Ajax Lab

1. Edit Register.js. Note that there is a method in the class called fetchCityAndState().

This method is called every time the user blurs from the postcode field. We're going to use it to request the city and state from a service called Zippapotamus. It's pretty simple; you make a GET request to http://api.zippopotam.us/<country>/<postcode> and it'll return a JSON object with the city and state among other data points.

2. Put a line of code inside fetchCityAndState. Something like this will do the trick:

```
fetchCityAndState(e) {
  store.dispatch(actions.fetchCityAndState(e.target.value));
}
```

   Of course when you run this, it doesn't work because fetchCityAndState is not a member of the actions object. Let's fix that.

Adding the action creator and the action type

3. Add your new action creator to actions.js. The action creator should receive a single parameter -- the postcode. It's return action object should look like this: { type: FETCH_CITY_AND_STATE, postcode: <whatever the user put in the postcode field>}.
4. You may have already guessed that the FETCH_CITY_AND_STATE should be added to our list of action types. Go ahead and do that now. (hint: look in action-types.js to add it.)
5. Run and test. Your web app should now run without error, but blurring from the postcode field doesn't do anything yet. Why not? _____

Writing the middleware function

If you said above that there is not reducer that handles a FETCH_CITY_AND_STATE action type, you were right. But if you thought we should add one, you are off base. Why? Because fetching is an asynchronous operation so it can't be done in a reducer! If you want asynchronous activities to happen, they must be done in middleware.

6. Write a middleware function called fetchCityAndStateMiddleware. It should do only one thing for now: if the action.type is FETCH_CITY_AND_STATE, then console.log() that it was called.
7. You'll have to register the middleware with the store. The registering should be done in store.js by passing fetchCityAndStateMiddleware to the applyMiddleware function kind of like this:

```
import { applyMiddleware, createStore } from 'redux';
// Then later ...
const middlewares = applyMiddleware(fetchCityAndStateMiddleware);
createStore(reducer, initialState, middlewares);
```

8. Run and test. Every time you dispatch any action -- including blurring from the postcode field -- you'll be running your middleware function and should see your console.log().

Some troubleshooting tips ...

- Did all of your SET_* dispatches suddenly stop working? Don't forget to call the *next* function.
- Did state suddenly become undefined? Don't forget to pass the action into *next* when you call it.

Making the API call

9. Inside your fetchCityAndStateMiddleware() function, create the proper URL. Something like this should do it:

```
const url = `http://api.zippopotam.us/us/${action.postcode}`;
```

10. Use whatever method you choose to make an Ajax call to that url. *fetch* is a good choice (http://bit.ly/fetchAPI). Whichever method you decide, you'll register a callback function. In that callback, you're going to want to convert the API's response string to a JSON object and then extract the city and the state from it.
11. Still in the callback, do this:

```
console.log(city, state); // This proves you got the city and state back okay
dispatch(actions.setCity(city));    // setCity and setState have already been
dispatch(actions.setState(state));  // written by you.
```

12. Run and test. Look in the console. You should see the correct city and state being logged. And if that works alright, you should also see a proper city and state being set in the UI.

Once you can set a valid postcode and see the city and state populate accurately, you can be finished.