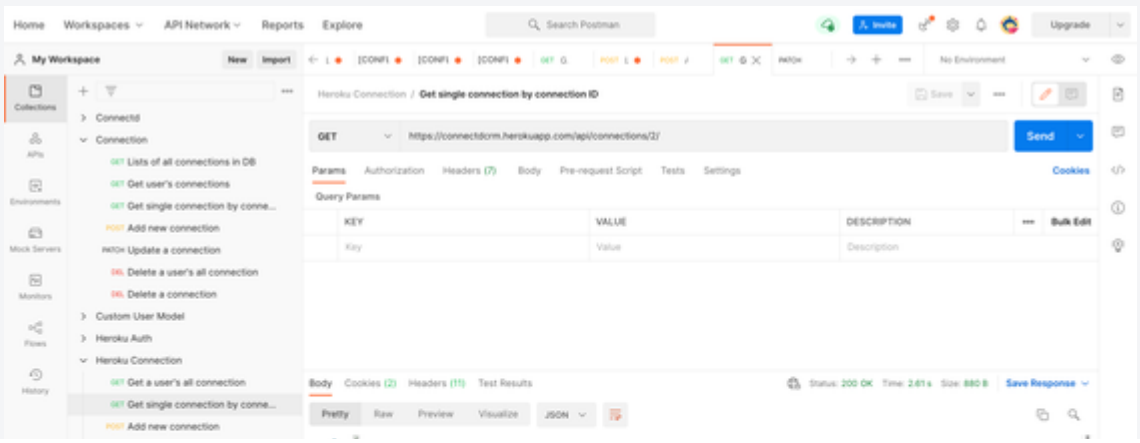
Test/Execution Type	**Manual**	**Designed By**	@ Han Liu
Test Case Name	Front-End Retrieve & Display User's Connections	**Designed Date**	22/10/2021
Test Case ID	4.2.5	**Last Modified By**	@ Han Liu
Test Priority	High	**Last Modified Date**	22/10/2021
Description	Testing if logged in user can retrieve connection details	**Executed By**	@ Han Liu
Final Results	PASS	**Execution Date**	22/10/2021

Pre-Condition	User logged in				
TC Step No.	**Setup** navigate to https://connectd-front.herokuapp.com/ and logged into account	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Navigate to connection page	All connection shows in connection page	All connection shows in connection page	☑	☐
Post-Condition	N/A				
Verified By	@ Han Liu				
Comments	Frontend successfully retrieve and displayed user's connection				
Screenshots					

Time Constraint	N/A

TC 4.2.6 Django Request - Get a Single Connection

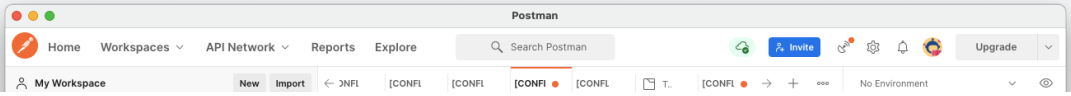
Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Get a single connection	**Designed Date**	19 Oct 2021
Test Case ID	4.2.6	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	28 Oct 2021
Description	Locally send a POST request to Django backend server to retrieve a single connection with its `id`.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	19 Oct 2021

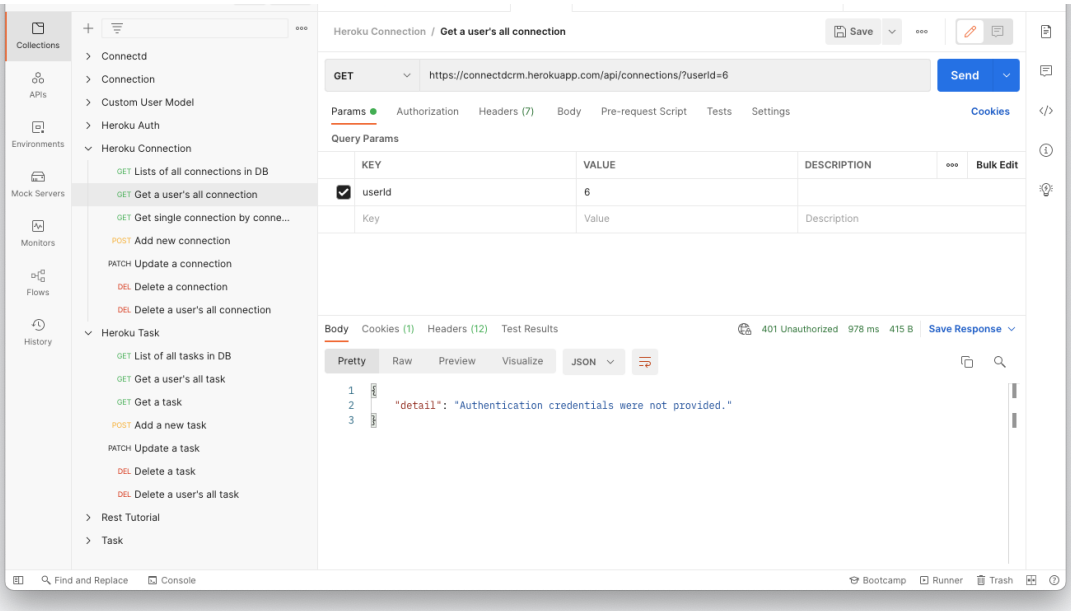
Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **GET** request in Postman			☑	☐
2	Add the value of connection `id` as the suffix in url		Appears like `.../connections/x/`	☑	☐
3	For example, we want to get a connection which the connection `id` is 2			☑	☐
4	Post to "https://connectdcrm.herokuapp.com/api/connections/2/"	Return a JSON file with the information of a connection given its `id`	Expected JSON response received with only a single connection with corresponding `id`	☑	☐
Post-Conditon	Retrieve the information of a single connection given its connection `id` already known				
Verified By	@ Jackson Hu				
Comments	The `id` attribute of a connection is generated automatically when the connection is created. Connections in database are filtered by its `id` attribute.				
Screenshots					

	
Time Constraint	< 3s

TC 4.2.7 Django Request - Get a User's Connection

Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Get a user's connection	**Designed Date**	29 Oct 2021
Test Case ID	4.2.7	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	29 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to retrieve all connections belong to a given user.	**Executed By**	@ Jackson Hu
Final Results	FAIL	**Execution Date**	29 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **GET** request in Postman			☑	☐
2	Add several new connections for user with `user Id` 6	Return JSON files with information of connections just created	Connections created successfully as expected	☑	☐
3	Add a `Query Parameter` such that key is `user Id` and value is the `userId` of a given user	Should add a suffix in url as well	Appears like `?userId = x`	☑	☐
4	For example, we want to get connections belong to a user whose `userId` is 6			☑	☐
5	Post to "https://connectdcrm.herokuapp.com/api/connections/?userId=6"	Return a JSON file with all connection related to the given user	Response returned with status code: `401 Unauthorized`	☐	☑
Post-Conditon	Returned `"detail": "Authentication credentials were not provided."`				
Verified By	@ Jackson Hu				
Comments	Seems the issue was caused by the new authentication mechanism just implemented.				
Screenshots					

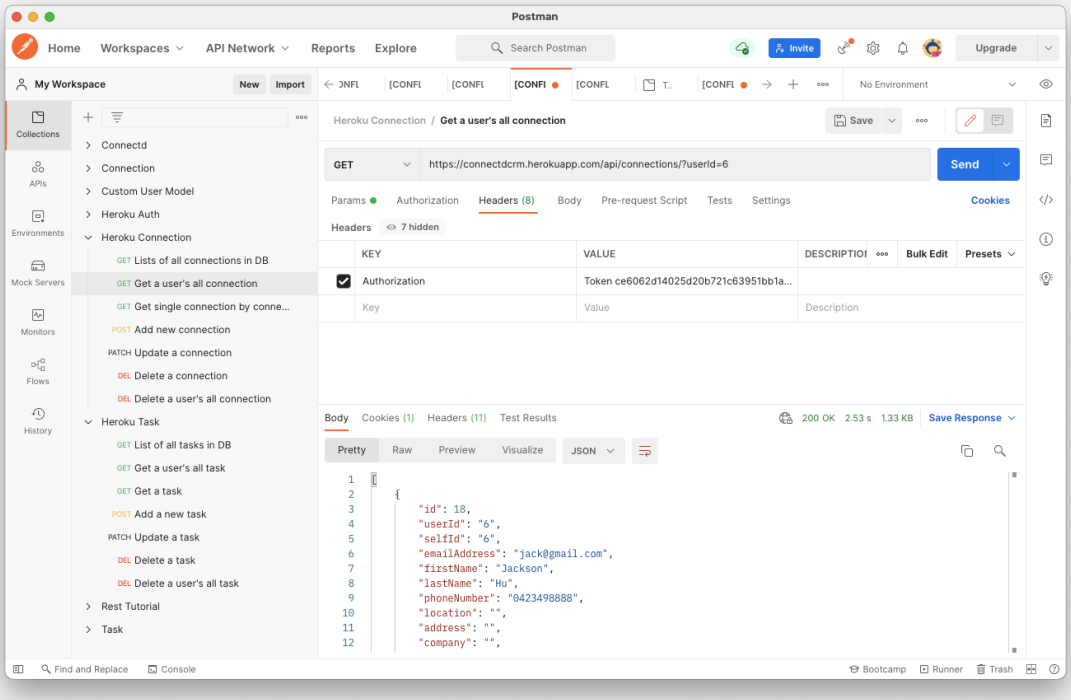


Time Constraint	< 2s

TC 4.2.8 Django Request - Get a User's Connection

Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Get a user's connection	**Designed Date**	29 Oct 2021
Test Case ID	4.2.8	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	29 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to retrieve all connections belong to a given user. Valid user token is now required to include in the request header according to new authentication mechanism implemented.	**Executed By**	@ Jackson Hu
Final Results	**PASS**	**Execution Date**	29 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **GET** request in Postman			☑	☐
2	Add several new connections for user with `user Id` 6	Return JSON files with information of connections just created	Connections created successfully as expected	☑	☐
3	Add a `Query Parameter` such that key is `user Id` and value is the `userId` of a given user	Should add a suffix in url as well	Appears like `?userId = x`	☑	☐
4	For example, we want to get connections belong to a user whose `userId` is 6			☑	☐

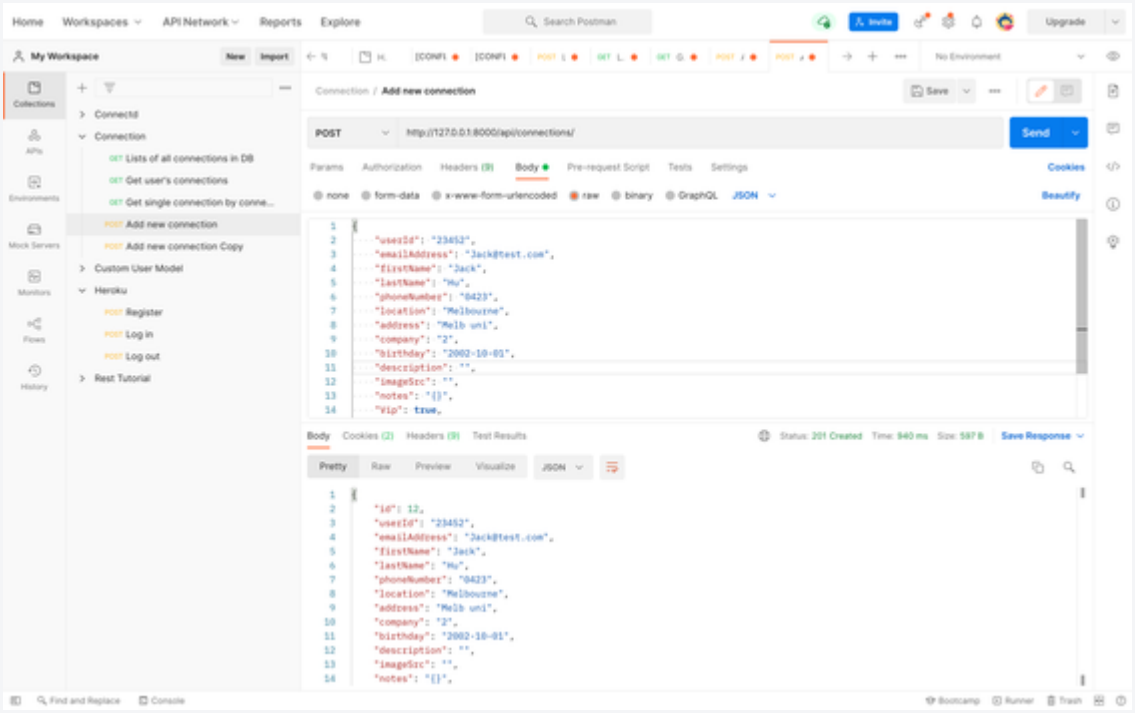
5	Add a “`Authorization`" key in HTTP header			☑	☐
6	Copy the valid token of this user (provided when successfully logged in) as the value with prefix “`Token xxxxx`“			☑	☐
7	Post to “https://connectdcrm.herokuapp.com/api/connections/?userId=6“	Return a JSON file with all connection related to the given user	Expected JSON response received which has list of JSON object for each connection record	☑	☐
Post-Condition	Get all connections belong to a given user.				
Verified By	@ Jackson Hu				
Comments	The new authentication mechanism will require to verify the validity of request by asking for valid user token in header. By doing this, one user can’t access the connection information of other users. This is one of the approaches against malicious user / attack.				
Screenshots					
Time Constraint	< 3s				

TC 4.3 Adding Connection Functionality

TC 4.3.1 Django Request - Add a New Connection

Test/Execution Type	**Unit / Manual**	**Designed By**	**@ Jackson Hu**
Test Case Name	Add new connection request	**Designed Date**	15 Oct 2021
Test Case ID	4.3.1	**Last Modified By**	@ Jackson Hu

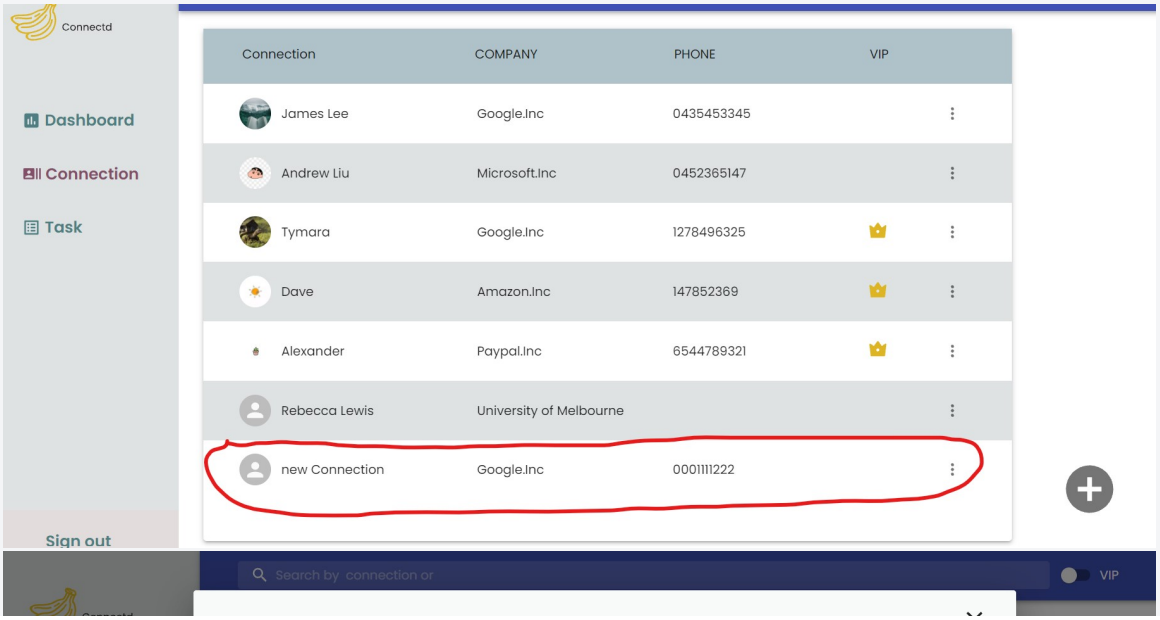
Test Priority	High	**Last Modified Date**	15 Oct 2021
Description	Locally send a POST request to Django backend server to add a new connection.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	15 Oct 2021

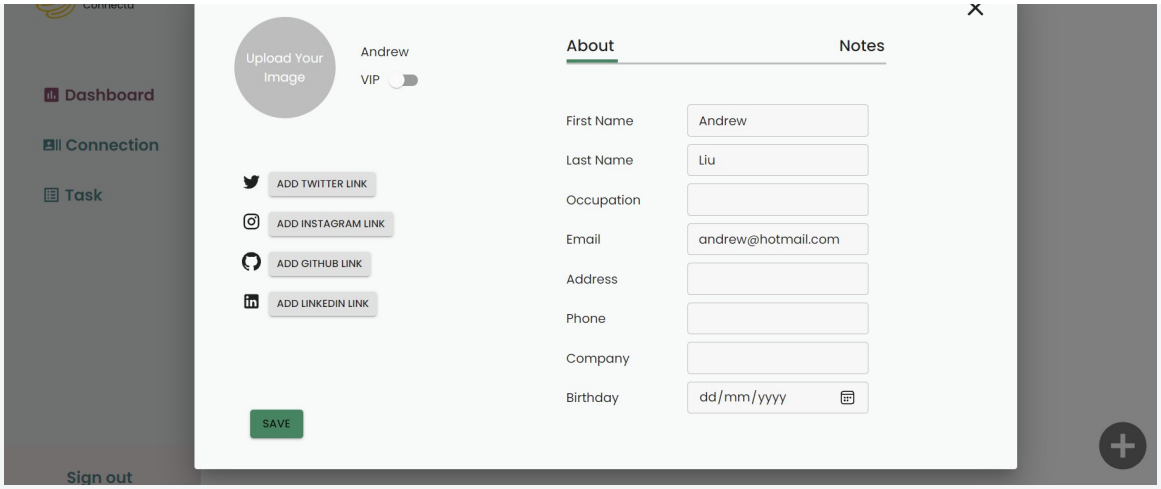
Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a POST request in Postman			☑	☐
2	Prepare a JSON file as the body of request with fields showed on the screenshot attached below			☑	☐
3	Fill in those fields with connection information as value. `userId` and `firstName` are compulsory, can drop other fields or leave the values as empty.			☑	☐
4	Post to "http://127.0.0.1:8000/api/connections/"	Return an HTTP response as JSON file with connection record `id` and serialised data	Expected JSON response received	☑	☐
Post-Condition	Successfully added a new connection record in database with given information.				
Verified By	@ Jackson Hu				
Comments	Currently just test locally. Notice that some unnecessary fields could be dropped when sending request to backend server: `userId` and `firstName` are compulsory, can drop other fields or leave the values as empty.				
Screenshots					
	< 1s				

Time Constraint	

TC 4.3.2 Front-End - User Add New Connection

Test/Execution Type	**Manual**	**Designed By**	@ Han Liu
Test Case Name	Front-End - User Add New Connection	**Designed Date**	22/10/2021
Test Case ID	4.3.2	**Last Modified By**	@ Han Liu
Test Priority	High	**Last Modified Date**	22/10/2021
Description	Testing if logged in user can add new connection	**Executed By**	@ Han Liu
Final Results	PASS	**Execution Date**	22/10/2021

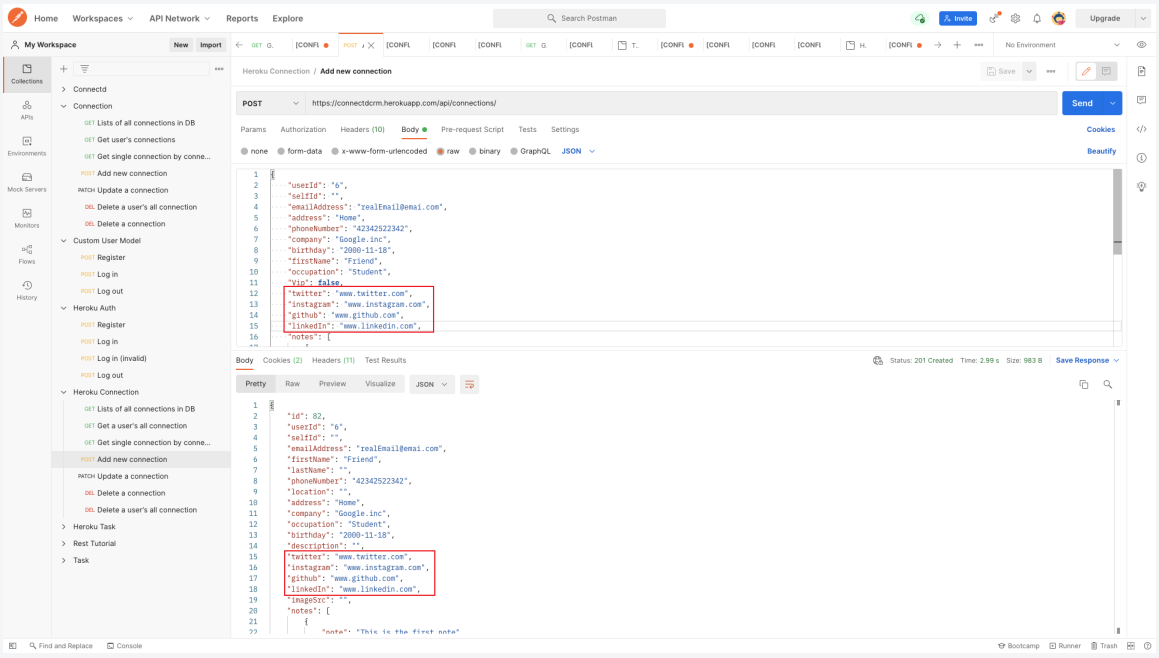
Pre-Condition	User logged in				
TC Step No.	**Setup** navigate to https://connectd-front.herokuapp.com/ and logged into account	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Click <Add Connection>, then fill in connection detail			☑	☐
2	Send connection detail to backend	Connection detail has been sent to backend	Connection detail has been sent to backend	☑	
3	Check if new connection has been added to database	new connection is showed in connection page	new connection is showed in connection page	☑	
Post-Condition	N/A				
Verified By	@ Han Liu				
Comments	User successfully added new connection via Front-End application				
Screenshots					

	
Time Constraint	N/A

TC 4.3.3 Django Request - Add a New Connection

Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Add new connection request	**Designed Date**	29 Oct 2021
Test Case ID	4.3.3	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	29 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to retrieve all connections belong to add a new connection. Valid user token is now required to include in the request header according to new authentication mechanism implemented. Can add social media links now with updated connection model.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	29 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **POST** request in Postman			☑	☐
2	Prepare a JSON file as the body of request with fields showed on the screenshot attached below			☑	☐
3	Fill in those fields with connection information as value. `userId` and `firstName` are compulsory, can drop other fields or leave the values as empty.			☑	☐
4	Add a "`Authorization`" key in HTTP header			☑	☐
5				☑	☐

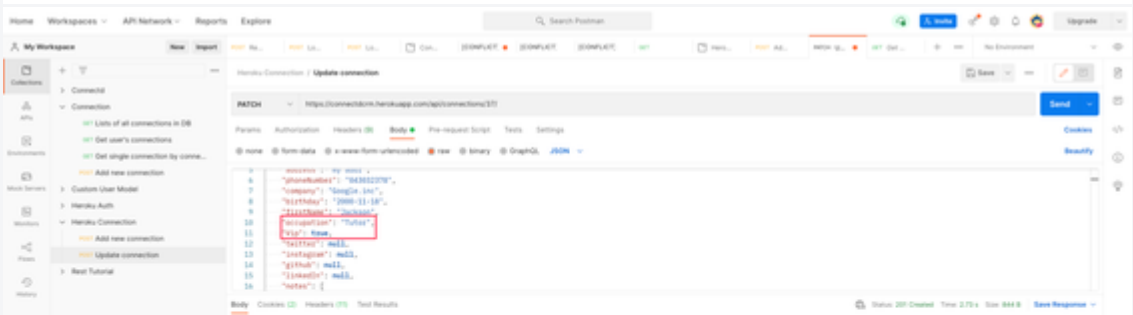
	Copy the valid token of this user (provided when successfully logged in) as the value with prefix "T`oken xxxxx`"				
6	Post to "https://connectdcrm.herokuapp.com/api/connections/"	Return an HTTP response as JSON file with connection record `id` and serialised data	Expected JSON response received	☑	☐
Post-Condiditon	Successfully added a new connection record in database with given information.				
Verified By	@ Jackson Hu				
Comments	Notice that some unnecessary fields could be dropped when sending request to backend server: `userId` and `firstName` are compulsory, can drop other fields or leave the values as empty. Connections can now store four social media links, including `twitter`, `instagram`, `github`, `linkedIn`.				
Screenshots					
Time Constraint	< 3s				

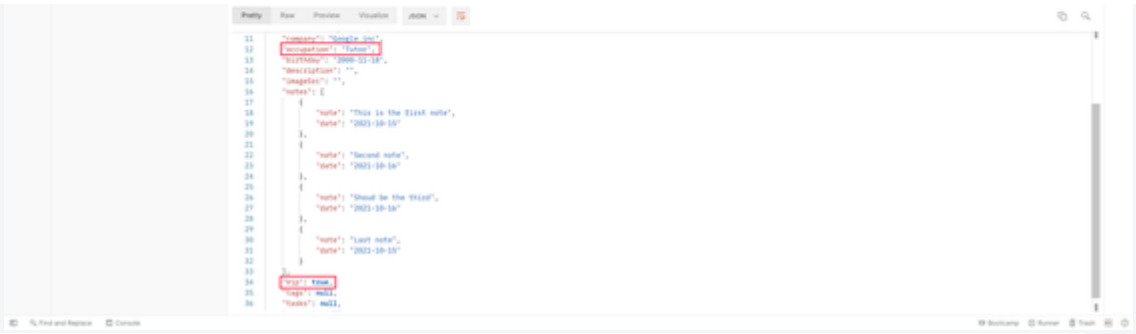
TC 4.4 Edit Connection Functionality

- TC 4.4.1 Django Request - Update a Connection
- TC 4.4.2 Front-End - User Edit Connection

TC 4.4.1 Django Request - Update a Connection

Test/Execution Type	**Unit / Manual**	**Designed By**	**@ Jackson Hu**
Test Case Name	Update a connection	**Designed Date**	16 Oct 2021
Test Case ID	4.4.1	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	16 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to update the information of a single connection with its `id`.	**Executed By**	@ Jackson Hu
Final Results	**PASS**	**Execution Date**	16 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **PATCH** request in Postman			☑	☐
2	Add the value of connection `id` as the suffix in url		Appears like `.../connections/x/`	☑	☐
3	For example, we want to update a connection which the connection `id` is 37			☑	☐
4	Prepare a JSON body including all information of the given connection you want to update			☑	☐
5	Post to "https://connectdcrm.herokuapp.com/api/connections/37/"	Return a JSON file with updated information of a connection given its `id`	Expected JSON response received with updated information in connection record	☑	☐
Post-Conditon	Updated the information of a connection given its connection `id` already known				
Verified By	@ Jackson Hu				
Comments	The `id` attribute of a connection is generated automatically when the connection is created. Connections in database are filtered by its `id` attribute. The attributes of a connection will be overwritten by the new values received. • Need to put those unchanged attributes in request as well.				
Screenshots	_id: ObjectId("616a7fabbcdb1fbbe15f02d4") id: 37 userId: "15" selfId: "15" emailAddress: "realEmail@emai.com" firstName: "Jackson" lastName: "" phoneNumber: "043032378" location: "" address: "my addr" company: "Google.inc" occupation: "Student" birthday: "2000-11-18" Vip: false description: "" imageSrc: "" notes: "[{"note": "This is the first note", "date": "2021-10-15"}, {"note": "S..." tags: null tasks: null groups: null 				

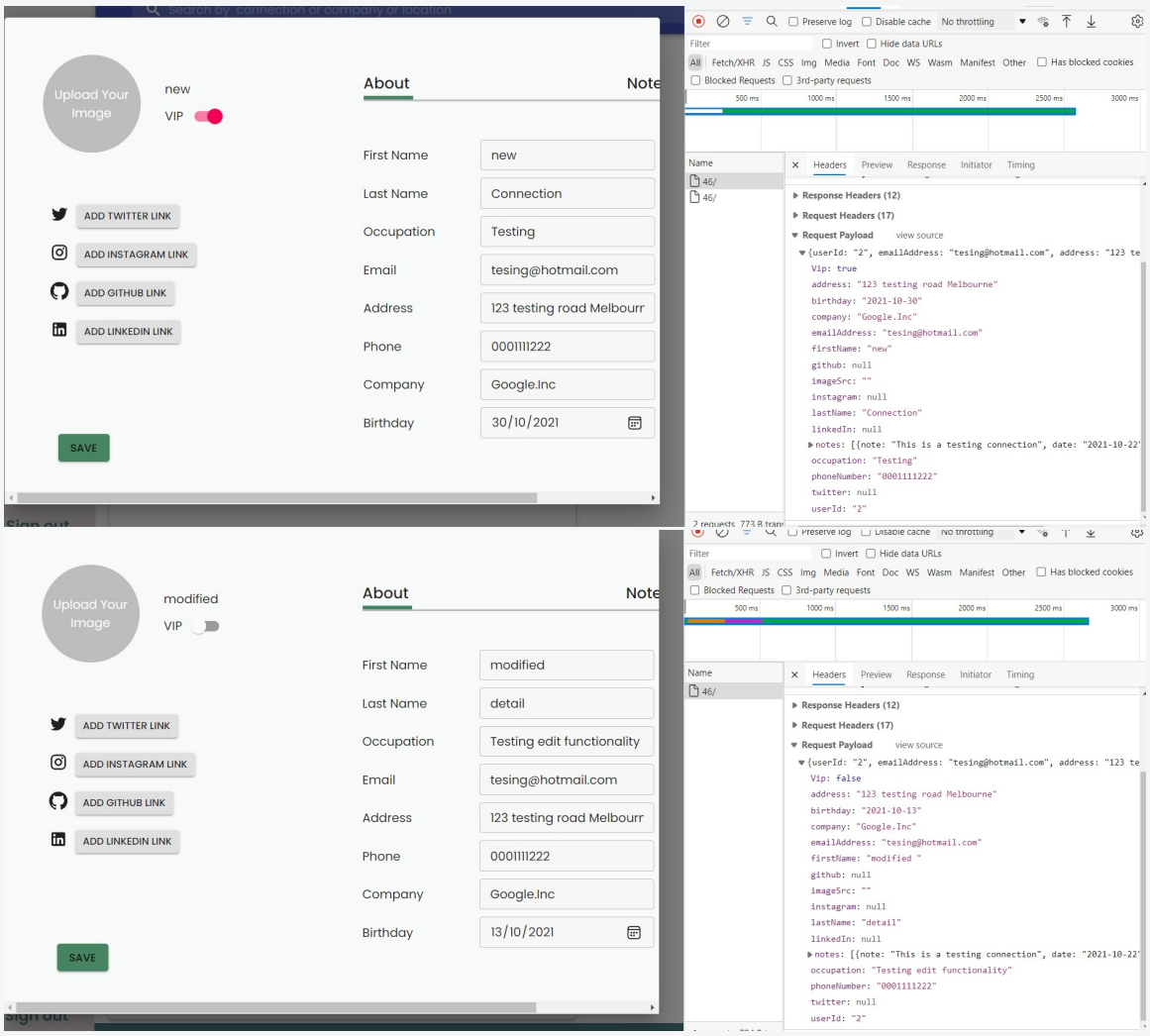


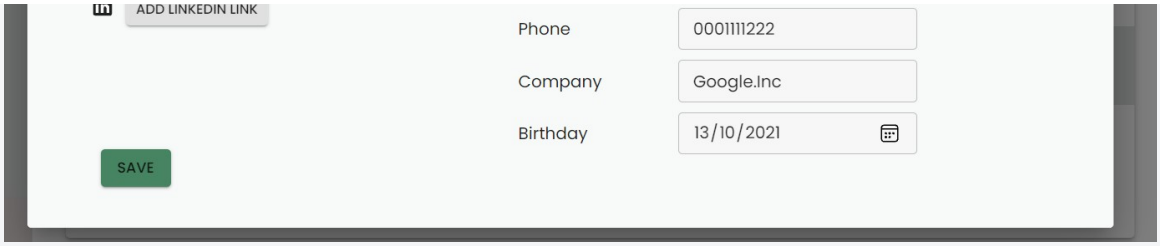
Time Constraint	< 3s

TC 4.4.2 Front-End - User Edit Connection

Test/Execution Type	**Manual**	**Designed By**	@ Han Liu
Test Case Name	Front-End - User Edit Connection	**Designed Date**	22/10/2021
Test Case ID	4.4.2	**Last Modified By**	@ Han Liu
Test Priority	High	**Last Modified Date**	22/10/2021
Description	Testing if logged in user can edit connection	**Executed By**	@ Han Liu
Final Results	PASS	**Execution Date**	22/10/2021

Pre-Condition	User logged in				
TC Step No.	**Setup**	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Click <Edit Connection>, then edit connection detail			☑	☐
2	Send new connection detail to backend	New detail has been sent to backend	New detail has been sent to backend	☑	
	navigate to https://connectd-front.herokuapp.com/ and logged into account				

3	Check if connection's detail has been modified	connection's detail has been modified	connection's detail has been modified		
Post - Condition	N/A				
Verified By	@ Han Liu				
Comments	User successfully edit connection via Front-End application				
Screenshots					

	
Time Constraint	N/A

TC 4.5 Delete Connection Functionality

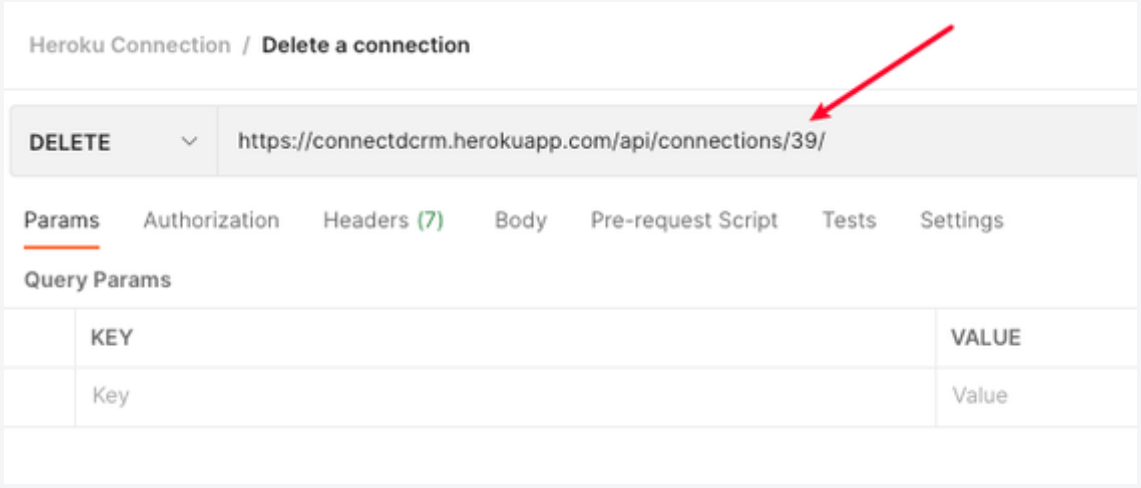
TC 4.5.1 Django Request - Delete a Connection

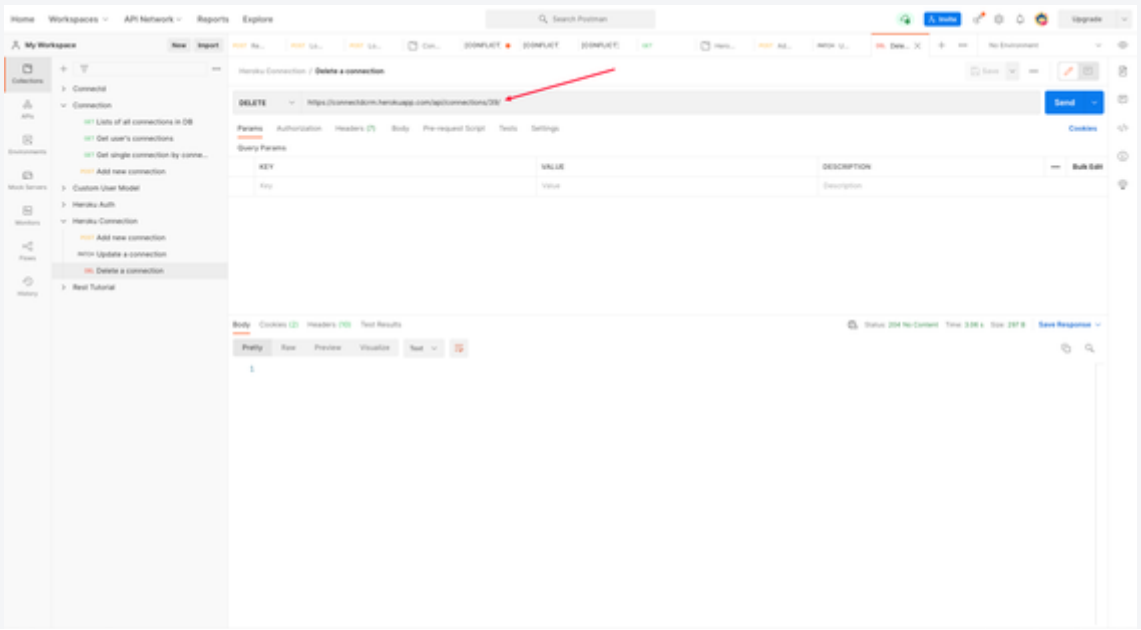
Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Delete a connection	**Designed Date**	16 Oct 2021
Test Case ID	4.5.1	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	16 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to delete a single connection with its `id`.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	16 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **DELETE** request in Postman			☑	☐
2	Add the value of connection `id` as the suffix in url		Appears like `... /connections/x/`	☑	☐
3	For example, we want to delete a connection which the connection `id` is 39			☑	☐
4	Post to "https://connectdcrm.herokuapp.com/api/connections/39/"	Nothing returned	Received `204 No Content`	☑	☐
Post-Condition	Delete a connection from database given its connection `id` already known				
Verified By	@ Jackson Hu				
	Connections in database are filtered by its `id` attribute.				

Comments	Need to know the `id` of the connection you want to delete.
Screenshots	


```
_id: ObjectId("616a832e0a62412d369baaa0")
id: 39
userId: "15"
selfId: ""
emailAddress: "realEmail@emai.com"
firstName: "Friend"
lastName: ""
phoneNumber: "42342522342"
location: ""
address: "Home"
company: "Google.inc"
occupation: "Student"
birthday: "2000-11-18"
Vip: false
description: ""
imageSrc: ""
notes: "[{"note": "This is the first note", "date": "2021-10-15"}, {"note": "S..."
tags: null
tasks: null
groups: null
```

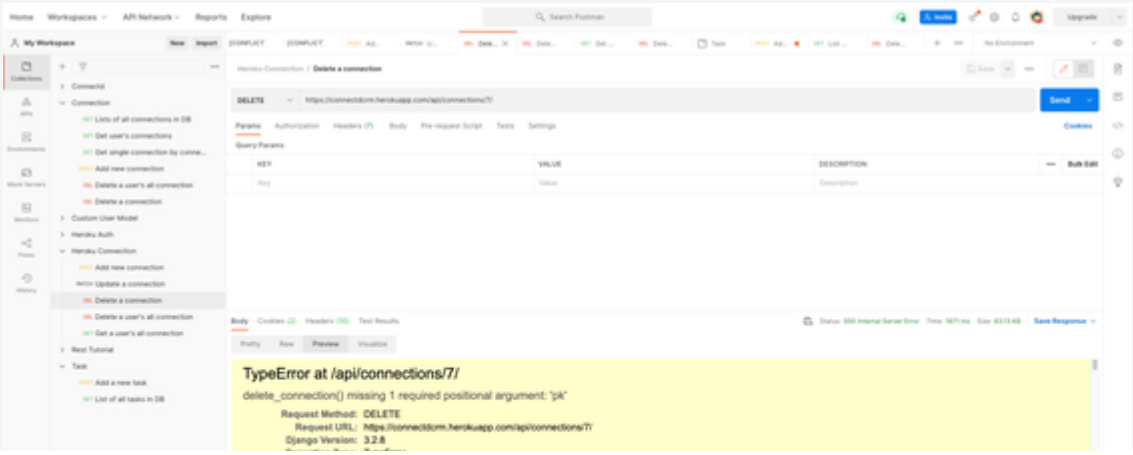





Time Constraint	< 3s

TC 4.5.2 Django Request - Delete a Connection

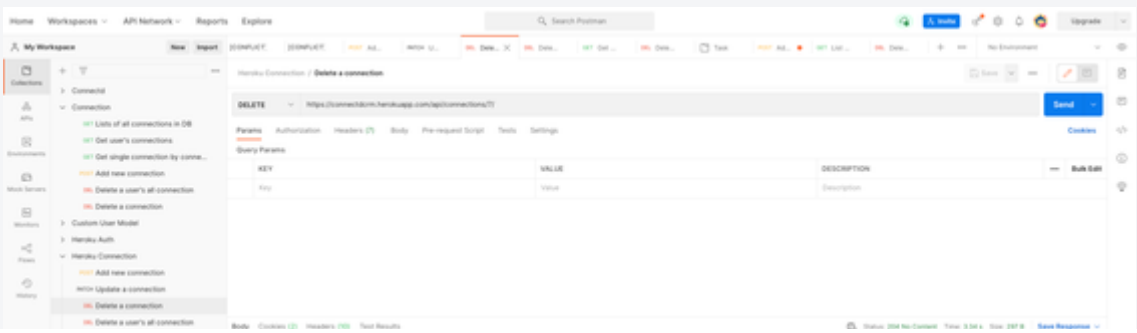
Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Delete a connection	**Designed Date**	17 Oct 2021
Test Case ID	4.5.2	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	17 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to delete a single connection with its `id`.	**Executed By**	@ Jackson Hu
Final Results	FAIL	**Execution Date**	17 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **DELETE** request in Postman			☑	☐
2	Add the value of connection `id` as the suffix in url		Appears like `... /connections/x/`	☑	☐
3	For example, we want to delete a connection which the connection `id` is 7			☑	☐
4	Post to "https://connectdcrm.herokuapp.com/api/connections/7/"	Nothing returned	Received `Type Error:` `delete_connection() missing 1 required positional argument: 'pk'`	☐	☑
Post-Condition	Unable to delete a connection from database				
Verified By	@ Jackson Hu				
Comments	Error raised in function responsible to delete the connection, didn't pass the argument of `pk` into function due to logic error.				
Screenshots					

	
Time Constraint	< 3s

TC 4.5.3 Django Request - Delete a Connection

Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Delete a connection	**Designed Date**	17 Oct 2021
Test Case ID	4.5.3	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	17 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to delete a single connection with its `id`.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	17 Oct 2021

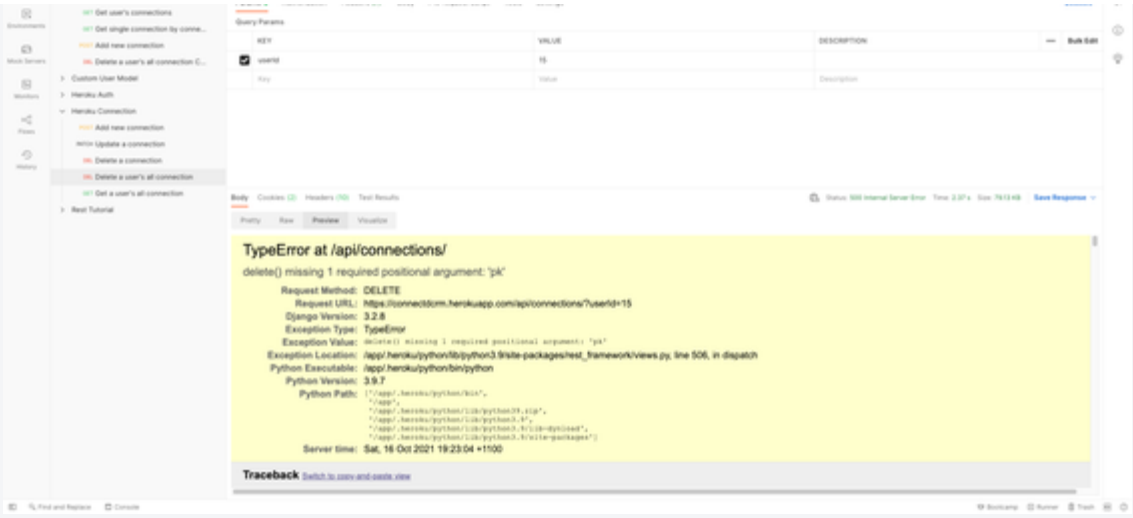
Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **DELETE** request in Postman			☑	☐
2	Add the value of connection `id` as the suffix in url		Appears like `... /connections/x/`	☑	☐
3	For example, we want to delete a connection which the connection `id` is 7			☑	☐
4	Post to "https://connectdcrm.herokuapp.com/api/connections/7/"	Nothing returned	Received `204 No Content`	☑	☐
Post-Condition	Delete a connection from database given its connection `id` already known				
Verified By	@ Jackson Hu				
Comments	Fixed the `Type Error` for unable to execute corresponding delete function correctly.				
Screenshots					

	
Time Constraint	< 3.5s

TC 4.5.4 Django Request - Delete a User's All Connections

Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Delete a user's all connections	**Designed Date**	16 Oct 2021
Test Case ID	4.5.4	**Last Modified By**	@ Jackson Hu
Test Priority	Medium	**Last Modified Date**	16 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to delete a user's all connections given the `userId`.	**Executed By**	@ Jackson Hu
Final Results	FAIL	**Execution Date**	16 Oct 2021

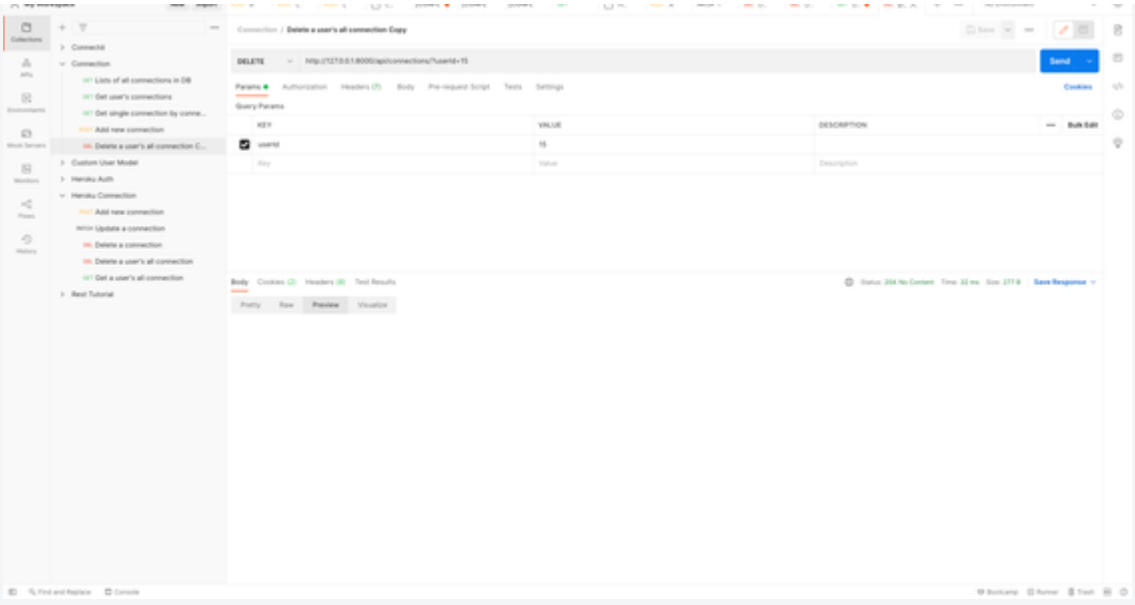
Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **DELETE** request in Postman			☑	☐
2	Add a `Query Parameter` such that key is `userId` and value is the `userId` of a given user	Should add a suffix in url as well	Appears like `?userId = x`	☑	☐
3	For example, we want to delete all connections with `userId` is 15			☑	☐
4	Post to "https://connectdcrm.herokuapp.com/api/connections/?userId = 15"	Nothing returned	Received `Type Error:` `delete_connection() missing 1 required positional argument: 'pk'`	☐	☑
Post-Condition	Unable to delete all connections of a user from database				
Verified By	@ Jackson Hu				
Comments	Error raised in function responsible to delete the connection, didn't pass the argument of `pk` into function due to logic error.				
Screenshots					

	
Time Constraint	< 3.5s

TC 4.5.5 Django Request - Delete a User's All Connections

Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Delete a user's all connections	**Designed Date**	16 Oct 2021
Test Case ID	4.5.5	**Last Modified By**	@ Jackson Hu
Test Priority	Medium	**Last Modified Date**	16 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to delete a user's all connections given the `userId`.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	16 Oct 2021

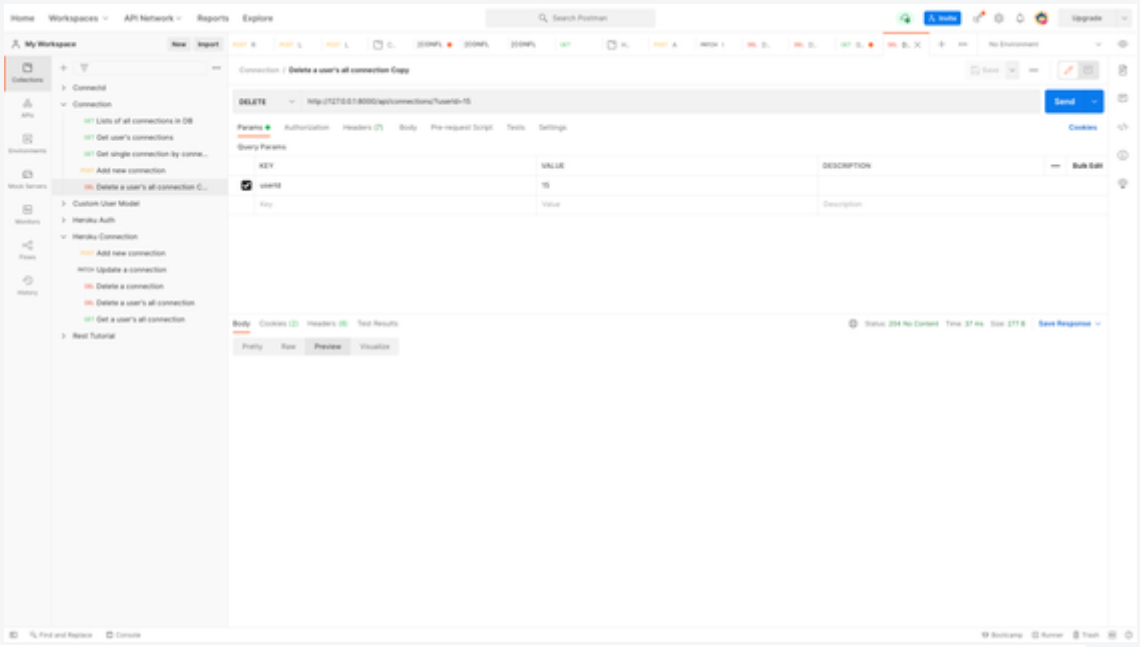
Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **DELETE** request in Postman			☑	☐
2	Add a `Query Parameter` such that key is `userId` and value is the `userId` of a given user	Should add a suffix in url as well	Appears like `?userId = x`	☑	☐
3	For example, we want to delete all connections with `userId` is 15			☑	☐
4	Post to "https://connectdcrm.herokuapp.com/api/connections/?userId = 15"	Nothing returned	Received `204 No Content`	☑	☐
Post-Condition	Delete all connections of a user from database				
Verified By	@ Jackson Hu				
Comments	Fixed the `Type Error` for unable to execute corresponding delete function correctly.				
					

Screenshots	
Time Constraint	< 3.5s

TC 4.5.6 Django Request - Delete a User's All Connections

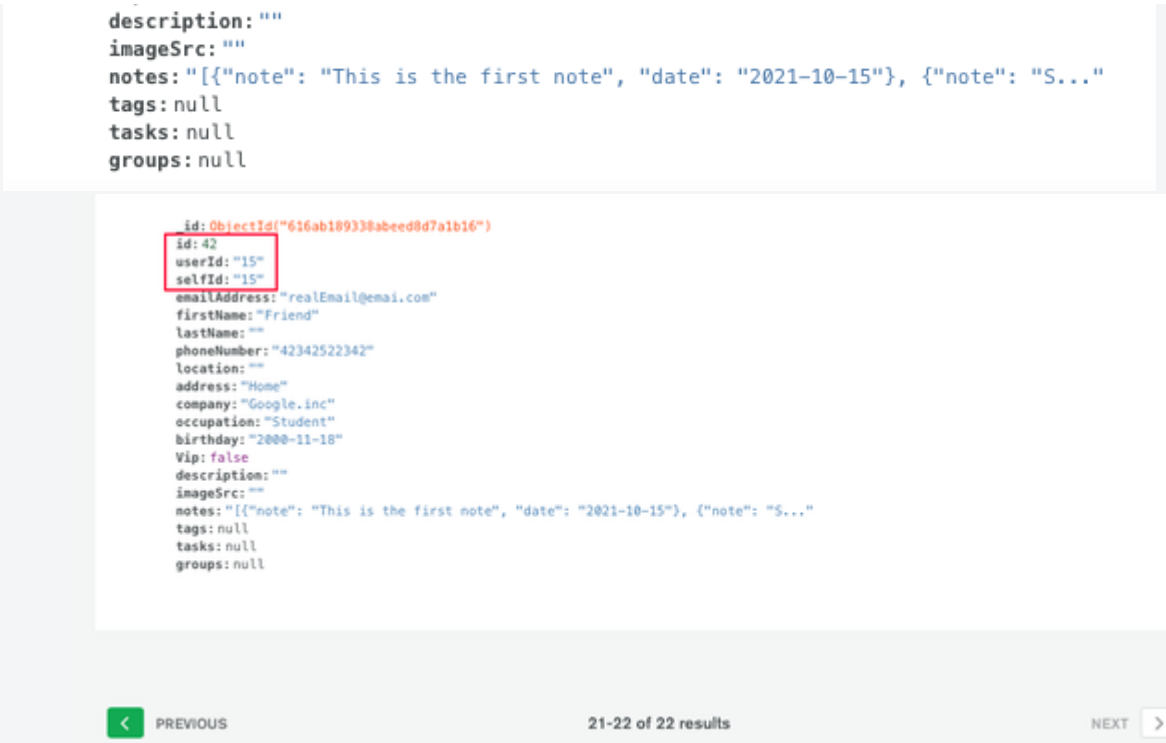
Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Delete a user's all connections	**Designed Date**	16 Oct 2021
Test Case ID	4.5.6	**Last Modified By**	@ Jackson Hu
Test Priority	Medium	**Last Modified Date**	16 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to delete a user's all connections given the `userId`, except user's own information.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	16 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **DELETE** request in Postman			☑	☐
2	Add a `Query Parameter` such that key is `user Id` and value is the `userId` of a given user	Should add a suffix in url as well	Appears like `?userId = x`	☑	☐
3	For example, we want to delete all connections with `userId` is 15			☑	☐
4	Post to "https://connectdcrm.herokuapp.com/api/connections/?userId = 15"	Nothing returned	Received `204 No Content`	☑	☐
Post-Condition	Delete all connections belong to this user from database				
Verified By	@ Jackson Hu				

Comments	Update the logic and only delete the connections belong to this user. Still keep the user's own information (as a connection record) in database, `userId` == `selfId`.
Screenshots	


```
_id: ObjectId("616ab1bc2d8f76a4a3e0fac7")
id: 43
userId: "15"
selfId: ""
emailAddress: "realEmail@emai.com"
firstName: "Friend"
lastName: ""
phoneNumber: "42342522342"
location: ""
address: "Home"
company: "Google.inc"
occupation: "Student"
birthday: "2000-11-18"
Vip: false
description: ""
imageSrc: ""
notes: "[{"note": "This is the first note", "date": "2021-10-15"}, {"note": "S..."
tags: null
tasks: null
groups: null
```

```
_id: ObjectId("616ab1cc9ec2f3e5f8ef72a8")
id: 44
userId: "15"
selfId: ""
emailAddress: "realEmail@emai.com"
firstName: "Friend"
lastName: ""
phoneNumber: "42342522342"
location: ""
address: "Home"
company: "Google.inc"
occupation: "Student"
birthday: "2000-11-18"
Vip: false
```

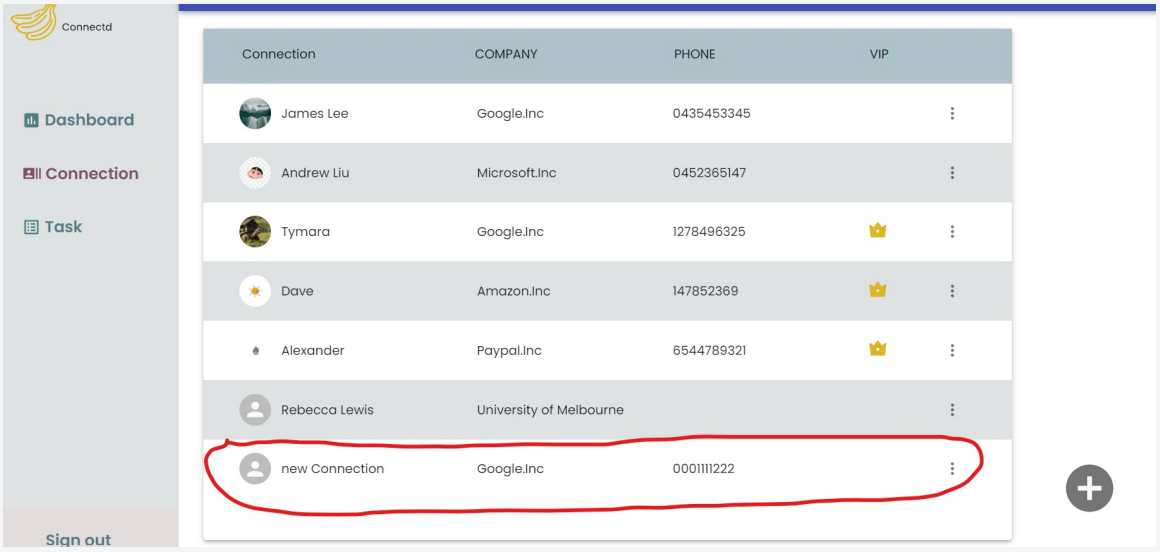
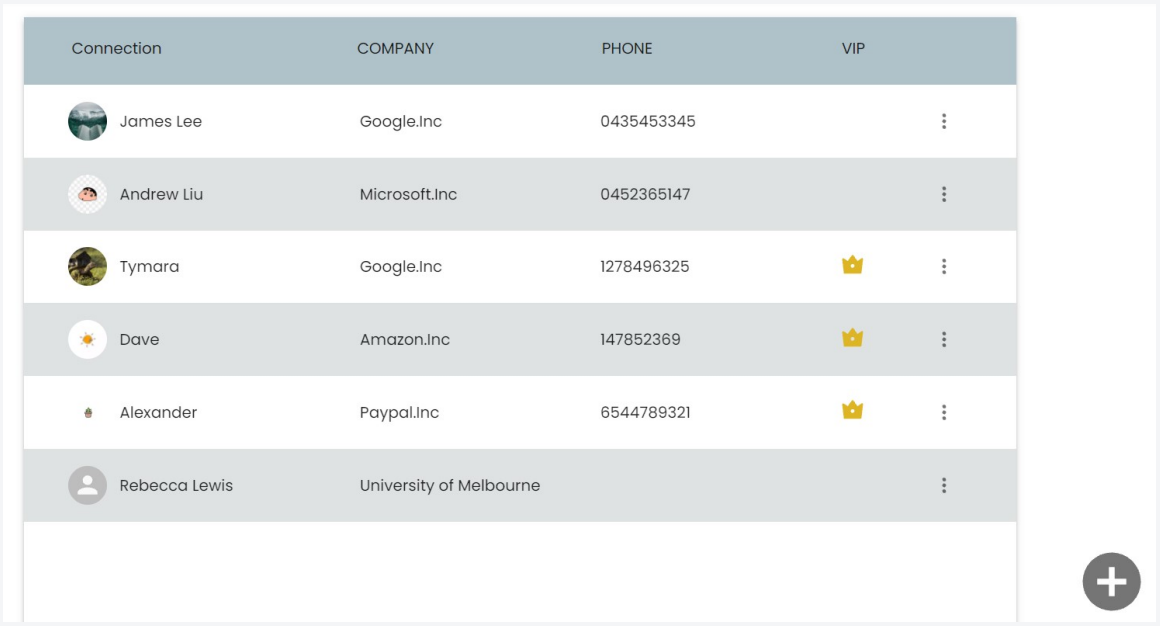


Time Constraint	< 3.5s

TC 4.5.7 Front-End - User Delete a Connection

Test/Execution Type	**Manual**	**Designed By**	@ Han Liu
Test Case Name	Front-End - User Delete a Connection	**Designed Date**	22/10/2021
Test Case ID	4.5.7	**Last Modified By**	@ Han Liu
Test Priority	High	**Last Modified Date**	22/10/2021
Description	Testing if logged in user can delete a connection	**Executed By**	@ Han Liu
Final Results	PASS	**Execution Date**	22/10/2021

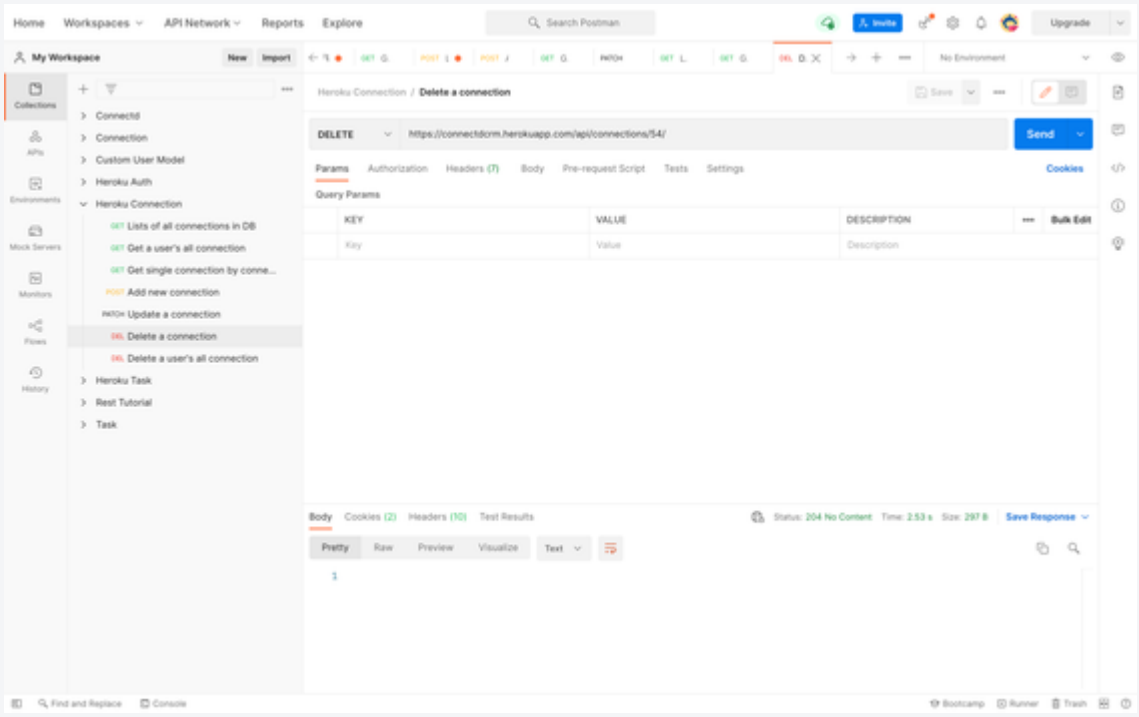
Pre-Condition	User logged in				
TC Step No.	**Setup** navigate to https://connectd-front.herokuapp.com/ and logged into account	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Click <Delete> button on connection page			☑	☐
3	Check if deleted connection has been removed from connection page	Delete connection is removed from connection page	Delete connection is removed from connection page	☑	
Post-Condition	N/A				

Verified By	@ Han Liu
Comments	User successfully delete a connection via Front-End application
Screenshots	 
Time Constraint	N/A

TC 4.5.8 Django Request - Delete a Connection

Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Delete a connection	**Designed Date**	26 Oct 2021
Test Case ID	4.5.8	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	28 Oct 2021
Description		**Executed By**	@ Jackson Hu

	Send a POST request to Django backend server deployed on Heroku to delete a single connection with its `id`.		
Final Results	**PASS**	**Execution Date**	26 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **DELETE** request in Postman			☑	☐
2	Add the value of connection `id` as the suffix in url		Appears like `... /connections/x/`	☑	☐
3	For example, we want to delete a connection which the connection `id` is 54			☑	☐
4	Post to "https://connectdcrm.herokuapp.com/api/connections/54/"	Nothing returned	Received `204 No Content`	☑	☐
Post-Condition	Delete a connection from database given its connection `id` already known				
Verified By	@ Jackson Hu				
Comments	Fixed the `Type Error` for unable to execute corresponding delete function correctly.				
Screenshots					
Time Constraint	< 3.5s				

TC 5 Tasks Testing

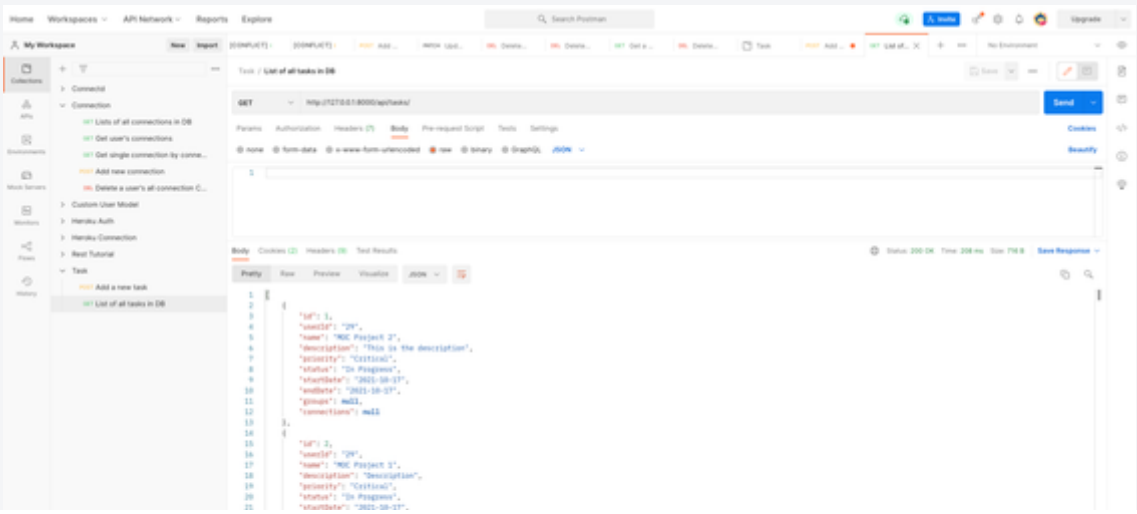
- TC 5.1 Retrieve Task Functionality

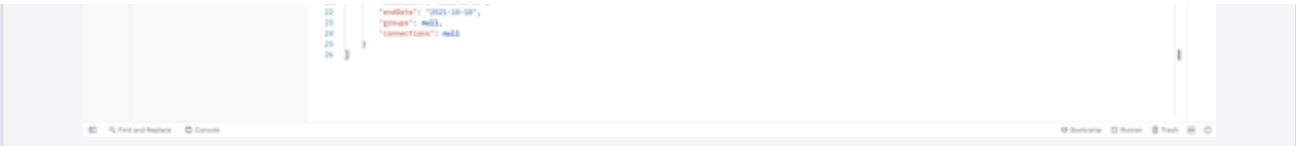
- TC 5.2 Adding Task Functionality
- TC 5.2 Edit Task Functionality
- TC 5.3 Delete Task Functionality

TC 5.1 Retrieve Task Functionality

TC 5.1.1 Django Request - Get All Tasks

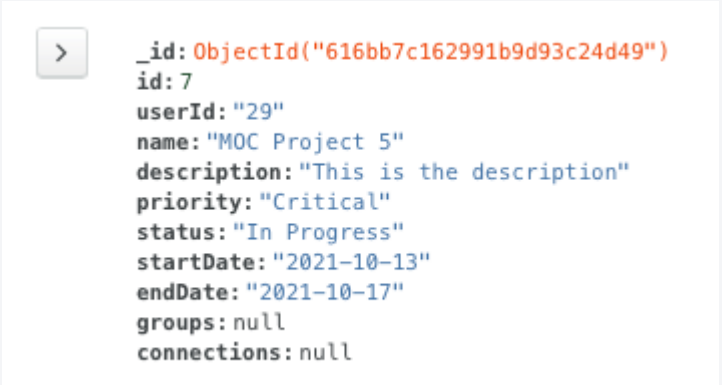
Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Get all tasks request	**Designed Date**	17 Oct 2021
Test Case ID	5.1.1	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	17 Oct 2021
Description	Locally send a POST request to Django backend server to retrieve all tasks stored in database.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	17 Oct 2021

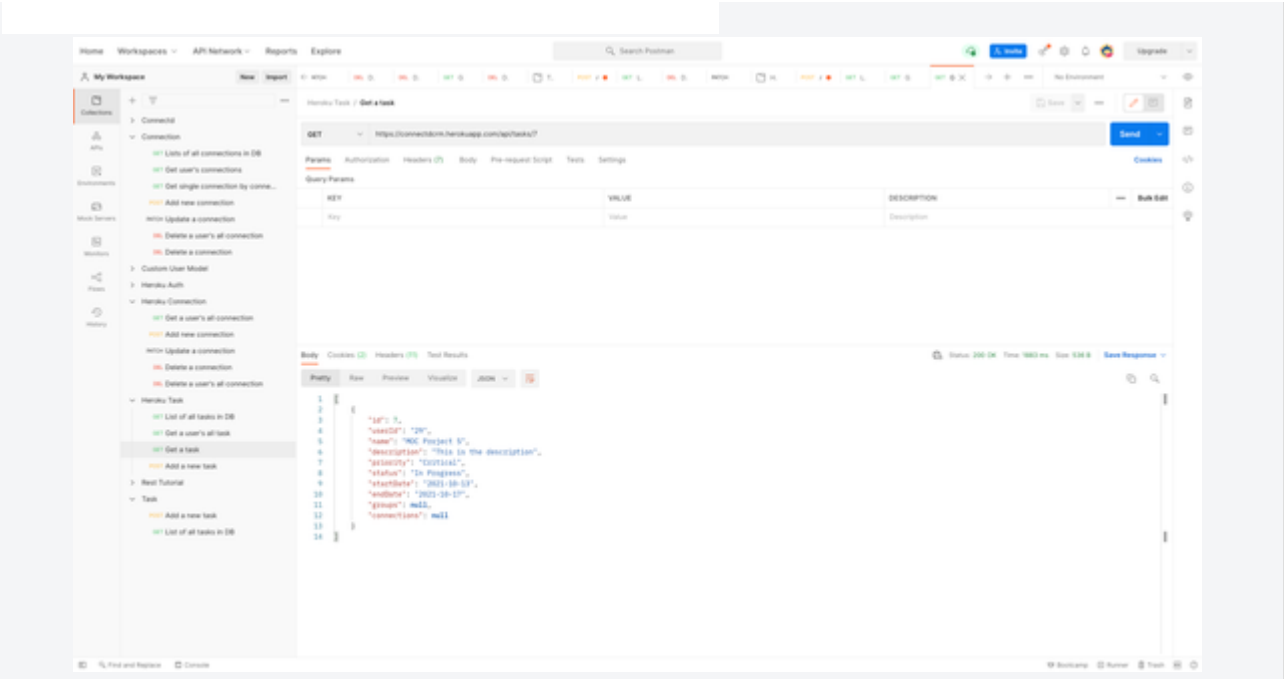
Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup** Activate virtual environment and run Django server locally	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **GET** request in Postman			☑	☐
2	No need to include a JSON file as body			☑	☐
3	Post to "http://127.0.0.1:8000/api/tasks/"	Return a JSON file with all task records currently stored in database	Expected JSON response received which has list of JSON object for each task record	☑	☐
Post-Condition	Retrieve all task records stored in database.				
Verified By	@ Jackson Hu				
Comments	Get all tasks belong to all users. Since we've set up the CORS so this API won't be used by normal users / malicious users.				
Screenshots					

	
Time Constraint	< 1s

TC 5.1.2 Django Request - Get a Single Task

Test/Execution Type	Unit / Manual	Designed By	@ Jackson Hu
Test Case Name	Get a single task	**Designed Date**	17 Oct 2021
Test Case ID	5.1.2	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	17 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to retrieve a single task with its `id`.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	17 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **GET** request in Postman			☑	☐
2	Add the value of task `id` as the suffix in url		Appears like `.../tasks/x/`	☑	☐
3	For example, we want to get a task which the task `id` is 7			☑	☐
4	Post to "https://connectdcrm.herokuapp.com/api/tasks/7/"	Return a JSON file with the information of a task given its `id`	Expected JSON response received with only a single task with corresponding `id`	☑	☐
Post-Condition	Retrieve the information of a single task given its task `id` already known				
Verified By	@ Jackson Hu				
Comments	The `id` attribute of a task is generated automatically when the task is created. Tasks in database are filtered by its `id` attribute.				
Screenshots					

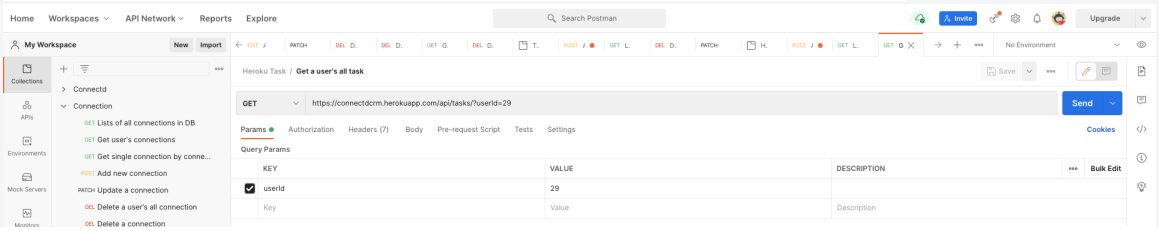
	
Time Constraint	< 3s

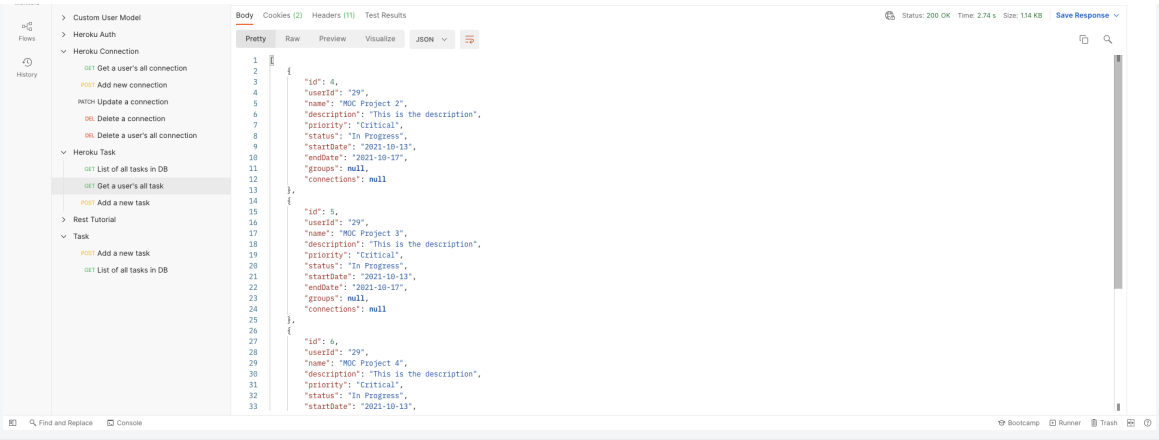
TC 5.1.3 Django Request - Get a User's Task

Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Get a user's task	**Designed Date**	17 Oct 2021
Test Case ID	5.1.3	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	17 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to retrieve all tasks belong to a given user.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	17 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **GET** request in Postman			☑	☐
2	Add a `Query Parameter` such that key is `user Id` and value is the `userId` of a given user	Should add a suffix in url as well	Appears like `?userId = x`	☑	☐
3	For example, we want to get tasks belong to a user whose `userId` is 29			☑	☐
4	Post to "https://connectdcrm.herokuapp.com/api/tasks/?userId=29"	Return a JSON file with all task related to the given user	Expected JSON response received which has list of JSON object for each task record	☑	☐
	Retrieve all tasks belong to a given user.				

Post-Condition	
Verified By	@ Jackson Hu
Comments	Get all tasks belong to a given user. Tasks in database are filtered by its `userId` attribute.

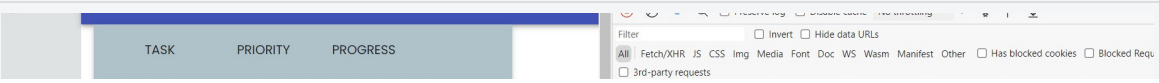
Screenshots	`_id: ObjectId("616bb7b262991b9d93c24d48")`
	`id: 5`
	`userId: "29"`
	`name: "MOC Project 3"`
	`description: "This is the description"`
	`priority: "Critical"`
	`status: "In Progress"`
	`startDate: "2021-10-13"`
	`endDate: "2021-10-17"`
	`groups: null`
	`connections: null`
	
	`_id: ObjectId("616bb7bad323ab4f720c578b")`
	`id: 6`
	`userId: "29"`
	`name: "MOC Project 4"`
	`description: "This is the description"`
	`priority: "Critical"`
	`status: "In Progress"`
	`startDate: "2021-10-13"`
	`endDate: "2021-10-17"`
	`groups: null`
	`connections: null`
	
	`_id: ObjectId("616bb7c162991b9d93c24d49")`
	`id: 7`
	`userId: "29"`
	`name: "MOC Project 5"`
	`description: "This is the description"`
	`priority: "Critical"`
	`status: "In Progress"`
	`startDate: "2021-10-13"`
	`endDate: "2021-10-17"`
	`groups: null`
	`connections: null`
	
	 

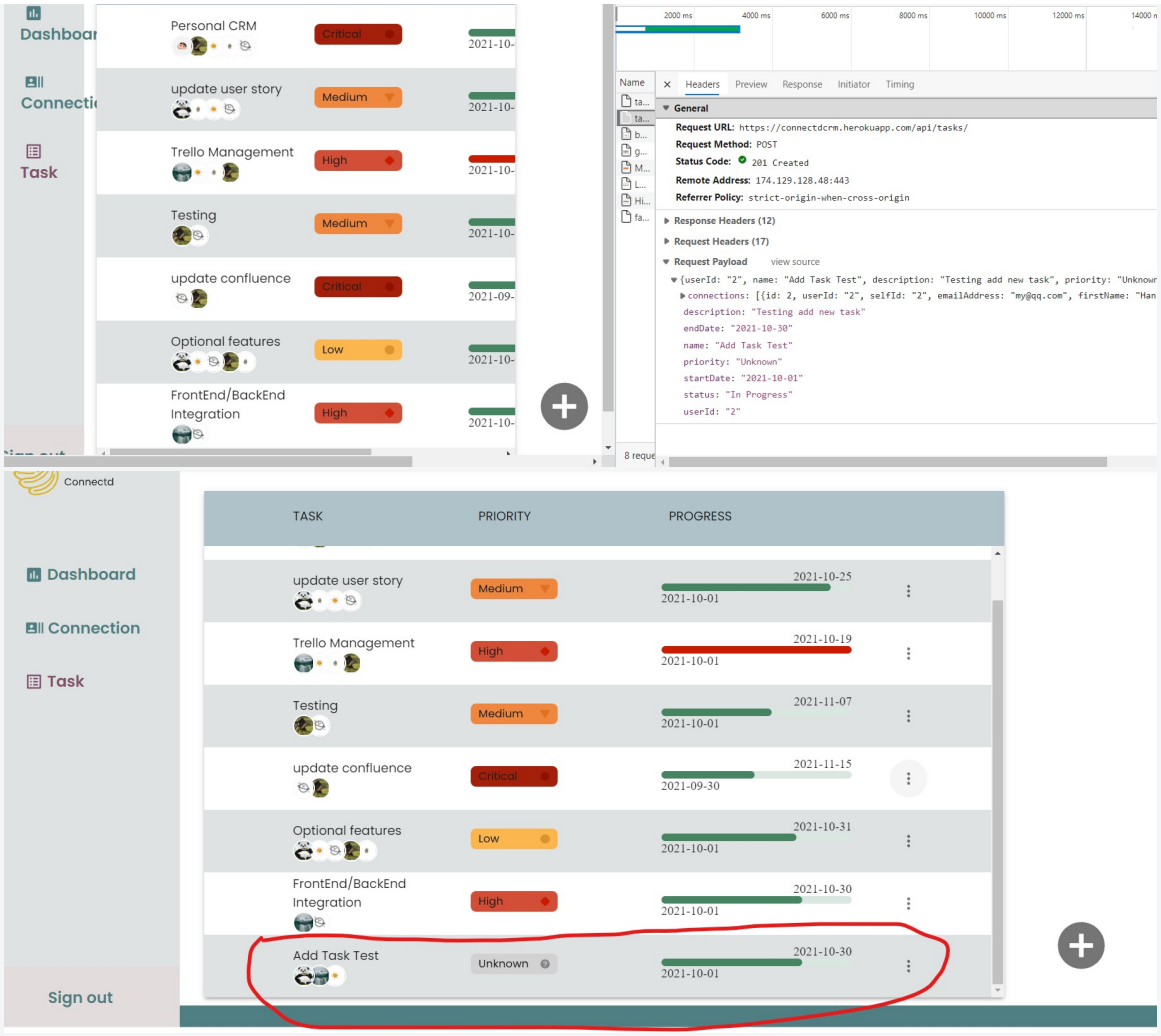
	
Time Constraint	< 3s

TC 5.2 Adding Task Functionality

TC 5.2.1 Front-End - User Add New Task

Test/Execution Type	**Manual**	**Designed By**	@ Han Liu
Test Case Name	Front-End - User Add New Task	**Designed Date**	22 Oct 2021
Test Case ID	5.2.1	**Last Modified By**	@ Han Liu
Test Priority	High	**Last Modified Date**	22 Oct 2021
Description	Testing if logged in user can add new task	**Executed By**	@ Han Liu
Final Results	PASS	**Execution Date**	22 Oct 2021

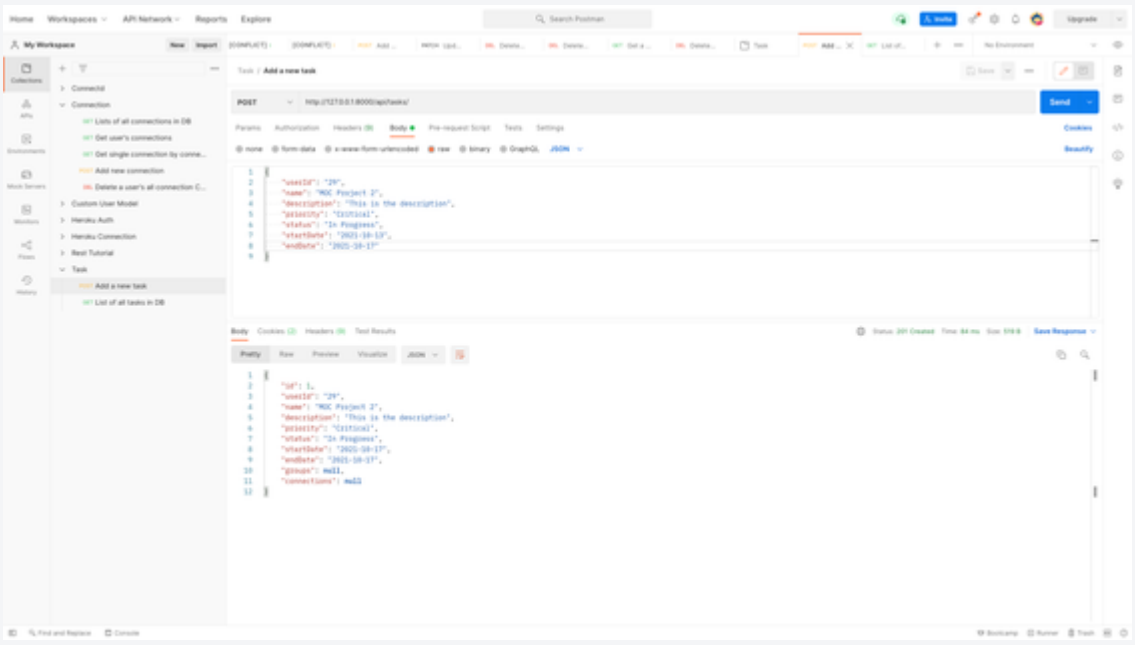
Pre-Condition	User logged in				
TC Step No.	**Setup** navigate to https://connectd-front.herokuapp.com/ and logged into account	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Click <Add Task>, then fill in task detail			☑	☐
2	Send task detail to backend	task detail has been sent to backend	task detail has been sent to backend	☑	
3	Check if new task has been added to database	new task is showed in task page	new task is showed in task page	☑	
Post-Condition	N/A				
Verified By	@ Han Liu				
Comments	User successfully added new task via Front-End application				
					

Screenshots	
Time Constraint	N/A

TC 5.2.2 Django Request - Add a New Task

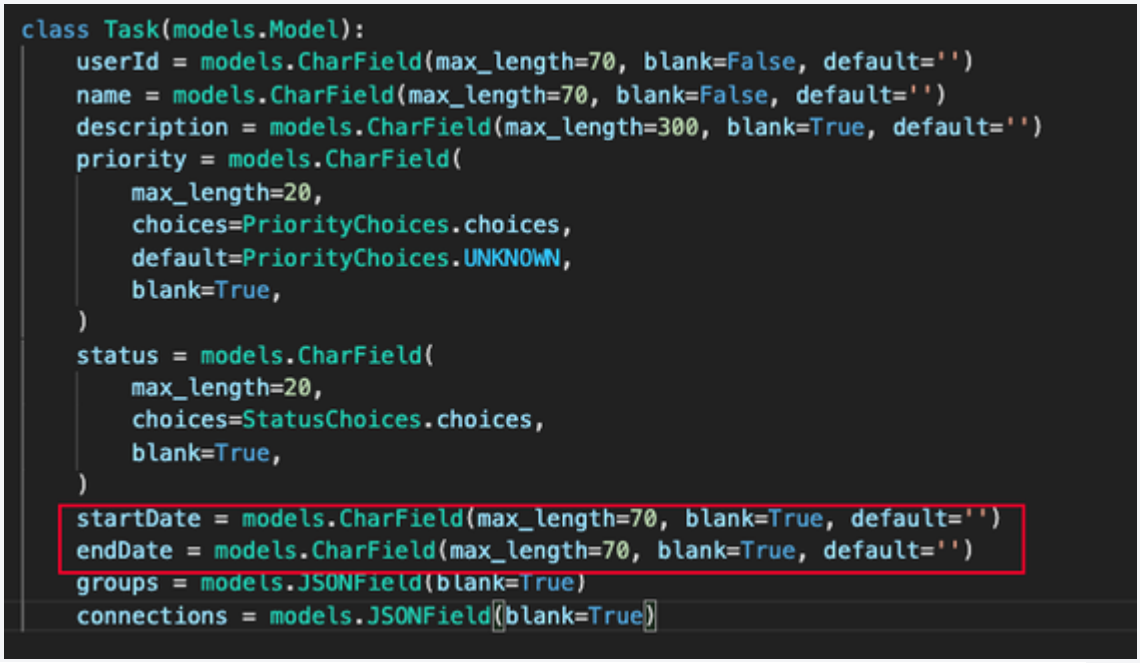
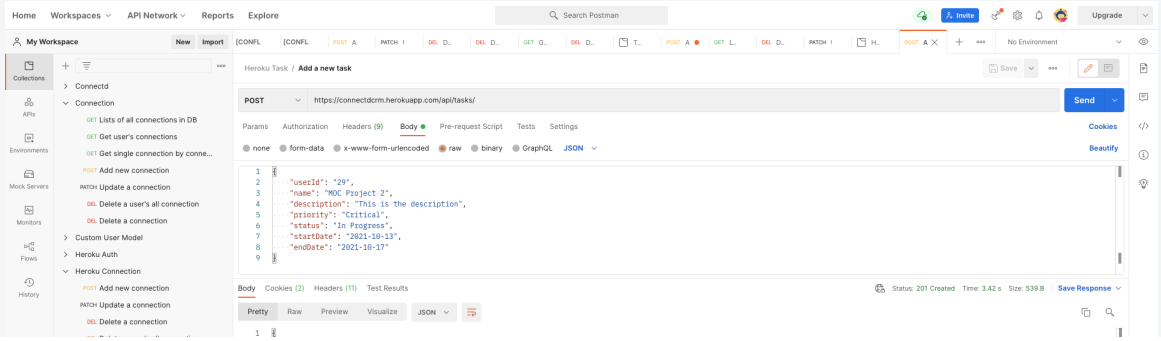
Test/Execution Type	**Unit / Manual**	**Designed By**	@ Jackson Hu
Test Case Name	Add new task request	**Designed Date**	17 Oct 2021
Test Case ID	5.2.2	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	17 Oct 2021
Description	Locally send a POST request to Django backend server to add a new task.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	17 Oct 2021

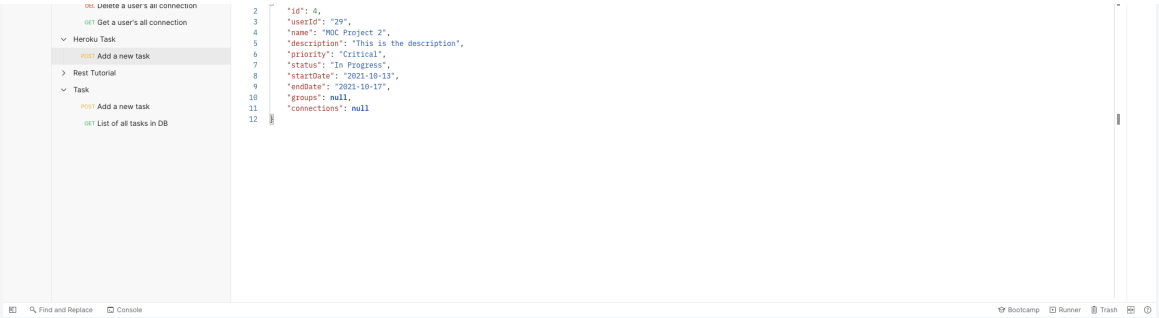
Pre-Condition	Locally running Django backend with port:8000				
TC Step No.	**Setup**	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
	Activate virtual environment and run Django server locally				

1	Create a **POST** request in Postman			☑	☐
2	Prepare a JSON file as the body of request with fields showed on the screenshot attached below			☑	☐
3	Fill in those fields with task information as value. `userId` and `name` are compulsory, can drop other fields or leave the values as empty.			☑	☐
4	Post to "http://127.0.0.1:8000/api/tasks/"	Return an HTTP response as JSON file with task record `id` and serialised data	Expected JSON response received	☑	☐
Post-Condition	Successfully added a new task record in database with given information.				
Verified By	@ Jackson Hu				
Comments	Currently just test locally. Notice that some unnecessary fields could be dropped when sending request to backend server: `userId` and `name` are compulsory, can drop other fields or leave the values as empty.				
Screenshots					
Time Constraint	< 1s				

TC 5.2.3 Django Request - Add a New Task

Test/Execution Type	**Unit / Manual**	**Designed By**	**@ Jackson Hu**
Test Case Name	Add new task request	**Designed Date**	17 Oct 2021
Test Case ID	5.2.3	**Last Modified By**	@ Jackson Hu
Test Priority	High	**Last Modified Date**	17 Oct 2021
Description	Send a POST request to Django backend server deployed on Heroku to add a new task. `DateField` replaced with `CharField`.	**Executed By**	@ Jackson Hu
Final Results	PASS	**Execution Date**	17 Oct 2021

Pre-Condition	Django backend server deployed on Heroku				
TC Step No.	**Setup** Ensure Django backend server is running on Heroku	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Create a **POST** request in Postman			☑	☐
2	Prepare a JSON file as the body of request with fields showed on the screenshot attached below			☑	☐
3	Fill in those fields with task information as value. `userId` and `name` are compulsory, can drop other fields or leave the values as empty.			☑	☐
4	Post to "https://connectdcrm.herokuapp.com/api/tasks/"	Return an HTTP response as JSON file with task record `id` and serialised data	Expected JSON response received	☑	☐
Post-Conditon	Successfully added a new task record in database with given information.				
Verified By	@ Jackson Hu				
Comments	Notice that some unnecessary fields could be dropped when sending request to backend server: `userId` and `name` are compulsory, can drop other fields or leave the values as empty. `DateField` was used to save the start date and end date of a task, but now it's been replaced with `CharField` for better compatibility with frontend.				
Screenshots	 				




```
_id: ObjectId("616bb689d323ab4f720c578a")
id: 4
userId: "29"
name: "MOC Project 2"
description: "This is the description"
priority: "Critical"
status: "In Progress"
startDate: "2021-10-13"
endDate: "2021-10-17"
groups: null
connections: null
```


Time Constraint	< 3.5s

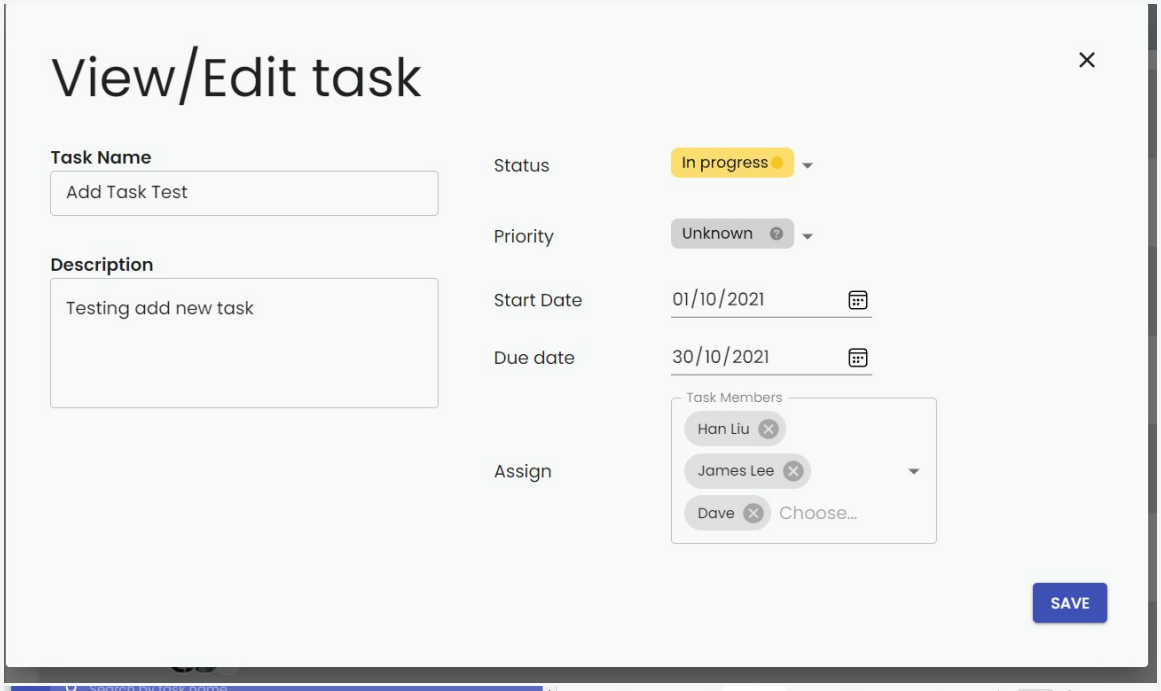
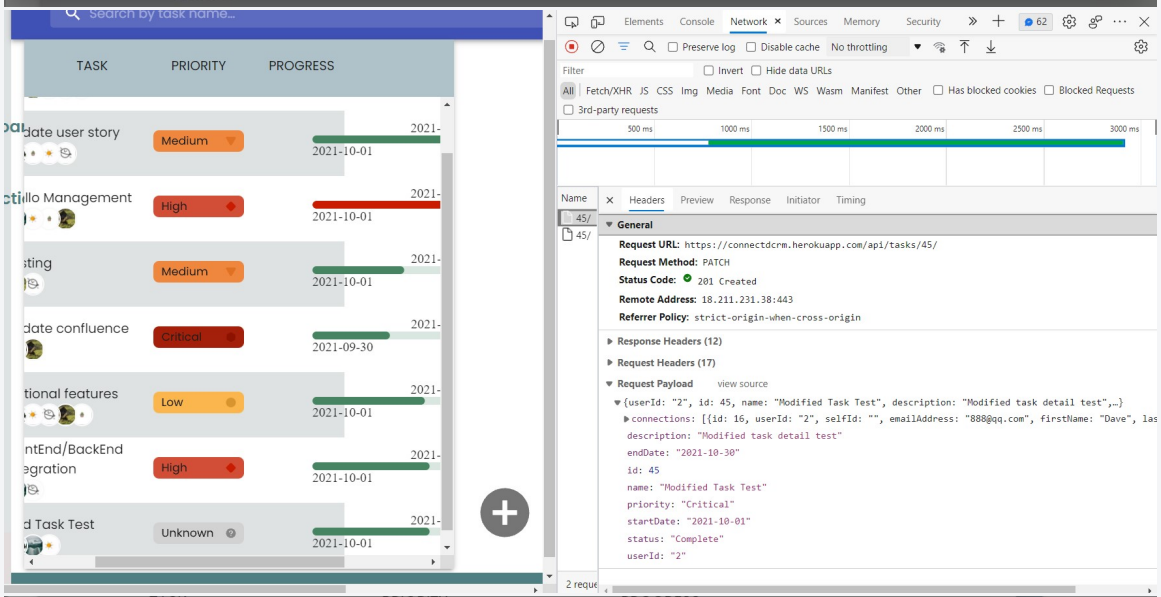
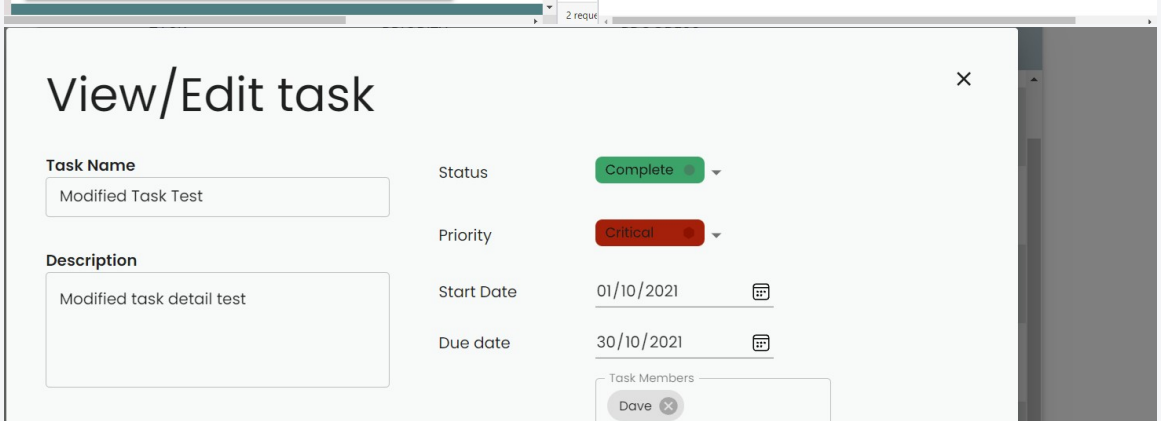
TC 5.2 Edit Task Functionality

- TC 5.2.1 Front-End - User Edit Task

TC 5.2.1 Front-End - User Edit Task

Test/Execution Type	**Manual**	**Designed By**	@ Han Liu
Test Case Name	Front-End - User Edit Task	**Designed Date**	22 Oct 2021
Test Case ID	5.2.1	**Last Modified By**	@ Han Liu
Test Priority	High	**Last Modified Date**	22 Oct 2021
Description	Testing if logged in user can edit task	**Executed By**	@ Han Liu
Final Results	PASS	**Execution Date**	22 Oct 2021

Pre-Condition	User logged in				
TC Step No.	**Setup** navigate to https://connectd-front.herokuapp.com/ and logged into account	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Click <Edit>, then fill in task detail			☑	☐
2	Send edit task detail to backend	edit task detail has been sent to backend	edit task detail has been sent to backend	☑	
3	Check if edit task has been added to database	edit task is showed in task page	edit task is showed in task page	☑	
Post-Condition	N/A				

Verified By	@ Han Liu
Comments	User successfully edit task via Front-End application
Screenshots	  

Time Constraint	N/A

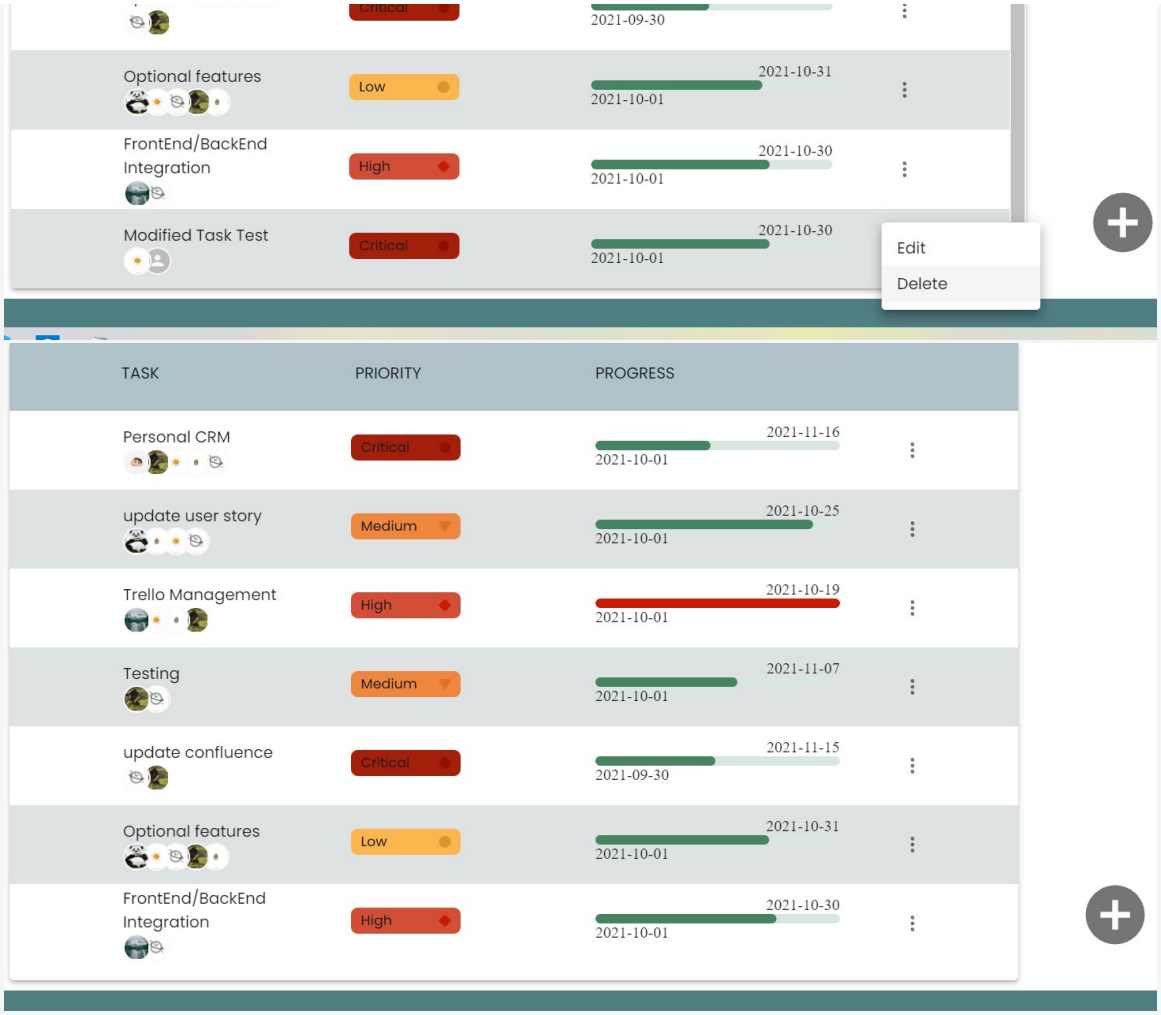
TC 5.3 Delete Task Functionality

- TC 5.3.1 Front-End - User Delete Task

TC 5.3.1 Front-End - User Delete Task

Test/Execution Type	**Manual**	**Designed By**	@ Han Liu
Test Case Name	Front-End - User Delete Task	**Designed Date**	22 Oct 2021
Test Case ID	5.3.1	**Last Modified By**	@ Han Liu
Test Priority	High	**Last Modified Date**	22 Oct 2021
Description	Testing if logged in user can delete task	**Executed By**	@ Han Liu
Final Results	PASS	**Execution Date**	22 Oct 2021

Pre-Condition	User logged in				
TC Step No.	**Setup** navigate to https://connectd-front.herokuapp.com/ and logged into account	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	Click <Delete >			☑	☐
3	Check if task has been deleted	task is not showed in task page	edit task is not showed in task page	☑	
Post-Condition	N/A				
Verified By	@ Han Liu				
Comments	User successfully delete task via Front-End application				
Screenshots					



Time Constraint	N/A

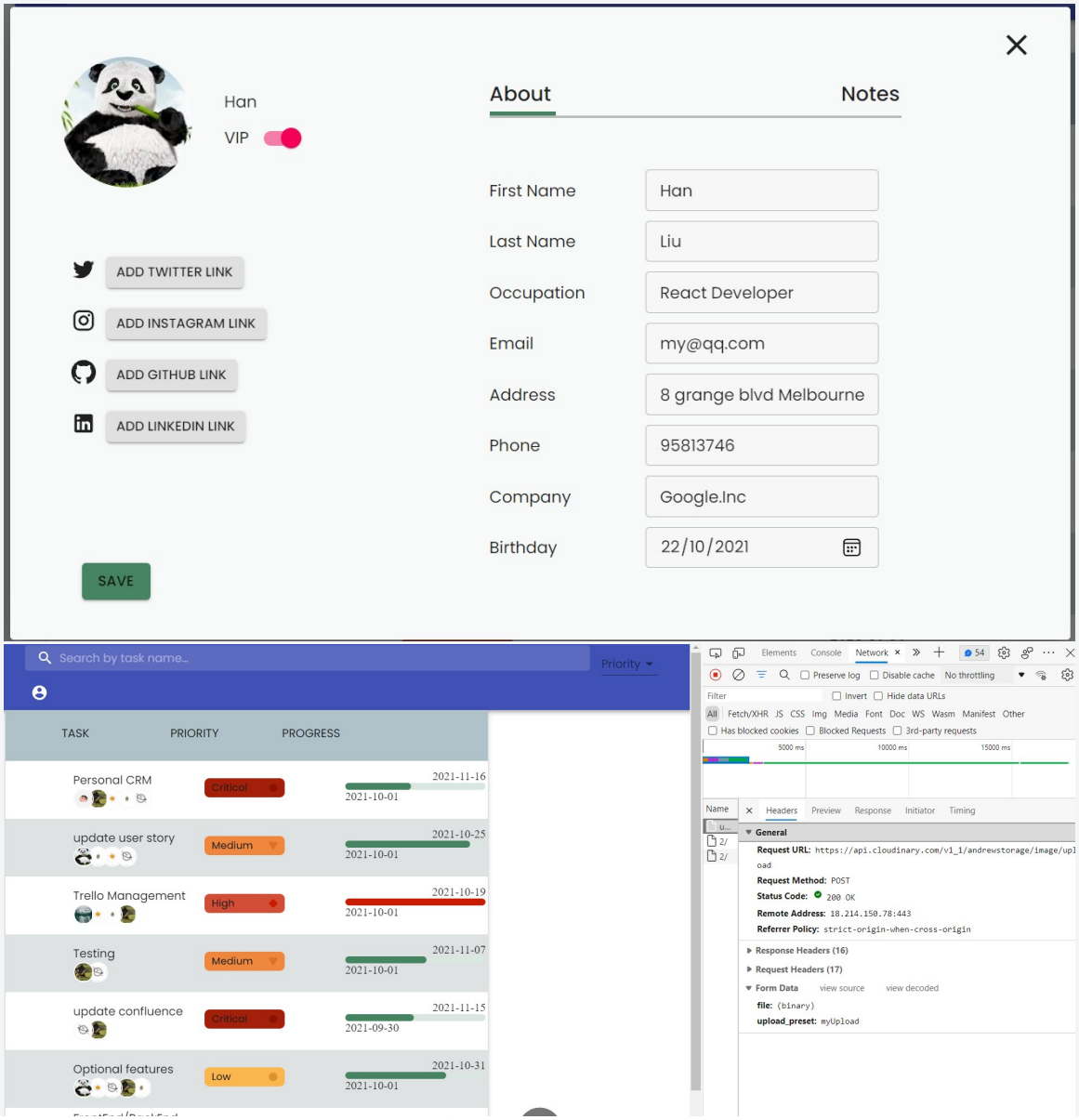
TC 6 Account Profile Functionality

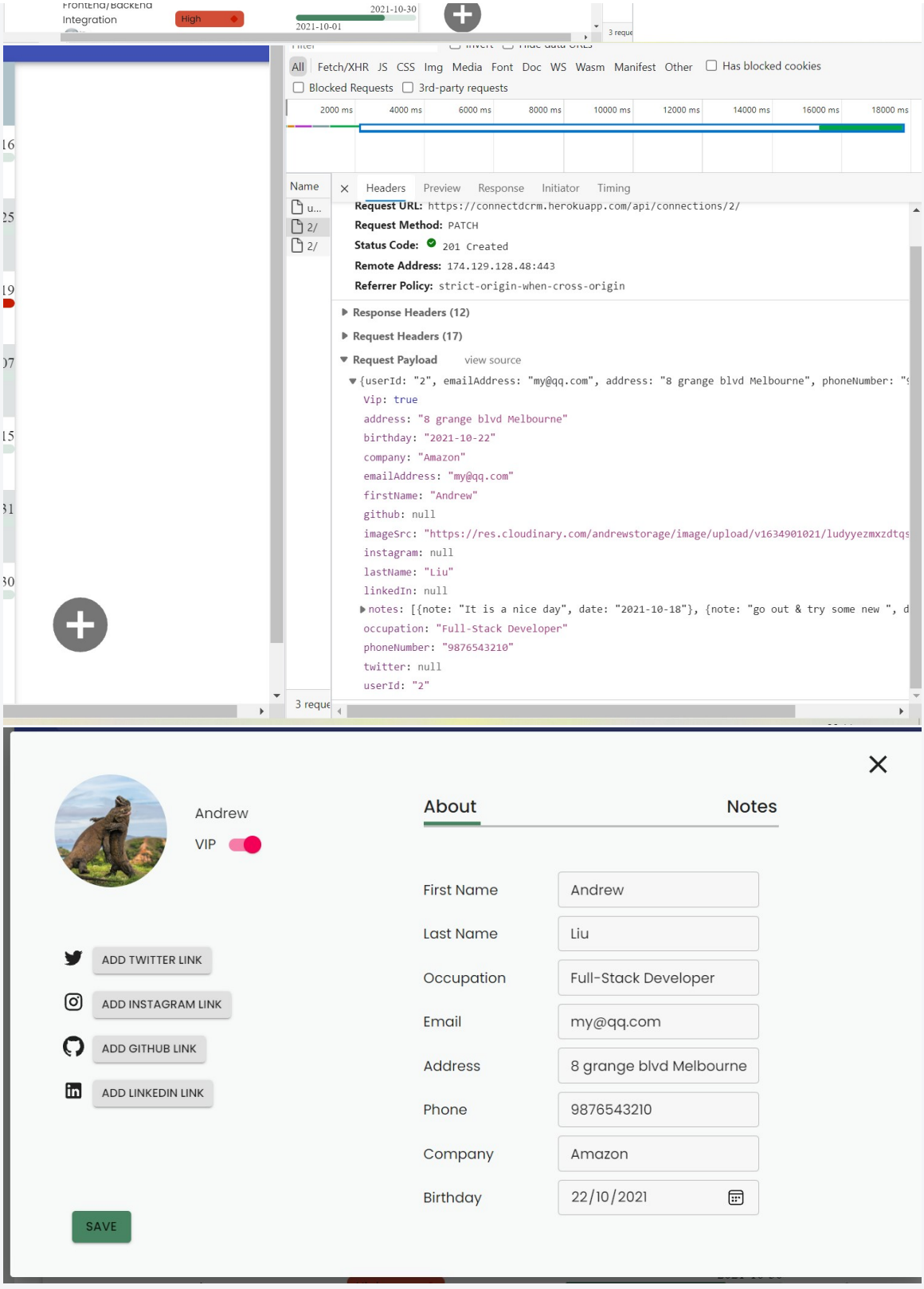
- TC 6.1 Front-End - User Profile Edit
- TC 6.2 Upload Photo

TC 6.1 Front-End - User Profile Edit

Test/Execution Type	**Manual**	**Designed By**	@ Han Liu
Test Case Name	Front-End - User Profile Edit	**Designed Date**	22 Oct 2021
Test Case ID	6.1	**Last Modified By**	@ Han Liu
Test Priority	High	**Last Modified Date**	22 Oct 2021
Description	Testing if logged in user can edit profile	**Executed By**	@ Han Liu
Final Results	PASS	**Execution Date**	22 Oct 2021

Pre-Condition	User logged in			
	Setup	**Expected Results**	**Actual Results**	**Results**

TC Step No.	navigate to https://connectd-front.herokuapp.com/ and logged into account			As Expected	Not As Expected
1	Click <Profile>, then edit in user's detail			☑	☐
2	Upload profile image	profile image has been uploaded to cloud	profile image has been uploaded to cloud	☑	
3	Send edit profile detail to backend	edited profile detail has been sent to backend	edited profile detail has been sent to backend	☑	
4	Check if edit profile has been added to database	profile is updated	profile is updated	☑	
Post-Condition	N/A				
Verified By	@ Han Liu				
Comments	User successfully edit profile via Front-End application				
Screenshots					



Time Constraint	N/A

TC 6.2 Upload Photo

Test/Execution Type	**<Unit, Functional/Manual, Auto>**	**Designed By**	
Test Case Name	<FILL>	**Designed Date**	
Test Case ID	<FILL>	**Last Modified By**	
Test Priority	<Optional, Low, Medium, High>	**Last Modified Date**	
Description	<FILL>	**Executed By**	<FILL>
Final Results	PENDING FAIL PASS	**Execution Date**	<FILL>

Pre-Condition	<FILL>				
TC Step No.	**Setup** <list the pre-conditions to carry thru this TC>	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	<FILL>	<FILL>	<FILL>	☐	☐
2	<FILL>	<FILL>	<FILL>	☐	☐
Post-Condition	<FILL>				
Verified By	<FILL>				
Comments	<FILL>				
Screenshots	<FILL>				
Time Constraint	<FILL>				

TC 7 Dashboard Functionality

TC 7.1 Summary of Tasks

Test/Execution Type	**<Unit, Functional/Manual, Auto>**	**Designed By**	
Test Case Name	<FILL>	**Designed Date**	
Test Case ID	<FILL>	**Last Modified By**	
Test Priority	<Optional, Low, Medium, High>	**Last Modified Date**	
Description	<FILL>	**Executed By**	<FILL>
Final Results	PENDING FAIL PASS	**Execution Date**	<FILL>

Pre-Condition	<FILL>				
TC Step No.	**Setup** <list the pre-conditions to carry thru this TC>	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	<FILL>	<FILL>	<FILL>	☐	☐
2	<FILL>	<FILL>	<FILL>	☐	☐
Post-Condition	<FILL>				
Verified By	<FILL>				
Comments	<FILL>				
Screenshots	<FILL>				

Time Constraint	<FILL>

TC 7.2 Summary of Connections

Test/Execution Type	**<Unit, Functional/Manual, Auto>**	**Designed By**	
Test Case Name	<FILL>	**Designed Date**	
Test Case ID	<FILL>	**Last Modified By**	
Test Priority	<Optional, Low, Medium, High>	**Last Modified Date**	
Description	<FILL>	**Executed By**	<FILL>
Final Results	PENDING FAIL PASS	**Execution Date**	<FILL>

Pre-Condition	<FILL>				
TC Step No.	**Setup** <list the pre-conditions to carry thru this TC>	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	<FILL>	<FILL>	<FILL>	☐	☐
2	<FILL>	<FILL>	<FILL>	☐	☐
Post-Condition	<FILL>				
Verified By	<FILL>				
Comments	<FILL>				
Screenshots	<FILL>				
Time Constraint	<FILL>				

TC 7.3 Connection Metrics

Test/Execution Type	**<Unit, Functional/Manual, Auto>**	**Designed By**	
Test Case Name	<FILL>	**Designed Date**	
Test Case ID	<FILL>	**Last Modified By**	
Test Priority	<Optional, Low, Medium, High>	**Last Modified Date**	
Description	<FILL>	**Executed By**	<FILL>
Final Results	PENDING FAIL PASS	**Execution Date**	<FILL>

Pre-Condition	<FILL>				
TC Step No.	**Setup** <list the pre-conditions to carry thru this TC>	**Expected Results**	**Actual Results**	**Results**	
				As Expected	**Not As Expected**
1	<FILL>	<FILL>	<FILL>	☐	☐
2	<FILL>	<FILL>	<FILL>	☐	☐
Post-Condition	<FILL>				
Verified By	<FILL>				
Comments	<FILL>				
Screenshots	<FILL>				

Time Constraint	<FILL>

TC 8 System Testing

TC 8.1 Create Account/Login

TC 8.2 Navigation

Test Case Log

Test Case	Test Case ID	Test Priority	Final Results
TC 4.3.3 Django Request - Add a New Connection	4.3.3	High	PASS
TC 4.1.3 Update Model - Add Social Media Links	4.1.3	Medium	PASS
TC 4.2.8 Django Request - Get a User's Connection	4.2.8	High	PASS
TC 4.2.7 Django Request - Get a User's Connection	4.2.7	High	FAIL
TC 5.2.2 Django Request - Add a New Task	5.2.2	High	PASS
TC 5.2.3 Django Request - Add a New Task	5.2.3	High	PASS
TC 5.1.3 Django Request - Get a User's Task	5.1.3	High	PASS
TC 5.1.2 Django Request - Get a Single Task	5.1.2	High	PASS
TC 5.1.1 Django Request - Get All Tasks	5.1.1	Medium	PASS
TC 3.9 Django Logout Request	3.9	High	PASS
TC 2.9 Django Login Request	2.9	High	PASS
TC 1.8 Django Registration Request	1.8	High	PASS
TC 4.5.8 Django Request - Delete a Connection	4.5.8	High	PASS
TC 4.2.6 Django Request - Get a Single Connection	4.2.6	High	PASS
TC 3.8 Django Logout Request (Script)	3.8	High	PASS
TC 2.8 Django Login Request (Script)	2.8	High	PASS
TC 1.7 Django Registration Request (Script)	1.7	High	PASS
TC 4.5.6 Django Request - Delete a User's All Connections	4.5.6	Medium	PASS
TC 4.5.5 Django Request - Delete a User's All Connections	4.5.5	Medium	PASS
TC 4.5.4 Django Request - Delete a User's All Connections	4.5.4	Medium	FAIL
TC 4.5.3 Django Request - Delete a Connection	4.5.3	High	PASS
TC 4.5.2 Django Request - Delete a Connection	4.5.2	High	FAIL
TC 4.4.1 Django Request - Update a Connection	4.4.1	High	PASS

TC 4.5.1 Django Request - Delete a Connection	4.5.1	High	PASS
TC 4.2.4 Django Request - Get a User's Connection	4.2.4	High	PASS
TC 4.1.2 Update Model - Update JSONField	4.1.2	High	PASS
TC 4.2.3 Django Request - Get a Single Connection	4.2.3	High	PASS
TC 4.2.2 Django Request - Get a User's Connection	4.2.2	High	PASS
TC 4.2.1 Django Request - Get All Connections	4.2.1	Medium	PASS
TC 4.1.1 Update Model - Include selfId	4.1.1	High	PASS
TC 4.3.1 Django Request - Add a New Connection	4.3.1	High	PASS
TC 3.6 Django Logout Request	3.6	High	PASS
TC 2.6 Django Login Request	2.6	High	PASS
TC 1.5 Django Registration Request	1.5	High	PASS
TC 2.4 Django Login Request	2.4	High	PASS
TC 3.4 Django Logout Request	3.4	High	PASS
TC 3.5 Django Logout Request	3.5	High	PASS
TC 1.4 Django Registration Request	1.4	High	PASS
TC 2.5 Django Login Request	2.5	High	PASS
TC 2.3 Django Login Request	2.3	High	FAIL
TC 1.3 Django Registration Request	1.3	High	PASS
TC 3.3 Django Logout Request	3.3	High	PASS
TC 3.2 Django Logout Request	3.2	High	PASS
TC 2.2 Django Login Request	2.2	High	PASS
TC 3.1 Django Logout Request	3.1	High	PASS
TC 1.2 Django Registration Request	1.2	High	PASS
TC 1.1 Django Registration Request	1.1	High	PASS
TC 2.1 Django Login Request	2.1	High	PASS
TC 7.3 Connection Metrics	<FILL>	<FILL>	<FILL>
TC 7.2 Summary of Connections	<FILL>	<FILL>	<FILL>
TC 7.1 Summary of Tasks	<FILL>	<FILL>	<FILL>
TC 6.2 Upload Photo	<FILL>	<FILL>	<FILL>

Resources

- Learning Resources

- Shared Files
- An Example CRM - Circles (by ZooWho)
- Lectures
- Workshops
- Confluence Official Guides

Learning Resources

Github

https://docs.github.com/en

https://guides.github.com/introduction/git-handbook/

Github Workflow

https://guides.github.com/introduction/flow/

Basic steps:

1. Create **Branch**
2. **Commit** changes
3. Open a **Pull Request**
4. Code **Review**
5. **Merge** into main branch

Basic Commands

https://docs.github.com/en/get-started/using-git

https://education.github.com/git-cheat-sheet-education.pdf

- Create empty Git repo in specified directory
  - `git init <directory>`
- Clone repo located at <repo> onto local machine
  - `git clone <repo>`
- Stage all changes for the next commit
  - Stage changes in a directory `git add <directory>`
  - Stage changes of specific files `git add <files>`
- Commit the staged snapshot with a descriptive message
  - `git commit -m "<message>"`
- List which files are staged, unstaged, and untracked
  - `git status`
- Fetch the specified remote's copy of current branch
  - `git pull <remote>`
- Push the branch to <remote>, creates named branch in the remote repo if it doesn't exist
  - `git push <remote> <branch>`

Git Actions

https://docs.github.com/en/actions/learn-github-actions

https://docs.github.com/en/actions/quickstart

Heroku

https://devcenter.heroku.com/categories/reference

Heroku CLI

https://devcenter.heroku.com/categories/command-line

- Heroku command-line interface (CLI) is used to enter commands via terminal to interact with Heroku services
- Can be used to link with our app deployed on Heroku and configure remotely from local devices
- Heroku CLI installation options
  - Windows - https://devcenter.heroku.com/articles/heroku-cli#download-and-install
  - macOS - install using `brew` via terminal with command `brew tap heroku/brew && brew install heroku`
  - Linux - install via terminal command `sudo snap install --classic heroku`

Basic Heroku CLI commands

https://devcenter.heroku.com/articles/heroku-cli-commands

- See full list of commands `heroku help`
- Link CLI with Heroku account `heroku login`
- Run commands for service deployed on Heroku `heroku run <app commands>`
- Check recent log `heroku logs`
- Check Heroku version / verity Heroku correctly installed `heroku --version`

Pipenv - Virtual Environment

https://pipenv.pypa.io/en/latest/

https://github.com/pypa/pipenv

https://realpython.com/pipenv-guide/#pipenv-introduction

Purpose

- Create and maintain an isolated development environment for different projects
- Keep dependencies required by different projects separate
- Can install different dependencies with different versions for different projects
- Won't mess up with system's global environment
- Easy for other developers to replicate running environment and launch projects to avoid errors caused by different configurations

Install Pipenv

- Should install `Python` and `pip` first
- Windows - type in command via Command Prompt or Windows Powershell `pip install --user pipenv`
- Unix - type in command via terminal `pip install pipenv`

Basic commands

- Install a package / dependency `pipenv install <package name>`
    - Install a package with specified version `pipenv install <package name>==<version number>`
- Activate a terminal under virtual environment at current directory `pipenv shell`
- Install all dependencies for existing virtual environment `piping sync`
- Check all installed dependencies `pipenv graph`

Run our project

- Make sure the whole project has been pulled from repository
- Open a terminal at project directory
    - Check your current path using command pwd
    - Direct to given directory using command `cd <directory path>`
- Ensure both `Pipfile` and `Pipfile.lock` files exist
- Type in command `pipenv sync` to install required dependencies
- After successfully installed all dependencies, run `pipenv shell` to activate virtual environment
    - Or type in `code .` to open in VS Code and open a new terminal there (should automatically running under virtual environment)
- Then you can just run the project as you normally do

> Correctly configured `pipenv` environment will have `Pipfile` & `Pipfile.lock` files at root directory, which will be used to track dependencies required for this virtual environment.

Django

https://docs.djangoproject.com/en/3.2/

Install Django

https://docs.djangoproject.com/en/3.2/intro/install/

- Windows - type in command via Command Prompt or Windows Powershell `py -m pip install Django`
- Unix - install with `pip` via command `python -m pip install Django`

Basic tutorial

https://docs.djangoproject.com/en/3.2/intro/tutorial01/

- Check if Django has been correctly installed `python -m django --version`
- Ensure `django` is on list when run `pip list`
- Create a new project `django-admin startproject <project name>`
- Run Django server `python manage.py runserver`
  - By default, Django server will run on port `8000`
  - Can change the server's port by `python manage.py runserver <port number>`
- Correctly running Django server will look like:

```
Performing system checks...

System check identified no issues (0 silenced).
October 06, 2021 - 02:53:23
Django version 3.2.7, using settings 'backend.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Database configuration

https://docs.djangoproject.com/en/3.2/ref/settings/#databases

https://docs.djangoproject.com/en/3.2/ref/django-admin/

https://docs.djangoproject.com/en/3.2/topics/migrations/

https://docs.djangoproject.com/en/3.2/ref/django-admin/#makemigrations

- By default, Django will use built-in SQLite as database (locally)

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'mydatabase',
    }
}
```

- Using other database to store records need to modify database settings in `settings.py`
- In terms of how to set up remote MongoDB as default database for Django, please see MongoDB
- Once data models have been modified, need to run `python manage.py makemigrations` to update data model in database
  - May need to solve some conflicts if any
- And then run `python manage.py migrate` to sync with database

```
% python manage.py makemigrations
  No changes detected
% python manage.py migrate
  Operations to perform:
    Apply all migrations: admin, api, auth, authAPI, contenttypes,
knox, sessions
  Running migrations:
    No migrations to apply.
```

URL

https://docs.djangoproject.com/en/3.2/topics/http/urls/

- All url redirection needs to be configured in `urls.py` file
- Project itself and each module will have their own `urls.py`
    - Project `urls.py` is under project name folder
    - Module (own created app) will have `urls.py` file under module directory
- A typical `urls.py` will look like this:

```
from django.urls import path
from .views import RegisterAPI, LoginAPI, LogoutAPI

urlpatterns = [
    path('register/', RegisterAPI.as_view(), name='register'),
    path('login/', LoginAPI.as_view(), name='login'),
    path('logout/', LogoutAPI.as_view(), name='logout'),
]
```

- Function `path` will specify which View is responsible to handle the request of given url
- Project `urls.py` may use `include()` to redirect sub urls to be processed by `urls.py` under specific module
- Following url pattern will redirect urls under `auth/` to `urls.py` of `authAPI` module:

```
from django.urls import path, include

urlpatterns = [
    ...
    path('auth/', include('authAPI.urls')),
    ...
]
```

View

https://docs.djangoproject.com/en/3.2/topics/http/views/

https://docs.djangoproject.com/en/3.2/topics/class-based-views/intro/

https://docs.djangoproject.com/en/3.2/topics/class-based-views/

- A view function takes a Web request and returns a Web response
- View can be considered as the action handler for HTTP request
- The logic of processing request is written in the view
- View will return a response as output, which could be:
    - HTML contents of a Web page
    - A redirect
    - 404 error
    - An XML document
    - An image
    - …
- A simple view which returns a HTML page showing current time will look like this:

```
from django.http import HttpResponse
import datetime

def current_datetime(request):
    now = datetime.datetime.now()
```

```
    html = "<html><body>It is now %s.</body></html>" % now
    return HttpResponse(html)
```

- A view could be either:
  - Function-based view
  - Class-based view

```
# Class-based view

class LoginAPI(KnoxLoginView):
    permission_classes = (permissions.AllowAny,)

    def post(self, request, format=None):
        serializer = AuthTokenSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        user = serializer.validated_data['user']
        login(request, user)
        return super(LoginAPI, self).post(request, format=None)
```

MongoDB

https://docs.mongodb.com

Link with Django

https://www.mongodb.com/compatibility/mongodb-and-django

- Make sure `djongo` is already installed
  - May need to install `dnspython` as well
- Open `settings.py` file under project directory
- Change the settings for `DATABASES` with your MongoDB database configuration such like:

```
DATABASES = {
        'default': {
            'ENGINE': 'djongo',
            'NAME': 'your-db-name',
            'ENFORCE_SCHEMA': False,
            'CLIENT': {
                'host': 'mongodb+srv://<username>:<password>@<atlas
cluster>/<myFirstDatabase>?retryWrites=true&w=majority'
            }
        }
}
```

- Run commands `python manage.py makemigrations` and `python manage.py migrate` to sync with database
- If failed to connect to database, check access settings on MongoDB

Shared Files

File	Modified

Group 79 Week 9 Checklist.pdf	Oct 05, 2021 by Jackson Hu
Group 79 Week 6 Checklist.pdf	Oct 05, 2021 by Jackson Hu
Client Meeting Agenda #1.pdf	Oct 05, 2021 by Jackson Hu
Client Meeting Agenda #2.pdf	Oct 05, 2021 by Jackson Hu
Mood Board.pdf	Oct 05, 2021 by Jackson Hu
PCRM-annotated.pdf	Oct 06, 2021 by Jackson Hu
Client Meeting Agenda #3.pdf	Oct 07, 2021 by Jackson Hu
Connectd-Team79-Presentation.pdf	Oct 19, 2021 by Tymara Metcalf
Presentation Slides (draft).pdf	Oct 19, 2021 by Jackson Hu
Group 79 Week 12 Checklist.pdf	Oct 22, 2021 by Jackson Hu
Product Handover Slides.pdf	Nov 05, 2021 by Jackson Hu
Acceptance Criteria.pdf	Nov 05, 2021 by Jackson Hu
Deployment Documentation.pdf	Nov 05, 2021 by Jackson Hu
User Manual.pdf	Nov 05, 2021 by Jackson Hu
Authentication API Specification.pdf	Nov 05, 2021 by Jackson Hu

Drag and drop to upload or browse for files

Download All

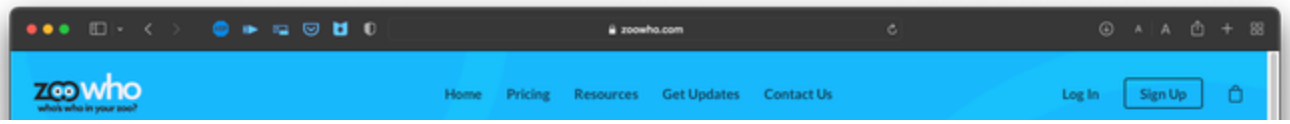
An Example CRM - Circles (by ZooWho)

Client used this existing CRM to demonstrate some features required for our product.

- Welcome Page
- Log in Page
- Dashboard
- Connections
- Contact Profile
- Circles - Group of Connections

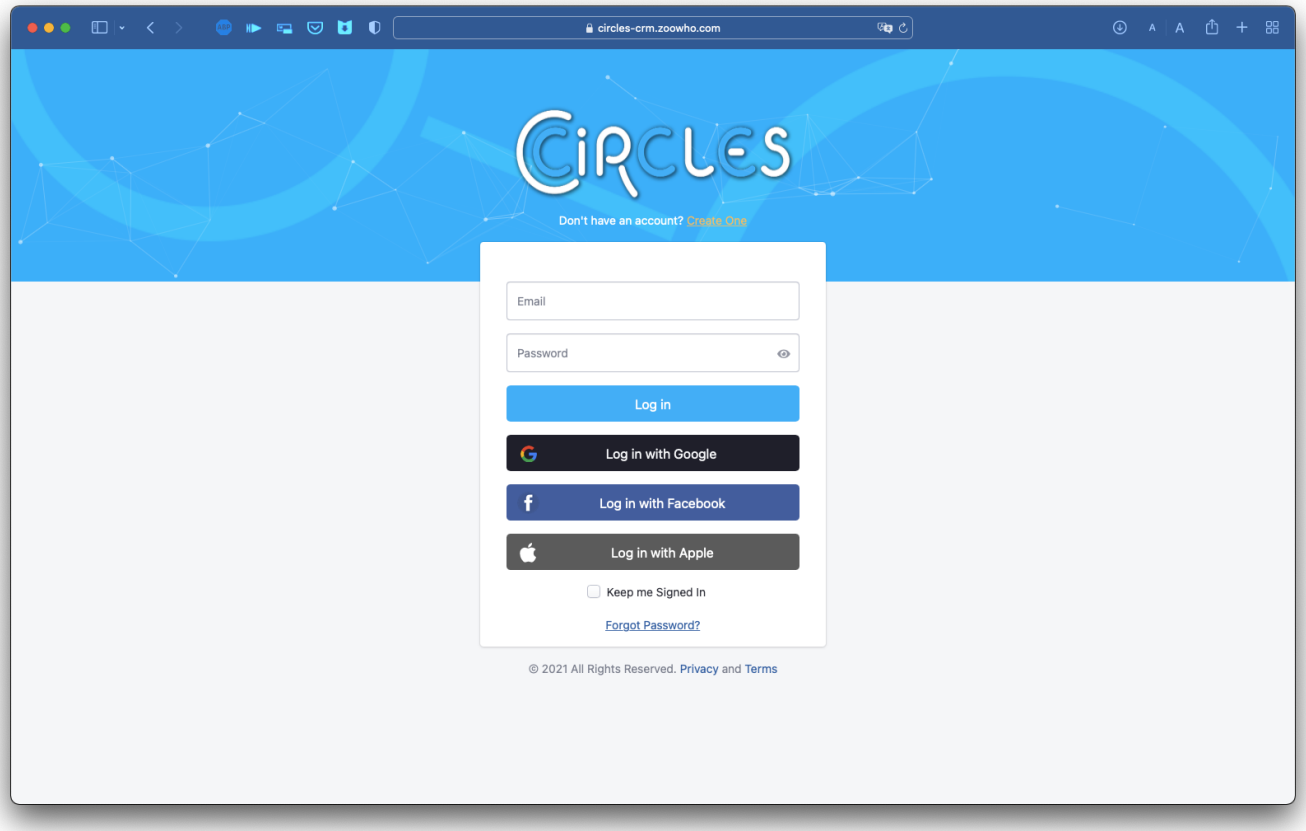
Welcome Page

- Company logo at the upper left corner
- Log in and Sign up buttons are at upper right corner
- A brief description of the product with screenshots
- Introduction of features can be seen by scrolling down the page



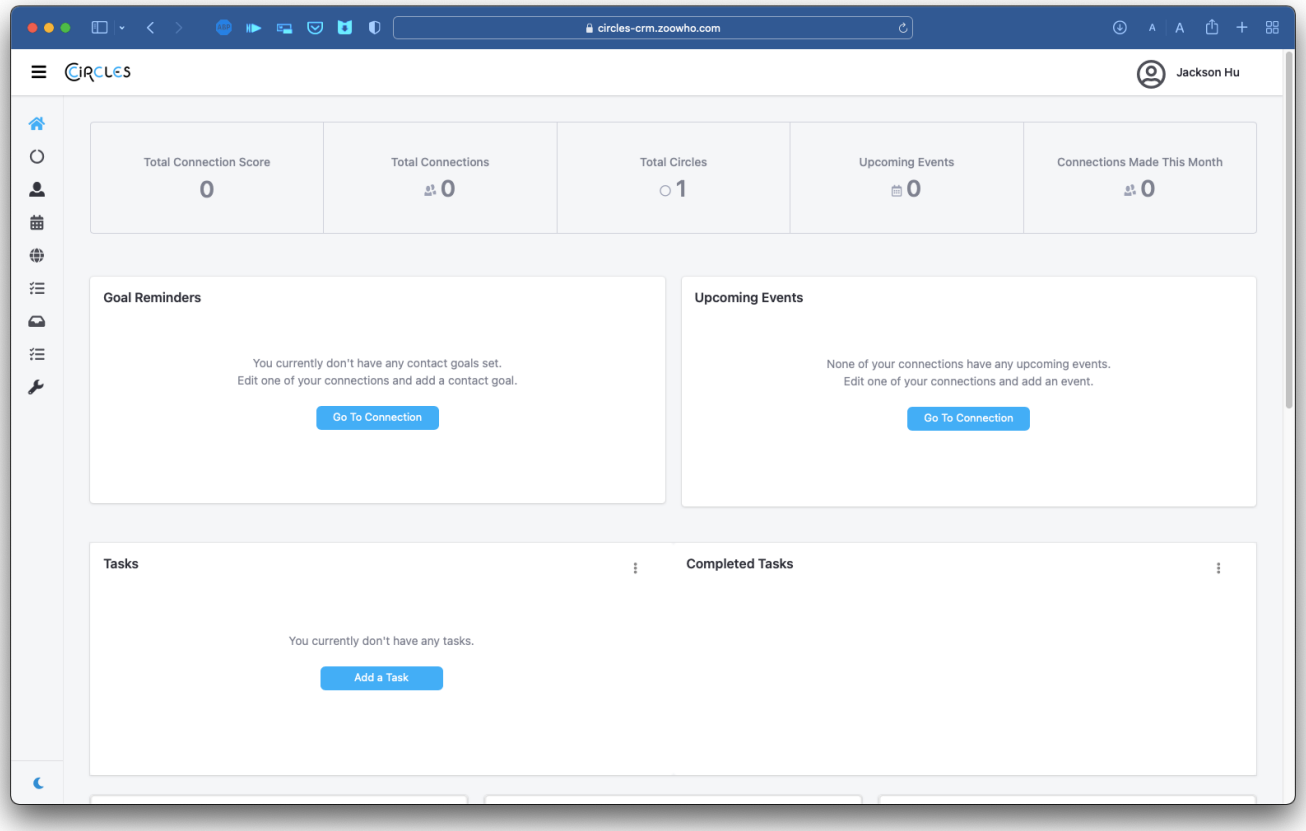
Log in Page

- Animated background with logo at the centre
- Input textfields for email and password
- Third party login options provided
- “Keep Signed In“ option
- Highlighted “Forget Password” link



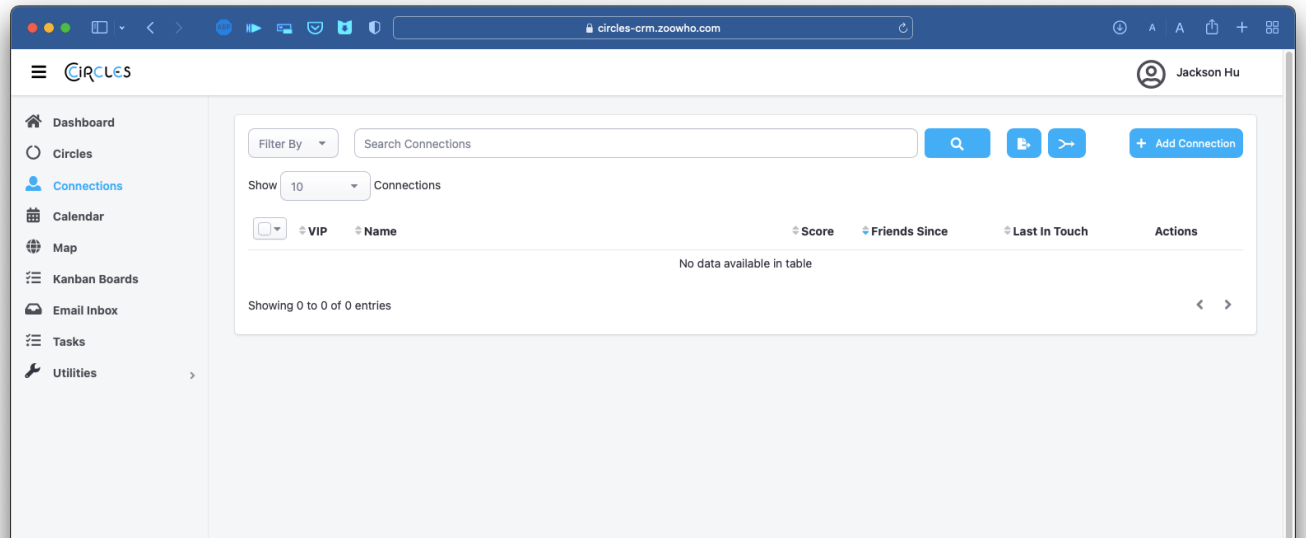
Dashboard

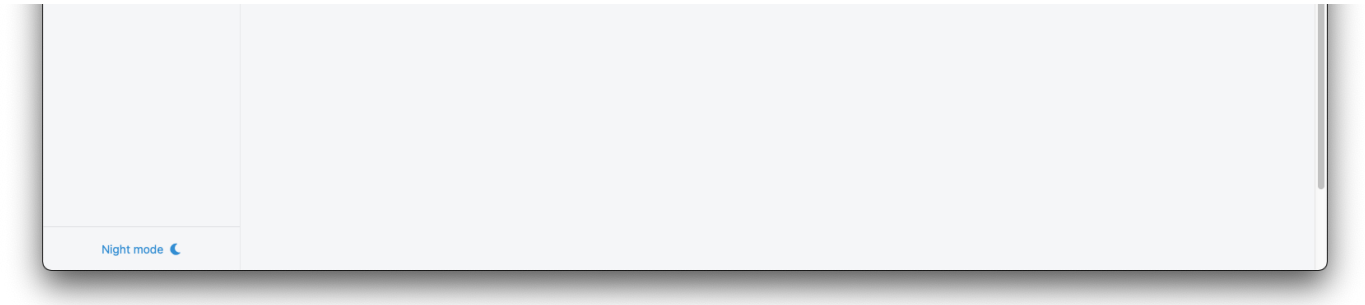
- Summary of total connections and circles
- Goal reminders
- Upcoming events
- Tasks
- Navigation bar on the left
- User profile picture and user name at upper right corner



Connections

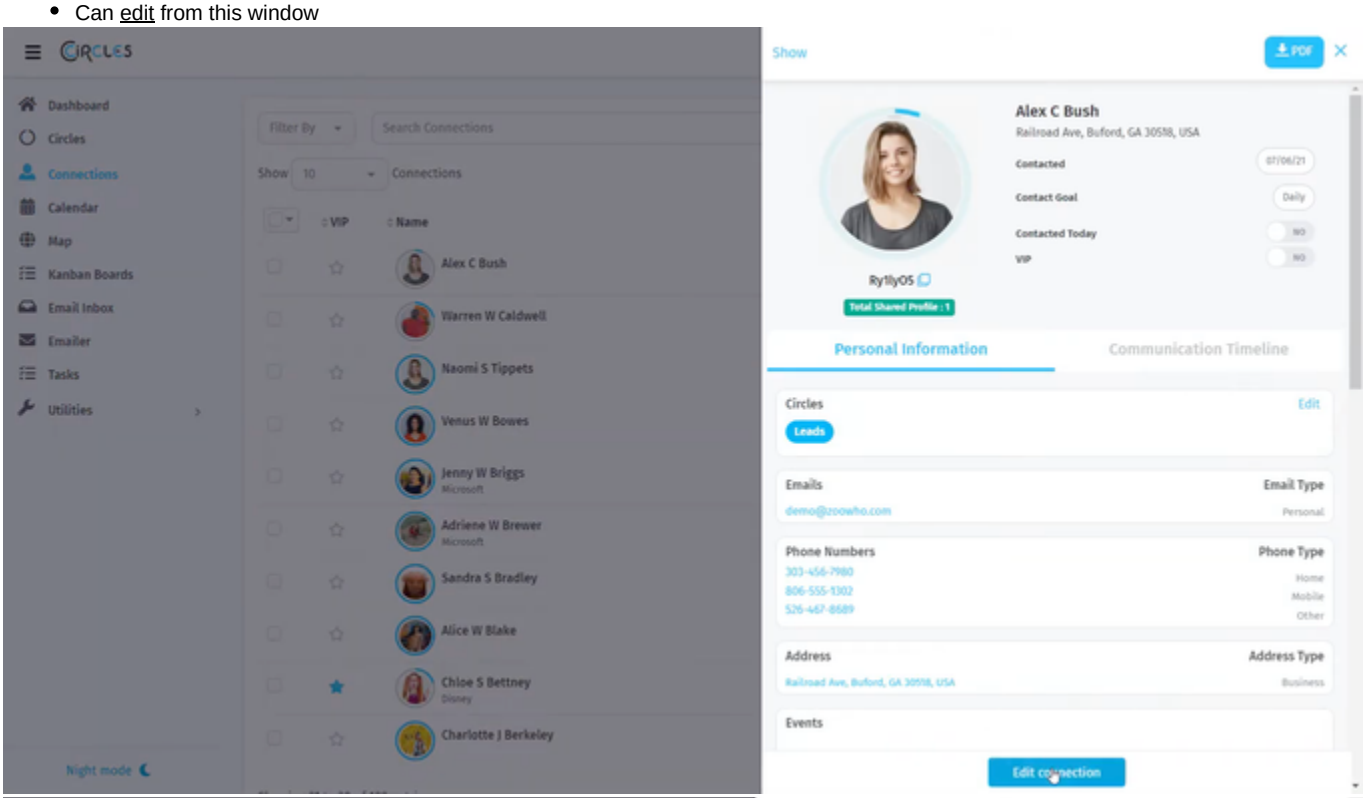
- Search bar at the top
- Can filter results displayed
- A button to add new connection on the right
- Key information of connections are showed in list

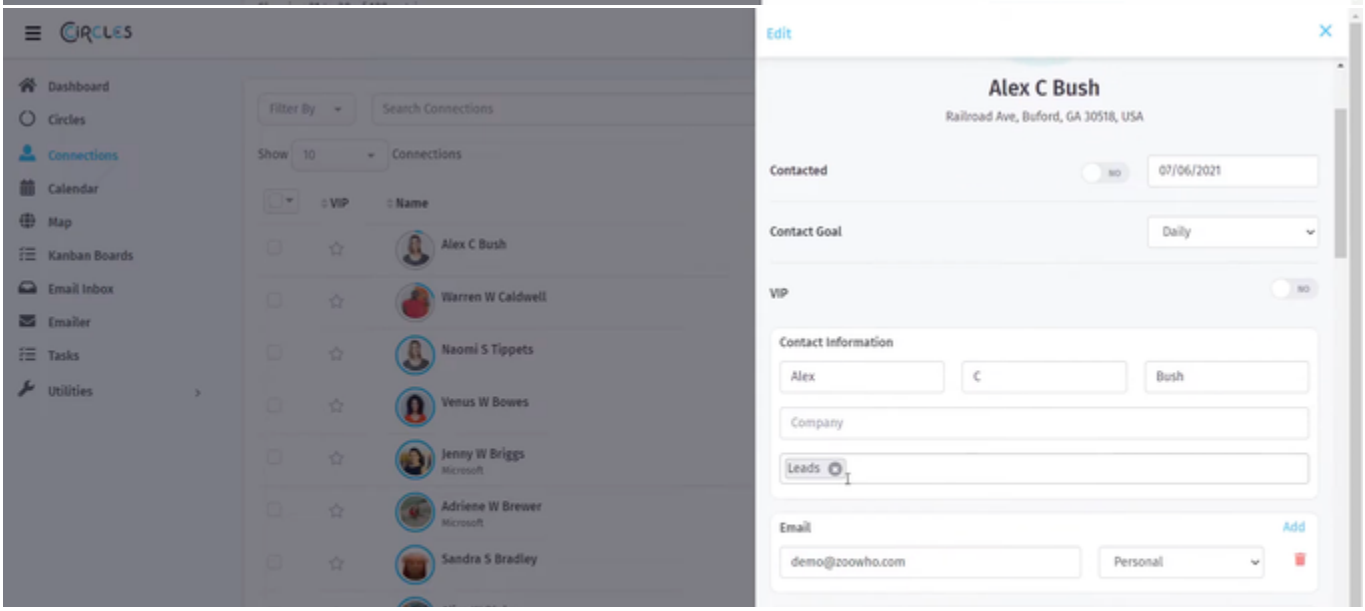




Contact Profile

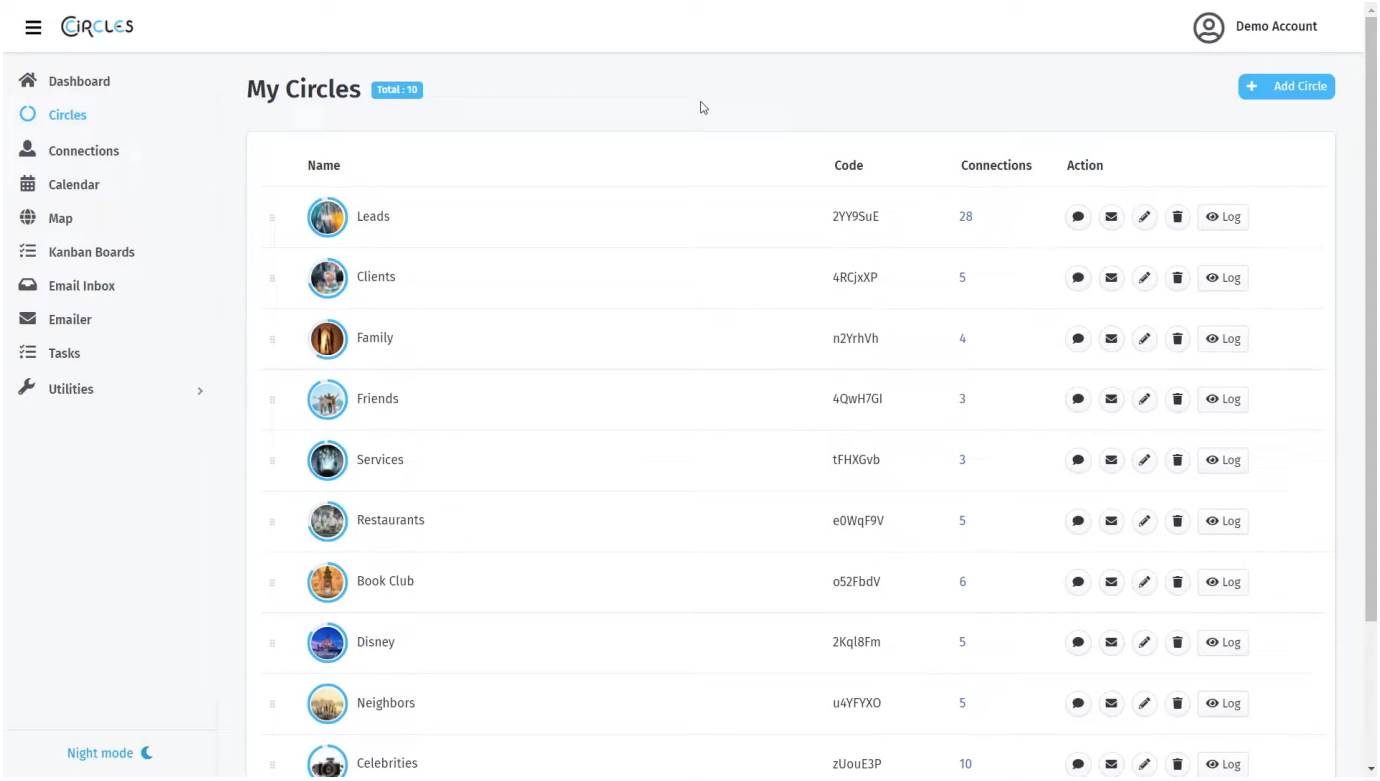
- A half size pop up window from right by clicking a connection
- Information is divided into different sections
- Can edit from this window

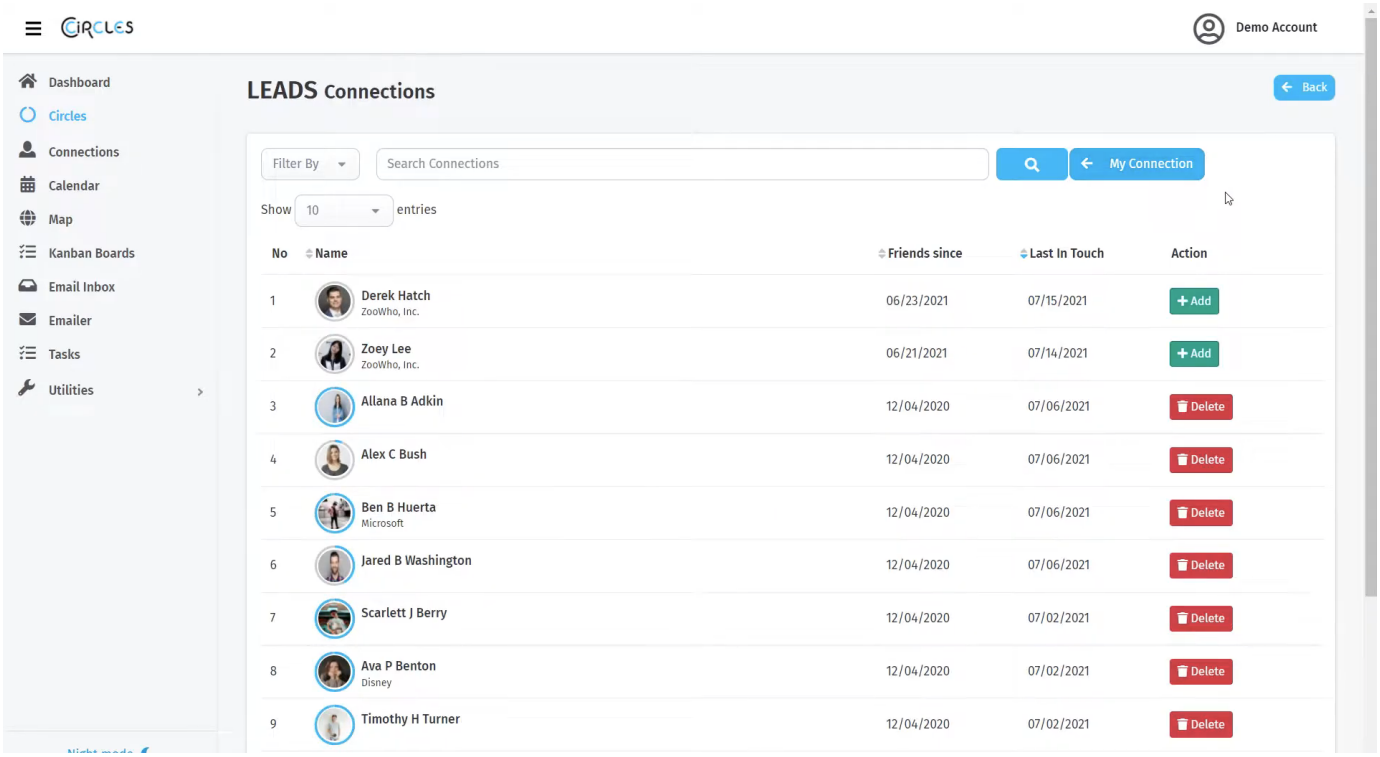




Circles - Group of Connections

- Similar list view for group of connections
- Can attach circle picture
- Quick actions on the right for each circle





Lectures

Week 1 Intro

Teaching Team
Leon Sterling leonss@unimelb.edu.au
Doc Wallace ian.wallace@unimelb.edu.au

Supervisors (Recent graduate or current student)
Hannah Ha, Rainer Selby, Luke Rosa, Andy Jiang, My Tien Tinh, Abhisha Nirmathalas, Chuanyuan Liu, Priyankar Bhattacharjee, Ayesha Ahmed, Mingye Li, Wei Liem Tan, Glenn Phillips, Thomas Bowes, Zhe Wang, Samuel Tumewa, Ishan Goyal, John McCleary

Mentor v tutor (not scrum master)
Support: Varsha Maram
Guest lecturers

2021 COMP30022 Project – Personal CRM
• Personal CRM
• What is a CRM?
• Customer relationship management (CRM) is a technology for managing all your company's relationships and interactions with customers and potential customers. The goal is simple: Improve business relationships. A CRM system helps companies stay connected to customers, streamline processes, and improve profitability.
• (from http://salesforce.com)
• Vexed Topic – Hard to do well
• CRMs proliferate at universities

Why a Personal CRM
• How to improve networking?
• Business cards
• Poor search capability
• Other uses of relationship management

Assessment

THE UNIVERSITY OF MELBOURNE

70% team
30% individual

Tools

- **Incredible flexibility**
- Discussed further in lectures and workshops over the next 2 weeks
- Repository – Github, Gitlab or Bitbucket
- Back end examples – Firebase, MySQL, MongoDB
- Front end examples – React.js, Node.js, vanilla Javascript, Python Flask
- Deployment examples – Heroku, AWS, Alibaba cloud
- Project management – Trello, JIRA
- Communication – Slack, Wechat

Communication Skills

- Essential
- Need to know what teammates are working on
- Think about meeting dynamics
- Be open to 'tone' of emails and conversations
- Practice reading and writing

Assessed through:

- Professional communication report
- Peer feedback
- Consideration of ethics

*Professional communication report (Reflections on guest lectures (500 words on what you learned/what you found interesting from the lecture –not a summary of the lecture))

Week 2 Development

Checklist

- ❑ Set up a channel or server for your team
  - ❑ Invite your Supervisor
- ❑ Create your documentation space
  - ❑ Familiarise yourself with it by filling in your team members details and contact information
- ❑ Set up a code repository
- ❑ Set up your task tracking tool
  - ❑ Estimate all tasks
  - ❑ Discuss progress of tasks at weekly meetings
- ❑ Decide on a technology stack
  - ❑ Front-end framework
  - ❑ Back-end framework
  - ❑ Testing framework
  - ❑ Deployment tool
  - ❑ API documentation

Useful links from the lecture:
https://learngitbranching.js.org/
https://lab.github.com/githubtraining/github-actions:-hello-world
https://learning.postman.com/docs/getting-started/introduction/ to Postman
https://www.postman.com/api-documentation-tool/

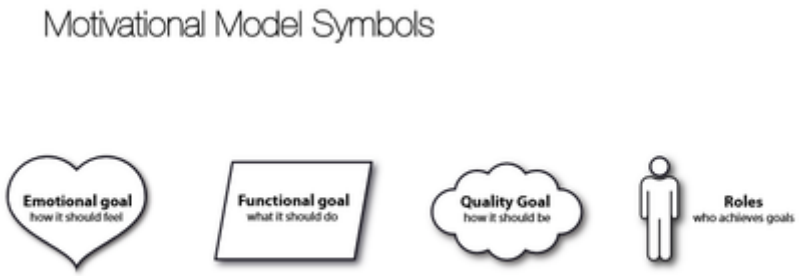
Week 3 Requirements

Lecture 3 - Requirements-1.pdf

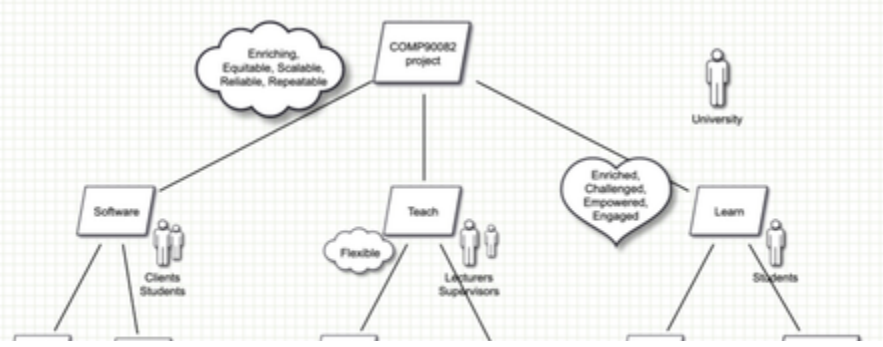
- Team name and logo
- Requirements
  - WHO/DO/BE/FEEL

Who	Do	Be	Feel
Students	Develop software	Enriching	Enriched
Clients	Maintain software	Equitable	Challenged
Lecturers	Conduct lectures	Flexible	Empowered
Supervisors	Review information (learn)	Scalable	Engaged
University	Communicate	Reliable	
	Develop skills	Repeatable	

  - Motivational modeling and its process



  - https://momo-staging.eresearch.unimelb.edu.au/ (Every team member needs to register separately, and Varsha will put you in with your team members.)



- Feedback to tweak model
- Ongoing maintenance

- User stories and Personas
  - Personas template: xtensio

- Expectations:
  - Document requirements, user stories.
  - DO/BE/FEEL recommended to develop an agreed model with your client, they may be familiar with motivational modeling.
  - Check consistency between user stories vs motivational model.
  - At the end of each sprint, check whether the model is still accurate and understood.

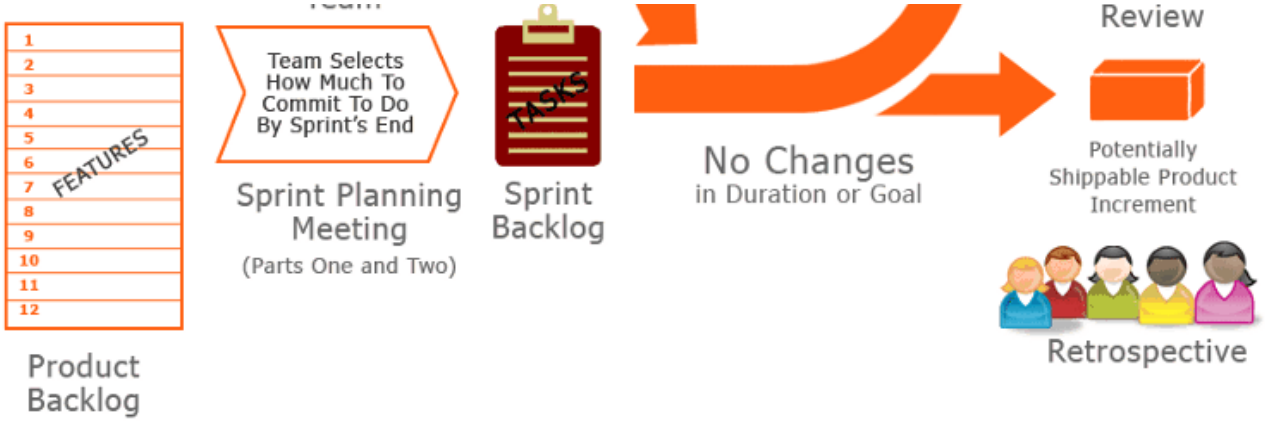
Week 4 Agile Process

Sometimes called the Iron Triangle

Ask do you have the right scope? Requirements, but also as you go along ask yourself/team what is the project suppose to be doing?

Agile Methods

- Scrum
- Extreme Programming
- Kanban



THE UNIVERSITY OF MELBOURNE

Agile in COMP30022

Appreciate that (most of) you have little background

- Stand ups (weekly, starting now)
- Sprints (3 x 3 weeks approximately)
- Retrospectives (end of each sprint)
- Scope considered dynamically each sprint
- Flexible
- Lightweight

Enough to fake it!

Note: Use Assessment checklist & Rubrics as a guide to make sure you're documenting and tracking well. No need to prepare separate documents apart from the tools you should be using, eg. Trello/Confluence.

Purpose of stand ups

- Industry practice
- Key feature of agile processes
- Give a regular snapshot on how the project is running

- An opportunity to identify any blockers
- Not usually the time to resolve conflicts or assign tasks
- Standing in a meeting different dynamic than sitting in a meeting
  - Up to team if you stand

THE UNIVERSITY OF MELBOURNE

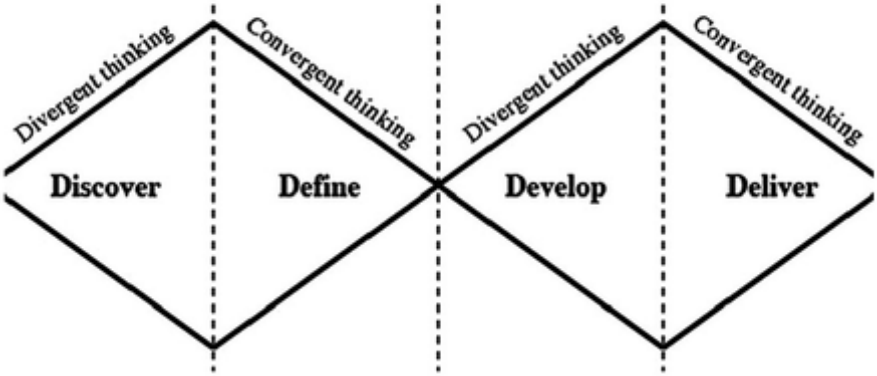
Participating in stand ups and meetings

- Go around the group (usually a circle) reporting on progress over the week
- Set order v where you are standing
- Speak clearly
- Speak slowly, especially initially

- Better with less technology, however may want to consider recording meetings

---

Double Diamond Model (from design)



4+1 Architectural View Models

Definition:

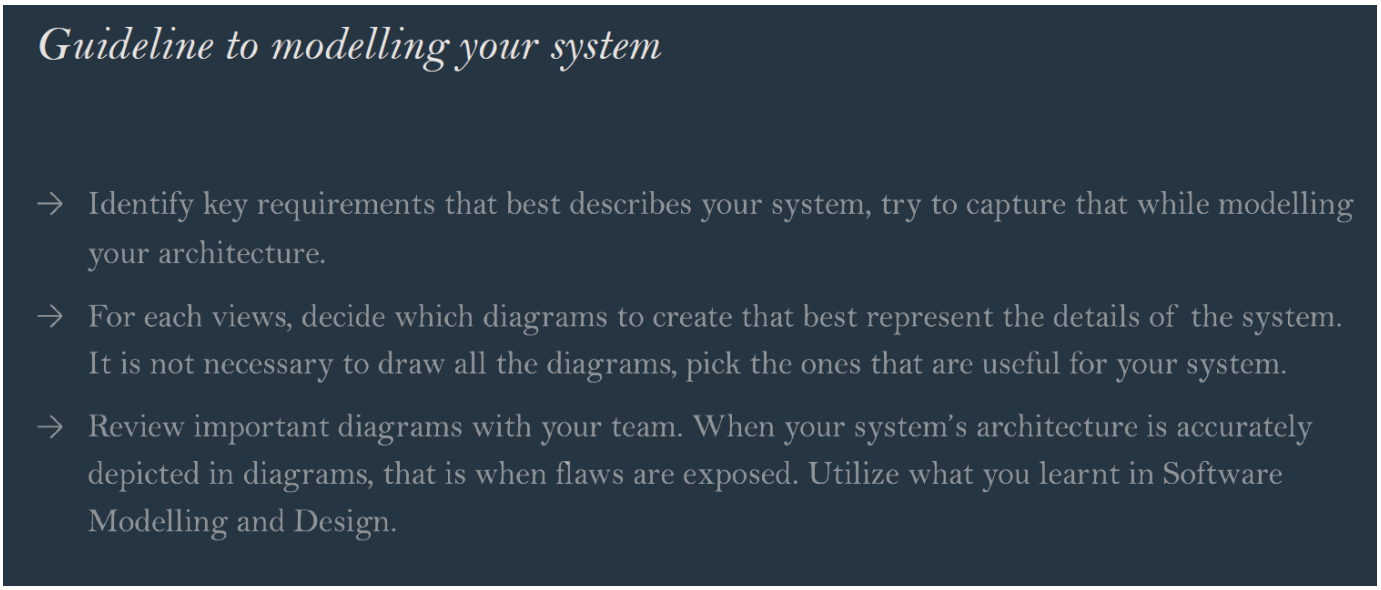
Engineering framework. A view describes the system from the viewpoint of different stakeholders, such as end-users, developers, and project managers.

Why:

- Organize design efforts and separation of concerns.
- Provides a way for developers and architects to prioritize modeling concerns
- Stakeholders can examine parts of system that are of interest to them
    - Developers: software components
    - System admin/engineers: deployment, hardware spec, network config

---

Guideline to modeling your system



Views:

1. **Logical:** functional requirements --- What the system should provide in terms of services to its users
2. **Process:** the dynamic aspects of the system, processes, and how they communicate.
   eg. Interaction between frontend and backend.
   The purpose is to capture the flow of information change. And the sequence and timing of these intercommunications.
   eg. if you call an API how does the flow go? Start off with the UI then you send a request to the backend.
3. **Development:** Illustrate the system from a programmer's perspective and software management.
   eg. Database or model diagram
4. **Physical:** System from engineer's POV. Describes mappings of software onto hardware.
   eg. Deployment, hardware specifications. What kind of server are you deploying to, what are the hardware specifications?
5. **Scenarios (Use-cases):** With the above 4 views put together they achieve certain use cases to describe sequences of interactions between objects and between processes. Forms the reason as to why all the other views exist
   You can use these use cases as a starting point for testing the system. Complete and consistent?

UML Diagrams:

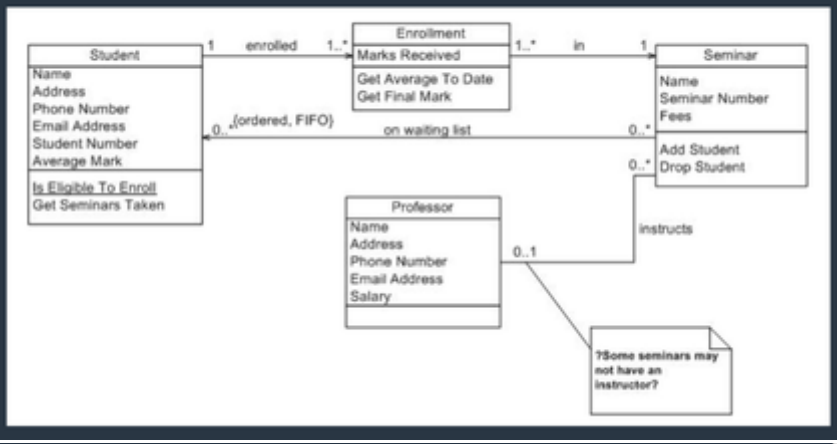
UML has many types of diagrams, each of them can be categorized into one of the the views.

- Behavior
- Interaction
- Structure
- More details: http://agilemodeling.com/essays/umlDiagrams.htm

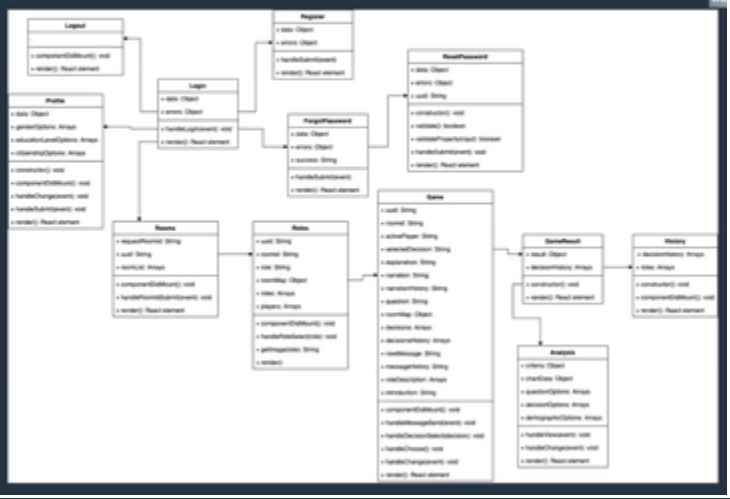
---

- → Viewer: End-users
- → Purpose: describes the functional requirements—what the system should provide in terms of services to its users.
- → Decomposed into a set of key abstractions taken (mostly) from the problem domain.
- → Shows components of the system and their interactions / relationships.
- → Diagrams: Class, Object, Package, State Machine, Composite Structure.

Conceptual Class Diagram / Domain Diagram



Design Class Diagram



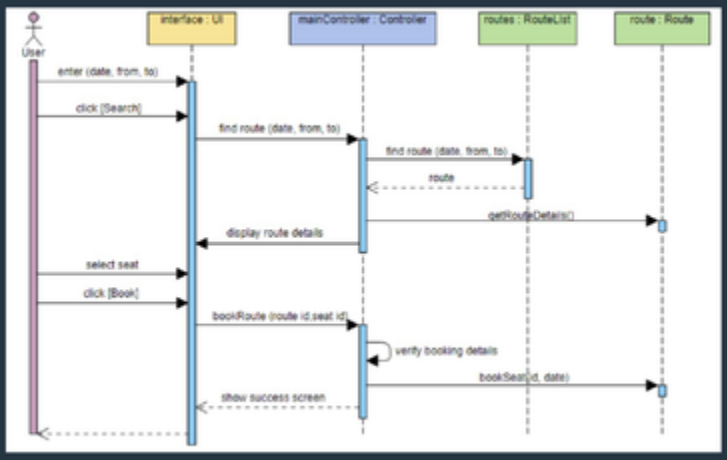
Process View

- → Viewer: Integrators
- → Purpose: Process decomposition; Shows the workflow of the system and how different processes interact with each other;

→ Diagrams: Sequence, Communication, Activity, Timing, Interaction Overview

Sequence Diagram

→ The process architecture can be represented at various levels of abstraction such as interactions between systems, subsystems and objects etc. based on the need.



Development View

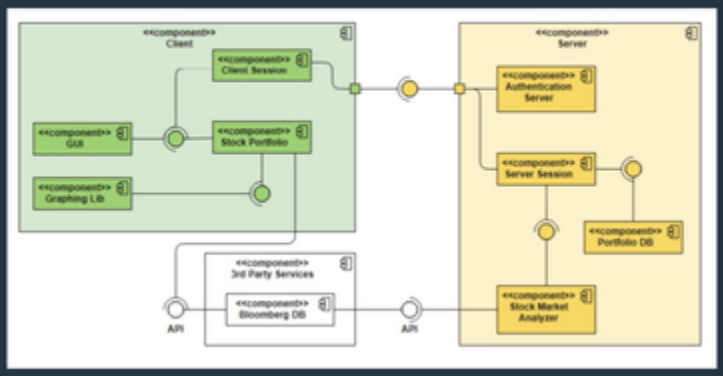
→ Viewer: Developers

→ Purpose: Illustrates a system from a programmer's perspective and is concerned with software management; building block view of the system, e.g. Packages Used, Execution Environments, Class Libraries and Sub systems utilized.

→ Diagrams: Component, Database

Component Diagram

→ Component diagram depicts how components are wired together to form larger components of software systems.



Database Model

→ A database model is a type of data model that determines the logical

structure of a database.

→ It fundamentally determines in which manner data can be stored, organized and manipulated.

Module Diagram

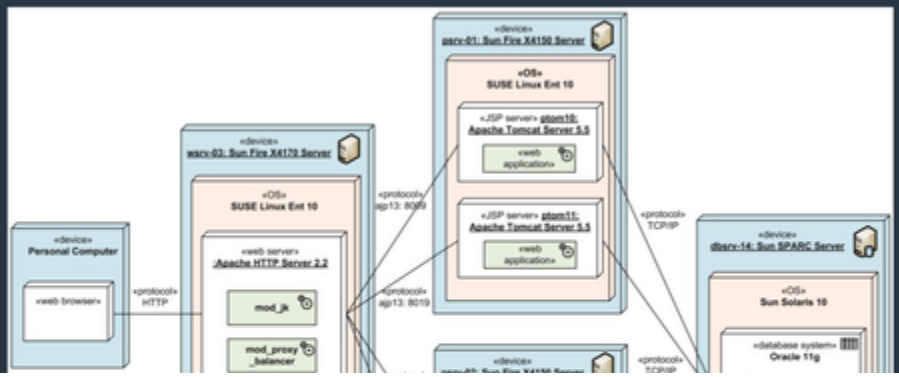
Deployment View

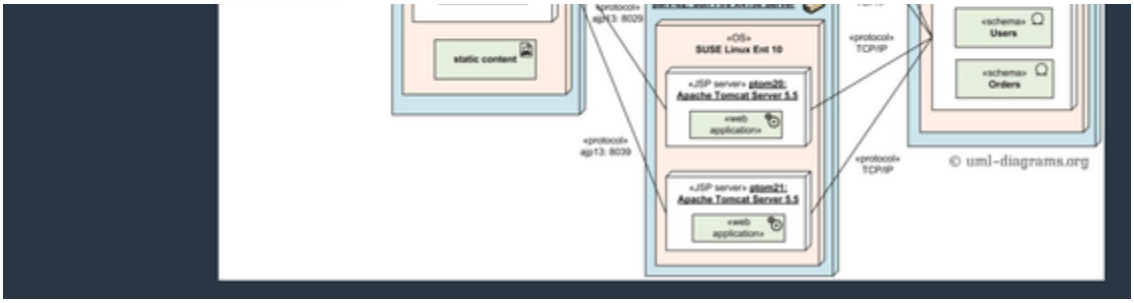
→ Viewer: System Engineers / System Admins

→ Purpose: Shows mapping of software onto hardware; Shows the system's execution environment; Concerned with deployment, network configuration etc.

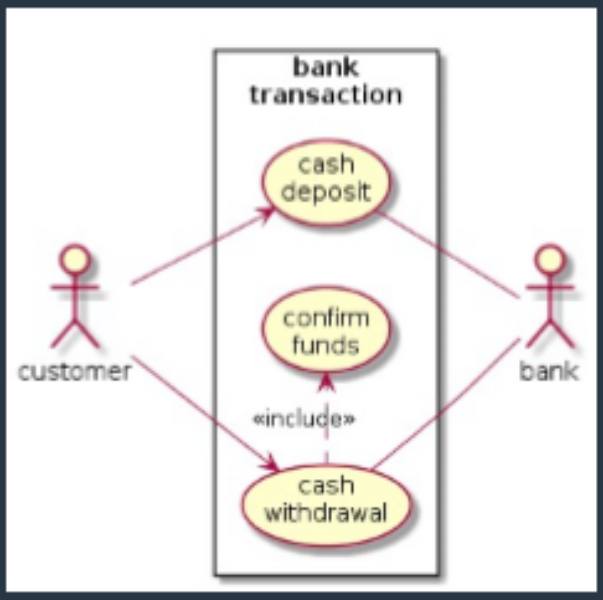
→ Diagram: Deployment

Deployment Diagram





Use Case Diagram



Coding

- Use coding standards – plenty of standards around
- Linters are useful
- Be aware of open source licensing agreements
  - https://opensource.org/licenses
- Coding reviews
  - See slides from Dr Patanamon Thongtanunam
- Consider pair programming
- Help your teammates

When we should do a code review?

- **For code, as soon as you** start implementing the system!

A general view of software development process

Why we should do a code review?

Knowledge Transfer, Team Awareness, & Share Code Ownership

- Novices can learn from experienced developers through code reviews
- Allow you to keep sharing and updating the knowledge within the team

How should we do a code review?:Best Practices

Code Review Checklists

- Consider all aspects of quality
  - Does the code follow documented architecture?
  - Does the code satisfy the acceptance criteria?
  - Is the code maintainable easily?
  - Does the code follow design principles, code convention?
  - Are the unit tests correct and sufficient?
  - Is the code comment informative and easy to understand?
  - etc.

How should we do a code review?:Best Practices

As a reviewer, you should

- Set aside dedicated, bounded time for reviews
- Review frequently, doing fewer changes at a time
- Provide feedback to authors as soon as possible
- Focus on core issues first; avoid nitpicking
- Give constructive, respectful feedback
- Choose communication channels carefully; talk face-to-face for contentious issues (Don't forget to document the conclusion!)
- Be prepared to iterate and review again

How should we do a code review?: Common practices

- A review typically takes 1 day
- Waiting time is about 1 hour to 1 day

- A review typically takes 4 hours
- Waiting time is about 1 hour for a small change and 5 hours for a large change

- Small code changes (11-44 lines were changed)
- Typically two reviewers per a review

- Small code changes (typically 24 lines were changed)
- The number of reviewers depends on the change size (typically one)

Remarks

- Finding defects is not the sole goal for reviewing
- Improving not assessing
- Not only code should be reviewed
  - All the deliverables (e.g., unit test, documentation) need a review as well
- Communicating, communicating, and communicating!
- Open for criticism
- Be ware of some subconscious biases
  - Experienced developers are not always right
  - Don't follow the crowd decision. Do raise a concern if you have ones.

Week 6 Software Testing

Seven Principles

1. Exhaustive testing is not possible
2. Defect clustering - a small number of modules contain most of the defects detected
3. Pesticide paradox
4. Testing shows a presence of defects
5. Absence of error is a fallacy
6. Early testing
7. Testing is context-dependent

Testing Types

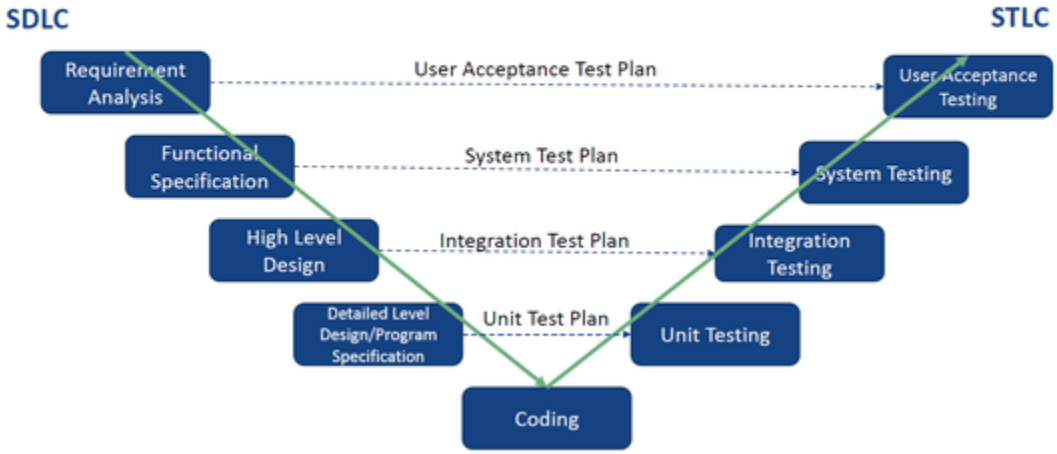
Functional Testing

- Unit
- Integration
- System
- Sanity
- Regression
- Acceptance

Non-functional Testing

- Performance
- Load
- Stress
- Volume
- Security
- Compatibility
- Install
- Recovery
- Reliability
- Usability
- Compliance
- Localization

V Model



Some useful terminology

- **Test suite**: a set of test cases.
- **Test case**: a set of inputs, execution conditions, and a pass/fail criterion.
- **Test** or **test execution**: the activity of executing test cases and evaluating their results.
- **Adequacy criterion**: a predicate that is true (satisfied) or false (not satisfied) of a ⟨program, test suite⟩ pair.

Where do test obligations/tasks come from?

- Functional (black box, specification-based): from software specifications
- Structural (white or glass box): from code
- Model-based: from model of system
- Fault-based: from hypothesized faults (common bugs)

Functional Testing

Acceptance Criteria

User Story ID	User Story	Given	When	Then
1	As an athlete user, I can see the latest update for my sessions in my dashboard's calendar so that I can track my current program's completion in a more informative way.	I have been doing exercises everyday	I check sessions in my dashboard's calendar	• I can see completion of my current and previous programs
2	As an athlete user, I can receive some badges for my achievements after I overcome some challenges during my sessions so that I can be motivated to stay on track for my "Building".	Many challenges related to my personal program listed in my session	I complete challenges in my session's program	• I receive badges for corresponding challenges

		I have received badge when I complete the challenge	I click on the social media button next to the badge section	• It shares my badges on the social platform
3	As an athlete user, I can see route map tracking in my session description & feedback so that I can get an overview of session feedback visually.	I am doing some outdoor exercises	I click session in the dashboard	• It show route tracking map • feedback

Test cases (templates)

US 01: As an admin,
I want to add new users to the system
so that they can login to it

TC 02: Add User (unsuccessful)

Test Type: Functional	**Execution Type:** Manual
Objective: Verify application behavior when user could not be added.	
Setup: *<List the pre-conditions to carry through this test case >* •	
Pre-Conditions: *<List the conditions after the test execution >* •	
Notes: [1] Start filling the form for a new user and cancel operation. User list is not modified. [2] Try to add user with an existent username. [3] Try to add user when username, name and/or email fields are empty. [4] Try to add user with invalid content type: [4.1] Using more than maximum allowed number of chars (username: 20, name: 55, email: 55). [4.2] Use special chars, accents, empty, spaces, only spaces to filled username, name and email fields. [4.3] Try to add user when email format is not valid. ** User name field allow the following special characters: _ . (underscore and dot).* ** Name field allow the following special characters: _ - (underscore and negative).* ** Email field allow the following special characters: - + _ . ' (negative, positive, underscore, dot and apostrophe).* ** The email field is validated*	
Time constraint: Minimum: 20 min Maximum: 25 min	

TestCase #	Test Case	Test Steps	Test Data	Expected Result
1	Verify Login	1) Go to Login Page 2) Enter UserID 3) Enter Password 4) Click Login	id= Guru99 pass= 1234	Login Successful

When correct password and id entered, it should login successfully

Confluence

Linking user stories and test cases in the sprint

Epic 1: Data Visualization

[adopt codes for US/AC/TC :: link them]

REQUIREMENT ID	REQUIREMENT DESCRIPTIONS	TC 001	TC 002	TC 003
SR-1.1	User should be able to do this.	x		
SR-1.2	User should be able to do that.	x		
SR-1.3	On clicking this, the following message should appear.		x	
SR-1.4			x	

SR-1.5		x		x
SR-1.6				x
SR-1.7			x	

Requirements vs. Tests

Systematic vs Random Testing

Random (uniform):
- Pick possible inputs uniformly
- Avoids designer bias
- But treats all inputs as equally valuable

Systematic (non-uniform):
- Try to select inputs that are especially valuable
- Usually by choosing representatives of classes of inputs that either fail *together* or *not at all*

Functional testing is systematic testing

Structural Testing

Structural testing in practice

Steps:
- Create functional test suite first
- Measure structural coverage to see what is missing
- Interpret unexecuted elements

Attractive because automated
- coverage are convenient progress indicators
- sometimes used as a criterion of completion
  - use with caution: does not ensure *effective* test suites

Functional vs Structural: Classes of faults

Functional testing is best for *missing logic* faults
- A common problem: Some program logic was simply forgotten

Structural (code-based) testing will never focus on code that isn't there!

UX - User-based testing

- We cannot design a user experience, only design **for** a user experience.
- UX is about "how people **feel** about a product and their **pleasure and satisfaction** when using it, looking at it, holding it, and opening or closing it. It includes their **overall impression** of how good it is to use, right down to the **sensual effect** small details have". (Preece et al., 2015, p. 12)

Usability

Products should be:

- Effective to use
- Efficient to use
- Safe to use
- Have good utility
- Easy to learn
- Easy to remember how to use

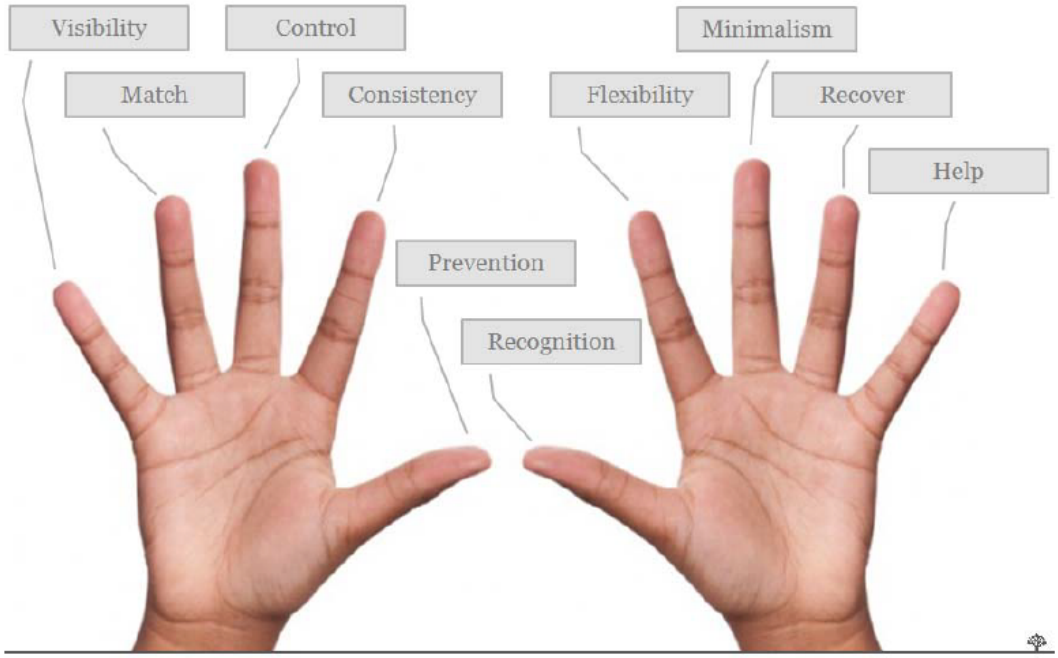
Preece et al. (2015)

The extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

ISO 25066:2016

Heuristic Evaluation

Experts compare the system to a set of heuristics or guidelines.



H1: Visibility of System Status

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

H2: Match between System and the Real World

- The system should speak the users' language, with words, phrases and concepts

- The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms.
- Follow real-world conventions, making information appear in a natural and logical order.

H3: User Control and Freedom

- Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue.
- Support undo and redo.

H4: Consistency and Standards

- Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

H5: Error Prevention

- Even better than good error messages is a careful design which prevents a problem from occurring in the first place.

H6: Recognition rather than Recall

- Make objects, actions, and options visible.
- The user should not have to remember information from one part of the dialogue to another.
- Instructions for use of the system should be visible or easily retrievable whenever appropriate.

H7: Flexibility and Efficiency of Use

- Accelerators - unseen by the novice user - may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users.
- Allow users to tailor frequent actions.

H8: Aesthetic and Minimalist Design

- Dialogues should not contain information which is irrelevant or rarely needed.

- Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

H9: Help Users Recognise, Diagnose, and Recover from Errors

Error messages should

- be expressed in plain language (no codes)
- precisely indicate the problem
- constructively suggest a solution

H10: Help and Documentation

- Troubleshooting: Make sure your system indicates errors and messages in plain language.

Week 3

⚠

- **Regardless of what tools you use...**

  Everything will need to be visible to myself, and later the subject coordinator (Leon) for assessment.

  The easiest way to do this is to create a folder in your repository called “documentation” and upload your confluence pages / trello screenshots / google files etc… into it at the end of each sprint.

  E.g.

  documentation > sprint1 > “my-files”.pdf

Checklist

- ❑ Setup Kanban Board
- ❑ Setup Documentation (Meetings, Team Resources, Sprints)
- ❑ Documentation folder in github/repository
- ❑ Add me to your github/repository (make sure it's private until the end of the semester)
- ❑ Assign roles
- ❑ After the lecture, prep for client meeting
- ❑ After the client meeting Setup Product Backlog (Or even just a sprint 1 backlog)
- ❑ Start Sprint 1

Week 4

Things to do

- ❑ Grant access of your main team spaces to me:
  - ❑ Repository (Github)
  - ❑ Documentation (Confluence / Google Drive)
  - ❑ glenn.phillips@unimelb.edu.au
- ❑ Consolidate your Requirements from your client meeting
- ❑ Set up your Sprint 1 Backlog
- ❑ Make a start on some design artefacts if you haven't already
- ❑ Get started on development if you want (and thus not have to work during your mid-sem break)

⚠️

- **Example Confluence/Google Drive Structure**

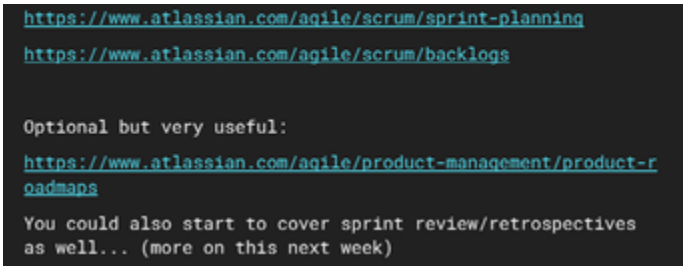
```
homepage/                                   (contact info, roles, important links)
|-- requirements/                           (initial client requirements, keep a log of any updates)
      |-- do-be-feel                        (and goal model)
      |-- personas                          (main stakeholders)
      |-- product-backlog
|-- design/                                 (can be merged with requirements, keep a log of updates) (bound to change)
      |-- moodboard
      |-- storyboard                        (a great artefact if you have time) (invision works well)
      |-- surprise-me                       (low-fidelity prototype? style-guide? test-plan?)
|-- sprints/
      |-- sprint-1/                         (x3)
            |-- sprint-1-backlog            (done at the start of each sprint) (trello/jira originate from here)
            |-- weekly-progress-updates/
                  |-- <trello/jira>-<date>  (weekly screenshots are fine, no need to invite me to these platforms)
            |-- sprint-1-retrospective      (done at the end of each sprint) (internal with team) (process based)
            |-- sprint-1-review             (done at the end of each sprint) (external with client) (product based)
|-- meetings/
      |-- meeting-template                  (date, time, attendees, agenda, minutes, actionable items)
|-- resources/
      |-- learning                          (add whatever resources you want here)
```

Don't feel limited by any of this - It is just a starting point.

Week 5

Some great articles by Atlassian

If you haven't already found these here they are:



https://www.atlassian.com/agile/scrum/sprint-planning

https://www.atlassian.com/agile/scrum/backlogs

https://www.atlassian.com/agile/product-management/product-roadmaps

Checklist for today

- ❑ Ensure you have a "sprint backlog" document in your confluence (under sprint 1).
- ❑ An overall "product roadmap" or even "product backlog" would be good additions if you want to provide a broader overview of the entire project.
- ❑ Deploy your frontend and backend by the end of sprint 1 (end of week 6)
- ❑ Sprint Review and Sprint Retrospective will be covered next week

Week 6

(Team) Sprint Retrospective

This is intended to be an ***internal*** reflection regarding the ***processes*** that the team has followed over the past sprint.

Remember a ***process*** is ***how*** you did something; not what you did.

E.g. The team created meeting agendas before each meeting

The team split into sub groups for pair programming

Be sure to reflect on both the ***positives*** and the ***negatives***, as well as mention what you will ***improve on*** in sprint 2.

(~ 3 of each)

(Client) Sprint Review

Very similar to the Sprint Retrospective except this time it is an ***external*** reflection with the ***client*** regarding the ***product*** that you have created thus far.

The ***product*** is the ***what***; not the how.

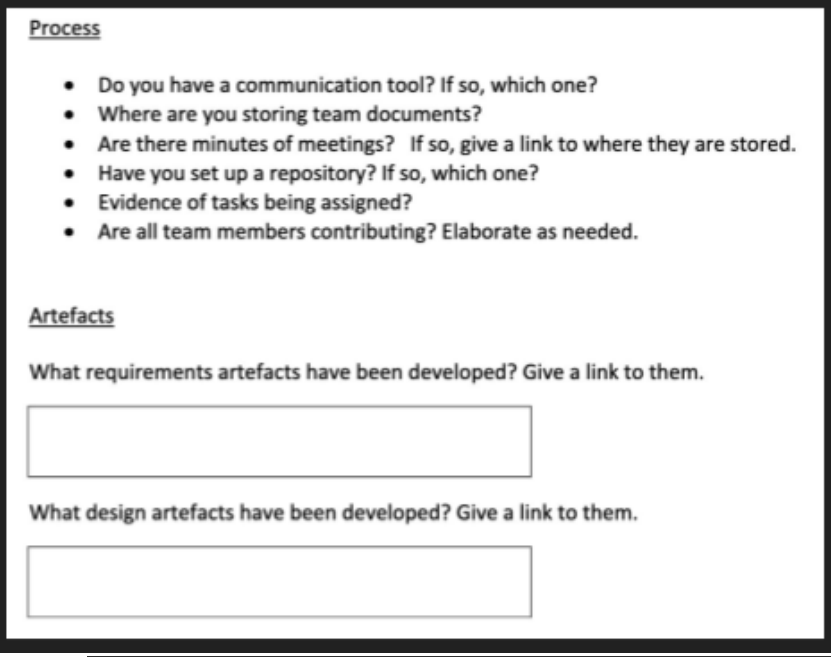
Focus on presenting the ***features*** / ***artefacts*** you have created in sprint 1 via some kind of ***presentation***.

Ensure you have someone take notes during this meeting.

Yet again be sure to reflect on the ***good***, the ***bad***, and ***areas of improvement*** for sprint 2.

(yet again ~3 of each)

Week 6 Assessment



Process

- Do you have a communication tool? If so, which one?
- Where are you storing team documents?
- Are there minutes of meetings? If so, give a link to where they are stored.
- Have you set up a repository? If so, which one?
- Evidence of tasks being assigned?
- Are all team members contributing? Elaborate as needed.

Artefacts

What requirements artefacts have been developed? Give a link to them.

What design artefacts have been developed? Give a link to them.

I asked Leon and he said links to confluence are fine.

I already have access to your spaces and if Leon requires access he will ask.

Ensure you provide a description of any links. A link alone is unsatisfactory.

Checklist

- ❑ Schedule a Sprint Retrospective Meeting with your team
- ❑ Schedule a Sprint Review Meeting with your client
- ❑ Submit the Week 6 Deliverable by this Friday
  - ❑ (there is a one week grace period if you have any difficulties but please get in touch with me if this is the case)
- ❑ Schedule a Sprint Planning Meeting with your team for Sprint 2

Confluence Official Guides

CRM - Getting started

Here are some tips to get you started. You can edit this page to see how it works!

On this page:

1. Create a page

- Click "Create" and select "Blank Page" to create your first page.
- New pages are created as children of the page you are currently viewing.

2. Add to your page

- Click "Edit" to enter the Confluence editor and use the page layouts feature to structure your content using sections and columns.
- Use headings to format your text and drag and drop images into your page to provide visual interest.
- Click "Insert" and select "Other Macros" to add macros for navigation, special formatting and other media.

3. Organise your pages

Here are some tips for organising your content.

- Change the page order

  The sidebar on the left displays your pages in a hierarchy. If you have Space Administrator permissions you can click "Space Tools" > "Reorder Pages" to move pages around.

- Add labels

  Labels help keep pages organised and make it easier for you to find the information you need. Click "Labels" at the bottom of a page to add or edit. The "Related pages" section on this page uses labels too!

- Make templates

  Standardise and speed up the page creation process with templates. You can create and format a template with page layouts, standard headings and instructional text for hints and guidelines. Check out our sample page on "Making a template"

Related pages

- CRM - Getting started
- CRM - Getting started - Making a template
- How-to article
- Master project documentation

CRM - Getting started - Making a template

Give your authors a helping hand by using templates in your documentation space. You will need Space Administrator permissions to create templates.

To create a template:

1. Go to "Space Tools" in the sidebar, select "Content Tools" and create a new template.
2. Click "Page Layout" and add sections and columns to your page.
3. Add headings and sub-headings as needed.
4. Choose "Instructional Text" from the "Template" menu and add text that is only visible in the editor.
5. Save your template.

Your template will become available in the Create dialog for this space.

Related pages

- CRM - Getting started

Useful hint
Confluence Administrators can also make templates that are available across your whole Confluence instance.

- CRM - Getting started - Making a template
- How-to article
- Master project documentation

How-to article

Instructions

Related articles

Master project documentation

This template is brought to you by Mural, a visual collaboration app.

❓ Unmet needs

🏹 Objectives

👥 User personas

💪 Jobs we want to cover

- When I
- I want to
- So I can

📜 Some history

👀 Constraints

🗺️ Explorations + Decisions

🗓️ Releases

Release Name	Value it adds	Scope	Status	Completed date
			TO DO / IN PROGRESS / BLOCKED / WAITING FOR FEEDBACK / DONE	

👣 Next steps

- [ ] *SET A STATUS*

📈 Impact

🔗 Other documents