Lecture 12: Deploying, Testing

Jordan Hayashi

Previous Lecture

- Performance Trade-Offs
- React Native Perf Monitor
- Chrome Performance Profiler
- Common Inefficiencies
  - Rerendering too often
  - Unnecessarily changing props
  - Unnecessary logic
- `Animated`

Deploying

- Deploy to the appropriate store by building the app and uploading it to the store
- Set the correct metadata in app.json
  - https://docs.expo.io/versions/latest/workflow/configuration
- Build the app using exp (command-line alternative to the XDE)
  - Install with `npm install --global exp`
  - Build with `exp build:ios` or `exp build:android`
  - Expo will upload the build to s3
  - Run `exp build:status` and paste the url in a browser to download

Deploying, cont.

- Upload to the appropriate store
  - https://docs.expo.io/versions/latest/distribution/building-standalone-apps
  - https://docs.expo.io/versions/latest/distribution/app-stores
- Deploy new JS by republishing from the XDE or exp
  - Rebuild app and resubmit to store to change app metadata

Testing

- The term “testing” generally refers to automated testing
- As an application grows in size, manual testing gets more difficult
    - More complexity means more points of failure
- Adding a test suite ensures that you catch bugs before they get shipped
- How do we know which parts of our app to test?

Test Pyramid

- Methodology for determining test scale/granularity
- Unit tests
  - Test an individual unit of code (function/class/method)
- Integration/Service tests
  - Test the integration of multiple pieces of code working together, independent of the UI
- UI/End-to-end tests
  - Test a feature thoroughly including the UI, network calls, etc.

Unit Tests

- Test an individual unit of code (function/class/method)
- Very granular and easy to tell what is breaking
- The most basic test is a function that notifies you when behavior is unexpected
- Testing frameworks give you additional benefits
    - Run all tests instead of failing on first error
    - Pretty output
    - Automatically watch for changes
    - Mocked functions

Jest

- "Delightful JavaScript Testing"
- A testing framework by Facebook
- Install with `npm install --save-dev jest`
- Run with `npx jest` or by adding script to `package.json`
    - Will automatically find and run any files that end in `.test.js` (or any other regex expression that you specify)

Jest: Testing Redux Actions

- We can replace our functions with Jest's `expect()`, `toBe()`, and `toEqual()`
- We can use snapshots to compare the output of a function to the previous output of the function
  - We get notified if the output changes
  - We don't need to rewrite tests if the change was intended
- Which should we use?
  - Use `toBe()` for primitives
  - Use `toEqual()` for objects that shouldn't change
  - Use snapshots for objects that may change

Jest: Testing Async Redux Actions

- Async functions add multiple difficulties
    - We have to wait for the result before testing against the result
    - Our tests may rely on imported libs
    - Our tests may rely on external services
- If we return a `Promise`, Jest will wait for it to resolve
    - Jest also supports `async/await`
- Jest supports mocking functions
- Dependency injection
    - Pass functions on which other functions rely as arguments
    - Allows us to mock functions that rely on external services

Integration Tests

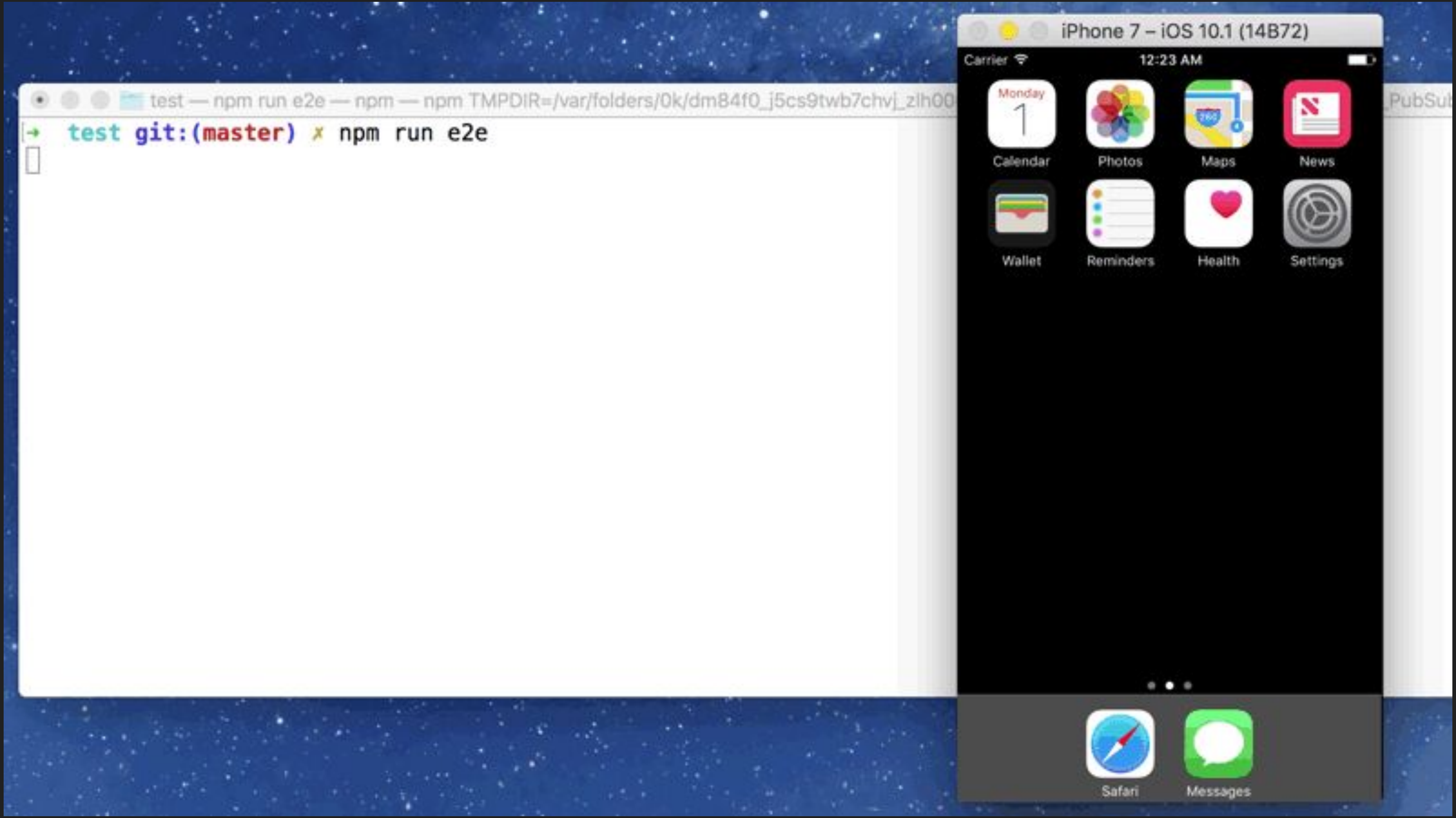
- We can use Jest's snapshot testing to test components
- `react-test-renderer` allows us to render components outside the context of an app
  - https://reactjs.org/docs/test-renderer.html
- jest-expo has all the configuration you need
  - https://github.com/expo/jest-expo

Code Coverage

- Metric for tracking how well-tested an application is
  - Statements: How many statements in the program have been executed?
  - Branches: How many of the possible code paths have been executed?
  - Functions: How many of the defined function been executed?
  - Lines: How many of the lines have been executed?
- Get coverage report by passing `--coverage` to `jest`

End-To-End Tests

- There is currently no easy way to run automated e2e tests in react native
- There is an awesome work-in-progress by Wix called Detox
  - https://github.com/wix/detox
  - https://github.com/expo/with-detox-tests
  - Lacks Android support



test — npm run e2e — npm — npm TMPDIR=/var/folders/0k/dm84f0_j5cs9twb7chvj_zlh00

```
➜  test git:(master) ✗ npm run e2e
```

Thanks!