**Chitter challenge – Detailed Plan**

# User Stories

## Part I

1. POST messages (with both username and real name)

As a trainee software engineer

So that I can let people know what I am doing

I want to post a message (peep) to chitter

1. View all messages in reverse chronological order

As a trainee

So that I can see what others are saying

I want to see all peeps in reverse chronological order

1. POST messages have a Date

As a trainee

So that I can better appreciate the context of a peep

I want to see the time at which it was made

## Part II

1. Registration

As a trainee

So that I can post messages on Chitter as me

I want to sign up for Chitter

1. Registration 2

As a trainee

So that I can post messages on Chitter as me

I must register by providing 4 required fields:

1. Email
2. Password
3. Name
4. Username
5. Log In

As a trainee

So that only I can post messages on Chitter as me

I want to log in to Chitter

1. Log Out

As a trainee

So that I can avoid others posting messages on Chitter as me

I want to log out of Chitter

## Part III – Additional:

1. Uniqueness

As a trainee

So that I can identify different users

Each user must have a unique username and email

1. Visibility

No registration or log in is required to view peeps.

## Part IV - Extended Acceptance Criteria

1. Tagging others in peeps

As a trainee

So that I can stay constantly tapped in to the shouty box of Chitter

I want to receive an email if I am tagged in a Peep

1. Replying to others in peeps

As a trainee

In order to start a conversation as a DFA trainee Software Engineer

I want to reply to a peep from another trainee.

(All criteria met. Just part IV left to do.)

# Detailed Implementation Notes:

## AllPeeps.jsx – formatPeeps()

*// sort in descending order of date*

*// then generate an array of Card components*

*// with the correct Name, Username, Date and Message*

*// and key set to the mongoID, namely, \_id*

## addPeep.js route

*// sort in descending order of date*

*// then generate an array of Card components*

*// with the correct Name, Username, Date and Message*

*// and key set to the mongoID, namely, \_id*

## allPeeps.js route

*// get request to retrieve all stored peeps in database*

*// Use Peep model to query database's peeps collection for all peeps*

*// Then send back the peeps as a json object*

## Login.js route

*// POST request containing user login info*

*// Validate data using express-router*

*// Use mongoose User model to query database for a matching email*

*// If a match exists, check the password*

*// If the password matches, return status 200*

*// and {"name": req.body.name, "username": req.body.username}*

*// If email is not found, return a json with "email not found"*

*// If password doesn't match, return a json saying so*

*// React should navigate to `/` once a res object is received*

## Register.js route

*// POST request received containing user registration info*

*// Validate user info using express-validator*

*// Use Mongoose User model to instantiate a document with the received info*

*// Check database to ensure the req.body.username and req.body.email are unique*

*// If so, save a new User document to the database users collection*

*// Then return a res object with status 200 and a message telling the user that they registered successfully*

*// If any errors occur, return status 400 and an object with an error message*

# Process

Planning stage

* Organised the user stories into related groups
* I then designed the User and Peep Schemas
* Then I planned out the project by planning the different routes and react components
* Listed all the packages that I would use

Project setup

* Initialised a npm project
* Installed dependencies
* Setup .gitignore files
* Setup mongoDB and Compass
* Setup dotenv and different environments for Node server

Implementation

* I first began with the core user stories: viewing all peeps in reverse order; adding a peep; adding dates to peeps
* Then I worked on the login/registration related user stories
* Finally, I attempted the extended acceptance criteria

Individual User Stories

* For each user story, I worked on the routes first and then the UI and finally the css styling
* I followed TDD strictly when working on the routes.
* But I was a little more flexible when it came to react components

# Reflections

* Sometimes need to click a button twice for state to be set – seems to be a problem with state setting functions being asynchronous?
* How do we test whether a function defined within a react function component is called?

We can’t seem to be able to mock it using jest.mock() or jest.fn() due to such functions being within the scope of the react function component

# Further – bcryptjs and jwt

Added bcryptjs.

Added jsonwebtoken, a secret, and middleware for verifying the jwt.

Fixed tests for routes so chai http requests also send a {headers: {x-access-token: jwt}} alongside a request body.

The use of local storage means that there is no longer any need to set the user manually when they login. Updated tests to take this into account.

Fixed react app to account for the addition of these two packages.