FHIR Questionnaires and Validation

The Dream Team

Objectives

- Fetch and Render FHIR Questionnaires
- Collect and validate answers
- Send Response to FHIR

What’s the deal?

- It’s complicated!
  - Many different types
  - Much to validate
  - Creating “QuestionnaireResponses”

Tech Stack

Frontend:

- ReactJS
- Formik

Backend:

- NodeJS
- ExpressJS

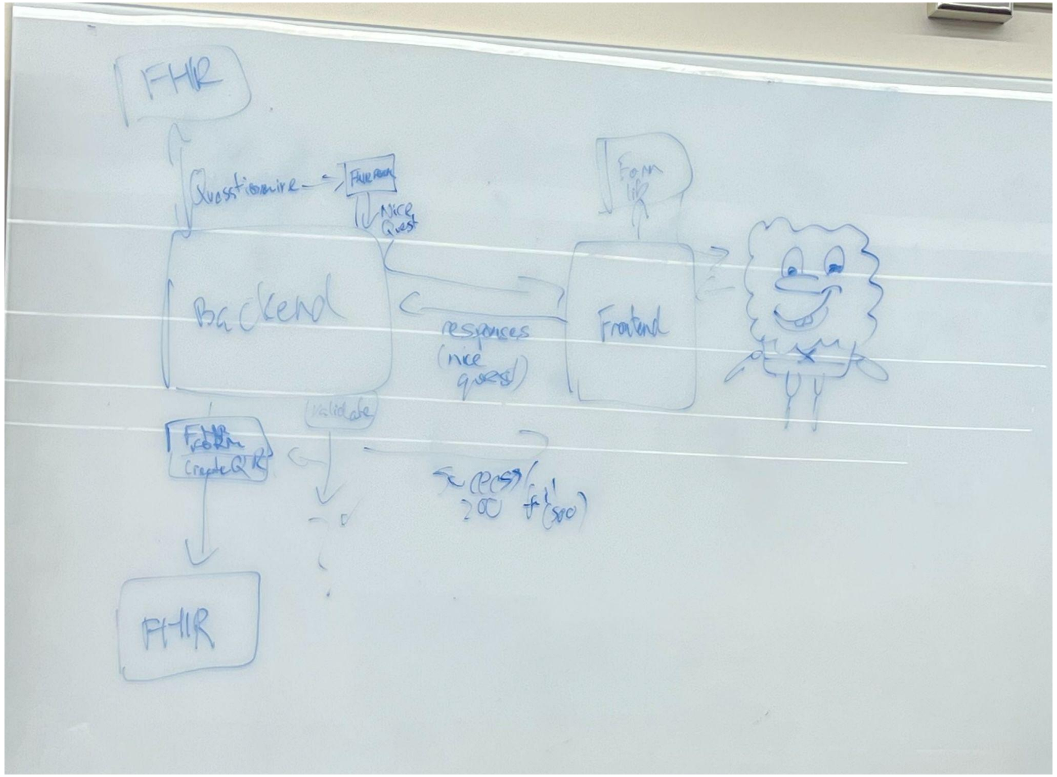
Other:

- Jest
- Docker

Why?

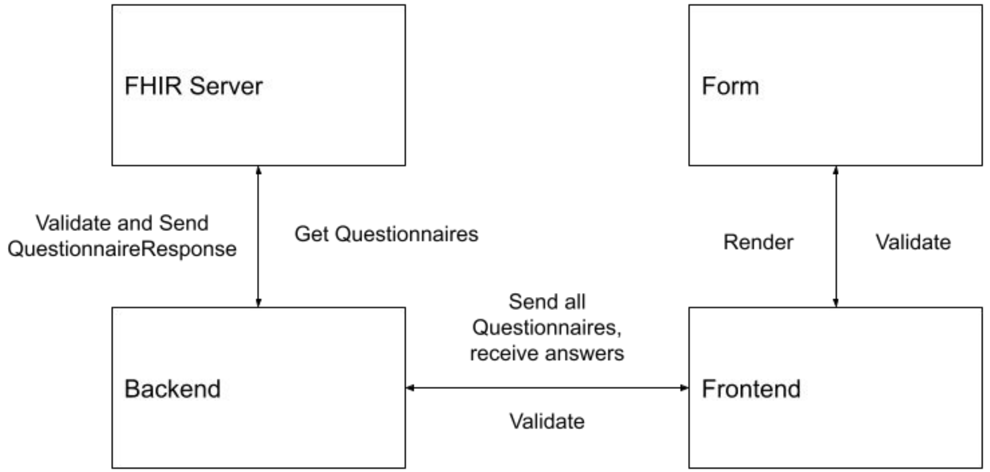
- Existing Team knowledge
- Popular
- Hardware agnostic
- Same stack as Parajuniper

System Architecture



A very early and *professional* drawing of our system and data flows

System Architecture


FHIR Server
Form
Validate and Send QuestionnaireResponse
Get Questionnaires
Render
Validate
Send all Questionnaires, receive answers
Backend
Frontend
Validate

An actual professional drawing of our system and data flows

Milestones

Milestone 1

- Setup Frontend/Backend App

-Investigate APIs and Documentation

-Questionnaire retrieval APIs

Milestone 2

-Render questionnaires in Formik

-Backend Validation

-Creating QuestionnaireResponse

Milestone 3

-Input Validation

-Frontend testing (Validation, Rendering Conditions)

-Backend tests (Validation, Requests)

Today

Delivered this very awesome presentation

Retrospective

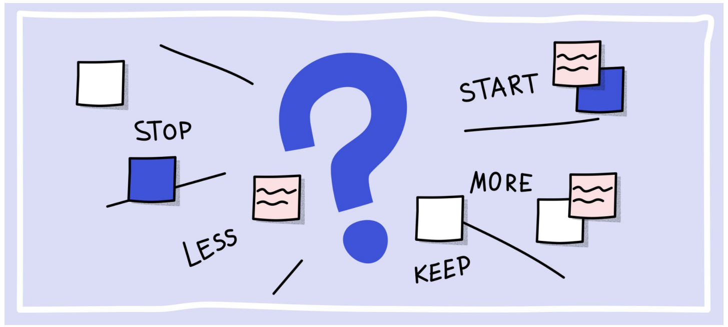
A few adjustments had to be made during each assignment due to unexpected problems and time constraints

Retrospective on A1

- Create QuestionnaireResponses (not using a library)
- Render a React form based on a Questionnaire (not using a library)
- Create a valid end-to-end prototype over testing features
- Incomplete validation on user submitted data

Retrospective on A2

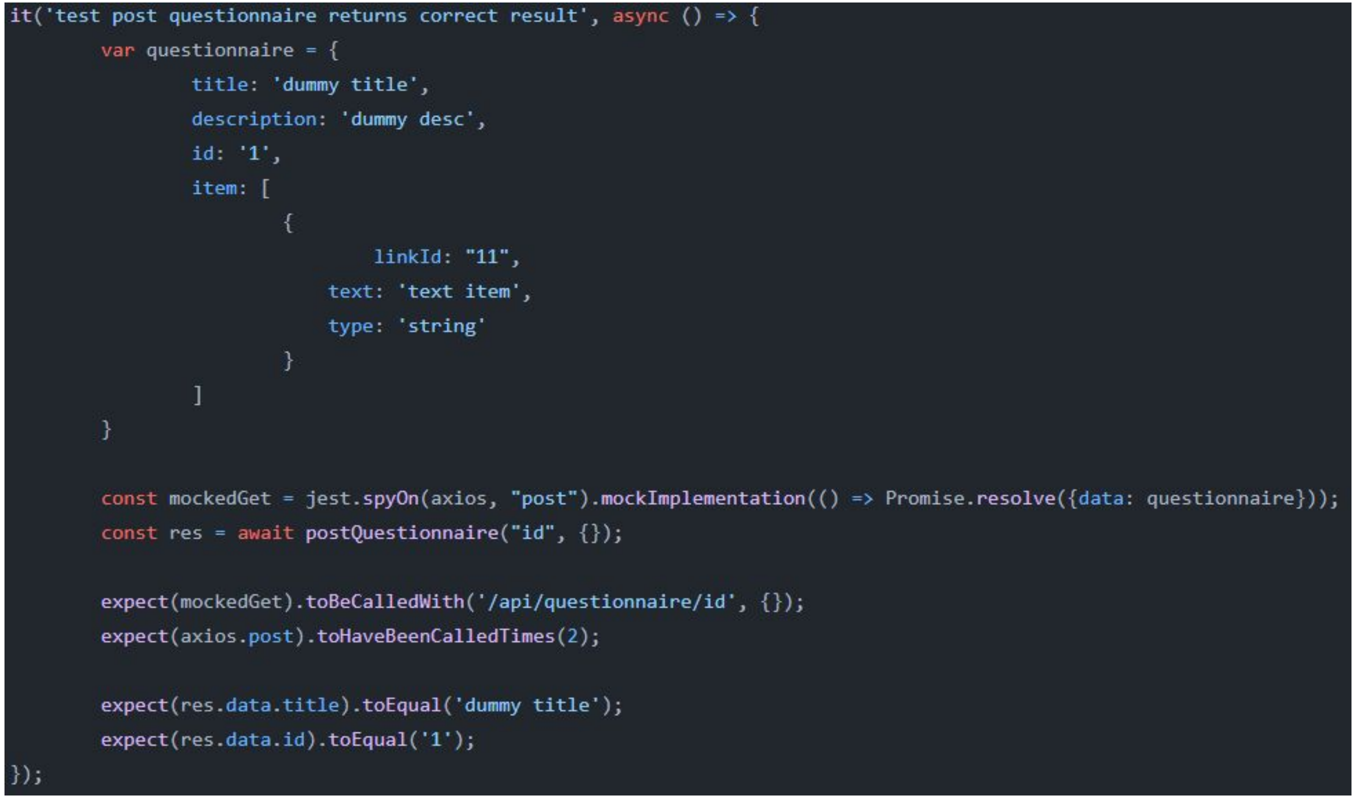
- Hosting our own FHIR server locally
- Would not implement a search bar functionality on frontend
- Improved validation on user submitted data but still incomplete



Testing

1. Frontend UI components rendering and content
2. Form rendering methods and questionnaire object parsing
3. Frontend API Requests
4. Backend data validation methods
5. Backend APIs

Example Test

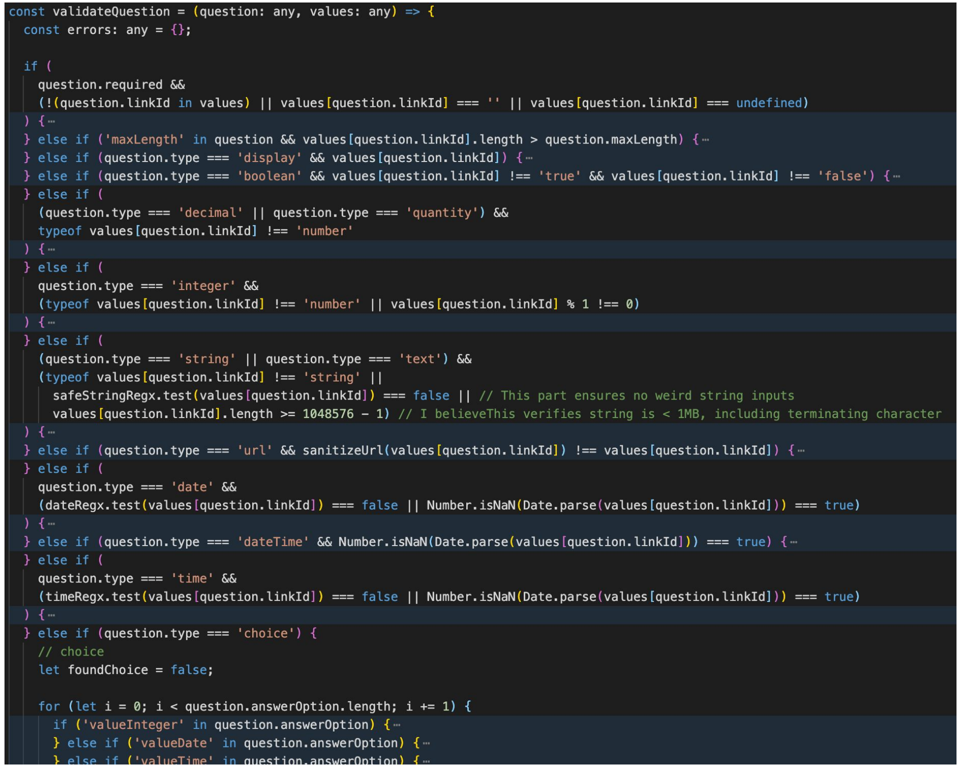


```
it('test post questionnaire returns correct result', async () => {
        var questionnaire = {
                title: 'dummy title',
                description: 'dummy desc',
                id: '1',
                item: [
                        {
                              linkId: "11",
                            text: 'text item',
                            type: 'string'
                        }
                ]
        }

        const mockedGet = jest.spyOn(axios, "post").mockImplementation(() => Promise.resolve({data: questionnaire}));
        const res = await postQuestionnaire("id", {});

        expect(mockedGet).toBeCalledWith('/api/questionnaire/id', {});
        expect(axios.post).toHaveBeenCalledTimes(2);

        expect(res.data.title).toEqual('dummy title');
        expect(res.data.id).toEqual('1');
});
```

Example test for submitting form data to the backend

Validation

- Answer type validation
  - Formatting
  - Input data types
  - Input sizes
  - Constraints on conditional questions, “enableWhens”
  - Required questions
  - etc...



```
const validateQuestion = (question: any, values: any) => {
  const errors: any = {};

  if (
    question.required &&
    (!(question.linkId in values) || values[question.linkId] === '' || values[question.linkId] === undefined)
  ) {⋯
  } else if ('maxLength' in question && values[question.linkId].length > question.maxLength) {⋯
  } else if (question.type === 'display' && values[question.linkId]) {⋯
  } else if (question.type === 'boolean' && values[question.linkId] !== 'true' && values[question.linkId] !== 'false') {⋯
  } else if (
    (question.type === 'decimal' || question.type === 'quantity') &&
    typeof values[question.linkId] !== 'number'
  ) {⋯
  } else if (
    question.type === 'integer' &&
    (typeof values[question.linkId] !== 'number' || values[question.linkId] % 1 !== 0)
  ) {⋯
  } else if (
    (question.type === 'string' || question.type === 'text') &&
    (typeof values[question.linkId] !== 'string' ||
      safeStringRegx.test(values[question.linkId]) === false || // This part ensures no weird string inputs
      values[question.linkId].length >= 1048576 - 1) // I believeThis verifies string is < 1MB, including terminating character
  ) {⋯
  } else if (question.type === 'url' && sanitizeUrl(values[question.linkId]) !== values[question.linkId]) {⋯
  } else if (
    question.type === 'date' &&
    (dateRegx.test(values[question.linkId]) === false || Number.isNaN(Date.parse(values[question.linkId])) === true)
  ) {⋯
  } else if (question.type === 'dateTime' && Number.isNaN(Date.parse(values[question.linkId])) === true) {⋯
  } else if (
    question.type === 'time' &&
    (timeRegx.test(values[question.linkId]) === false || Number.isNaN(Date.parse(values[question.linkId])) === true)
  ) {⋯
  } else if (question.type === 'choice') {
    // choice
    let foundChoice = false;

    for (let i = 0; i < question.answerOption.length; i += 1) {
      if ('valueInteger' in question.answerOption) {⋯
      } else if ('valueDate' in question.answerOption) {⋯
      } else if ('valueTime' in question.answerOption) {⋯
```

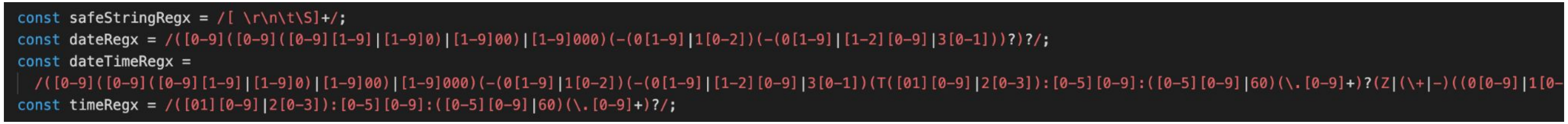
Example of some questionnaire answer type validation done in the backend

Technical Challenges

- FHIR server unreliability
  - Crashed often rendering our product useless
  - Inconsistent Data

- Rendering Questionnaire
  - Too many types, nested items, conditionals!

- Validation
  - Wide acceptance formats
    (Example below of Regex validation)



```
const safeStringRegx = /[ \r\n\t\S]+/;
const dateRegx = /([0-9]([0-9]([0-9][1-9]|[1-9]0)|[1-9]00)|[1-9]000)(-(0[1-9]|1[0-2])(-(0[1-9]|[1-2][0-9]|3[0-1]))?)?/;
const dateTimeRegx =
  /([0-9]([0-9]([0-9][1-9]|[1-9]0)|[1-9]00)|[1-9]000)(-(0[1-9]|1[0-2])(-(0[1-9]|[1-2][0-9]|3[0-1])(T([01][0-9]|2[0-3]):[0-5][0-9]:([0-5][0-9]|60)(\.[0-9]+)?(Z|(\+|-)((0[0-9]|1[0-
const timeRegx = /([01][0-9]|2[0-3]):[0-5][0-9]:([0-5][0-9]|60)(\.[0-9]+)?/;
```

What was learned?

- Research and planning before development
- Organization, communication and timelines
- Documentation
- Healthcare system still has a long way to go

Why does our work matter?

- Helps Chronic Illness Patients
- In-home care and monitoring
- Help Parajuniper bring the healthcare system into the 21st Century #FHIRisFire #TotallyProfessionalPresentation

Demo