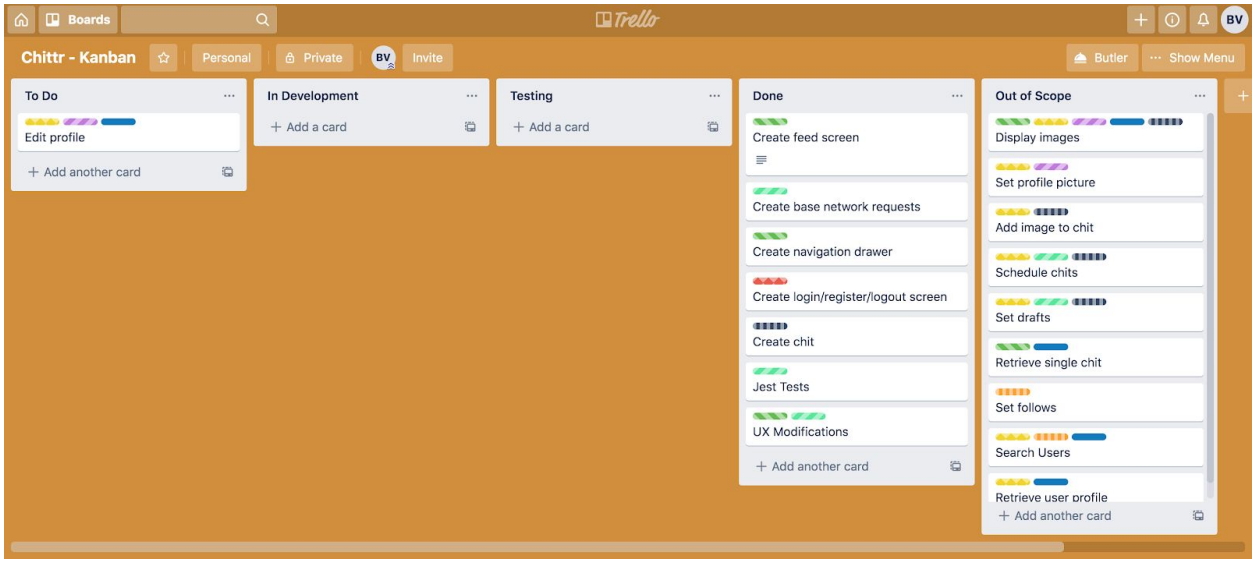
Mobile Applications Development (6G6Z1104_1920_9Z6)

Assignment 2

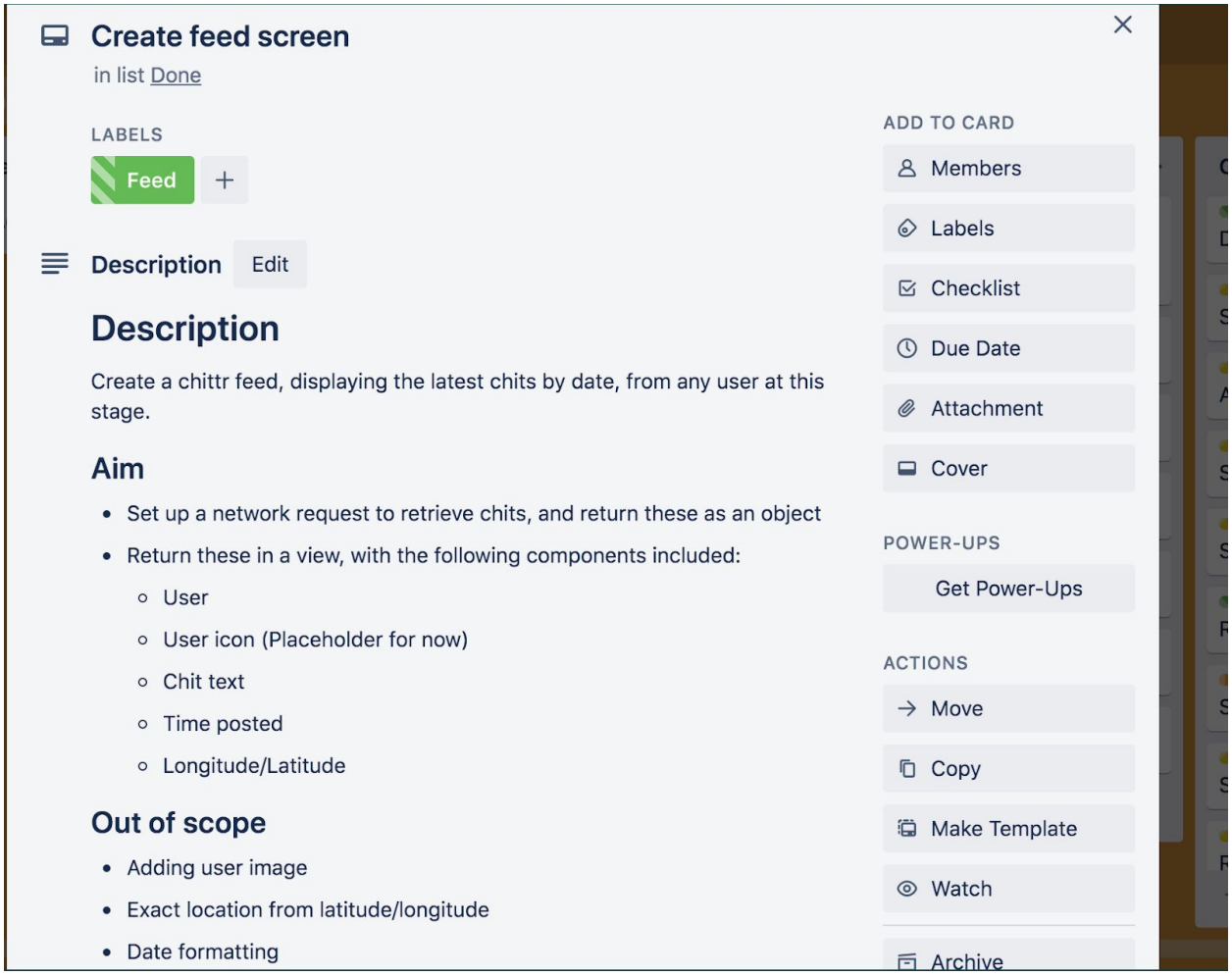
Management

In order to do this assignment, I chose to use Kanban and set up a trello board. In Kanban, tickets are created for each task and brought in when they are ready to do. Each one goes through several stages in order to check that it is functional, before being marked as done and closed off.

Trello board:



I chose to use Trello for this, and dragged each component across as it was merged. The more time went on with the assignment, the more components I moved to mark as out of scope for my current timeframes, in order to visualise and get done as much as I saw possible.

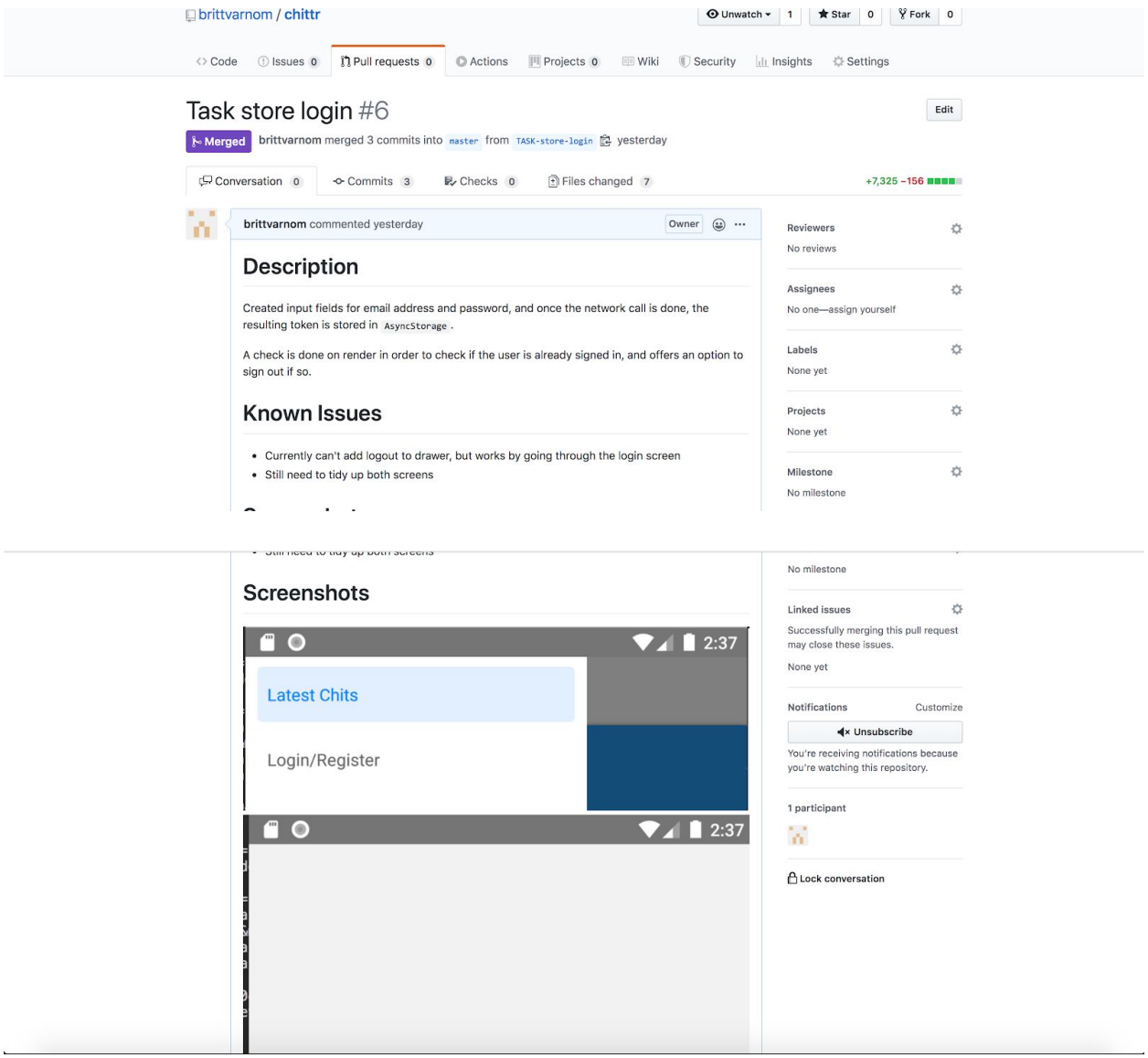


I laid out tickets with a clear structure - a overall description of what the ticket aims to do, a breakdown of the aim, and any features of the component that may be out of scope of the current ticket, to be done later.

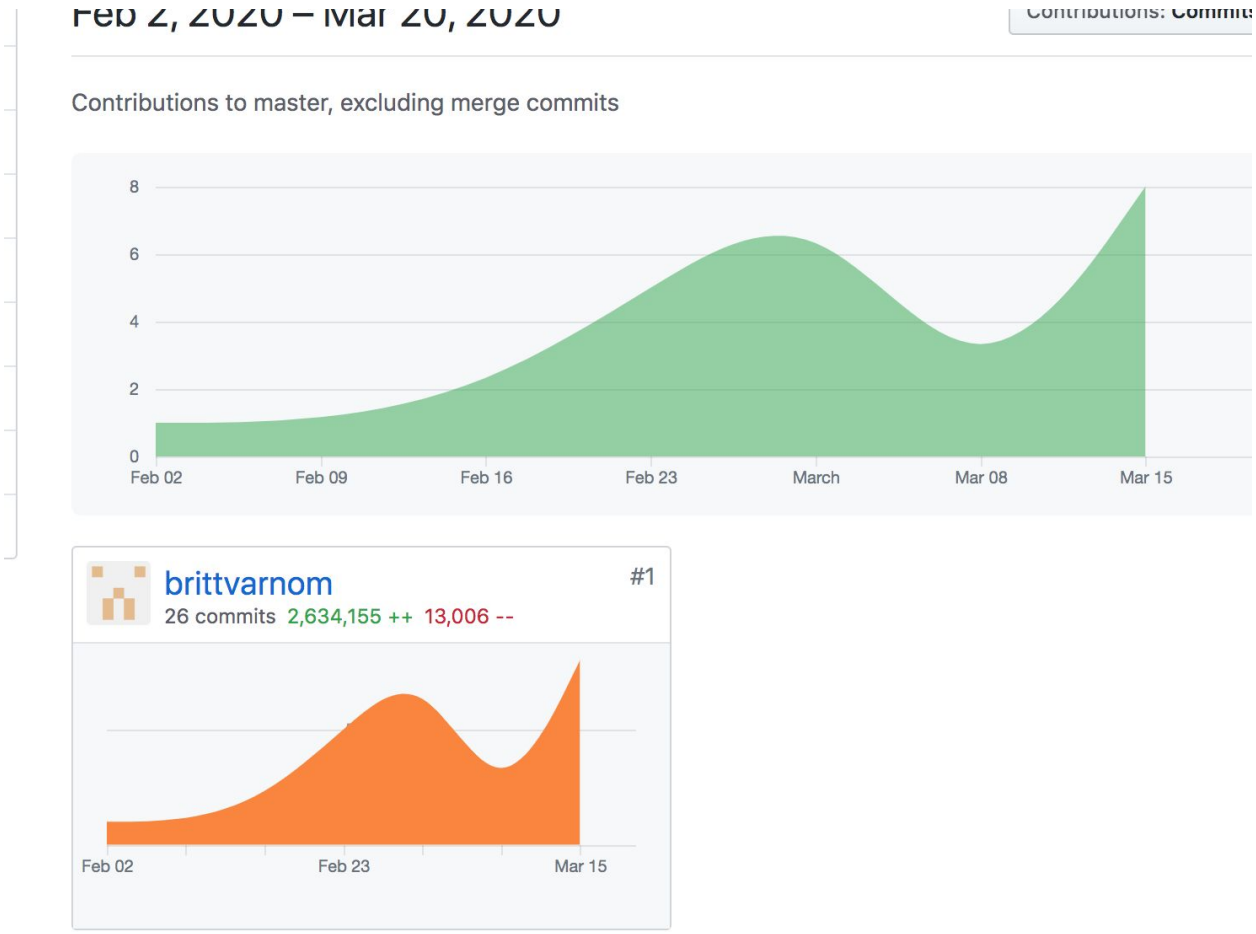
Version Control

For version control, I used GitHub, and set up my project as a repository, which can be found here: https://github.com/brittvarnom/chittr/

I set up new branches for each task I undertook, and created new pull requests for each to document what had been done, why, and how it looked at the time.



By doing this, it becomes more clear why a pull request was done as reference in the future, especially if an issue later pops up that may have been caused by this pull request.

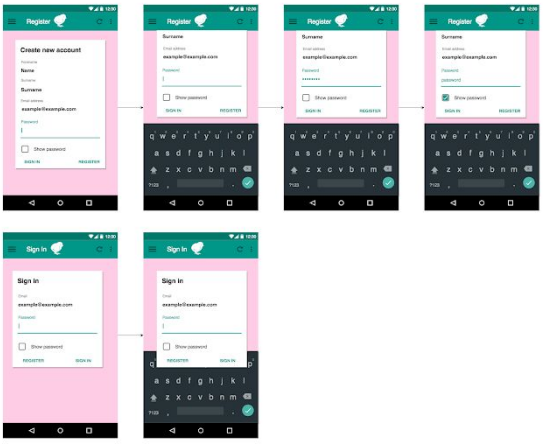


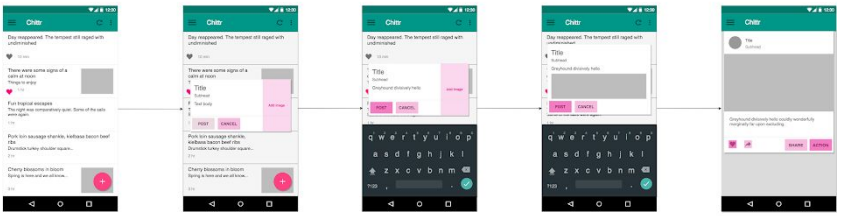
I committed regularly as much as I can, but was delayed by other work and uni projects. A steady increase in commits happened as I started to improve the feed I had created, slowing down when responsibilities came up at work, and increased again towards the deadline as extra components were added and bugs were found.

Wireframing

When I first started on the project, I had issues accessing the server and started on the wireframes while I waited to be able to get help resolving these. I started with a few basic screens that I had in mind, with some random colours that I tossed together, as follows:

Brittany Varnom 16062790 App Development - Assignment 2





This had a few planned user journeys, but the final results were limited by the limits of React Native, as well as my remaining time and the structure of my codebase.

Data Retrieval

Login/Logout

When a user clicks the login button, a function calls the API, and if it succeeds, the resulting token is stored in the state and AsyncStorage for use later. The token is accessed in order to verify if a user is already logged in (to avoid prompting them to log in/register again, as well as to allow the posting of new chits.

In the future, the user id should also be stored in order to be passed around, allowing the retrieval of the logged in user's profile with editing functionality, follows, and following lists.

```
async postUserLogin(email, password) {
    const data = JSON.stringify({
        email: email,
        password: password
    });

    try {
        const response = await fetch('http://10.0.2.2:3333/api/v0.0.5/login', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: data
        });
        const responseJson = await response.json();
        //sets login token and current user id
        this.setState({ user_id: responseJson.id, user_token: responseJson.token });
        console.log('>>> LOGIN SUCCESS', `Logged in! User id: ${this.state.user_id}, Token: ${this.state.user_token}`);
        this.setValueLocally();
        this.props.navigation.goBack();
    }
    catch (error) {
        this.setState({ loginMessage: 'Failed to log in, wrong email/password' });
        console.log('>>> LOGIN FAILED', `Failed to login - wrong email/password. ${error}`);
    }
}
```

```
setValueLocally = () => {
    AsyncStorage.setItem('@LOGIN_TOKEN', this.state.user_token);
}

getValueLocally = () => {
    AsyncStorage.getItem('@LOGIN_TOKEN').then((value) => this.setState({ user_token: value }));
}

deleteValueLocally = () => {
    try {
        AsyncStorage.removeItem('@LOGIN_TOKEN');
    }
    catch (exception) {
        return;
    }
}
```

setValueLocally is called inside of the login function, and sets the token using the state. The same technique is used to retrieve the token, as well as delete it from state when a user has logged out.

The register button calls the register function, which takes in the value of each editText on the page as a prop and sends this as data to the server. If this succeeds, the user is navigated back to the home screen, and the account is created. They are not auto-logged in, however, and should then proceed to login from the navigation drawer.

```
postUserRegister(gName, fName, email, password) {
    const data = JSON.stringify({
        given_name: gName,
        family_name: fName,
        email: email,
        password: password
    });

    return fetch('http://10.0.2.2:3333/api/v0.0.5/user', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: data
    })
        .then((response) => response.json())
        .then((responseJson) => {
            //sets login token and current user id
            console.log('>>> REGISTER SUCCESS', `Account created! Account id: ${responseJson.id}`);
            this.props.navigation.goBack();
        })
        .catch((error) => {
            this.setState({ loginMessage: 'Failed to register, fill all fields' });
            console.log('>>> REGISTER FAILED: ', error);
        })
}
```

The logout function works similarly to the login, making the logout API call, but also deletes the token from storage so that it can't attempt to continue using it.

Feed

My feed is retrieved as the initial screen in App.js, and renders a simple flatlist containing a Chit object. The chit is passed JSON data containing recent chits, and these are accessed as props inside the component.

```
render() {
    if (this.state.isLoading) {
        return (
            <View>
                <ActivityIndicator />
            </View>
        )
    }

    return (
        <View style={{ flexDirection: 'row', backgroundColor: 'chocolate' }}>
            {/* List rendering all chits */}
            <FlatList
                data={this.state.responseJ}
                // Map each item
                renderItem={({ item }) =>
                    <View>
                        {/* Individual chit component */}
                        <Chit item={item} />
                    </View>}
                keyExtractor={({ id }, index) => id} />
        </View >
    )
}
```

Render function of Feed.js.

```
render() {
    const url = `http://localhost:3333/api/v0.0.5/user/${this.props.item.user.user_id}/photo`;
    return (
        //Chit
        <View style={{ backgroundColor: 'white', marginBottom: 8 }}>
            {/* User's info */}
            <View style={spacing, { flexDirection: 'row', backgroundColor: 'moccasin' }}>
                {/* Profile pic */}
                <Image
                    style={{ height: 50, width: 50 }}
                    source={url}
                />
                {/* Username */}
                <Text style={[fontSize.pica, spacing]} onPress={() => { Alert.alert("test"); }}>
                    {`${this.props.item.user.given_name} ${this.props.item.user.family_name}`}
                </Text>
            </View>
            {/* Chit */}
            <Text style={[fontSize.body, spacing]} onPress={() => { Alert.alert("test"); }}>
                {this.props.item.chit_content}
            </Text>
            <View style={{ backgroundColor: '#f0f0f0' }}>
                {/* Timestamp */}
                <Text style={[spacing, fontSize.brevier]}>
                    {Date(this.props.item.timesamp)}
                </Text>
                {/* Location (If the chit has one) */}
                {this.props.item.location &&
                    <Text style={[spacing, fontSize.brevier,]}>
                        {this.getLatitudeAndLongitude(this.props.item.location)}
                    </Text>
                }
            </View>
        </View >);
}
```

Render function of Chit.js. The url takes in the user id of the user who posted the chit in order to retrieve their account image if one exists, and display this next to their name. Date and latitude/longitude are also included (Explanations of the layout of this file are included in my screencast).

Improvements

There were several components I was not able to implement - either at all, or to their full potential:

- Images on chits
- User profile images - Call is made on feed, but no images have been uploaded

- Image upload
- Follows functionality
- View individual chit
- View user profile
- Edit profile

These are all things I would like to have implemented, and would look out for in the future. The main constraint I had with this was the way I am passing data back - I've been unable to retrieve a user id in most areas of my app, and my resulting json across the feed and a user's data seems to come through differently in a way that won't support the same component retrieving the data in the same way.

Testing

I was unable to implement full testing functionality, but started writing out a simple jest test in order to check that a component was being rendered, stored in a describe block and using a beforeReach in order to set up a shallow render of the component correctly.

```
1   import React from 'react';
2   import { shallow } from 'enzyme';
3   import Feed from './Feed';
4
5   describe('OrderingPracticeContainer', () => {
6       describe('component', () => {
7           let wrapper;
8
9           beforeEach(() => {
10              wrapper = shallow(<Feed />);
11          });
12
13          it('should render the Drag and Drop context provider', () => {
14              const flatlist = wrapper.find('Flatlist');
15
16              expect(flatlist).toExist();
17          });
18      });
19  });
20
```

This, however, was not picked up by jest when I ran the tests through npm run test. I based this test off my knowledge of react.js rather than react native, so as an improvement I would choose to further investigate the differences between how testing works across both.

Logout

While I have logout functionality, logging out navigates to the login page, with a conditional render that will display a logout button if a user token exists, and a login form if not. If possible, my aim would've been to instead make it so that once the logout button is clicked in the navigation drawer, the user is logged out and the app refreshes, without having to log into another screen.

```
render() {
    if (this.state.isLoading) {
        return (
            <View>
                <ActivityIndicator />
            </View>
        )
    }

    if (this.state.user_token != "" && this.state.user_token != undefined) {
        console.log('DBUEGGGGG: ', this.state.user_token);
        return <View styles={spacing, { backgroundColor: 'moccasin' }}>
            <Text>Log out of your account:</Text>
            <Button color="chocolate" title='Log out' onPress={() => { this.postUserLogout(this.state.user_token) }} />
        </View>
    }

    return (
        <View style={spacing, { alignItems: 'center' }}>
            <TextInput
                autoCompleteType='email'
                onChangeText={(email) => { this.setState({ email }) }}
                keyboardType='email-address'
                placeholder='Email Address'
                style={spacing, { width: 300, backgroundColor: 'moccasin' }} />
            <TextInput
                autoCompleteType='password'
                onChangeText={(password) => { this.setState({ password }) }}
                placeholder='Password'
                secureTextEntry={true}
                style={spacing, { width: 300, backgroundColor: 'moccasin' }} />
            <Button title='Login' color="chocolate" style={{ margin: 8, width: 100 }} onPress={() => { this.postUserLogin(th
            <Text>{this.state.loginMessage}</Text>
        </View >
```