DAMSELS IN DEVELOPMENT

inkā

TEAM 13

Project Report

Te Tuhi Disability Training Cafe Platform for Trainees

Saacha Moyo	Team Leader/Front-end & UI/UX Developer
Eisen Belleza	Backend Developer
Istefat Sawda	Full-stack Developer
Zainab Shehzad	Full-stack Developer
Natania Thomas	Project Manager/Security Engineer

Version Updated: 2 June 2024

Executive Summary

Employment inequality remains a significant challenge for people with disabilities. Statistics from Statistics NZ reveal that individuals with disabilities face an unemployment rate more than double that of those without disabilities. The disparity is even more pronounced for people with intellectual disabilities, who are over four times as likely to be unemployed compared to their non-disabled counterparts. This alarming gap underscores an urgent need for effective solutions to promote inclusive employment opportunities. The UN Convention on the Rights of People with Disabilities affirms that employment and accessible workplaces are fundamental rights, highlighting the importance of creating supportive job environments. Inspired by this mission, we, Damsels in Development, have focused on enhancing the invaluable work done at Te Tuhi Cafe, a project committed to fostering inclusion.

To support Te Tuhi Cafe's objectives, we have developed Inka, a cutting-edge mobile application designed to improve the employment prospects of individuals with disabilities. Inka serves as a comprehensive repository, a one-stop shop that stores trainee progress and staff evaluations, streamlining the training and development process. The app features three distinct interfaces, each tailored to different user roles: trainees, support staff, and administrative staff. The trainee interface provides users with access to training modules, enabling them to learn and progress at their own pace. The support staff interface allows for the monitoring and evaluation of trainee progress, ensuring that each individual receives the guidance and support they need. The administrative staff interface offers data management capabilities, facilitating efficient organisation and manipulation of trainee information and training modules.

However, there is still room for improvement. To address this, future developments should focus on incorporating user feedback to enhance the user experience of the application, ensuring it meets the evolving needs of trainees and staff alike. Implementing features such as in-app notifications and versatile login options will further streamline operations. Integrating a fully-matched evaluation system identical to the paper-based system currently used at Te Tuhi Cafe, along with the capability to export trainee data, will ensure consistency and ease of use. These only outline a few of the several ways in which the impact of Inkā can be maximised.

While Inkā aims to provide support, it is essential to recognise that Te Tuhi Cafe, its dedicated staff and its partners, Rescare Homes Trust and University of Auckland’s School of Psychology, are the driving forces behind real change. Inkā enhances their efforts by providing a streamlined, efficient and personalised approach to training and development, potentially contributing to the creation of inclusive employment opportunities in New Zealand. As a final-year project, we can only hope that Inkā proves helpful to Te Tuhi Cafe’s mission and demonstrates the impact of collaborative efforts in

fostering a more inclusive community.

Table of Contents

Introduction

Before the commencement of this project, Te Tuhi Cafe had its training resources and data dispersed across drop-boxes, Excel sheets, and, in some cases, emails, requiring manual sharing among the cafe and its partners. Therefore, our aim is to streamline and enhance the valuable work done at Te Tuhi Cafe.

To achieve this aim, our primary objective is to deliver a fully functional and intuitive mobile application that consolidates Te Tuhi Cafe's initiatives. Our application, Inka, serves as a one-stop shop for the training programs developed by the School of Psychology at The University of Auckland, as well as other partners (Rescare Homes Trust). It functions as a platform for monitoring cafe trainee progress and fostering personalized learning experiences for trainees within the cafe environment.

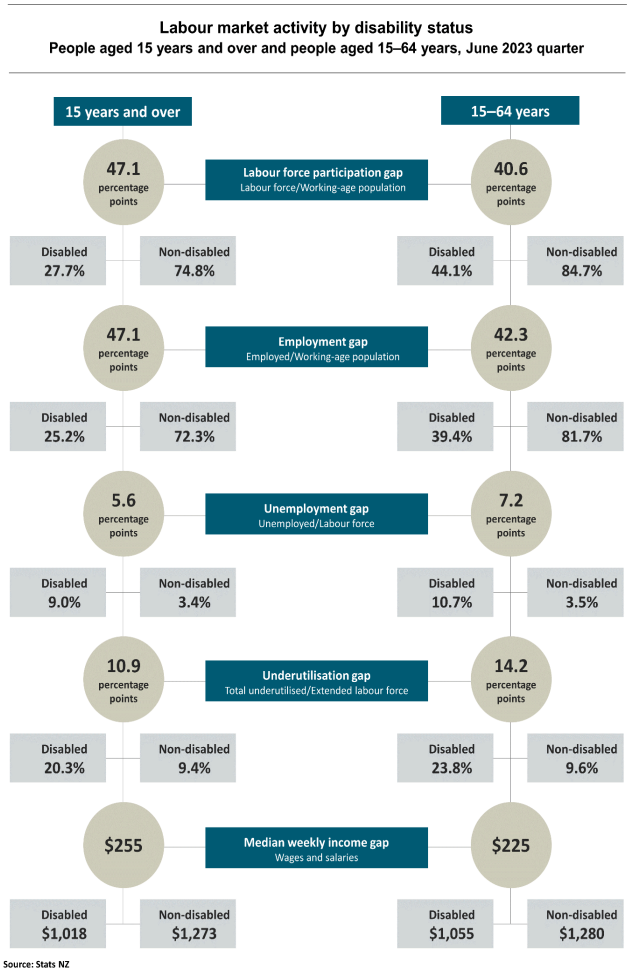
The target audience for this project includes the trainees at Te Tuhi Cafe, the support workers, and the administrative staff responsible for overseeing these training programs. By catering to these groups, we aim to facilitate a seamless and efficient training process.

The scope of the project encompasses the development and deployment of the Inka mobile application, which includes features for tracking trainee progress, storing all training materials, and allowing manipulation of data by the support workers and administrative staff. Additionally, the project involves integrating existing data into the new system to ensure a comprehensive and cohesive platform.

Our approach in carrying out this project involved agile project management methodologies, which allowed us to break down the project into manageable sprints. Each sprint focused on specific features and functionalities of the Inka application, ensuring steady progress and flexibility to adapt to changing requirements. In addition, we maintained regular communication and collaboration through weekly meetings with our tutor, and fortnightly meetings with our clients. These meetings served as checkpoints to review progress, gather feedback, and make any necessary adjustments to the project plan. Between these meetings, there were many exchanges of emails between us and our clients, discussing updates, clarifying requirements, and addressing any issues that arose. Furthermore, we held several in-person team meetup. These meetings allowed for brainstorming and problem-solving, which was crucial in delivering a successful project.

The important outcomes of our work include the successful creation of Inka, which consolidates and simplifies access to training resources, improves the efficiency of monitoring trainee progress, and enhances the overall training experience at Te Tuhi Cafe. This project addresses the immediate needs of Te Tuhi Cafe and hopefully sets a precedent for future digital solutions in similar training cafe environments.

Background



Our app aims to provide Te Tuhi Cafe with a digital repository to store their training materials as well as a medium to perform evaluations of their trainees online through the help of the support workers. To put the problem into context, we will dive deeper into the employment sector and disparities faced by people with disabilities.

One of the key barriers faced by people with impairments include disparity in employment opportunities. As seen in *figure 1*, in the June 2023 quarter, the employment rate for disabled 15 to 64-year-olds was 39.4%, compared with 81.7% of non-disabled people[1]. This is one of the key problems the Te Tuhi Cafe hopes to address by ensuring that people with disabilities can learn in a supportive environment, be better prepared to enter the working environment and showcase their value to society.

Figure 1: Labour Market Activity [2]

Currently, the cafe is partnered with the School of Psychology at the University of Auckland, which helps formulate personalised learning plans for trainees. However, there is a problem with aligning these plans with each individuals' learning pace. Moreover, the data needed for these plans is stored in drop-boxes and manually shared via email, making the learning plan development process tedious and difficult. This is where Inkā comes into play. While there are existing apps to facilitate learning various skills, our app will stand out from the rest by ensuring a personalized learning experience specifically suited for use in the cafe by people with various intellectual disabilities.

Furthermore, Inkā will aim to address the storage issue by having a repository that is easily accessible by the various clients to ensure the data is in one place. Inkā will then use this data to enable real-time tracking to allow the trainees to receive constant feedback through their evaluations (done by the support workers) and guarantee an efficient and continuously evolving learning platform.

By implementing Inkā, the concept of action learning will be put into use. Firstly, the

trainees will be able to receive instructions through the app (through visual cues and interactive text) to learn various tasks in the cafe. The support workers can then guide the trainees along this journey by checking if the trainees are performing most of the tasks correctly through checking off the associated task steps. This is followed by a judgement call screen where the support workers can choose from one of five different categories (*Independant, Verbal, Gestural, Physical Prompt, Not Completed*). These categories represent that specific trainee's performance for that task. The sessions for each task are recorded on a separate screen (after selecting the task) where the corresponding (most recent) judgement call, task notes, and feedback can be viewed for that specific trainee. In this way, the whole training process is designed to be iterative simulating real world training scenarios i.e. trainees will be evaluated for the same task for multiple sessions. The app stands out from other training apps in the sense that a trainee will never truly be "done" with a task so the app helps to facilitate this agile learning approach.

Inkā will be designed to be used seamlessly in conjunction with the prompt fading and the gestures/vocal support from the support workers to aid the trainees' learning further. Furthermore, it will include the element discrimination cards already used by the cafe. The user-friendly UI will support this multi-faceted approach and guide not only the trainees but also the admin and support staff.

Finally, Inkā's real-time feedback will allow users to gain more confidence in their skills and support them in hopes of building more employment opportunities for adults with intellectual disabilities.

By developing Inkā, we hope to create a platform that is accessible, easy to use and personalised to help fulfil Te Tuhi Cafe's goal of integrating people with disabilities in the community, providing the trainees with better access to job opportunities and helping them enjoy their learning journey. We hope to make the training process easier for the trainees, the support workers and the clientele and give the trainees the platform they need to grow.

The platforms and tools utilised throughout this project are listed in the next page.

Tools and Technologies

Logo	Name	Category	Description
	Flutter	Front-end	Flutter was used as our main front end language as it has many useful in built packages e.g. flutterTTS for text to speech translation and widgets to support our design
	Amplify	Back-end	Amplify was chosen as it can be easily linked to DynamoDB databases which was useful for our project. It was also used to link to Amazon S3, GraphQL API, AWS Cognito which were all key tools used in our app.
	GraphQL	Back-end API	GraphQL API was chosen as it could easily model the relationships between our tables and is supported by Amplify
	Amazon S3	Storage	Amazon S3 was used for storage as it can automatically scale to growing data which is useful for the app as the admin can add as many images as they want (for the tasks). It is also good for protecting data and ensuring accessibility over time.
	AWS Cognito	Authentication	AWS Cognito was useful for our app as it allowed us to easily obtain temporary user credentials to test our login features, as well as creating our final users that we shall give to the client. It is also useful to easily manage user profiles as our app has two different logins - admin and support.
	Lucid Chart	Database design (ERD)	The final version of our ERD was created using LucidChart as it was easier to manage our tables and link them with different cardinalities.
	Jira	Project Management	The kanban boards were used for agile project management and to keep track of deadlines, as well as a good way for us to note what we needed to get done.

	GitHub	Version Control	Github and Github desktop were used for working on different branches which was especially useful as we had a lot of screens for this app. Our work process was more efficient as we were easily able to merge and commit changes made to the same screens.
	Figma	UX/UI Design	Figma was used extensively for the initial low fidelity prototypes. We also used it to iterate on our designs and implement client feedback. Designing the low fidelity prototype on Figma made our front end process much more efficient.
	Discord	Collaboration	A discord server was made specifically for brainstorming, debugging, collaborating and delegating tasks. This allowed our team to communicate more effectively. We also had several different channels for each part of the project e.g. front-end/back-end development
	ClickUp	Collaboration	This tool was used for task management and time tracking specifically by making a Gantt chart.
	Visual Studio Code	Coding	Our team was most familiar with this IDE and it has many useful extensions and intuitive controls as well as easily identifiable syntax highlighting for different errors.

Specification

User Requirements

Our project can be separated into three main use cases: storing trainee data and training materials, performing trainee evaluations (support worker interface), manipulating data (admin edit/add/delete functionality). We can use a UML use case diagram (shown in the next page) to summarise the user's interactions with our app.

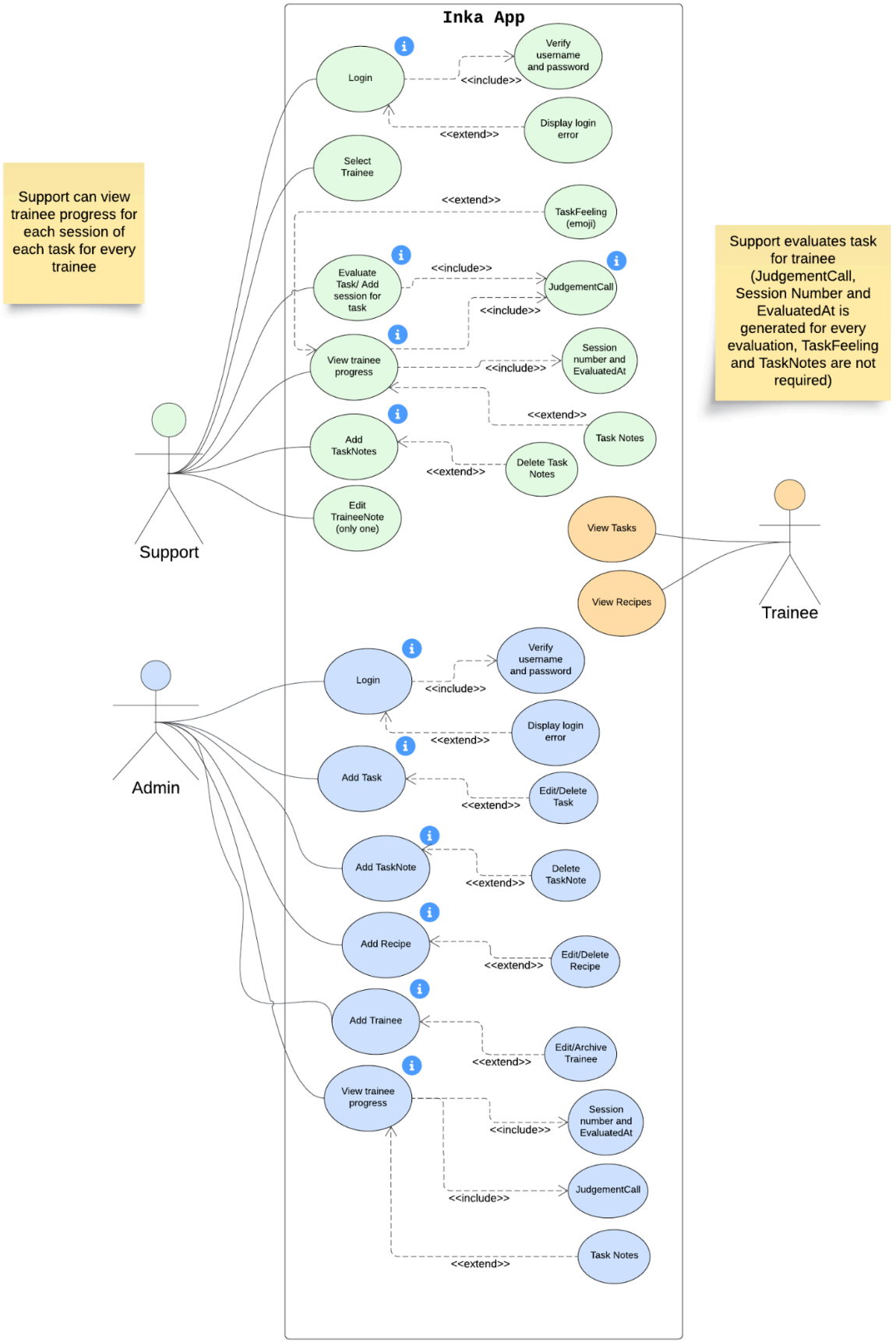


Figure 2: UML case diagram

As seen from the UML diagram in *Figure 2*, there are three different user roles: support, trainee and admin. Support and admin have separate logins, while the trainee is treated as a secondary user who can access all the training modules (recipes and tasks) without needing to log in. Support workers can edit and add new task notes, edit trainee notes, and perform evaluation based on the specific tasks and trainees after logging in. They have access to a progress screen with features such as judgement calls, task feelings, task notes, session numbers, evaluation dates, etc. Admins have all the capabilities of support workers, except for creating sessions and editing trainee notes and also have additional access to manipulate data for trainees, tasks, recipes and task notes.

The table below lists out the different types of requirements based on the user.

Actors	Requirements
Trainee	This user wants to access the training modules available but is not interested in manipulating the stored data.
Support Worker[1]	This user wants to manipulate the stored data and publish new data to the server.
Admin[2]	This user wants to manipulate the stored data and publish new data to the server.

1 - *includes all the functionalities that a trainee has i.e. view recipes and tasks*
2 - *includes all the functionalities that a support worker has except to manipulate evaluation data of trainees like trainee notes and session data for each task.*

Use Cases

Key Use Cases

The following listed below are our key use cases for this application as these use cases best encapsulate the services that our application provides.

1) Accessing Training Modules

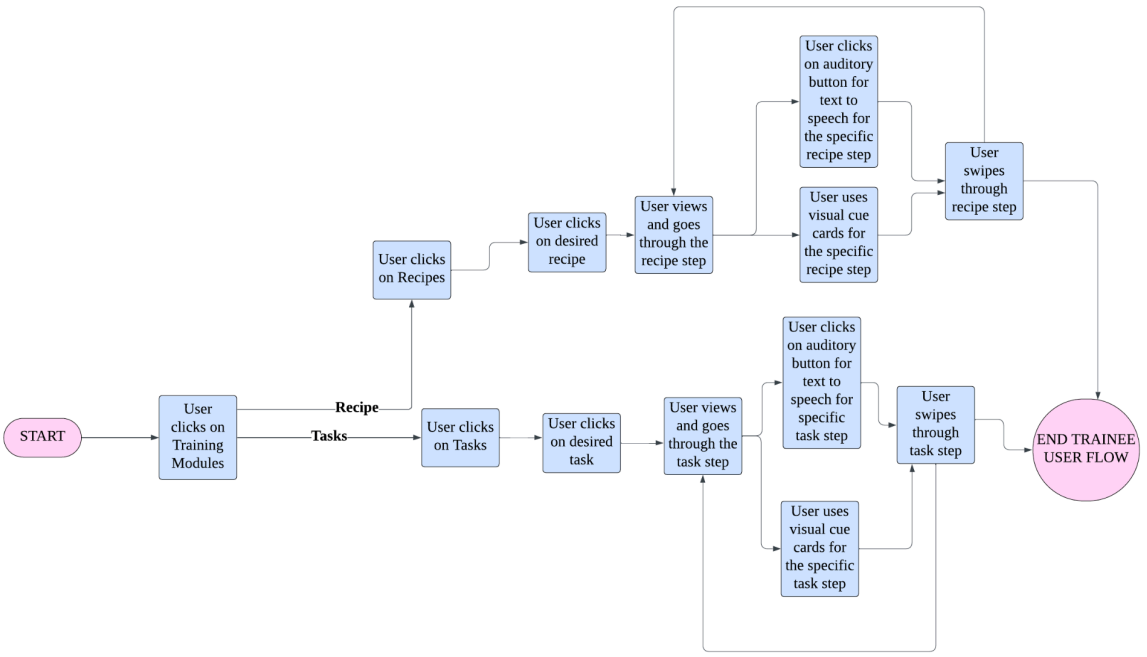


Figure 3: UML user flow for training modules

The training modules are accessible without a login, primarily serving to aid the trainees in viewing instructions in a user friendly manner. Additionally, support workers and admin can also access these training modules the same way without authentication if they wish to (see *Figure 3*).

The trainees can visualise cue cards and utilise auditory cues via a text-to-speech button for each task or recipe step. They can navigate through task or recipe steps by swiping, mimicking the experience of physical training materials. All actors will be able to see all the new tasks and recipes added by admin to the database.

- Training Module Homepage screen
  - Provides direct access to task view and recipe view screens.
- Task and Recipe view screens
  - Trainees can scroll, search and select desired tasks or recipes to follow through the steps.
- Selected Task/Recipe screens
  - Trainees can progress through each task or recipe step, utilising visual and auditory cues while a progress bar captures their advancements.

2) Manipulating existing data - Support

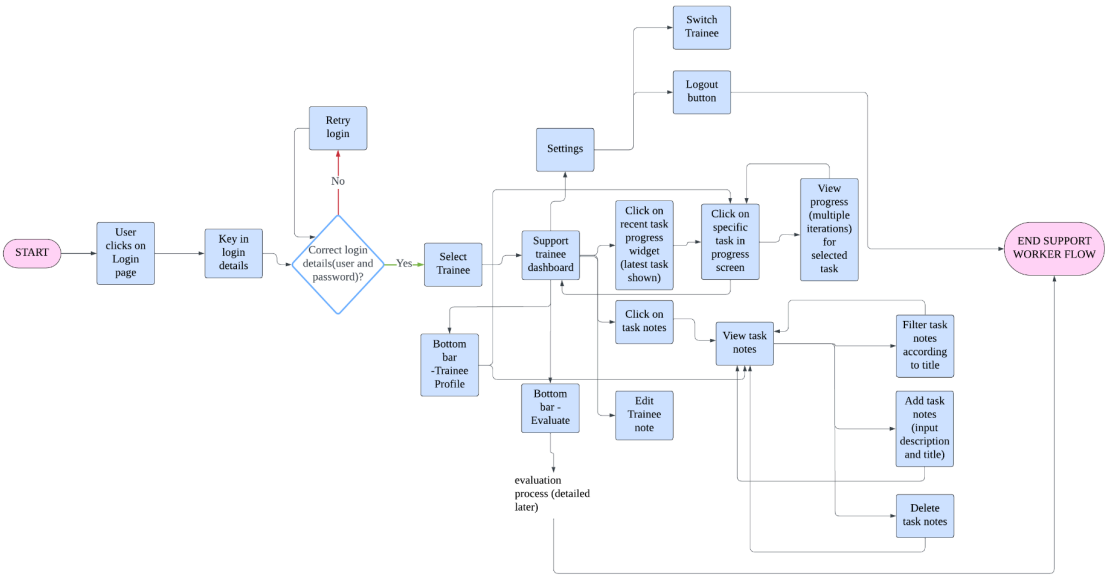


Figure 4: UML user flow for Support

Support workers have the capability to manipulate existing data, including trainee and task notes, and evaluate trainee progress. While admins have broader access to manipulate all data (including access to trainee progress), they are unable to evaluate the trainee. Other than the evaluation process, trainee note and specific task notes, the support workers are unable to manipulate trainee, task and recipe data (which can be done by admin).

Support workers can add/remove a task note. This can be done through the Support Trainee Dashboard. They can add/remove task notes for a particular trainee in 2 different ways:

1. Through the Support Trainee Dashboard:
   Simply add a task note with a specified title (this will be helpful if there is a specialized task meant for a trainee which does not require admin to create a new task in order to specific task note).

2. Through the evaluation procedure:
   The title of the task note will be the task title itself.

Support can evaluate the trainees and record their progress, storing the data into the database.

- Trainee Selection screen
  - Support workers can select a trainee for evaluation. They can also view/edit trainee notes or view/delete/add task notes.

- Support Trainee Dashboard
  - This dashboard includes recently evaluated tasks, the most recent task

note, and trainee note for the selected trainee. Support workers can access the All Task Notes screen from here to view all task notes for the selected trainee.

- All Task Notes screen
  - Support workers can view all task notes by selecting the desired task from the drop down menu. They can also delete a task note or add task notes, either for existing tasks or specialised tasks.

- Evaluation screens to add session using trainee progress (*detailed later*)
  - This includes storing/updating the task feeling for trainees, judgement call, session date and specific task note into the database for tracking progress through the Progress screen.

3) Manipulating existing data - Admin

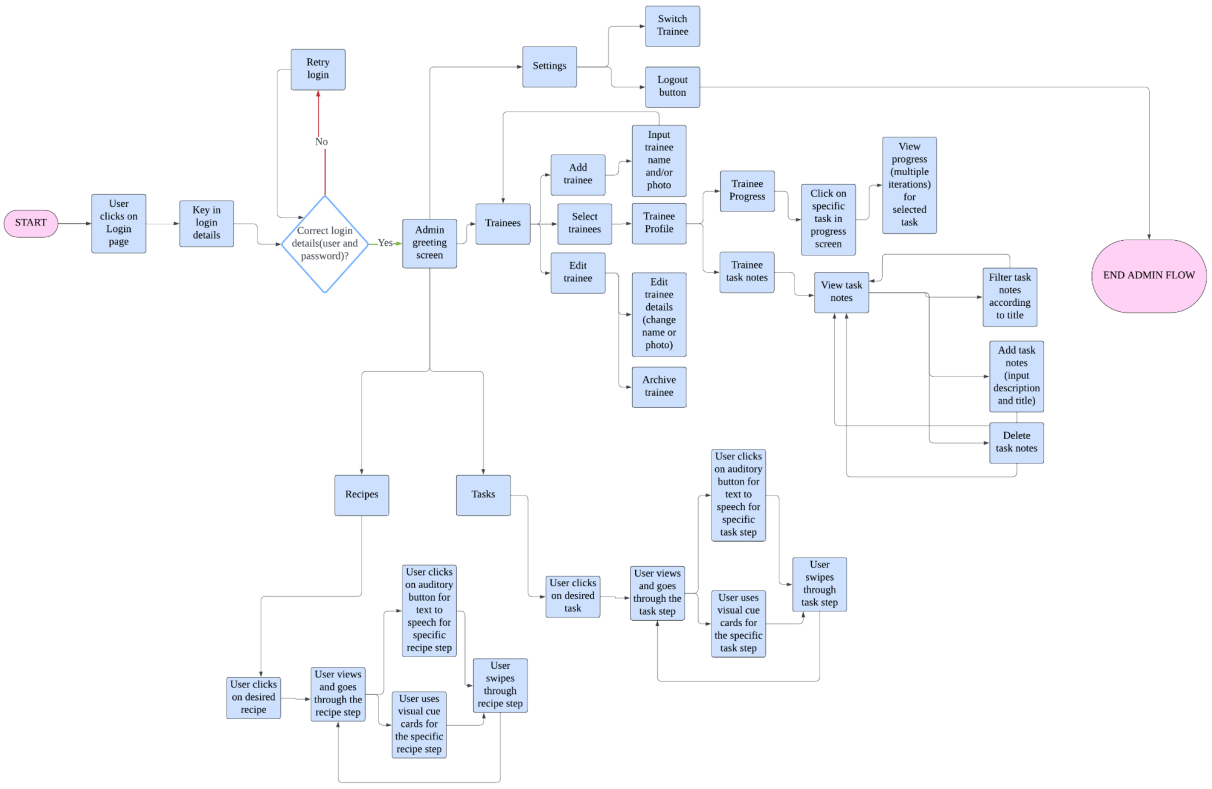


Figure 5: UML user flow for Admin

Admin has CRUD operations for recipe, trainee and tasks.

Trainees:

- *view*
  - Admin can select a trainee to view the progress and task notes for that trainee, just like a support can.
- *add*
  - Admin can add a trainee by entering their name and optionally adding a picture.
- *edit*

- Admin can edit the trainee details.

- *archive*
  - Admin can archive the trainees that have left the program with their details still visible in the view screens.

Recipe/Task:

- *view*
  - Admin can view the tasks and recipes the same way Trainees view and select.
- *add (+ add step)*
  - A new task/recipe can be added by inputting the task/recipe name, cover image and at least 1 step with optional step image.
- *edit (+ delete steps + add step)*
  - An existing task/recipe can be edited. In addition, steps can be added or deleted.
- *delete*
  - A task/recipe can be deleted.

4) Evaluating a trainee

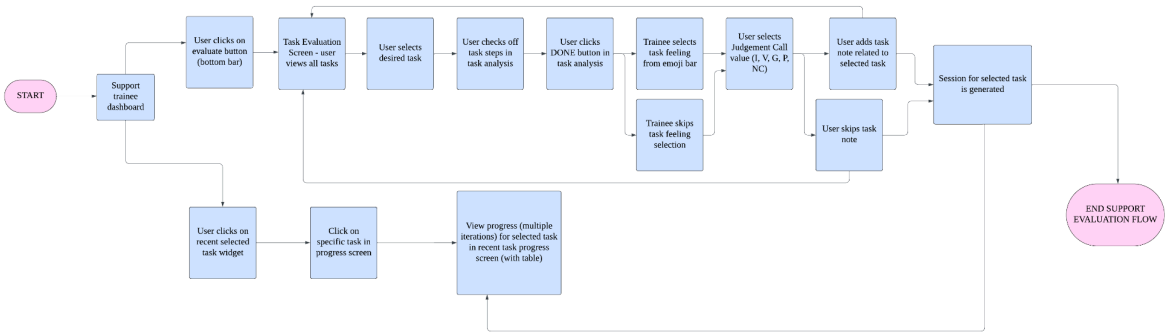


Figure 6: UML user flow for trainee evaluation

Support workers can initiate the evaluation procedure for a trainee by selecting a task to evaluate. They can analyze the task steps completed by the trainee, gather feedback, make judgement calls, and add detailed task evaluation notes.

Sessions created during the evaluation process are stored in the progress table and the progress can be viewed through:

1. Navigating to the Profile screen in the bottom bar and clicking on the progress tab.
2. Clicking on the recent task progress tab on the Trainee Dashboard screen.

- **Evaluate Task Screen**
  - *Selecting a task for evaluation for a specific trainee.*

- **Task Analysis Screen**
  - *Keeping track of steps done by the trainee:*
    - Support workers can perform task analysis by checking all steps

successfully performed by the trainee and clicking done. This will lead to the feedback screen.

- **Feedback Screen**
  - *Collecting input from the trainee regarding how they felt:*
    - The support worker can hand the iPad to the trainee so they can either select a task feeling or skip.

- **Judgement Call Screen**
  - *Getting support worker's input based on task performance:*
    - The iPad is handed back to the support worker who can select one of the five judgement calls. This leads to the Evaluation notes screen.

- **Evaluation Notes Screen**
  - *Adding further details for the trainee evaluation:*
    - The support worker can either leave a detailed note or skip.
  - This will create a session which can be viewed through the progress screen.

- **Session Screen**
  - Repository storing session information in table format, including session number, evaluated date, judgment call, feedback, and specific task notes.

Design

Technologies Utilised and System Architecture

System Architecture

Our app, Inka, embodies a modern serverless architecture, harnessing the capabilities of Flutter, AWS Amplify (cloud tech), and GraphQL to deliver a scalable and resilient mobile app experience.

The client-side encompasses the Flutter mobile application. Flutter's versatility empowers us to deliver a consistent and immersive user experience. The server-side infrastructure operates on a serverless model, abstracting away the complexities of server management and enabling seamless scalability. AWS Amplify orchestrates the integration between client-side and server-side components, streamlining communication and data exchange.

Handling user authentication and authorization, AWS Cognito ensures secure access to our application. For scalable storage of images, we rely on Amazon S3, tightly integrated with AWS Amplify for efficient storage and retrieval using tokens. Additionally, to power our real-time data querying and synchronization needs, we make use of AWS AppSync—a serverless API layer powered by GraphQL. GraphQL's flexible query language optimizes network efficiency and enhances user experience by fetching precisely the required data. Lastly, Amazon DynamoDB serves as our responsive and highly available database solution, seamlessly integrated with AWS AppSync to cater to the dynamic data requirements of our application.

Together, these components form the backbone of Inka, enabling us to deliver a seamless and responsive mobile application experience to our users.

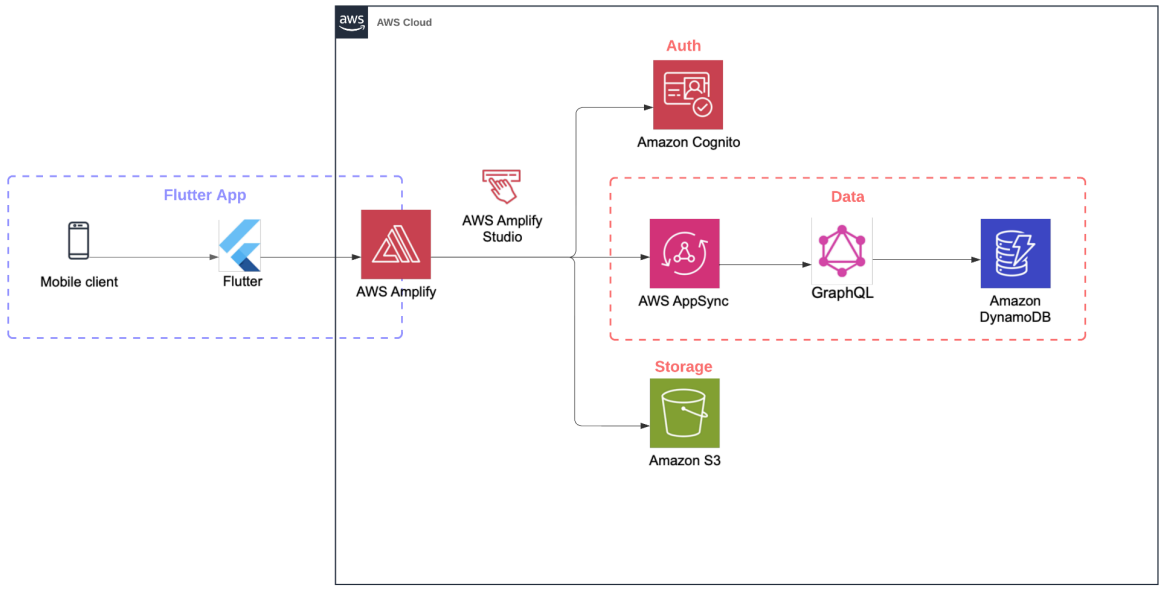


Figure 7: AWS system architecture diagram (serverless)

Main Algorithms

Our app made use of many different algorithms due to the three different interfaces and the variety of use cases. The main algorithms are highlighted below:

1) Uploading/Reuploading and viewing images - Admin

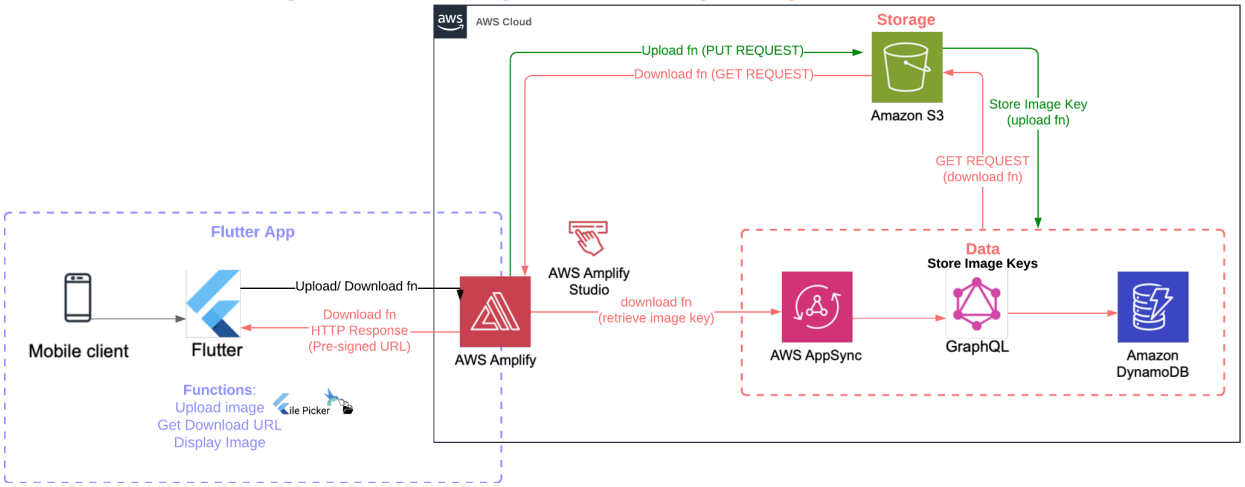


Figure 8: AWS upload/download system architecture diagram (serverless)

AWS Amplify handles the interaction with AWS S3 for image uploads, retrieval of signed (download) URLs for displaying images and storage of images keys in the database. The Flutter App uses the Amplify SDK to upload to S3 (via PUT requests) and retrieve pre-signed URLs for downloading images.

Uploading an Image

After the admin selects an image using FilePicker, the image is uploaded to AWS S3. The flutter app sends a PUT request to the S3 bucket with the image data and the HTTP PUT

request is constructed internally and the S3 bucket processes this request and stores the image key into the database. The image key is used to specify the unique identifier for the image in S3 bucket.

Displaying an image using signed URL

Wherever an image needs to be displayed in the app, the Flutter app retrieves the image key from the database and a download URL is created. The flutter app requests a pre-signed URL from AWS S3 using the *getDownloadUrl* function for the uploaded file using the image key to specify the object. The url includes temporary credentials that allow the client to access the object and this url is used to make an HTTP GET request to retrieve the image. Finally, the image is displayed in the app using the downloaded URL.

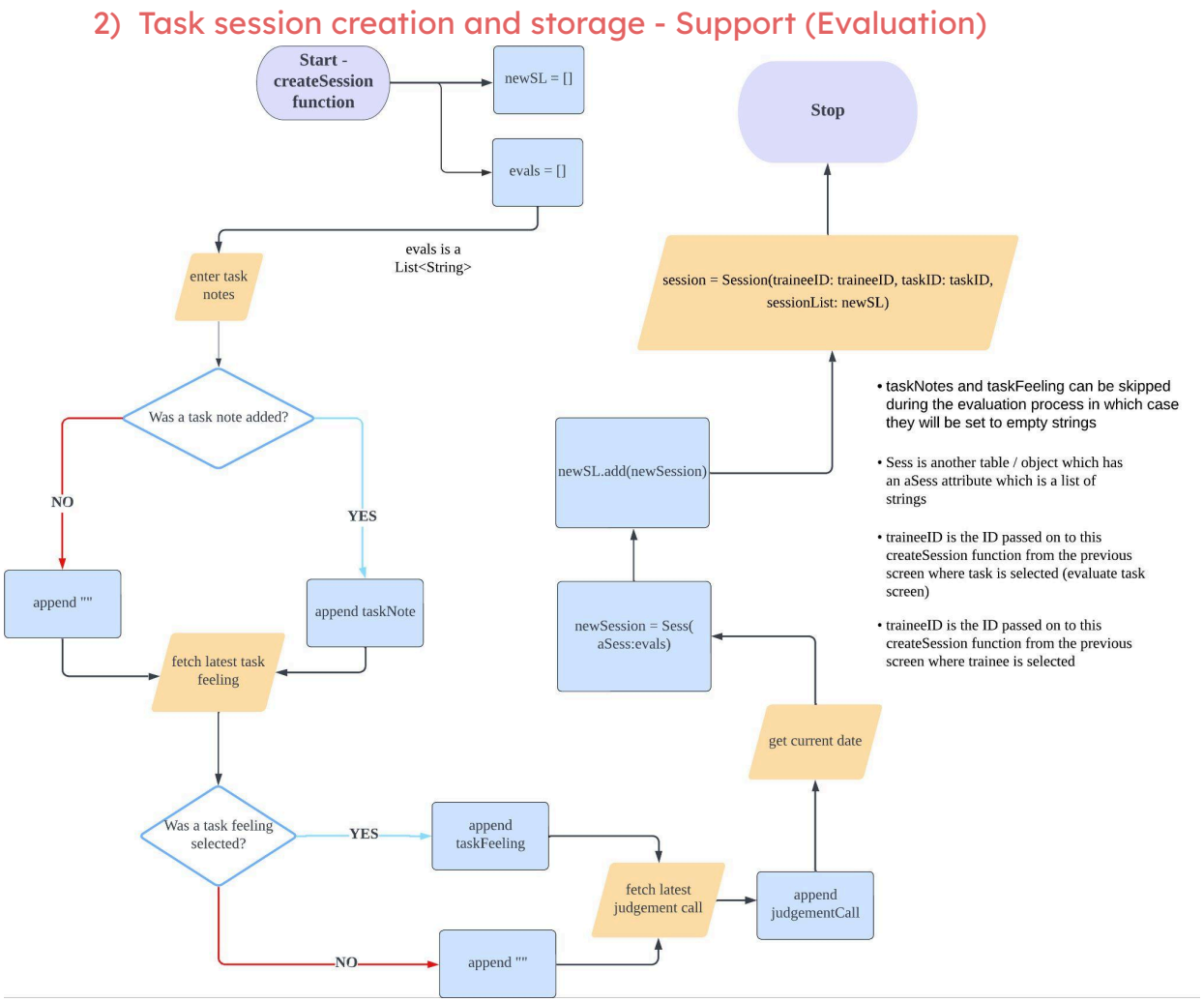


Figure 9: Task session creation and storage (inside Evaluation Section) for Support Workers

The tasks are linked to each trainee, and the sessions are connected to both the trainee and the specific task chosen by the support worker. The session is created on the Evaluation Notes screen, incorporating the latest judgment call, evaluation task note input, latest task feeling, and the current date of creation.

Additionally, the app includes several validation mechanisms. For instance, an admin will receive a snackbar error popup if they attempt to add a trainee without entering both a first and last name. Similarly, the admin cannot save a task or recipe without adding a

cover image and ensuring there is at least one step for the task or recipe.

Frontend

Flutter was utilized as the primary frontend language due to its flexibility and support for various in-built packages and modules essential for the app's functionalities. Given the 12 week time frame and Flutter's easy to follow and up to date documentation, we decided Flutter would be the better suited for our app's development compared to React Native. Moreover, Flutter's customizable Material UI based widgets provided a simpler and consistent look to help us develop a more polished looking application. Flutter also has better performance compared to React Native as it uses Dart, which proved to be an easy to grasp language.

Our UI was designed to be intuitive and user-friendly, with a focus on accessibility for all users, including trainees with intellectual disabilities. The training modules are accessible to all users without the need for authentication, featuring visual and auditory cues to guide trainees through tasks and recipes.

Backend

Due to AWS Amplify's streamlined development process and our restricted time frame, AWS Amplify was the core framework for our app's backend development. We were initially considering the traditional backend languages for our app i.e. NodeJS/Express. However, we realised that using these languages would require manual setup and integration of various third-party services, which can be time-consuming. Moreover, since the team was already committing to learning a new language for the front end (Flutter), Amplify was seen as the better choice. This was because using Amplify abstracts the infrastructure management and backend complexities allowing us to focus more on the application logic rather than the server maintenance and scaling.

Since Amplify was largely connected with other AWS services and pre-linked applications, we made use of the other services provided for us via Amplify, such as GraphQL API and AWS Cognito, allowing us to fully maximise our time into developing backend services that were required per the clients' needs.

- **AWS Amplify**: Greatly simplified the process of configuring, building, and managing AWS resources, enabling us to focus on delivering the necessary functionalities.

- **GraphQL API:** Utilized for flexible querying and modeling relationships between different tables within the DynamoDB database.

- **AWS AppSync**: Supports real-time data synchronization, enabling live updates and instantaneous feedback for trainees during their evaluations

- **Amazon S3**: Used for scalable storage of images and other multimedia resources required for tasks and recipes. Ensured that all training materials and evaluations are stored securely and easily accessible.

- **AWS Cognito**: Employed for authentication purposes, allowing for secure user login and management of user profiles

Database

For the database development, we chose DynamoDB. Aside from DynamoDB being AWS' in-house database service, we deemed that using DynamoDB would be the best option to model our tables as our database required many complex, dynamic data models and relationships. Moreover, DynamoDB was a good fit for our app as it automatically scales and ensures consistent performance. The Amplify API simplified the creation and management of the database. Along with this, the GraphQL API was utilised as it can be easily connected to Amazon's DynamoDB and allows for flexible queries. The documentation provided for Amplify API and GraphQL API was easy to read and allowed us to understand the data models and queries easily.

We started modelling the database through ERD diagrams to better envision the relationships and the overall schema.

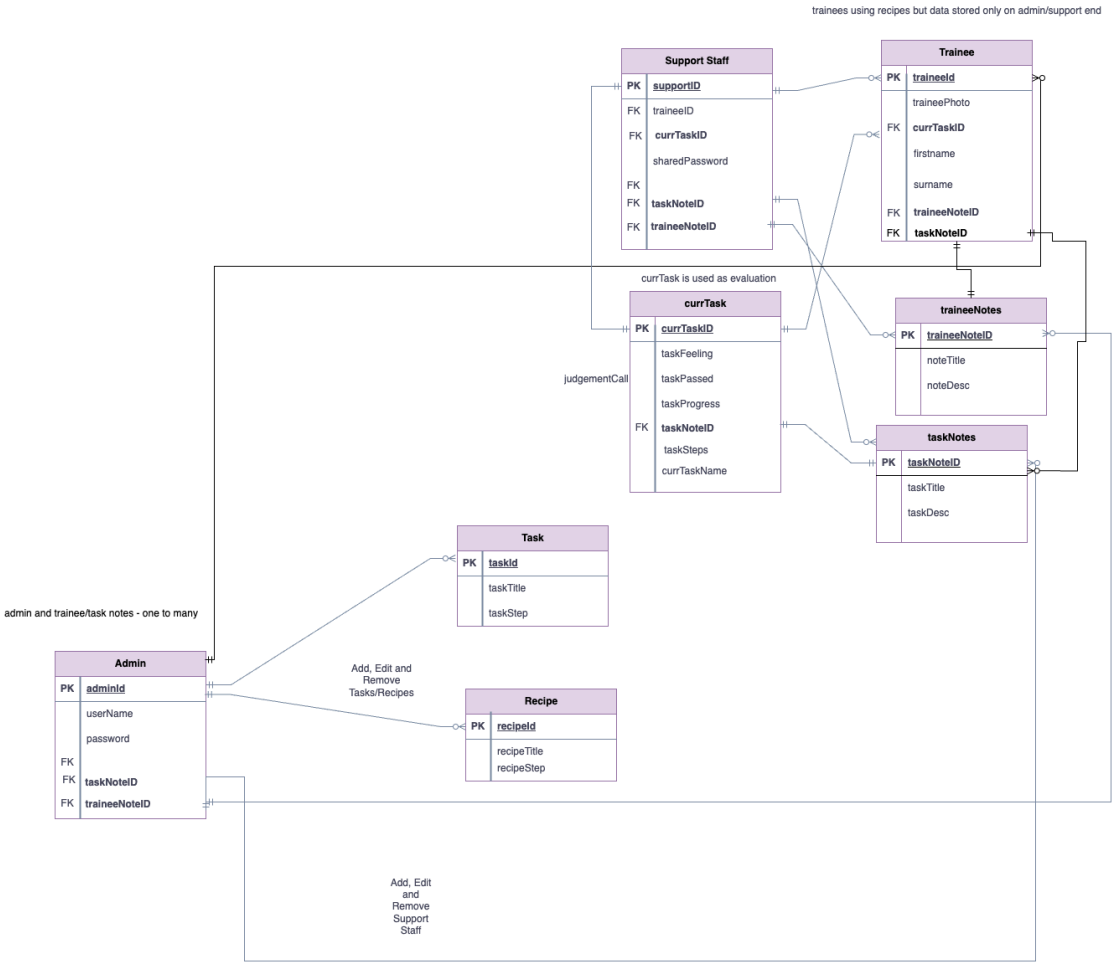


Figure 10: Initial ERD design

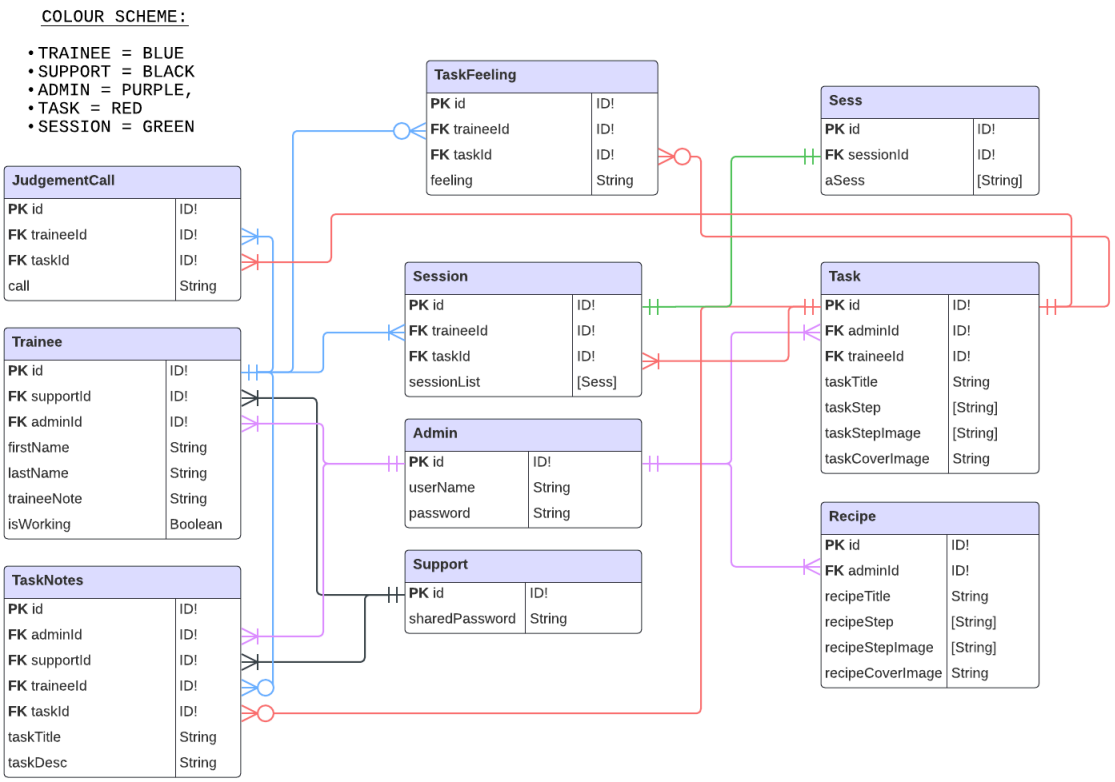


Figure 11: Final ERD Design

For our initial ERD design (see *Figure 10*) , we had some redundant tables e.g. currTask table that was removed later on as using the task table directly for the selected task proved to be more efficient (see *Figure 11*) . Our other tables remained the same. The traineeNote was added as an attribute of the Trainee table as accessing and manipulating it as an attribute was easier than having it as a separate table. Some new tables (e.g. JudgementCall, Session, Sess, TaskFeeling) were added and the cardinalities were changed. The initial idea of the app was targeted for the trainees but after several meetings with the client, it became clear that the app was meant more for the support worker and admin to use to manage the trainees evaluations and as a repository.

Our final ERD design (*Figure 11*) included new tables i.e. JudgementCall, TaskFeeling, Session, Sess. JudgementCall and TaskFeeling were separated as we found it easier to link to both task and trainee if they were different tables. The Session table was created in the later iterations of the app as the tasks would be completed by the trainees repeatedly and the iterations needed to be recorded in the database. The Session had a sessionList attribute which was a list of Sess objects where Sess is a separate table containing a list of strings attribute i.e. aSess. In this way, we were able to implement a nested list structure to record all iterations of each task and link it to the current trainee as well as display the latest task feeling, task notes and judgement call for each iteration. Most of the attributes had a string data type as they allow for easy manipulation. A few attributes were modelled as a list of strings e.g. taskSteps, recipeSteps as these contain multiple values

(more than one step). The ID attributes have the ID! datatype as seen in Amplify's data modelling to differentiate them as primary keys and ensure that they do not change once assigned.

The logic of the *Figure 11* final ERD is explained below:

- JudgementCall table
    - Represents the judgement calls i.e. I, V, G, P, NC categories are chosen after the trainee goes through a task evaluation (in support worker interface). It is an unskippable option and is linked to each trainee and task through Foreign Keys i.e. traineeId, taskId.

- TaskFeeling table
    - Represents the task feelings i.e. emoji bar. This is when the support worker passes the screen to the trainee and the trainee can choose how they felt about the particular task from a range of options e.g. Good, Not so Good etc. This is a skippable option and is linked to each trainee and task through Foreign Keys i.e. traineeId, taskId.

- Trainee table
    - This represents all the trainees (added by admin) in the database (including archived/past trainees). Each trainee can have no more than one trainee note which is editable and is displayed on the trainee dashboard for the support worker. The isWorking attribute is used to distinguish between old and current trainees. The trainees are linked to the support worker and admin through Foreign Keys i.e. supportId, adminId.

- Task Notes table
    - This represents the task notes (added by support) associated with each task and trainee. The task notes can be added either through the notes screen linked on the dashboard or after the support evaluates a trainee on a specific task. Not all tasks have to be associated with a task note. The task notes table has two main attributes - the title and the description. It is linked to the admin, support, trainee and task through their IDs as foreign keys.

- Session table
    - The session table represents all the iterations of the task and contains a list of Sess objects to keep track of the sessions. It is linked to the trainee and task through their respective IDs as foreign keys.

- Sess table
    - This is an object that is linked to the session table through the sessionID as a foreign key. The aSess attribute is a list of strings. Each Sess object is the current iteration of the task.

- Task table
    - The task table represents all of the tasks added by the admin. The taskCoverImage is required whereas the taskStepImage is not required i.e.

taskSteps do not have to have images. There is at least one task step associated with each task. The task table is linked to the admin and trainee through their respective IDs as foreign keys. The selected task is set using this table.

- Recipe table
    - The recipe table represents all of the recipes (added by admin). This is logically similar to the task table. It is linked to the admin through the adminID as a foreign key. There is no evaluation done based on the recipe so it does not need to be linked to the trainees.

- Admin table
    - This represents the admins (there can be more than one admin but they are treated as one entity). The admin has permission to edit/add/delete tasks, recipes, trainees and view task notes.

- Support table
    - This represents the support workers in the cafe (there can be more than one support worker but they are treated as one entity). The support workers have their own interface and perform the task evaluations. The support worker can add/view judgementCalls, taskFeeling and add/delete taskNotes. They can also view and edit traineeNotes attributes in the trainee table. The sessions are added and stored by the support worker.

<u>Cardinalities:</u>

- Support to Trainee:
    - *One to many:* One support can have one or many trainees but one trainee is linked to one and only one support.

- Support to TaskNotes:
    - *One to many:* One support can have one or many task notes for each trainee but one task note is linked to one and only one support. Assuming there is at least one trainee (linked to support) with a task that has a task note (i.e. there is at least one task note in the database).

- Trainee to TaskNotes:
    - *One to many:* One trainee can have zero or many task notes but one task note is linked to one and only one trainee. A trainee can have zero task notes.

- Admin to Trainee:
    - *One to many:* One admin can have one or many trainees but one trainee is linked to one and only one admin.

- Admin to Task:
    - *One to many:* One admin is linked to one or many tasks but one task is linked to one and only one admin

- Admin to Recipe:
    - *One to many:* One admin is linked to one or many recipes but one recipe is linked to one and only one admin

- Admin to TaskNotes:
    - *One to many*: One admin is linked to one or many task notes but one task note is linked to one and only one admin

- Task to TaskNotes:
    - *One to many:* One task can have zero or many task notes but one task note is linked to one and only one task.

- Task to Session:
    - *One to many:* One task can have one or many sessions but each session is linked to one and only one task. Assuming at least one session is recorded in the database.

- Trainee to Session:
    - *One to many:* One trainee can have one or many sessions (one or more iterations of tasks) but one session is linked to one and only one trainee.

- Session to Sess:
    - *One to one:* One session has one and only one Sess object while one Sess object is linked to one and only one session.

- Trainee to TaskFeeling:
    - *One to many:* One trainee can have zero or many task feeling (emoji bar is skippable) but one task feeling is associated with one and only one trainee.

- Task to TaskFeeling:
    - *One to many:* One task can have zero or many task feeling (emoji bar is skippable) but one task feeling is associated with one and only one task.

- Trainee to JudgementCall:
    - *One to many:* One trainee can have one or many judgement calls (judgement call option is not skippable) but one judgement call is associated with one and only one trainee.

- Task to JudgementCall:
    - *One to many:* One task can have one or many judgement calls (judgement call not skippable) but one judgement call is associated with one and only one task.

Authentication

Authentication consisted of three main factors - Differentiating the levels of Access for our two different interfaces and users, secondly, a reset password function, thirdly, two minor functions of validating the passwords within the reset password page (with a function to

ensure that password security standards were being met) and that the users would be able to see their passwords when entering them, with an 'eye' function. AWS Cognito was the key in supporting and enabling this. All users were configured within a user pool, with temporary passwords sent to a dummy email account created by the team. We added the functionality for the user to view their passwords if they would like to check what password they are typing in. With the reset password function, there is a validation function to ensure that the clients are utilising good security standards, where there is a red x indicating that the validation has not been met and then a green tick replaces that once the validation is met.

Version Control

All team members opted for GitHub as the service to keep track of the versions. Different platforms were used to create this application, MacOS and Windows, so we decided to use GitHub to retrieve any new or old versions. Near the start of the project, we were mostly utilising GitHub and pushing/pulling through terminal commands. In the later stages of development, however, we switched to GitHub desktop as we found it more intuitive and efficient to use. Instead of having many different branches, we set our prototype as the main branch and committed changes (with detailed summaries of new features added) directly to the main branch to prevent future confusion and more conflict errors later on.

To prevent conflicts while merging, we coordinated and delegated tasks using Kanban boards on Jira. Due to this, no two people were working on the same feature simultaneously, leading to increased overall efficiency during the development process.

UI/UX Design

UI Design Overview

The user-interface (UI) design drew inspiration from existing applications across a variety of genres, including fitness/health, education, and e-commerce. By leveraging the strengths of these diverse applications, we aimed to create a versatile and user-friendly interface that caters to our target audience, who are, as aforementioned, the cafe trainees, support workers, and administrative staff. This approach ensured that our design is both functional and visually appealing while meeting the needs of our audience effectively.

Gestalt Law

Our UI is heavily influenced by Gestalt Law which is a set of perception laws that serve the purpose of organising visual information such that it enhances user engagement and improves the overall user experience. The following Gestalt laws applied are:

- **Similarity**
  The law of similarity posits that elements with similar visual characteristics (size, shape, colour) are perceived as part of the same group, helping users quickly identify related items and navigate the interface more efficiently.

- **Continuation**
  The law of continuation suggests that we tend to perceive elements as continuous shapes rather than disjointed parts. This principle ensures smooth navigation and flow within the interface.

- **Proximity**
  The law of proximity suggests that elements close to each other are seen as a group, structuring information logically and intuitively.

- **Symmetry**
  The law of symmetry posits that symmetrical designs are perceived as complete and harmonious, naturally attracting our attention. This principle contributes to a balanced and pleasing visual design.

Design Principles

In addition to Gestalt laws, a few key design principles were considered:

- **Balance**
  Balance refers to the distribution of visual elements to create a sense of cohesion.

- **Emphasis**
  Emphasis involves highlighting certain elements to draw attention. It helps direct the focus to key areas or actions within the interface.

- **Unity**
  Unity refers to the integration of various elements to create a unified whole. Consistency in design elements, such as typography and colours, help achieve unity.

Colour

Colour played a significant role in the design process, particularly in adhering closely to our client's branding. The primary color in their branding is a brilliant red which proved too intense for the key elements of our interface. Therefore, we opted for a more subdued shade of red to maintain their brand consistency. In tandem with this choice, we incorporated warmer colours to supplement the colour palette, as warm colours are perceived as inviting and close, making the interface feel more welcoming to our audience. Additionally, we introduced blue for contrast.

Figure 12: Inkā Colour Palette

- **Design**
  To prevent users, especially trainees, from feeling overwhelmed, the 80/20 rule was applied to the colour palette, with the majority of the interface in neutral tones

(#FFFFFF and #E0E0E0), reserving the other colours for key elements.

- **Accessibility**
  We verified that the chosen colours complied with accessibility standards for colour contrast, ensuring that all users could read the content easily. This involved conducting tests for colour contrast ratios using WebAIM and making necessary adjustments to meet the guidelines.

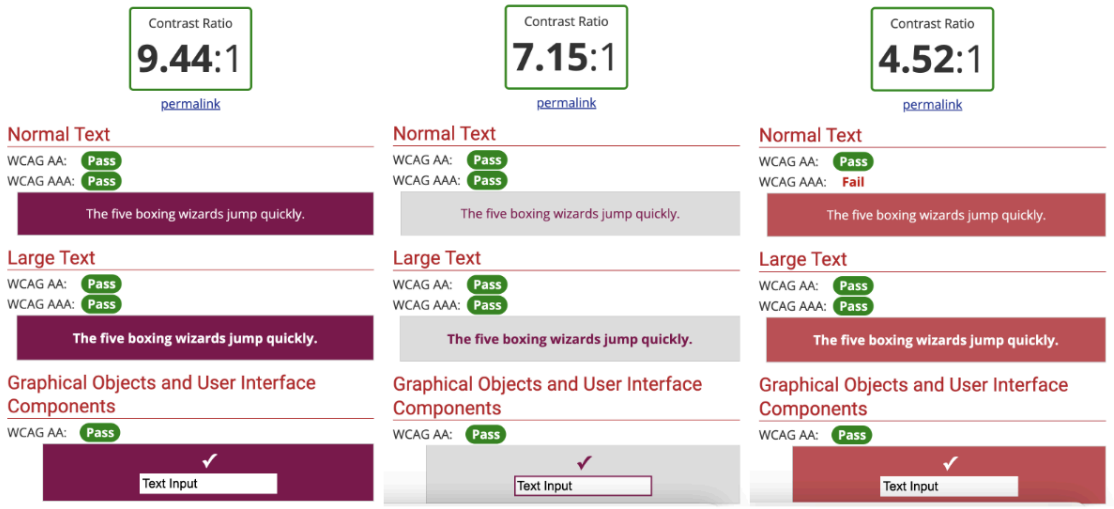


Figure 13: Examples of WebAIM's Contrast Checker

UI Elements

The individual UI elements were designed with careful consideration of both aesthetic appeal and functionality.

- **Buttons and Icons**
  The buttons and icons serve as interactive elements guiding users through the application. Uniform styling ensures users can easily recognise the functions.

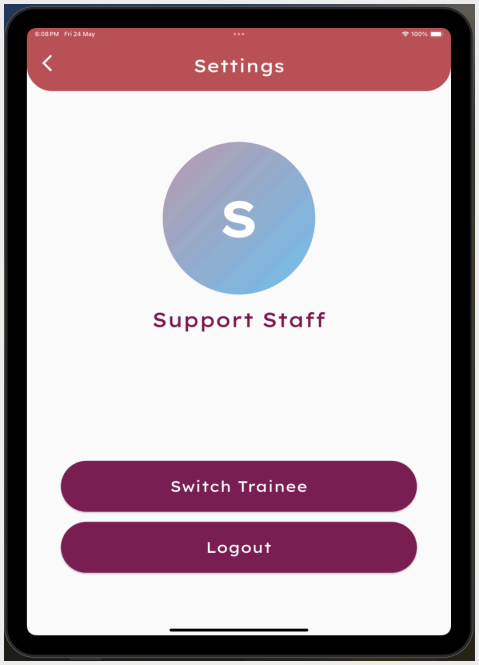
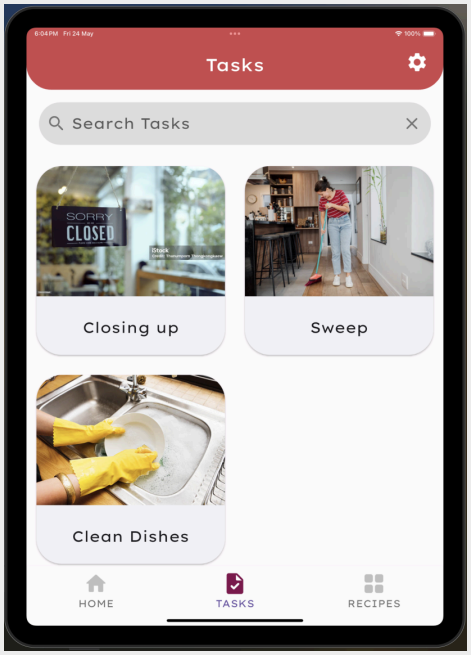
- **Forms and Input Fields**
  The forms and input fields facilitate data entry and interaction. Clear labels and hint texts were used.

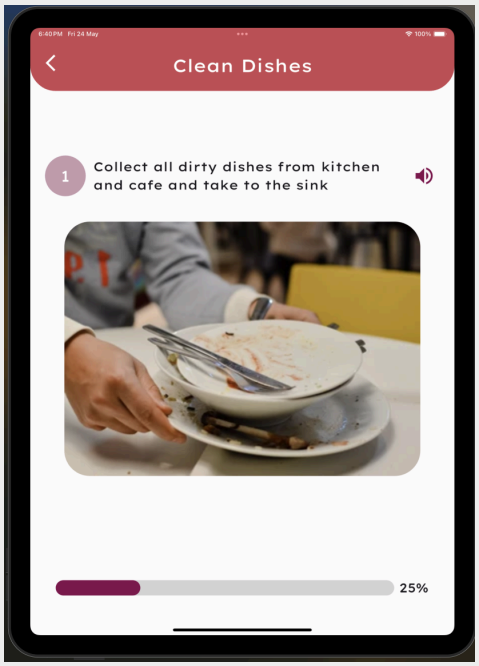
- **Navigation Bar**
  The navigation bar enables intuitive navigation within the application with clear labelling to help users understand their current location and available functions.

- **Typography**
  The application uses legible fonts, ensuring that users can easily read and understand the content.

Application

Below are some examples of how the outlined Gestalt laws, design principles and colour palette are used in the UI design.

Example	Principles Used
	**Proximity** The 'S' circle avatar and 'Support Staff' label are placed close to each other, suggesting they are part of the same group. **Similarity** In addition to close proximity, the 'Switch Trainee' and 'Logout' button share identical visual characteristics, also suggesting they are part of the same group. **Balance / Symmetry** All of the elements are centre-aligned, achieving symmetrical balance. **Unity** The typography is consistent throughout, unifying the design.
	**Proximity** The tasks are grouped closely together, indicating they belong to the same group. The grouping helps the user understand that these elements are part of the same functional area. **Similarity** The tasks share identical layout, making them visually consistent and easy to recognise as related elements. This helps the user to quickly identify the pattern and understand the layout of the interface. **Continuation** The alignment and flow of elements guide the user's eye naturally from the search bar to the individual task cards, creating a seamless experience as the user scrolls through the tasks. **Unity** The typography, colour palette, task card shapes are consistent throughout, unifying the design.



Proximity
The task instruction, image, audio button are placed close to one another, indicating they are related.

Continuation
The visual flow guides the user's eye from the task title down to the instruction step, and then to the image, naturally leading to the progress bar at the bottom.

Balance/Symmetry
The design maintains visual balance with the center-aligned elements.

Emphasis
The image, the main facilitatory cue for the user is emphasised through size, making it stand out.

User Experience Overview

User experience (UX) encompasses crucial factors which we took into consideration:

- **Usability**
  Usability is the measure of the quality of a user's experience when interacting with the application. The quality depends on the ease of learning, efficiency of use and memorability. For the trainees with intellectual disabilities, the application was designed to be as simple and intuitive as possible. We ensured the buttons are large and easy to tap, text is clear and concise, and visual and audio cues are used to guide them through the task and recipe instructions. Support workers and administrative staff also benefit from an easy-to-use interface, allowing them to efficiently navigate the application while supporting trainees.

- **Usefulness**
  Usefulness refers to the ability to use the application to accomplish goals of work. The application is designed to assist trainees in various tasks and recipes, whereas support workers and administrative staff can access the application for monitoring progress, evaluating trainees and providing guidance.

- **Emotional Impact**
  Emotional impact is the affective component which focuses on the user's feelings and satisfaction with the application. Recognising the importance of a supportive and positive environment, the application incorporates features for the support workers and administrative staff to provide encouraging notes and feedback of the trainees' performance.

- **Meaningfulness**
  Meaningfulness refers to the user's long-term personal relationship with the

application. The plays a vital role in the lives of the trainees, providing them with opportunities for skill development, empowerment and personal growth. Through regular use, the trainees can develop pride in their work and build long-term connections with the wider community. Support workers and administrative staff play an important role in nurturing these connections, providing guidance and encouragement to help trainees realise their full potential.

In designing our application tailored for our clients, we integrated the foundational elements of UX outlined above. Additionally we drew upon Nielsen's Heuristics as guiding principles for the design evaluation, ensuring factors such as visibility of the system status, user control and freedom, consistency and standards were upheld for an intuitive, inclusive and efficient user experience.

Project Implementation

Frontend Implementation

The frontend implementation of our project began with the creation of a high-fidelity prototype using Figma. This tool allowed us to design an intuitive and visually appealing user-interface that met the needs and preferences of our clients. We essentially had free reign to select the colour palette, font family and more. The prototype included detailed layouts, interactive elements, and design specifications, providing a clear blueprint for the development phase.

Once the prototype was finalised, we translated the designs into functional code using Flutter/Dart. Flutter provides a rich set of pre-designed widgets, which made it an ideal choice for our project as we were unfamiliar with the particular framework. Hence, the learning curve proved to be more manageable. By leveraging Flutter's widgets, we were able to ensure the final product closely resembled our prototype created in Figma, maintaining consistency in design and user experience. The specific widgets used for some functionalities are described in the backend implementation section.

Our finalised prototype comprised more than 50 screens. Therefore, we initially divided the work among us, each person working on 10 screens. However, in the end, we decided to delegate frontend implementation to one person to allow for a streamlined and uniform design.

During the process, we created Kanban boards on Jira to keep track of the main functionalities to do as well as to delegate tasks. Initially, we did not have any specific roles set up, as we were all focused on learning Flutter, Figma, Jira and Git and helping each other out. We divided up the work according to the different interfaces and merged our screens manually to avoid conflicts.

Backend Implementation

For the backend implementation, we had to split the development period between the database design and the backend services, as the main functionalities needed depended on the data that had to be retrieved and manipulated from the database. We split off into specific roles after the initial prototype was developed, with one person focusing on the front end and the rest focusing on the backend. Our key backend services and database implementation are written below.

Key Backend Services

Our client asked us to help automate their training process for their trainees with intellectual disabilities. The key functions that they wanted us to include within this app are the following listed below:

- **Search Bar and Autocomplete functionality**
  The search bar is implemented using a *'TextField'* widget so users can input their search terms in the *'_textController'* field. The AutoComplete widget is then used to provide search suggestions. The optionsBuilder is part of this widget and it filters the list of task notes (*allTaskNotes*) based on the search entered by the user. It returns a suggestion if the text entered is either a substring of the task title or task description. If the search text is empty, an empty iterable is returned indicating no suggestions. When the user selects a suggestion from the autocomplete list, the *'onSelected'* callback updates the state with the selected task note. The *'searchResults'* list is updated to include the selected task note and the search text displays only the selected task note's title. The _onSearchTextChanged function is called whenever the text in the search bar changes. It filters the 'allTaskNotes' list based on the search text and updates the '*searchResults'* list accordingly. The search bar is rendered using the fieldViewBuilder function, which builds the *'TextField'* with appropriate decorations, including a clear button to reset the search.

- **Drop down and filter functionality for Task Notes**
  The drop down is implemented using the 'DropdownButtonFormField' widget. The *'_selectedGroup'* variable holds the current selection in the drop down. The items in the drop down list include *'All'* to display all the task notes and unique task title names from the *'groupedTaskNotes'* map. This map is created by the '_groupTaskNotesByTitle' function. The _groupTaskNotesByTitle function iterates over the list of task notes and groups them by their titles. It returns a map where the keys are task titles and the values are lists of task notes with that title. Finally, the '*ListView.builder'* widget is used to display the task notes. The list's content depends on the selected group:
    - If 'All' is selected, it displays either the *searchResults* (if search text is not empty) or *allTaskNotes*.
    - If a specific task title is selected, it displays the task notes corresponding to that title from the *groupedTaskNotes* map.

- **Text-to-Speech functionality for Task/Recipe cue cards**
  The text-to-speech (TTS) Flutter plugin was utilised along with methods to control speech rate, volume and pitch by TTS which is configured during initialisation. When a trainee swipes through either the task or recipe steps, a volume icon is provided next to the step description. When the user taps the TTS button, the app triggers the Flutter TTS plugin to narrate the step description aloud using synthesized speech. This functionality enhances accessibility and user experience, allowing users to listen to the instructions without reading them.

- **Manipulating and Storing Data**
  Our client intends to migrate all of their accumulated trainee data to the application, necessitating the creation of backend services to manage, manipulate, and store this information. We have added appropriate functions in each of the screens titled 'view', 'add', and 'edit'. We implemented CRUD operations based on the information provided by the actors. For example, the **createRecipe** function uses two helper functions: **addStepButton**, which dynamically populates the step list, and **deleteStepButton**, which allows actors to delete a step if needed. When an actor creates a recipe (*or other data models*), a validator ensures that the information passed is valid, preventing illegal transactions. If required fields are left empty, a pop-up informs the actor of the specific missing information, such as *'Please add a recipe title!'* if the title is missing.

- **Authentication Implementation - Functions and Issue**
  Our primary issue was accessing the application once the temporary passwords expired and we were unable to change them ourselves. With further Administrative Access granted to us, we were able to utilise commands on terminal to force change the passwords and to enable the team to utilise AWS Cognito to reset passwords there. If the user inserts the wrong username, they will get the error '*Error signing in: User does not exist*'. If the password is wrong, they will get the error '*Error signing in: Incorrect username or password*'.

  After this, we realised that there was a security issue i.e. the support login would still be able to access the admin interface which contained all the edit and delete functionalities. We utilised AWS Cognito to create custom attributes and added specific dummy users to groups labelled 'Admin' and 'Support', however the code ended up searching for the custom attribute 'custom:role' and the respective dummy users would have the value of either admin or support to differentiate their access. The next step was then to do a differentiation which would take simple *if* statements and redirect the user based on the username/password combination utilized. However we came across issues here, and thus we introduced two separate buttons which ultimately would be user friendly and allow for the user to select which interface they would like to use. After conducting extensive testing, the support user was no longer able to access the admin interface without admin credentials and vice versa. If they insert the wrong access credentials, they will get the error, *'Access denied: You are not an Admin*' or ' *Access denied: You are not a Support Worker*'.

  The final step of authentication was adding a reset password functionality - which

the client had expressed interest in. The clients will be able to reset any passwords, via a verification code being sent to the email address registered with the AWS Cognito dashboard, as long as the passwords meet password security requirements. There is a validation in place to ensure that this happens, as well as double checking that the same password is inserted the second time. In addition to this, we added an 'eye' feature that will allow the user to view the password if they are unsure of the password that they have entered.

Database Implementation

For the implementation of the database, we have included photos of our initial and final designs (*Figure 10, 11*). For the backend functions to work as expected, we had to design the database thoroughly to ensure that all the data being sent and stored were correctly routed when the functions were triggered. We used Jira and Discord for brainstorming and tried to think of the app's overall functionality when designing the tables to avoid getting too overwhelmed with the details.

Unforeseen Issues

For the duration of this project, we encountered many unforeseen challenges that tested our problem-solving skills and strategy. Our problems ranged from using the AWS services to client-related expectations and using GitHub. We have listed the issues we faced and how we overcame them below.

Most of our issues initially stemmed from using AWS and its various in-house services. AWS presented a learning curve for all team members since this was our first time utilising this cloud service. We encountered several issues while working on the backend, particularly with linking Amplify to our Flutter project, due to our need for more expertise. However, we overcame these challenges by thoroughly reading Amplify's documentation and consulting various online resources, including YouTube videos and other website tutorials.

We encountered a challenge related to our backend integration. Initially, we explored using AWS Lambda functions, a serverless computing service, to run our code without managing servers. However, we spent considerable time trying to link the backend to the frontend using these Lambda functions. Upon re-evaluating our approach and conducting further research, we decided to use AWS Amplify API and GraphQL API to connect our DynamoDB database directly to the frontend. By incorporating backend code directly into our Flutter screens, we adopted an unorthodox but efficient solution. This approach saved us significant time and avoided potential future issues, thanks to Amplify's comprehensive documentation on using GraphQL API operations for CRUD functions. Through resource sharing and pair programming, our team made substantial progress in a short period. This collaborative effort ensured we could swiftly implement a robust and effective backend solution.

Our team also found that Amplify lags significantly when two or more users work on the same thing. While creating our database on Amplify, the interface performed slower when multiple users attempted to create tables. As a team, we agreed only to have one person

use Amplify and inform everyone of the changes. This process was tedious; however, we lessened the probability of using a slow service and unsaved changes if everyone used Amplify simultaneously.

Another issue we encountered with AWS was storing images. Image storage in the database was integral to this application since Te Tuhi Cafe used visual cues for the trainees, which was a client requirement. Initially, we created the download URL for the uploaded image and stored it in S3 as a pre-signed URL. However, this approach was problematic because the tokens would expire, and we were not utilising Amplify's automatic refresh. As a result, the download URLs were not updated in the database, requiring the client to re-upload images every hour due to the token's 1-hour expiry. We overcame this issue by changing our image upload method. Instead of using generated URLs, we switched to storing keys. These keys can be used to generate URLs whenever images need to be displayed, significantly improving our image handling and ensuring that the images are consistently accessible.

Additionally, as stated previously, we ran into some authentication issues as we were given limited access to AWS Cognito. We got Administrator Access with AWS Cognito, which allowed us to force change and reset passwords, ultimately getting it past the 'Force change password' confirmation status to the 'Confirmed' confirmation status. Another challenge was the differentiation part, where we originally planned for everyone to access only one login page, and then the credentials would re-route them to their expected interfaces. However, this did not work, and we ran into errors. After some brainstorming, we decided on the two buttons (which initially were backups in case the differentiation did not work out), and after, utilising similar code, we got it to work.

We also encountered a client-related challenge concerning the evaluation section of our app. Given the tight timeline, we aimed to gather as much client feedback as possible while adding functionality to numerous screens. Our initial design for the progress and evaluation section for support workers conflicted with the client's vision, leaving us uncertain about the next steps. However, after a productive team meeting, we understood the client's requirements better and revised our evaluation screens accordingly. The client preferred tasks to be iterative, with each iteration recorded in the database. As a result, we removed the progress bar entirely and focused on creating new tables and functionalities to log these "sessions" in a new progress table linked to the specific task and trainee. Additionally, we made several design and functionality changes to align with the client's vision. This collaborative effort allowed us to transform the initial setback into an opportunity to improve the app's overall design and functionality, ensuring it met the client's needs more effectively.

Lastly, we had to familiarize ourselves with Git, as it was not a tool we had used extensively. Initially, managing the process of pushing and pulling changes through numerous terminal commands felt tedious. Additionally, with so many screens to manage, tracking changes became challenging. In the later stages of our app development, we decided to switch to GitHub Desktop. This transition made fetching changes from the repository and committing updates much more efficient. It also allowed us to view version history and real-time changes clearly and concisely. Our team found GitHub Desktop to be more intuitive than using Git via the command line or the website. This change significantly streamlined our workflow and improved our version control process.

Results & Evaluation

Project Goals

Our main goal was to complete the project fully and to have a prototype - if given the opportunity - it would be able to go live immediately to our hand in the project. We wanted to ensure that the clients would be happy with our project and could potentially see use with this project. We would say that the clients enjoyed interacting with the application and seeing this come alive over the last couple of weeks. There was feedback naturally and other ideas to implement and improve on, but we can say that the client's overall were satisfied with the product that we gave them. Some things that could hinder the immediate go live, would be transferring the code base over to their system as well as the client's requirement to set up AWS and potentially moving this application over to the apple app store as the application is aimed for use on iPads.

Goals that we had based on the client's expectation included, having a personalized learning experience for the trainees, being able to track trainee progress in the tasks that they completed - also to note here that we were working under the impression that the trainees would never fully complete a task, rather if they actually mastered the task one time, they would still continue to do the task - and finally to build a repository for all the training materials that the clients would like to store on the application - as their current process includes storing everything between dropbox, email and paper copies.

The overall goal for us, in conjunction with the Te Tuhi Cafe would be that the users that this application has been created for, will be able to use it with ease and understanding. That there will be a smooth transition from a paper based learning environment to a digital based learning environment. Our hope was that any trainee that comes through the programme will have an enhanced learning experience through the application we have developed.

Testing The System

We tested the system alongside the creation of a function. Once a function was completed, we tested the prototype's functionality by utilising dummy users (*for authentication*), dummy recipes, tasks, task notes, and trainees (*CRUD operations*). Additionally, we ensured that the functionalities worked even when new commits to the code were made on GitHub. After confirming that a function worked correctly, we created additional test cases to further validate our system. These test cases, documented in the Inka Test Case Document (*Ref C*), ensured that the system could handle both legal and illegal transactions with the database, regardless of the user's actions.

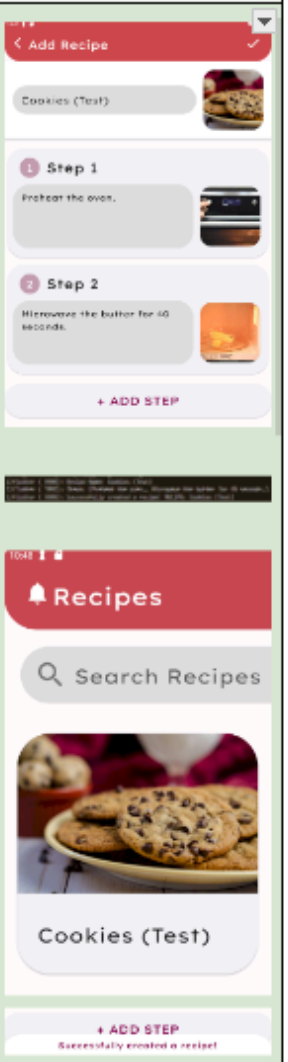
Response	Test Case	Expected Result	Reference
EXPECTED, CORRECT RESULT	As a business user, I can create a recipe by inputting the appropriate information and appropriate images.	Successfully created a recipe. On the terminal, it should return the following statements: `Recipe Name: (name of the recipe) Steps: (list of recipe steps) Successfully created a recipe! RECIPE: (name of the recipe)` [Second Image] There will be a snackbar popping up afterwards to inform the user of the unsuccessful process. [Third Image] On the application, the added recipe should be shown in the recipe list in the Edit Recipe screen [Fourth Image]	

Figure 14: createRecipe test case

An example of how we created a test case is seen in Figure 14. This is one of the expected and correct test cases under **createRecipe.** Each row contains a response, test case, expected result and reference for users to expect when performing a selected test case.

Evaluation

The evaluation reflects the strengths and weaknesses of our project. We aimed for the system to be accessible to users of all ages and varying levels of tech-savviness. Most feedback indicated that the system is intuitive and easy to use. We employed different testing methodologies—usability, integration, and exploratory testing—to thoroughly assess the system's performance.

Evaluation results varied depending on the testing method and user, but all feedback was

consistent. We conducted three types of testing with external testers:

1. Independent Exploration: Testers navigated the application on their own, testing functionalities without input from the team.
2. Demonstration: The team demonstrated the application's functionalities to the testers, explaining how everything worked.
3. Prompted Testing: Testers were given specific tasks to perform, such as adding a task or archiving a user, to evaluate the system's response and ease of use.

Strengths

- **Usability**
  The application is easy to use, allowing even non-technical users to quickly understand its purpose. For example, during client demonstrations, initial requests for detailed documentation evolved into remarks about the application's simplicity and ease of use.

- **Interface**
  The UI/UX design adheres to the Gestalt principles, which was something we tried to meet to cater to the needs of individuals with intellectual disabilities.

- **Functions**
  We implemented secure authentication with distinct access levels for support workers and admins. Although the system doesn't store much personal information, this security ensures that any stored data is protected according to the user's access level, particularly concerning add/edit/delete/archive functionalities.

- **Goal**
  We were successfully able to meet our goal of creating a fully functional prototype

Weaknesses

- **Not in the application**
  A weakness for the team was definitely anything we weren't able to implement (*as noted in future work*), as well as coming up with different functionalities down the line and not having enough time to implement

- **Road Block**
  A bit of a road block for us, was our non-experience and the amount of time it took for us to get started on the project, even though we were aware of this, we probably focussed more on trying to work on one part together (which did take time), as well as trying to troubleshoot majority of the functions together.

- **UI/UX**
  Some of our testers, outside of the team, found the UI design a bit confusing and were unsure of how to navigate through the system. Additionally, one of our testers was unsure of what the purpose of the profile was - was it for the support worker or for the trainee

- **Bug**
  Lastly, a bug we found was that if the selected task is set as the same task for more than one trainee, the progress widget in support's dashboard does not properly show the latest selected task for both trainees. The latest task in the dashboard also resets after a while but this is a minor bug and we aim to fix it as part of future enhancements.

Future Work

Features we were unable to implement

- **Evaluation Task Steps Table**
  We would like to further enhance the evaluation section to make this as similar as possible to the system that the cafe has in place at the moment. This system includes a feature where the grading system is marked against every task analysis section, rather than the task as a whole.
- **Notifications**
  We wanted to implement a function that would pop up as a notification whenever a task/recipe/trainee was added to the system or if anything was deleted/archived from the system. It would be accessed through the notification bell that would be at the top of the screen.
- **Alternate Email**
  We wanted to allow the users to use a different email or phone number, anything that might be associated with the AWS Cognito account, to get a verification code to change the password, however this is something that can be implemented in the future.

Future Enhancements

- **Logging**
  If any changes are made to the system e.g. adding/editing/deleting/archiving, it would be handy for the app to have a history log, showing exactly what the changes were, in the case that something needed to be reviewed. If the user requests a log report, this could be sent directly to an email connected to the account.
- **Exporting**
  Based on the trainee's sessions and their progress, if the cafe would like to download their work in either pdf or excel format, they would have this option. Another option could be to allow them to view the sessions in a monthly format to help the support workers oversee the training in a different and more consolidated way.
- **Distribution of the application**
  During the duration of the project there were conversations about how this application could be used in different countries. It would be beneficial for the cafe, if we could implement something that ensured that the different cafes could have different applications with no information from the separate cafes, for example, they would start with a whole new application and build that from the ground up.
- **Customisation**
  We could include a customisation feature where If the clients would like to change the theme, or customize the application specific to the trainee , then it would be saved within the specific trainee's profile.

- **Extra Aesthetics**
    - Distinction in background for support and admin
    - An information button for new users who are unsure of the functionalities
    - Auto correct function
    - Time out the session if exceeding a long period of time or a message pop up asking if a session continuance is required - determine whether there needs to be a session limit (something the cafe can configure)
    - To maximise screen space, we can move the navigation bar at the bottom and create a burger menu for this
    - In the event that the app crashes, the application should retain the information, like a cache memory stored locally on the iPad which couldl be loaded once the app restarts
    - Should be more obvious which trainee is being evaluated or trained - with the name at the top of the screen

Conclusion

The primary aim of this project was to consolidate all of the work done by Te Tui Café by developing a functional and intuitive application that is Inkā. This app aimed to enhance and streamline all of Te Tuhi Café's initiatives, which ultimately would lead to a platform that helped foster a personalized learning experience for the trainees and a way to track their progress in the work they did.

The key findings as mentioned above, highlight that we indeed managed to have a successful creation of our mobile application Inkā. We were able to consolidate the majority of the paperwork and dropbox work that they have currently to give them an application where they can have everything in one place. This, we believe, can only enhance the training of individuals at the café and assist in a personalized training plan. In specific, Inkā was able to address the need for storage, by implementing a repository (being able to create, store and delete tasks/recipes/trainees), which also can keep track of the progress and any notes that the support workers may have for the trainees.

Some outcomes from this project that could be considered quite significant is that Inkā has a personalized learning experience tailored to individuals with intellectual disabilities and in this regard, Inkā stands out from other learning applications. Inkā also supports the learning of individuals in a café environment, allowing the trainees to do things at their own pace. Additionally, the aim of the application is to ensure that intellectually disabled individuals have the same opportunity to work in hospitality as any other individual and we believe that Inkā will assist in this, as this is another way for the trainees to learn and gather skills. Finally, we believe that Inkā makes the training process in general more efficient and easier for both support workers and trainees.

Overall, as a team we believe that the development of Inkā will make an improvement in the efficiency and training processes that are already in place at Te Tuhi Café, as was our goal. Our goal to also make an impact in this community can be shown through the final prototype of this application and we are pleased that we were able to make a contribution in this community and hopefully in the learning journey for the trainees who will utilise Inkā.

References

(1) *Labour market statistics (disability): June 2023 quarter | Stats NZ*. (n.d.). https://www.stats.govt.nz/information-releases/labour-market-statistics-disability-june-2023-quarter/#:~:text=labour%20force%20participation%20rate%20for,a%2042.3%20percentage%20point%20gap)

(2) Stats NZ. (2023) . Labour market Activity [infographic]. NZ. https://www.stats.govt.nz/information-releases/labour-market-statistics-disability-june-2023-quarter/#:~:text=labour%20force%20participation%20rate%20for,a%2042.3%20percentage%20point%20gap

Declaration of Authorship

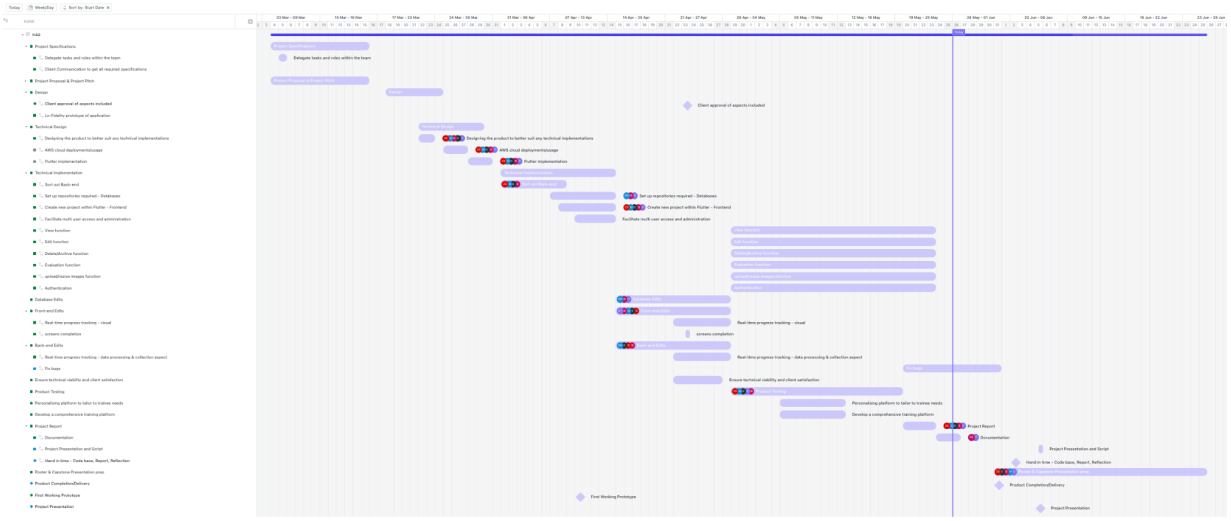
Section	Author(s)
Title Page	Saacha Moyo
Executive Summary	Saacha Moyo
Introduction	Saacha Moyo
Background	Zainab Shehzad
Project Specification	Everyone
Project Design	Everyone
Project Implementation	Everyone
Results & Evaluation	Natania Thomas, Zainab Shehzad, Eisen Belleza
Future Work	Natania Thomas
Conclusion	Natania Thomas
References	Everyone

Appendices

A. Gantt Chart

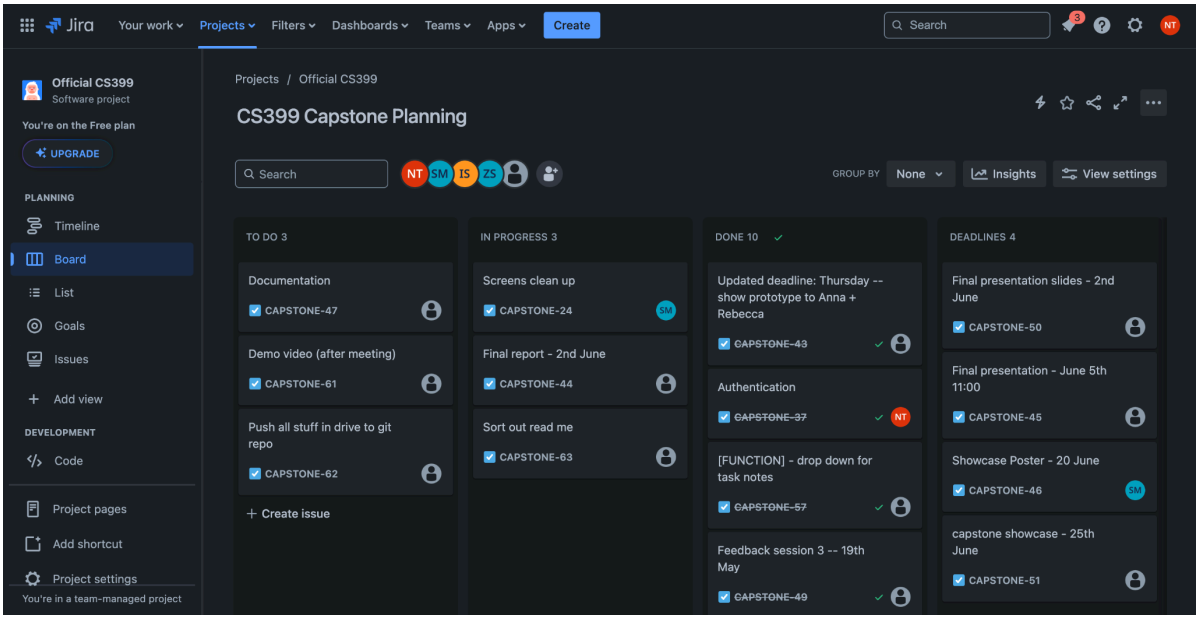
Link to Gantt Chart:

https://app.clickup.com/9016213572/v/li/901601710023



A.1 Jira

Link to Jira board:
https://capstone-cs-399.atlassian.net/jira/software/projects/CAPSTONE/boards/1?atlOrigin=eyJpIjoiMDM0ZDMyZmQ3OTZiNGMzODg4YmJlODAyNmU3MGI3ZjMiLCJwIjoiaiJ9



B. Inka Demo Video

Link to demo:

https://youtu.be/zaYUskN_kiw

C. Inka Test Case Documentation

Link to test case documentation:
https://docs.google.com/document/d/1XAhk4qNlOTb1GPFFwL1Eew_QfBtGGWhFafHnu6uACUM/edit?usp=sharing

D.1 UML user flow diagrams

Link to UML user flow diagrams:
https://lucid.app/lucidchart/b0e42189-1495-4b20-8379-eab9d24852c4/edit?viewport_loc=407%2C4185%2C2682%2C1529%2C0_0&invitationId=inv_6ea54e62-2d22-49d1-80ee-9c6a3fba6f23

D.2 UML user case diagram

Link to UML user case diagram:

https://lucid.app/lucidchart/16b1e8ad-df26-42a1-89c6-9c6036c10deb/edit?viewport_loc=-488%2C-923%2C3981%2C2270%2C.Q4MUjXso07N&invitationId=inv_f76ba200-5d5e-48fa-b304-f6b83a1cbc68

E. Final ERD diagram

Link to final ERD diagram:
https://lucid.app/lucidchart/e56863e2-e3a3-4485-98e2-dd06a0832619/edit?invitationId=inv_713c0488-cf6f-4700-9ba9-fb2a106e0e2f&page=0_0#

F. AWS Setup Documentation :

https://docs.google.com/document/d/17ZWbsLJndr8A0jwiOuvkP9uwAZ9P8AkxaGnu-Ex6398/edit

G. Link to Final Build:

https://github.com/uoa-compsci399-s1-2024/capstone-project-2024-s1-team-13.git