Mini-MELD fest
08–Feb–2021

MELD - learning and testing

Activities undertaken as part of SSI3 sustainability for DH software

SSI3 MELD activity (from 2020-11 MELD-fest)

(An exemplar of) Best practice for testing in DH software development

- Realised as a testing framework for MELD
- Building upon command line MELD client developed in the later stages of FAST
- Extending this as a unit testing approach for MELD app development
- Testing framework could be validated against existing MELD apps
- A reflective case study of software sustainability in DH projects. While the MELD framework itself benefited from the substantial research efforts of FAST, additional apps have been developed with more limited resourcing of humanities projects.
- Outputs:
  - The software testing framework for MELD (public repo).
  - Unit tests for (some/all of) the above studies/apps (public repo).
  - Documentation and a best practice report. (public, online).

Changes from original plan

- The original plan was to focus on the network HTTP interfaces used by MELD applications. This would allow tests to be written separately, and run as stand-alone programs interacting with a running MELD application service.
- Early examination of the MELD libraries highlighted that much of the support logic was provided by MELD library calls from code running in the browser, and there would be limited testing access possible using just HTTP. So we started down the route of using React testing frameworks (initially Mocha, then Jest).
  - Using these testing frameworks would have an added advantage of being easier to deploy in a continuous integration environment, by not needing to be run against a separate service.

What's happened so far (1)

- Create (and document) working environment(s) for running simple MELD apps
  - See oerc-music/meld-hello-meld
- Examine existing demo app (`trompamusic/selectable-score-demo`)
  - Many "moving parts" for an introductory app
  - MELD built using React and Redux, which take some learning
- Decided to strip back to something *really* simple: "Hello MELD"
  - `hello-meld-1`: a very simple React app, not using any MELD elements, with tests
  - `hello-meld-2`: audio player and score rendering using MELD
  - `hello-meld-3`: like `hello-meld-2`, but using MELD graph retrieval for configuration
  - `hello-meld-4`: (planned) incorporate `selectable-score` user-interation

What's happened so far (2)

- MELD application tests based on "Hello MELD" apps
  - `hello-meld-1:` tests OK, using Jest (a widely used testing framework for React)
  - `hello-meld-2`: tests written, but not working using Jest
  - `hello-meld-3`: no testing yet
  - `hello-meld-4`: no implementation yet
- I've run into a range of problems getting the environment and tests running.
  - some appear to be caused by API changes between library versions;
  - some by module loading differences between Node and test environments.

What have we learned so far? (1)

- Starting with very simple applications has been really helpful for coming to terms with the MELD environment (and its dependencies)
- Testing has proved to be surprisingly challenging
- The Node/React ecosystem is very dynamic, with many complex dependencies that are difficult to understand and keep track of
- The Node/React ecosystem is very fragile: a support library can easily get into a state hat requires a complete rebuild for an application to work again. This in turn leads to a lot of misleading and confusing errors.
- The Mocha/Jest testing frameworks do not exactly duplicate the NodeJS running environment: applications that run OK directly under NodeJS often fail when the exact same code is run in a test environment.

What have we learned so far? (2)

- Testing code written for a complex platform environment (in this case Node and React) has proved to be surprisingly challenging.
- Testing code that is intended to run run a browser environment is more complex than testing server side rendering code. Arguably, complex platform environments make this easier, or even possible, but only when the test environment accurately reflects the browser application environment.
- Retro-fitting testing to an existing codebase can be very challenging. I can't be certain, but I suspect that if even very rudimentary testing had been included when the code was originally developed, some of the present problems might have been avoided. This would at least ensure that the code is testable, and any impediments to testing would be identified early and should be easier to work around.

What have we learned so far? (3)

- A number of issues observed have been recorded, to be used as input to a report on sustainability to be produced for the SSI3 project:

  github.com/oerc-music/meld-testing-ssi3/blob/main/Notes/MELD-sustainability-issues-observed.md

-

What next?

- I still believe that automatic tests are fundamental to software sustainability
- Currently I'm unable to get automated tests running for MELD applications
- I've also very recently run into problems with Redux application state (store) access
- Options under consideration include:
  - Adding test logic to the "Hello MELD" apps, which would provide a degree of testing, though not something that would be easily run in Continuous Integration.
  - Setting up Continuous Integration for the "Hello MELD" apps. This might at least indicate that the software can be installed successfully into a clean environment.

These slides at:
@@@