Up-Cycle Makerspace Year Summary / FAQ

Confluence Summary

Question - Answer Space / Multiple Inventories

An inventory on the website is a collection of EconomicResource objects. EconomicResources can have different individual locations and different relationships with users. Any item on the website should be available for distribution (there should be little need for “private” inventory items).

Technical Documentation / API Documentation

See separate API documentation

Sprint 4 Documentation / Sprint 4 Friday Meeting Notes (1/12/18)

This meeting took place after an event that was done at Ward 4 which included many stakeholders for the project. During this event, discussion was conducted between various groups as the topic of our marketplace was picked apart.

There was a bit of concern regarding turnover and how to properly incentivize people. A few stakeholders asked about the membership fee and the possibility of site-only credits: how would we handle credit for people who run out, do we allow people to cash out, etc. There was also concern regarding the inherent risk of receiving goods from people online, both in the quality and safety of these goods, but in how they’re catalogued as well. It was also brought up that people may want to track their goods in case they are audited or wish to see what it was used for. Similarly, some people may have specific places in mind that they want their goods to go to.

During our stakeholder meeting the idea of “recipes” was brought up. A recipe would be some type of project or activity that someone else has submitted to the website. These recipes would list all the required materials needed to build it and feature instructions on how to do so. This would allow teachers and instructors to share ideas on how to engage their students and allows for more complex features, such as displaying recipes based on the items in one of your organization’s inventories. This is an idea that many people were enthusiastic about and interested in seeing, and is worth keeping in mind for the eventual future.

There are numerous other talking points in this document that would be worth bringing up in the future as they become more relevant. Some points are important to the development team whereas others are more important for the product owner himself.

Technical Documentation

All important documentation in this section of confluence can be found elsewhere, such as the documentation for GraphQL and the project build instructions.

Sprint Planning in Hindsight

Making sure your user stories are appropriately sliced up is vital. No one should only have 1-3 user stories to work on for a single sprint as this incurs too much risk. It’s okay to have stories worth a lot of story points in the backlog. However, these should be sliced up into multiple smaller stories. Doing this is the difference between a sprint with 8/40 points complete and a sprint with the same amount of work with 36/40 points complete.
A groomed backlog should have two sprints worth of sprint-ready user stories.

Sprint Reviews and Retrospectives in Hindsight

Developing SMART goals based on what went wrong and what went well during the previous sprint will help you have strong process improvement.

FAQ

Tools

Regarding the tools being employed

Error Messages

Regarding common errors returned by the project

Tools

GraphQL

What is GraphQL?

GraphQL is a graph database API tool made by Facebook. It is designed for extracting related data from a graph database, but also has uses in typical relational databases where the information returned could be very large. Basically, it only returns back what was requested, and can return as much or as little data as it needed. It relies heavily on relations between object in the database, which is what makes it attractive for graph databases, but the relations can be abstracted in the resolver to apply it to a normal API endpoint. In the end, what is really important is understanding that it is a framework for designing APIs to connect front ends and databases together.

How does GraphQL differ from REST APIs?

GraphQL relies on relations between object in the database. It can traverse many relationships and return nested data really well. It returns only what was asked for, so the payload remains relatively small compared to a similarly scoped REST API. A REST API returns a specific set of data for any query, and relies on the client to interpret that data and fetch any additional queries. GraphQL uses a more advanced approach to the query structure to interpret the request and pull any additional queries out, returning them all as one.

What kind of data do we get back?

The data returned from the GraphQL query is structured JSON following the same format the data was requested in.

For example, if you requested the ID of all agents related to the agent with id 56, your request would look similar to this

```
viewer(token: “sdrgnj3t34534523efjkkn43456”) {
        getAgentById(id: 56) {
                agents {
                        id
                }
        }
}
```

The data you got back would follow the same form, but it would have the data requested in it.

```
viewer: {
        agent: {
                agents: [
                        {id: 1},
                        {id: 4},
                        {id: 10}
                ]
        }
}
```

What’s a resolver?

The resolver is the function in the API that interprets the request and generates the resulting data. It connects directly to the database and fetches only what was requested in the GraphQL query. You do not need to write those for this project, we recommend reaching out to the existing development team and asking them to write for you. They’ll write it upstread so it benefits multiple teams and you can just pull that code down into your repository.

What’s the difference between a query and a mutation?

A query merely fetches existing information from the database. It can accept arguments and use those to fetch the data, but nothing is changed in the backend. This is the READ database operation. A mutation modifies data in the database. It covered CREATE, UPDATE, and DELETE operations.

What’s this “token” and “viewer” stuff?

The token is the authentication for the currently logged in user. It is generated from the username and password for the user. It is required for almost all queries and mutations. The viewer is a common design pattern for GraphQL queries where the viewer is the root node of the query, and all other queries are underneath it. This is useful for the token requirement, because instead of each new query requiring the token, they can just be subqueries and inherit the authentication of the viewer.

What do we test GraphQL queries?

Queries and mutations can both be tested using the GraphiQL web interface. From the backend, localhost:8000/api/graphiql opens a web app that allows you to send commands directly to the backend api endpoint using plain GraphQL syntax. This allows you to ensure your query or mutation is correct before writing a ui-binding for it. It also allows your to perform API actions quickly that might not require integration into rea-app. The GraphiQL interface is also useful for previewing what you get back from the API. I makes it easier to dissect the data and extract what is useful to you if you can see the full JSON response.

How do we make GraphQL queries in the code?

Please see the example commented file in the rea-app project at
`packages/ui-bindings/EconomicEvent/CreateEconomicEvent.tsx`

How do we make GraphQL mutations in the code?

Please see the example commented file in the rea-app project at
`packages/ui-bindings/EconomicEvent/CreateEconomicEvent.tsx`

What are Fragments?

Fragments are partial queries that can be reused in other queries by simply referencing them in the query structure. For example, a Person might have an id, first name, last name, and email address. Every time a Person is returned from the database, let’s say you want back all four of those properties for the person. Instead of writing those into every query that returns a person, you can define a Person fragment containing the properties you want back, and just include the fragment name. This saves lots of time when larger data types are returned that might have dozens of properties.

How do you make fragments in the code?

This explains it better than I could (and it has an interactive sandbox to test them out).
http://graphql.org/learn/queries/#fragments

Apollo

What is Apollo?

Apollo is a popular tool for making advanced GraphQL queries in Javascript. It has specific React support as well. It combines the query with any query fragments and injects variables. It also allows pulling variables from Redux stores while constructing the query. It passes the data back in either the props or as a JSON object, depending on optional arguments when constructing the query.

Why are we using Apollo?

We are using Apollo because it is one of the most popular tools for integrating GraphQL into a project. It has React support and allows for a lot of additional options when constructing the query.

How are we using ui-bindings?

In the rea-app project, there is a ui-bindings directory. We are treating everything in there as an API middleware layer. We are defining interfaces, fragments, queries, and mutations for all of the API calls. This way, we can simply import the query into the front end page we are working on. We have better code reuse since we don’t have to redefine the same query in every page that needs it.

What is going on in the API code?

Please see the example commented file in the rea-app project at
`packages/ui-bindings/EconomicEvent/CreateEconomicEvent.tsx`

React

What is React?

React is a Javascript framework for making web pages out of reusable components. Everything on the page is a component, and can be reused anywhere else in the project. Each component has its own state and properties that make it unique from the other components.

Why are we using React? ← Good question….

Primarily, we are using React because the project existed before we started. Another development team from individual contributors around the world have been working on this project for a few years now. Using React does have benefits for the developers in making the project scale easier as development continues.

What is a React Component?

A React Component is the building block of the web page. A React component has its own properties and state. It is a class in Typescript that allows local variables and functions. Every React component must know how to render itself on the screen, but that can be accomplished easily using TSX (described below).

What is the state?

Every React component has its own state. This is where it maintains important information about the object. Every time the state changes, it redraws the component on the screen. This allows you to dynamically change components on the screen without registering event handlers. By simply changing a variable (that is tracked in the state) React will update the component in the browser for you. You can define the state yourself, and include any variables you want inside the JSON object.

What are props?

Properties are passed to React components inside their brackets when defining them in HTML. These props allow customization of the React component, or passing variables / functions to the component from the parent. Props are immutable in React which means that they can't change. The component can't re-assign any of the props. If a value that the parent passed to the React component changes, it triggers a redraw of the entire component.

What does it mean that React is "stateless"?

React is not stateless. It does encourage the use of functional Javascript methods and components which *are* stateless, but it also supports using stateful React components. Stateless just means not having a state, and describes the functional programming approach.

JSX / TSX

What is JSX?

JSX stands for JavaScript XML. It is a way to define HTML inside Javascript that became very popular with React. It does not follow all of the normal HTML rules, but translates directly to HTML on a page.

What is TSX?

TSX is Typescript XML and works the same way as JSX, but with Typescript.

What limitations come with TSX?

Since React is interpreting the TSX and converting it to HTML, you cannot always use HTML directly. For example, in HTML components you list the “class” something belongs to for CSS. In React TSX, you list the “className”. Any components you define cannot start with a lowercase letters. Those are interpreted as pure HTML, instead of your component.

You can also include Javascript in your TSX. This is useful for looking up values, mapping over an array, or conditionally rendering something. Anything inside curly braces will be interpreted as Javascript.

VirtualBox

What is Virtualbox?

Virtualbox is a virtual machine emulator for Windows. It allows you to run other operating systems inside of Windows. It is most commonly used for Linux distributions to run inside Windows, without setting up a dual boot system. It should be installed from Software Dev 3 (and Vagrant).

Why are we using Virtualbox?

Some of the tools used to start up the project are built for Linux only. They have been tested with Ubuntu 16.04 LTS, so that is the OS we are developing with. Since our laptops are not available to dual boot (without the bios password) we are using VirtualBox to run Ubuntu so we can execute the project.

Why not Windows development?

Believe me, we’ve asked ourselves that at least once every sprint. Windows development would certainly be easier. Unfortunately, we have devoted significant time to converting the project to Windows compatible and it seems to be very tightly coupled with Linux tools.

Why not Linux Subsystem?

We tried that a couple of sprints too. Since it isn’t purely Linux and is still built on Windows, some of the Linux only features still don’t work as expected.

Isn’t this really slow and frustrating?

Yup. But what’s more slow and frustrating is repeatedly trying to get it to work on Windows and wasting lots of development hours with no results.

What’s the difference between Shut Down and Save State?

When closing the Virtual Machine, you have two options for how to shutdown the VM. Shut Down will close all running processes and shutdown the Ubuntu OS. This would be the equivalent of shutting your computer down. Save State will take all current processes and RAM and save them to the disk. It then terminates the processes and closes the VM window. Both options leave nothing running on the VM after you close it. If you selected Save State, it will restart all of the processes and restore the RAM so the VM is in the exact same state it was before closing it.

Python Django

What is Django?

Django is a Python web framework for building sites in Python. The valuenetwork project (back end) is currently using it.

Do I need to know Django?

Nope. The original development team is happy to assist in Python or Django work.

Who do we go to for Django help?

If you have any questions about something in the valuenetwork project or Django in general, reach out to the original dev team on Slack and they can help you out.

Slack

What is Slack?

Slack is a workspace for team communication. It works like an instant messaging application keeping each project in its own scope. It supports channels which help separate the context of a conversation which allows for concurrent discussions without getting confused.

What are the channels we use?

# blockers - Post here when you are stuck on something. Even if you aren't looking for assistance, someone else might see it and know exactly what to do to get you moving forward again. It is also useful for documenting what went wrong, so when it is fixed, you can post what you did to solve that problem.
# general - General communication.
# links - Links to resources that helped you.
# process - Post the process of how to do things here. Examples include setting up the dev environment, making a new page on the website, or deploying to Google Cloud Platform. This area is meant to be step by step instructions.
# pull-requests - This is an integration from GitHub that posts when pull requests are opened or closed. Users can also post in here to explain anything about their pull request (such as new packages needed) or to pester students to review their pull request.
# random - Random stuff.

# students - Private space for just the students. The other development team and product owner are not in this private channel.
# xp - We used this as a continuous standup similar to what might be used in the XP model. We had communication problems in our team, so we posted status updates after we stopped working on something.
# builds - This is the Travis integration. It posts the status of builds whenever a pull request is updated.

What integrations do we already have set up?

GitHub - Currently tracks pull requests that are opened or closed.
Travis - Currently builds on pull requests. It reports back the build status in the #build channel
Polly - Useful for polls and surveys. Can be used for anonymous peer evaluation surveys.

Why Slack?

Slack is useful for separating concurrent conversations by context. It also allows a number of useful integrations to manage the project from one place.

Travis

What is Travis?

Travis is a build server which periodically attempts to build the code base and lets you know if it was successful or not.

When does Travis run?

Travis runs every time there is a pull request, or once every day if there have been no pull requests.

How do we see the results of Travis?

We linked Travis with Slack so the results are posted in the #builds channel.

How do we configure Travis?

Go to https://travis-ci.org/LearnDeepMilwaukee/rea-app. You sign in with your Github credentials.

NodeJS

What is NodeJS?

NodeJS is a Javascript engine for executing JS server side. It contains lots of common functions for serving up web pages and managing server side data. In simple words, it allows execution of headless Javascript without using a Javascript engine from a browser.

NPM

What is NPM?

NPM stands for Node Package Manager. It is used to keep track of external packages being used in the project. For NodeJS, many times it is easier to bring in an external package than reinvent the wheel for common web tasks. Additionally, these packages are often independently tested and validated by a community of active users who help ensure the package's stability.

Yarn

What is Yarn?

Yarn is an alternative to NPM developed by Facebook. In the early days of NPM, it required an internet connection to pull the packages from their repository. Facebook wanted to have an offline version of NPM for their testing environment. Instead of following common open-source practices and forking a copy of NPM and improving it by adding their own changes, they started from the ground up and built their own solution. Relatively soon after Yarn was released, NPM released the additions that solved everything Yarn brought in. Yarn is most commonly used now because it pulls packages from a global archive of cached packages. This increases the speed (if your download speed is slow) but requires an increase in storage capacity to maintain a cache of packages.

Can we use NPM or do we have to use Yarn?

You can most certainly use NPM. We have found it to be significantly faster than Yarn in our project. The different is how NPM and Yarn maintain their package list. NPM uses a list of dependencies inside package.json in the project root. Yarn uses yarn.lock (also in the project root). The way they store the information is slightly different, but both support the same capabilities as far as requiring a certain version of the package. To install a package using NPM, type "npm install -S package-name". The -S tells it to save the package in the dependencies list. Using Yarn, simply type "yarn add package-name". It does the rest for you.

Why do we use Yarn instead of NPM?

We are using Yarn for most of the packages because the developers before us were using Yarn. It was easier to just keep it in the project compared to converting everything to NPM. For many new packages, we have used NPM to add them.

Why is Yarn *SO SLOW?*

We're not entirely sure. Yarn is supposed to be incredibly fast because it pulls the packages from the file system cache instead of downloading them. Perhaps the reason it is slow in this project is because of the number of packages it is managing, and all different versions of them.

Typescript

What is Typescript?

Typescript is a typed superset of Javascript. It allows more formal class creation, and type validation. It transpiles directly into Javascript to be run in a browser.

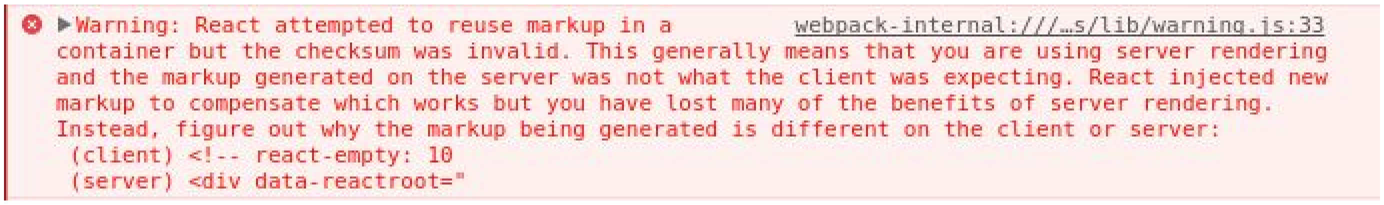
Why are we using Typescript?

Typescript promotes fewer errors by catching incompatible types at transpile time. It can alert you that something is an error before that error would come up in normal testing. It also allows better intellisense abilities because the properties of a class are predetermined.

Error Messages

Console Errors

“React attempted to reuse markup in a container but the checksum was invalid.”



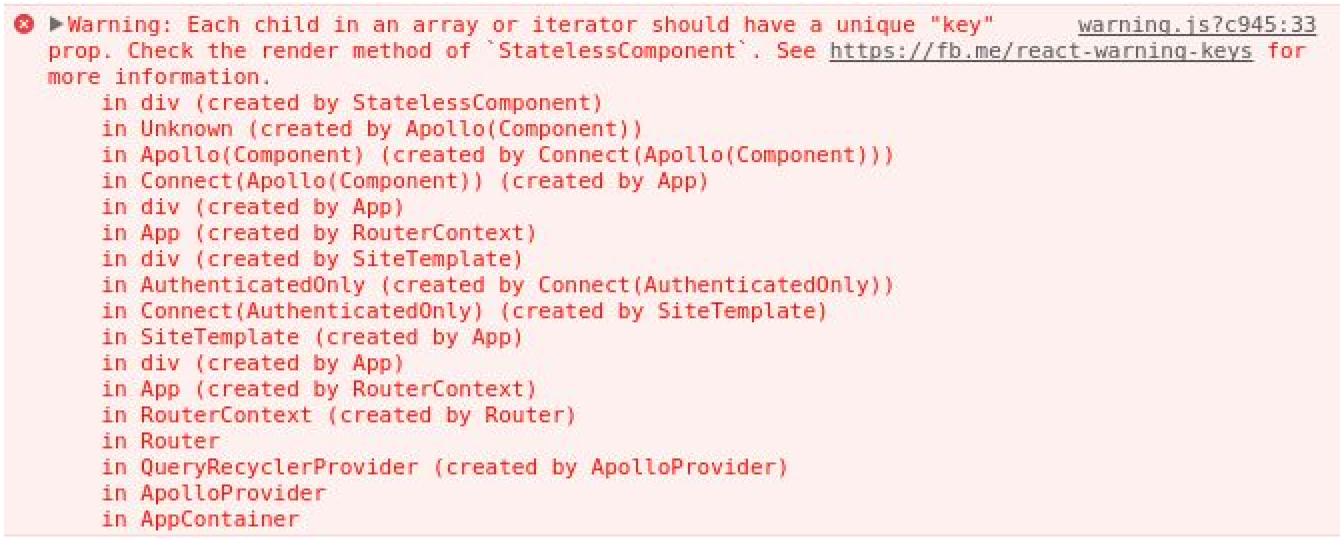
```
▶Warning: React attempted to reuse markup in a        webpack-internal:///…s/lib/warning.js:33
container but the checksum was invalid. This generally means that you are using server rendering
and the markup generated on the server was not what the client was expecting. React injected new
markup to compensate which works but you have lost many of the benefits of server rendering.
Instead, figure out why the markup being generated is different on the client or server:
 (client) <!-- react-empty: 10
 (server) <div data-reactroot="
```

This is a React error that occurs when using server rendering and the checksum is not what was expected. What is happening is the page is being rendered on the server and then matched against what was rendered by the client. For some reason, the client output does not match the server output. The checksum is simply an internal consistency mechanism used to ensure the server and client outputs are the same.

The invalid checksum is a pre-existing issue that will propagate to every page of the site. We are not sure where the source of this issue lies, but it has something to do with the route order. Any page that exists under the *App* component in *routes.tsx* will have this issue. This has been tested and confirmed, where the same page outside of the “/” path generates no checksum error whereas inside the “/” path it does. Since this error is being propagated from somewhere higher in the load order, it’s impossible to tell if any of the newly introduced pages are also generating their own checksum errors.

Trying to fix this issue could prove to be a lengthy venture, as it would mean modifying code that is reused in every single page on the site. For a full release, this is something that would need to be fixed, but for our purposes we did not feel it was a top priority.

“Each child in an array or iterator should have a unique “key” prop.”



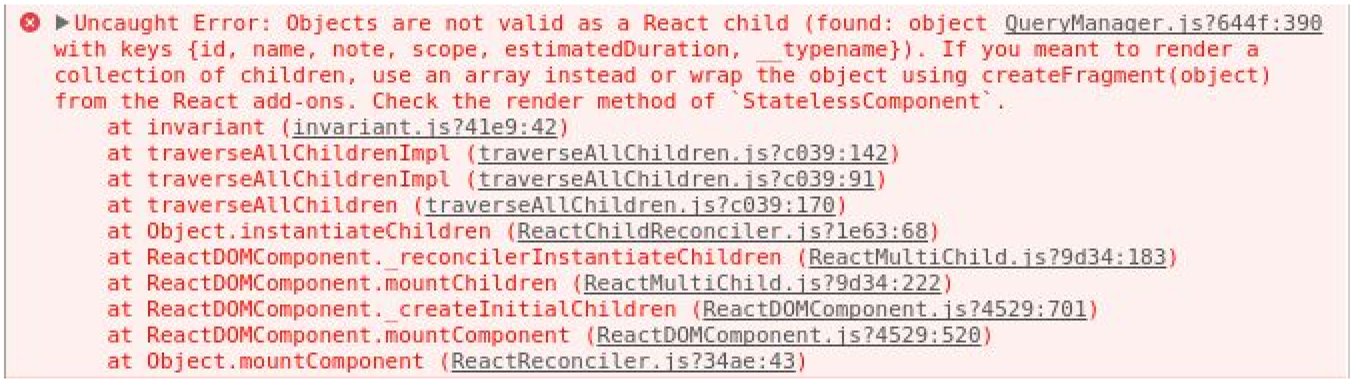
```
▶Warning: Each child in an array or iterator should have a unique "key"      warning.js?c945:33
prop. Check the render method of `StatelessComponent`. See https://fb.me/react-warning-keys for
more information.
    in div (created by StatelessComponent)
    in Unknown (created by Apollo(Component))
    in Apollo(Component) (created by Connect(Apollo(Component)))
    in Connect(Apollo(Component)) (created by App)
    in div (created by App)
    in App (created by RouterContext)
    in div (created by SiteTemplate)
    in AuthenticatedOnly (created by Connect(AuthenticatedOnly))
    in Connect(AuthenticatedOnly) (created by SiteTemplate)
    in SiteTemplate (created by App)
    in div (created by App)
    in App (created by RouterContext)
    in RouterContext (created by Router)
    in Router
    in QueryRecyclerProvider (created by ApolloProvider)
    in ApolloProvider
    in AppContainer
```

This is an error message that will appear if you are using the *Array.prototype.map()* function without setting a key in an outer element. The map function repeats some given operation on a an array of objects for each object inside. Notice the use of “for each” in that last sentence, as this is essentially a fancy for each loop, including the lack of an iterator index.

Why does each child of the array need a specified key prop? React uses keys to track changes, the addition and removal of items. To fix this, all that needs to be done is give some outer DOM element in the map function a key attribute.

More reading on the topic (with examples) can be done at: https://reactjs.org/docs/lists-and-keys.html

“Objects are not valid as a React child.”

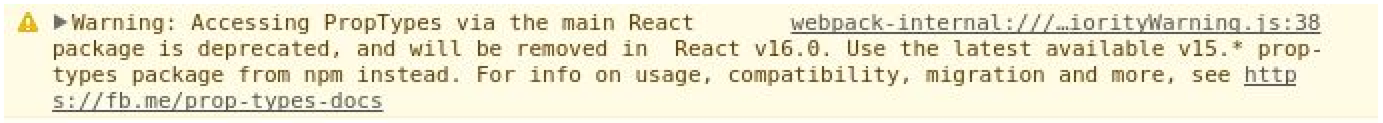


```
▶Uncaught Error: Objects are not valid as a React child (found: object QueryManager.js?644f:390
with keys {id, name, note, scope, estimatedDuration, __typename}). If you meant to render a
collection of children, use an array instead or wrap the object using createFragment(object)
from the React add-ons. Check the render method of `StatelessComponent`.
    at invariant (invariant.js?41e9:42)
    at traverseAllChildrenImpl (traverseAllChildren.js?c039:142)
    at traverseAllChildrenImpl (traverseAllChildren.js?c039:91)
    at traverseAllChildren (traverseAllChildren.js?c039:170)
    at Object.instantiateChildren (ReactChildReconciler.js?1e63:68)
    at ReactDOMComponent._reconcilerInstantiateChildren (ReactMultiChild.js?9d34:183)
    at ReactDOMComponent.mountChildren (ReactMultiChild.js?9d34:222)
    at ReactDOMComponent._createInitialChildren (ReactDOMComponent.js?4529:701)
    at ReactDOMComponent.mountComponent (ReactDOMComponent.js?4529:520)
    at Object.mountComponent (ReactReconciler.js?34ae:43)
```

This error occurs when attempting to render an object somewhere, which React doesn’t know how to do. In turn, this is usually because an object was passed as a prop. When attempting to render an object, and not just an attribute of that object which is a string or number, *toString()* should be used to convert it to a form that React can render.

Console Warnings

“Accessing PropTypes via the main React package is deprecated, and will be removed in React v16.0.”

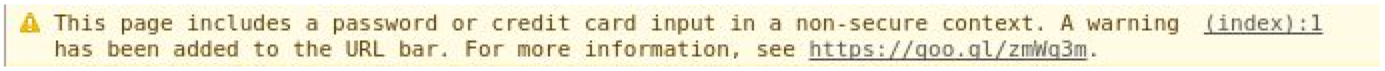


```
▶Warning: Accessing PropTypes via the main React        webpack-internal:///…iorityWarning.js:38
package is deprecated, and will be removed in  React v16.0. Use the latest available v15.* prop-
types package from npm instead. For info on usage, compatibility, migration and more, see http
s://fb.me/prop-types-docs
```

PropTypes, in a nutshell, is a way to implement type checking for the props of a component in your application. As of the most recent React version this is done via a separate package called *prop-types*, but since we are using React v15.4.1 this isn’t a problem. This would only be an issue if you ever plan on updating the React version. Currently, PropTypes can be used via *React.PropTypes* rather than the separate *prop-types* package. A quick check through the code doesn’t reveal any uses of PropTypes this way, which leads us to believe it’s either a constant warning, or a false-positive triggered by typescript.

More reading on the topic can be done at: https://reactjs.org/warnings/dont-call-proptypes.html

“This page includes a password or credit card input in a non-secure context.”

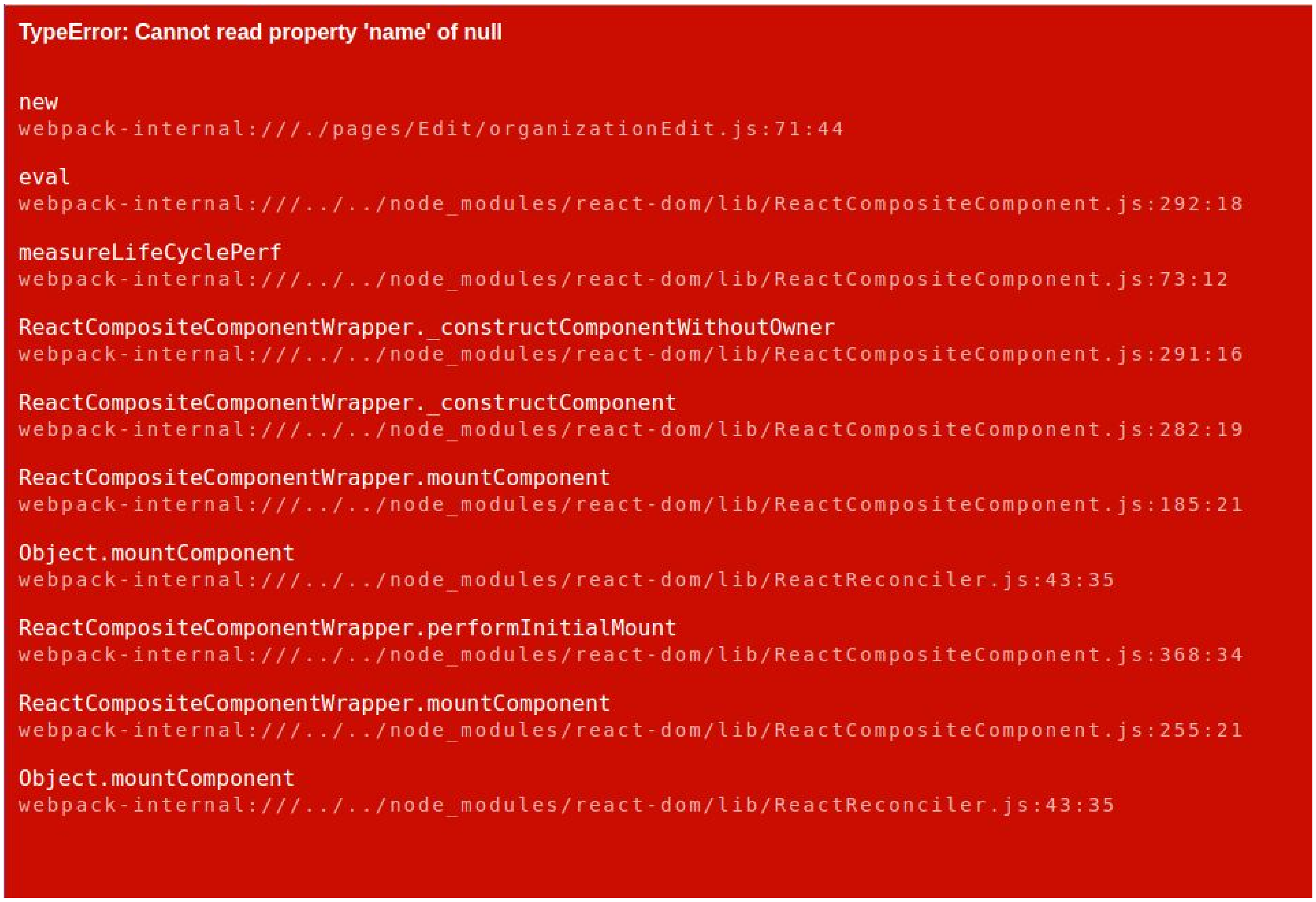


```
This page includes a password or credit card input in a non-secure context. A warning  (index):1
has been added to the URL bar. For more information, see https://goo.gl/zmWq3m.
```

This is a warning from Chrome that is generated because we’re passing around information on a non-secure (non-HTTPS) channel. This is an issue that would need addressing when a proper web server is set up to host the website. For now, it’s just something to worry about later.

Red Screens Errors

“Cannot read property of null”



```
TypeError: Cannot read property 'name' of null

new
webpack-internal:///./pages/Edit/organizationEdit.js:71:44

eval
webpack-internal:///../../node_modules/react-dom/lib/ReactCompositeComponent.js:292:18

measureLifeCyclePerf
webpack-internal:///../../node_modules/react-dom/lib/ReactCompositeComponent.js:73:12

ReactCompositeComponentWrapper._constructComponentWithoutOwner
webpack-internal:///../../node_modules/react-dom/lib/ReactCompositeComponent.js:291:16

ReactCompositeComponentWrapper._constructComponent
webpack-internal:///../../node_modules/react-dom/lib/ReactCompositeComponent.js:282:19

ReactCompositeComponentWrapper.mountComponent
webpack-internal:///../../node_modules/react-dom/lib/ReactCompositeComponent.js:185:21

Object.mountComponent
webpack-internal:///../../node_modules/react-dom/lib/ReactReconciler.js:43:35

ReactCompositeComponentWrapper.performInitialMount
webpack-internal:///../../node_modules/react-dom/lib/ReactCompositeComponent.js:368:34

ReactCompositeComponentWrapper.mountComponent
webpack-internal:///../../node_modules/react-dom/lib/ReactCompositeComponent.js:255:21

Object.mountComponent
webpack-internal:///../../node_modules/react-dom/lib/ReactReconciler.js:43:35
```

Note: This error can potentially show up as a console error instead.

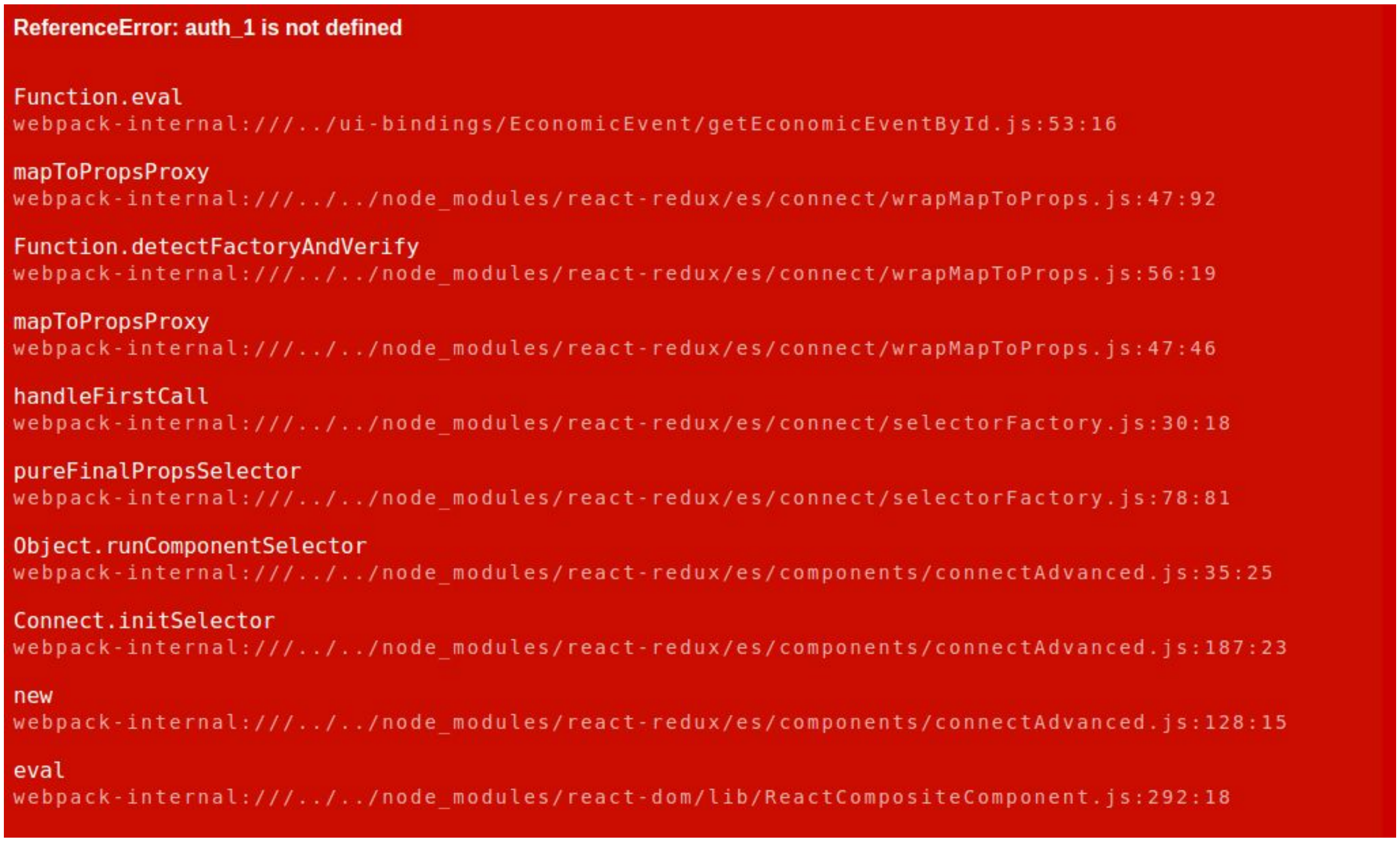
The cause of this error is simple but finding where it’s happening might be tricky depending on your code. When this error appears it’s because you’re trying to reference an attribute on an object that does not exist (hence the “null” part).

This error is usually caused by trying to reference an attribute on a queried object before the query finishes.

If you are using a GraphQL query from ui-bindings to fetch some data, make sure you’re using the **getValidation** function from *app/pages/Common/common.tsx* correctly. Querying data is not instantaneous, and so there will be a short period of time where the object does not exist. We use the *loading* and *error* states to help specify when data can be accessed, and components properly rendered.

However, this error could just as easily be due to a typo, either in the object or attribute name.

“auth_1 is not defined”



```
ReferenceError: auth_1 is not defined

Function.eval
webpack-internal:///../ui-bindings/EconomicEvent/getEconomicEventById.js:53:16

mapToPropsProxy
webpack-internal:///../../node_modules/react-redux/es/connect/wrapMapToProps.js:47:92

Function.detectFactoryAndVerify
webpack-internal:///../../node_modules/react-redux/es/connect/wrapMapToProps.js:56:19

mapToPropsProxy
webpack-internal:///../../node_modules/react-redux/es/connect/wrapMapToProps.js:47:46

handleFirstCall
webpack-internal:///../../node_modules/react-redux/es/connect/selectorFactory.js:30:18

pureFinalPropsSelector
webpack-internal:///../../node_modules/react-redux/es/connect/selectorFactory.js:78:81

Object.runComponentSelector
webpack-internal:///../../node_modules/react-redux/es/components/connectAdvanced.js:35:25

Connect.initSelector
webpack-internal:///../../node_modules/react-redux/es/components/connectAdvanced.js:187:23

new
webpack-internal:///../../node_modules/react-redux/es/components/connectAdvanced.js:128:15

eval
webpack-internal:///../../node_modules/react-dom/lib/ReactCompositeComponent.js:292:18
```

This is an error that occurs when changing anything related to the files in UI-Bindings and loading a page that uses that code. To solve this error, you need to rebuild the entire project (run *npm run dev* in the rea-app directory). We aren’t exactly sure why this error occurs. Our best guess would have to do with the current token getting invalidated somehow. Fortunately, rebuilding the project has resolved this error whenever it appears.