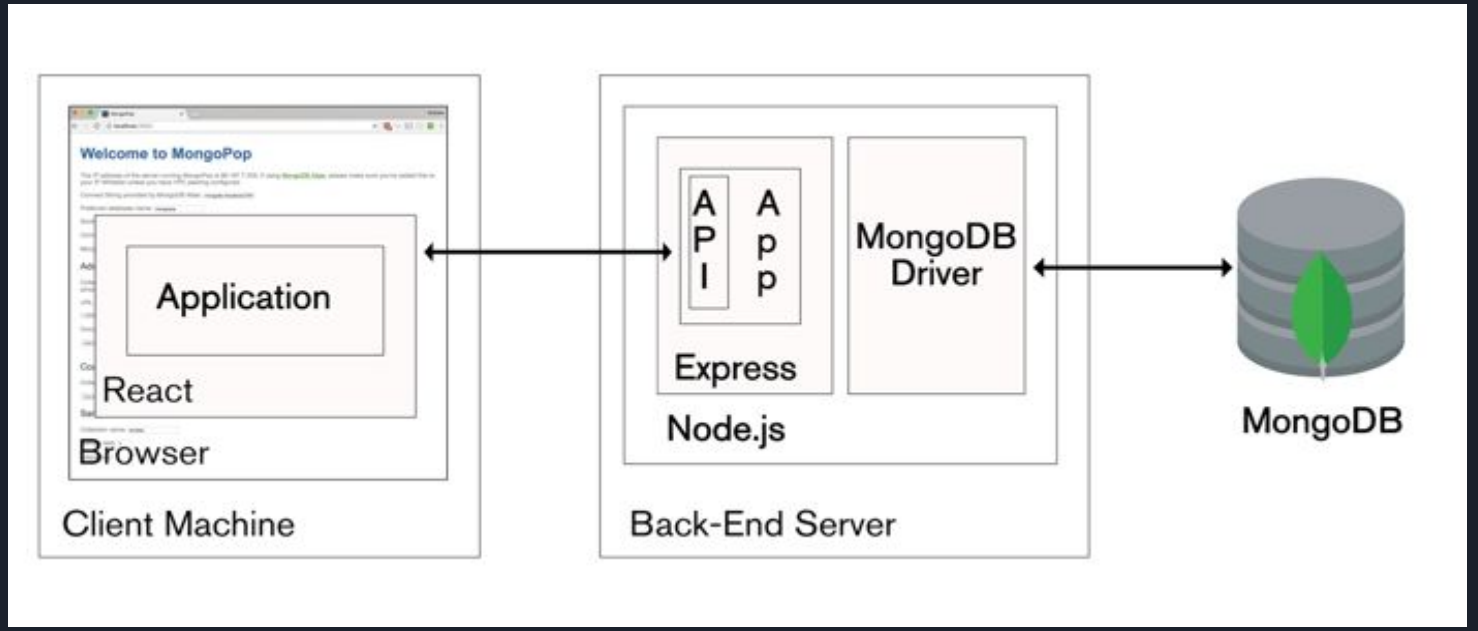
# Psychiatric Evaluation Web App

**Contributors: George Engel, Troy Oster, & Henry Soule**

# System Architecture:



# System Design:

## Patterns used:

- **Model View Controller, as it is natively implemented in react by way of it's component based workflow.**
- **Adapter Pattern**, this is used in our Express server. It acts as the adapter between the react front end and the MongoDB backend
- **Abstract Factory,** creates several instances of our components (classes) that are abstract
- **Abstraction**, all of our components are abstracted
- **Composite**, again from our component based architecture
- **State**, object/component behavior is changed when the state changes

# Design Choices: Why/How

**Maintainability & Extensibility:**

We used **react** for front end development due to the component based nature of it as a framework. Due to the innate modularity of the design principles associated with react, we determined early on that it would be our best choice for the sake of ***long term maintainability*** and ***extensibility***.

# Quality Assurance Methods:

**Travis:** Deloys the project after each commit and push & runs our Jest test cases to ensure nothing breaks from build to build.

**Jest testing:** Provides coverage of X% of the code base to ensure the code compiles & works as intended.

**Selenium Tests:** Emulates a user doing a certain task with various inputs.

**StoryBook:** Individual component testing & debugging

**User testing reports:** Non-CS related users to simulate product usage.

# Testing: Jest & Enzyme

Jest - standard test runner for React, a form of white box testing as we are targeting as much of the code base as possible for coverage.

Enzyme - Testing utility for React.

Enzyme utilizes ‘shallow rendering’ for component testing to ensure child components are not rendered for unit tests.

For integration testing, Enzyme provides a jQuery-like DOM-traversal API to mount components in memory and test their interactions. Utilizes ‘mock’ functions to test interactions with the Database.

Configurable with Travis CI

~80% code coverage

# StoryBook - In Depth:

**StoryBook** is an open source tool for developing, debugging, and testing UI components in isolation for React, Angular, & Vue.

It is arguably a black box styled testing suite, as it isolates the component in a sandbox for testing hard to reach edge cases, and merely provides input from the developer interacting with the isolated component. When it comes to debugging it simply tells you whether or not the component breaks when you try to interact with it in a certain way.

Essentially it only informs you of inputs that yield malformed/incorrect output, or when inputs break things.

# User Testing Reports:

**Who we had test:**

- Harold Oster; Infectious disease specialist/practitioner (Doctor)
- Karen Solberg; Student (Math Major Junior Year)

**What we had them do:**

- Harold: Access the website through a practitioner account & evaluate a patient's progress & diagnosing the appropriate methods of treatment.
- Karen: Sign up as a patient & take an evaluation test

# User Testing Reports: (continued)

**Results:**

- Both took some time to find the login page
- Karen successfully took the evaluation tests after registering & signing in, however the results reported where not correct.
- Harold added patients just fine, and accurately decide on the next steps in the patient's treatment plan, though commented on how bad the treatment plan page looked.

**What we Learned:**

- The main page could use some reworking
- The Depression treatment plan page needed major revamping
- Evaluation tests where apparently broken

# Special Features:

- Scalable to fit any screen size thanks to React Grids
- Easy to reuse Components for future development
- Searchable tables
- Dynamically rendered forms that can be made from how objects are defined in the database for ease of scalability