Meteor.js + ReactJS

MongoDB + d3.js

Flask API + Python

Alan Xie '16
CS50 Seminar
November 2016

Technology

React allows us to focus on building components and automatically manages UI updates

Meteor.js full-stack framework allows rapid prototyping with pre-built functionality and works with React + Node.js

MongoDB provides JSON-format storage on the Meteor back-end and works with Python data science pipeline

Build fast & deploy fast…but a few caveats on scale.

Meteor is usually overkill.

MongoDB documents might not be right for you.

Setup

Meteor automatically installs MongoDB.

We can install **React** via the meteor or node package managers (along with fun things like jQuery and d3).

```
$ meteor npm install --save react react-dom
$ meteor add mizzao:jquery-ui
```

https://github.com/alan-xie/cs50seminar-meteor-react

Default boilerplate. No real content but all **CSS** lives here. Calls **imports/startup/client**.

dump folder created by **mongodump**.

Think of these files as being “imported” into **main.html**.

imports/startup is called at startup.

imports/ui/components is where we build things.

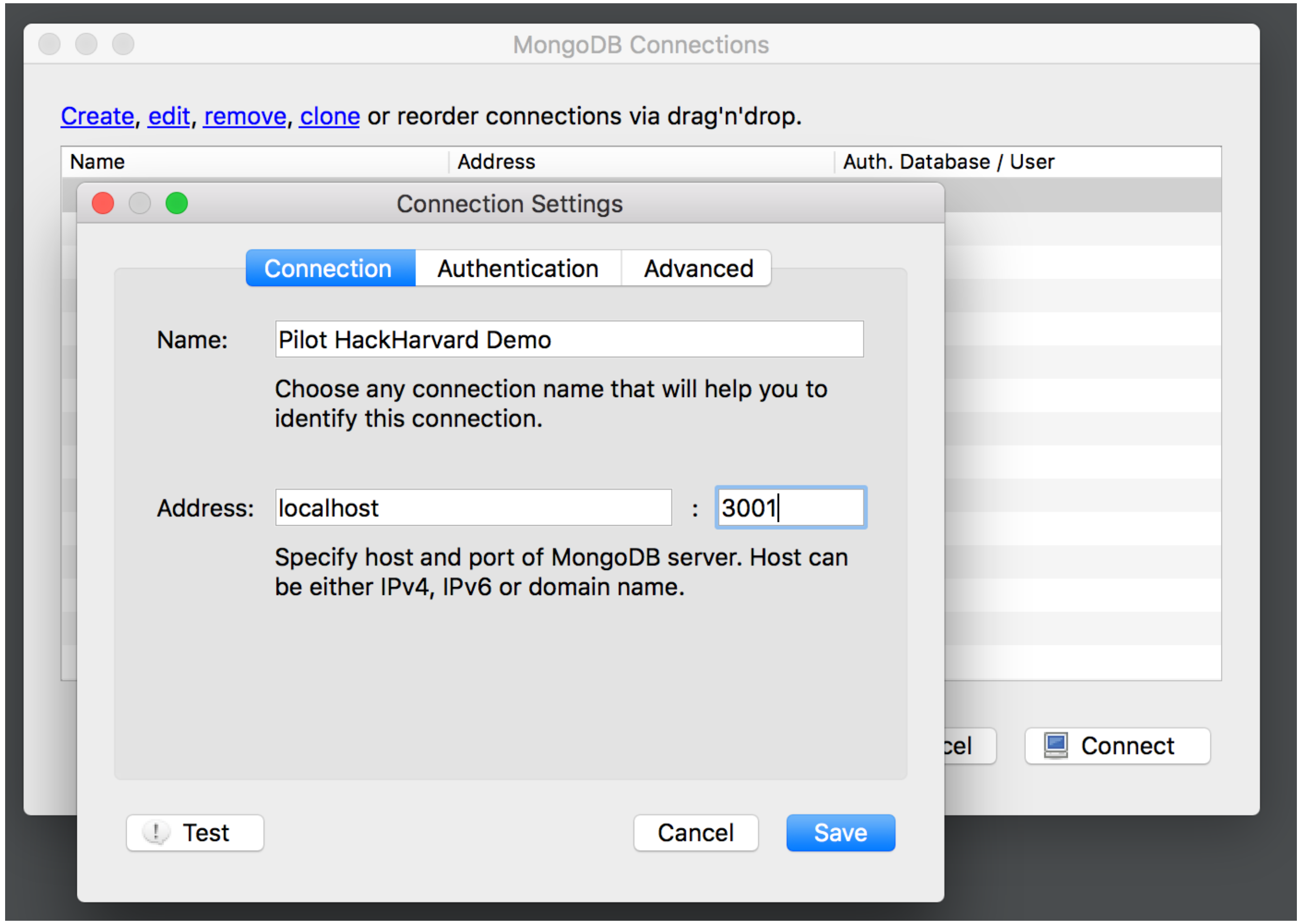
Not accessible except via Meteor server methods.

Accessible via client.

Calls **imports/startup/server**.

```
.
├── README.md
├── client
│   ├── main.css
│   ├── main.html
│   └── main.jsx
├── dump
│   └── meteor
│       ├── meteor_accounts_loginServiceConfiguration.bson
│       ├── meteor_accounts_loginServiceConfiguration.metadata.json
│       ├── people.bson
│       ├── people.metadata.json
│       ├── titles.bson
│       ├── titles.metadata.json
│       ├── users.bson
│       └── users.metadata.json
├── imports
│   ├── api
│   │   └── films.js
│   ├── startup
│   │   ├── client
│   │   │   ├── accounts-config.js
│   │   │   └── index.js
│   │   └── server
│   │       ├── films.js
│   │       ├── index.js
│   │       └── security.js
│   └── ui
│       ├── components
│       │   ├── AccountsUIWrapper.jsx
│       │   └── Actor_RevChart.jsx
│       └── layouts
│           └── App.jsx
├── package.json
├── people.json
├── private
│   └── names.csv
├── public
│   ├── Pilot.png
│   └── favicon.ico
├── server
│   └── index.js
└── titles.json
```

Robomongo Setup

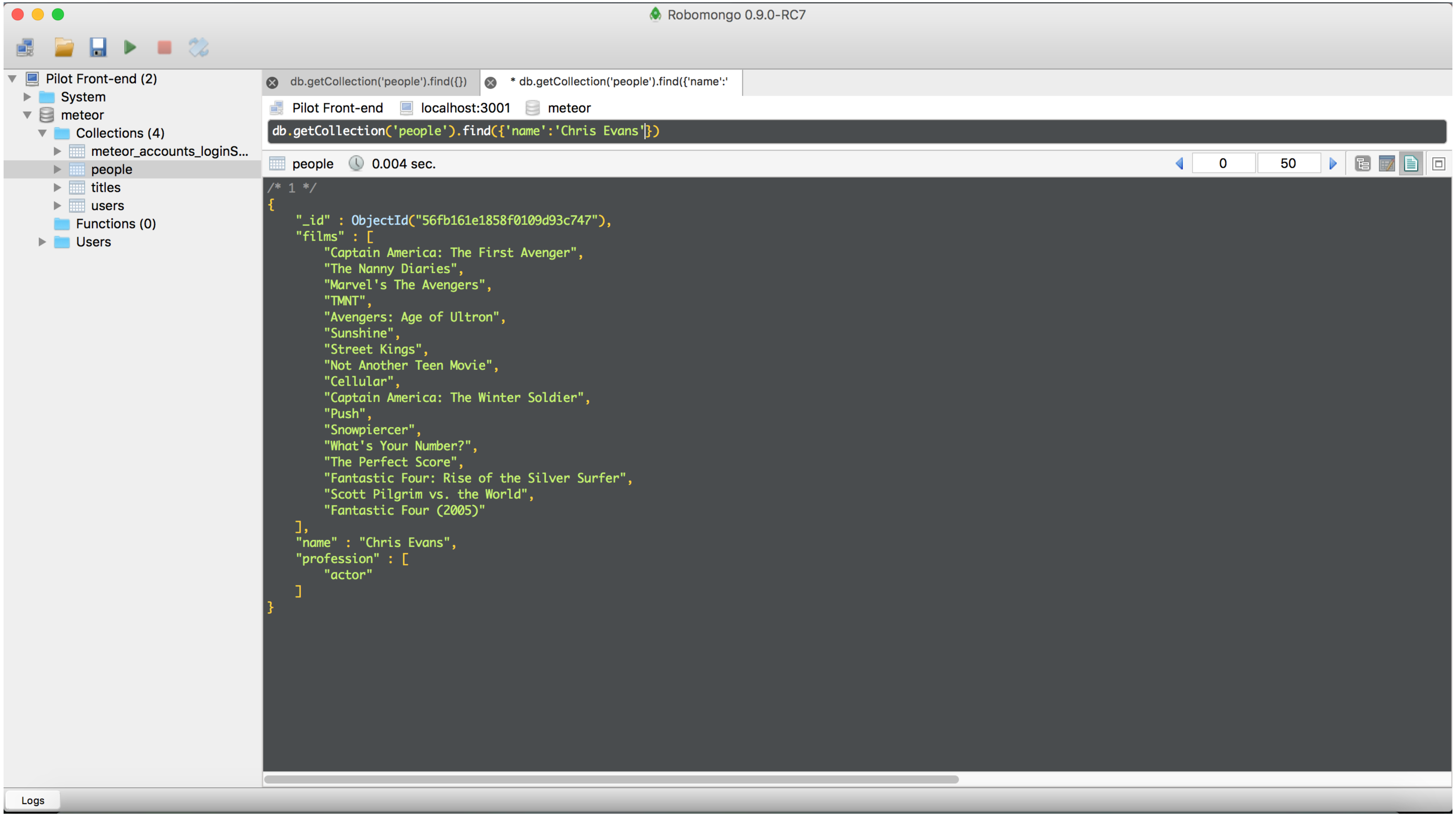


Name: Pilot HackHarvard Demo

Choose any connection name that will help you to identify this connection.

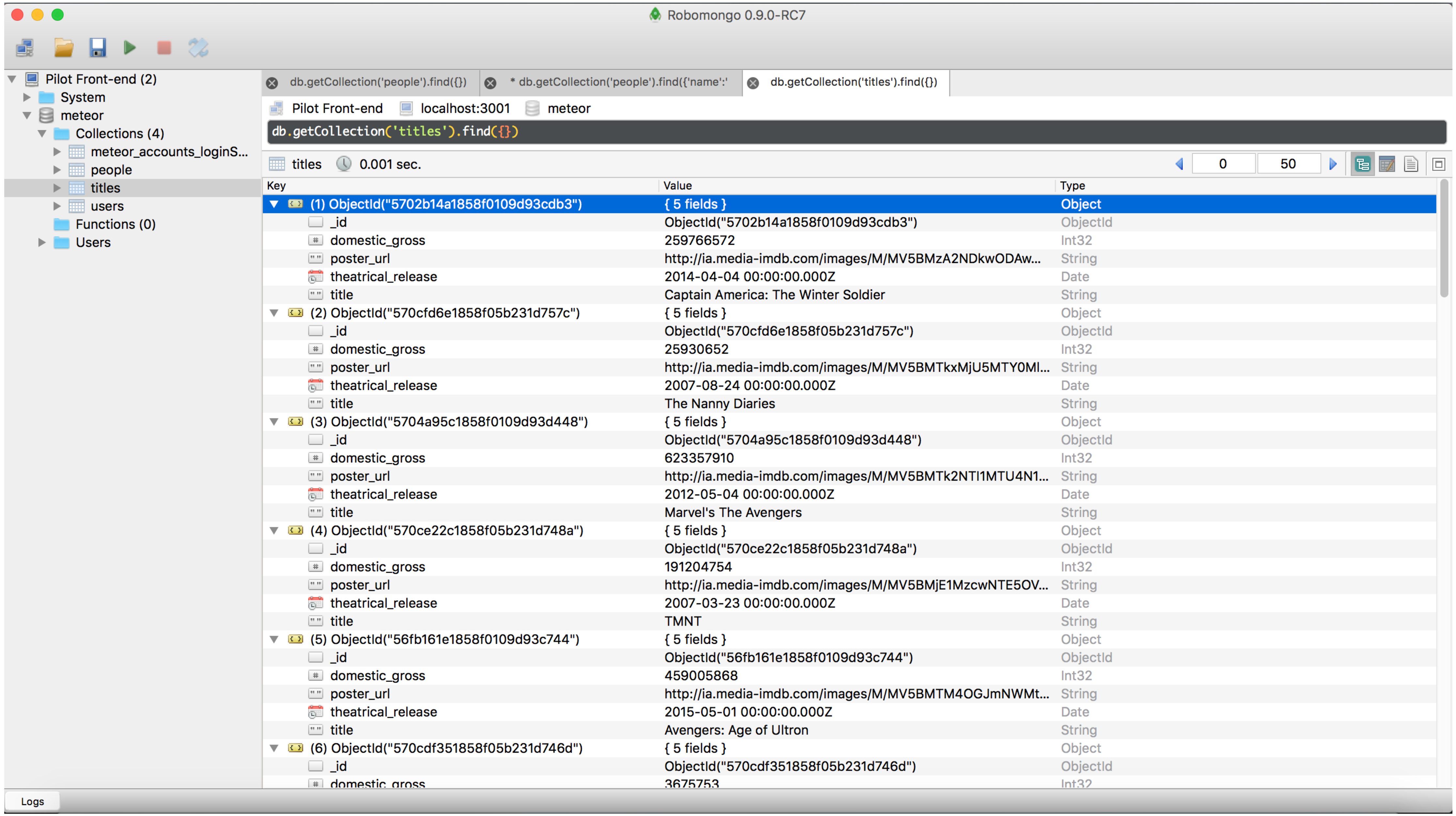
Address: localhost : 3001

Specify host and port of MongoDB server. Host can be either IPv4, IPv6 or domain name.



```
db.getCollection('people').find({'name':'Chris Evans'})
```

```
/* 1 */
{
    "_id" : ObjectId("56fb161e1858f0109d93c747"),
    "films" : [
        "Captain America: The First Avenger",
        "The Nanny Diaries",
        "Marvel's The Avengers",
        "TMNT",
        "Avengers: Age of Ultron",
        "Sunshine",
        "Street Kings",
        "Not Another Teen Movie",
        "Cellular",
        "Captain America: The Winter Soldier",
        "Push",
        "Snowpiercer",
        "What's Your Number?",
        "The Perfect Score",
        "Fantastic Four: Rise of the Silver Surfer",
        "Scott Pilgrim vs. the World",
        "Fantastic Four (2005)"
    ],
    "name" : "Chris Evans",
    "profession" : [
        "actor"
    ]
}
```



Robomongo 0.9.0-RC7

- Pilot Front-end (2)
  - System
  - meteor
    - Collections (4)
      - meteor_accounts_loginS...
      - people
      - titles
      - users
    - Functions (0)
    - Users

db.getCollection('people').find({}) | * db.getCollection('people').find({'name':' | db.getCollection('titles').find({})

Pilot Front-end   localhost:3001   meteor

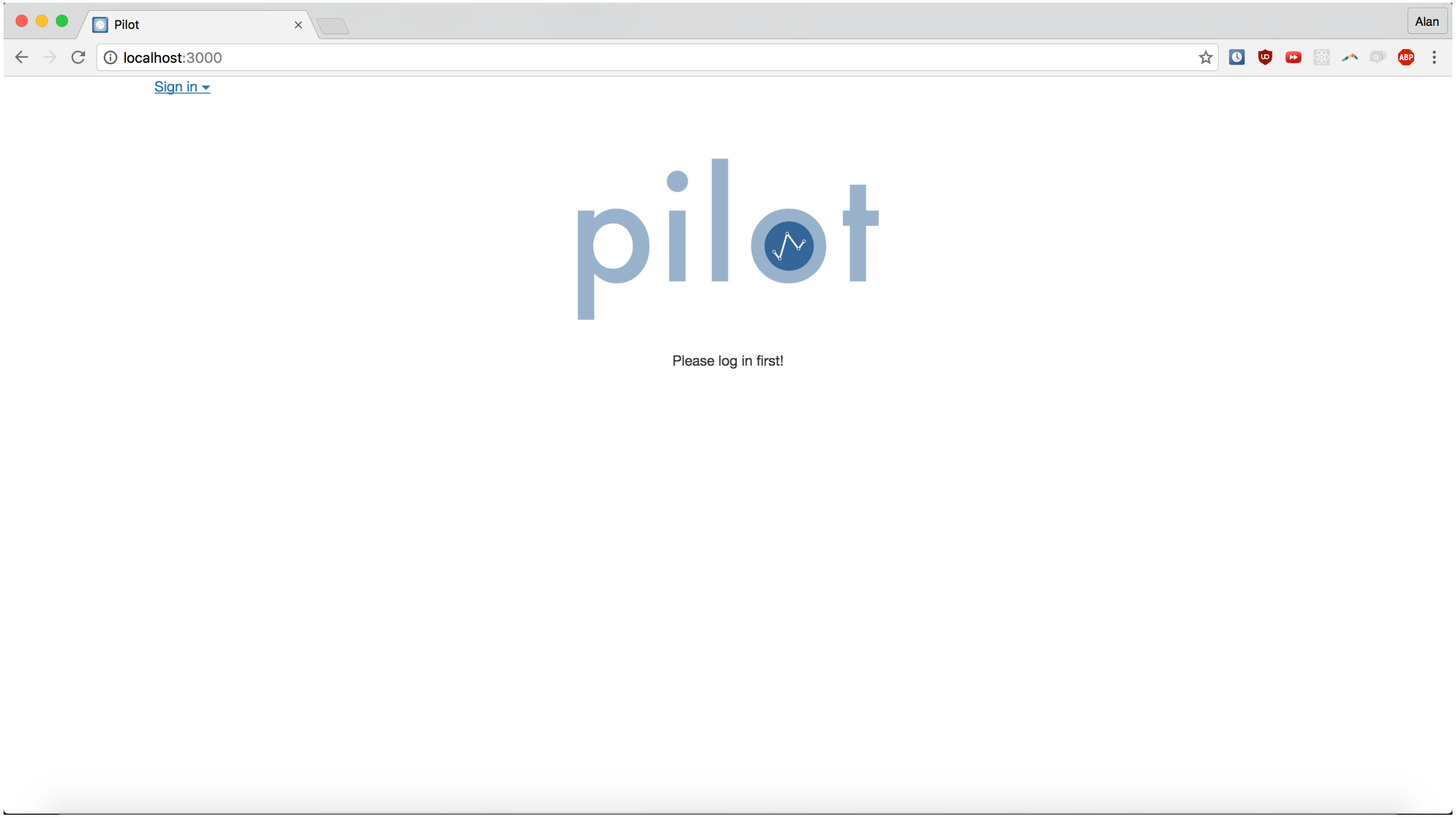
```
db.getCollection('titles').find({})
```

titles   0.001 sec.   0   50

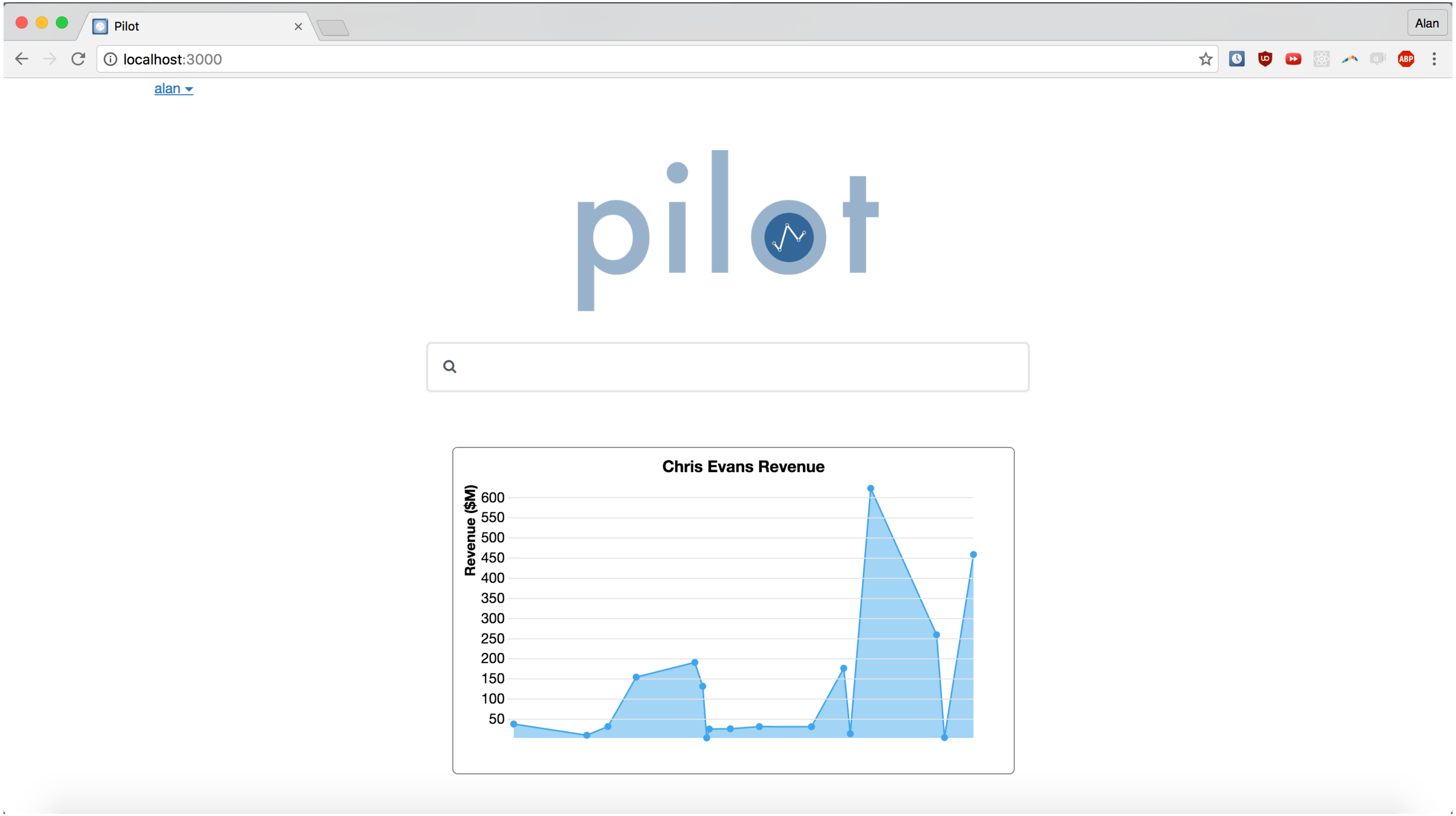
Key	Value	Type
(1) ObjectId("5702b14a1858f0109d93cdb3")	{ 5 fields }	Object
_id	ObjectId("5702b14a1858f0109d93cdb3")	ObjectId
domestic_gross	259766572	Int32
poster_url	http://ia.media-imdb.com/images/M/MV5BMzA2NDkwODAw...	String
theatrical_release	2014-04-04 00:00:00.000Z	Date
title	Captain America: The Winter Soldier	String
(2) ObjectId("570cfd6e1858f05b231d757c")	{ 5 fields }	Object
_id	ObjectId("570cfd6e1858f05b231d757c")	ObjectId
domestic_gross	25930652	Int32
poster_url	http://ia.media-imdb.com/images/M/MV5BMTkxMjU5MTY0Ml...	String
theatrical_release	2007-08-24 00:00:00.000Z	Date
title	The Nanny Diaries	String
(3) ObjectId("5704a95c1858f0109d93d448")	{ 5 fields }	Object
_id	ObjectId("5704a95c1858f0109d93d448")	ObjectId
domestic_gross	623357910	Int32
poster_url	http://ia.media-imdb.com/images/M/MV5BMTk2NTI1MTU4N1...	String
theatrical_release	2012-05-04 00:00:00.000Z	Date
title	Marvel's The Avengers	String
(4) ObjectId("570ce22c1858f05b231d748a")	{ 5 fields }	Object
_id	ObjectId("570ce22c1858f05b231d748a")	ObjectId
domestic_gross	191204754	Int32
poster_url	http://ia.media-imdb.com/images/M/MV5BMjE1MzcwNTE5OV...	String
theatrical_release	2007-03-23 00:00:00.000Z	Date
title	TMNT	String
(5) ObjectId("56fb161e1858f0109d93c744")	{ 5 fields }	Object
_id	ObjectId("56fb161e1858f0109d93c744")	ObjectId
domestic_gross	459005868	Int32
poster_url	http://ia.media-imdb.com/images/M/MV5BMTM4OGJmNWMt...	String
theatrical_release	2015-05-01 00:00:00.000Z	Date
title	Avengers: Age of Ultron	String
(6) ObjectId("570cdf351858f05b231d746d")	{ 5 fields }	Object
_id	ObjectId("570cdf351858f05b231d746d")	ObjectId
domestic_gross	3675753	Int32

Logs

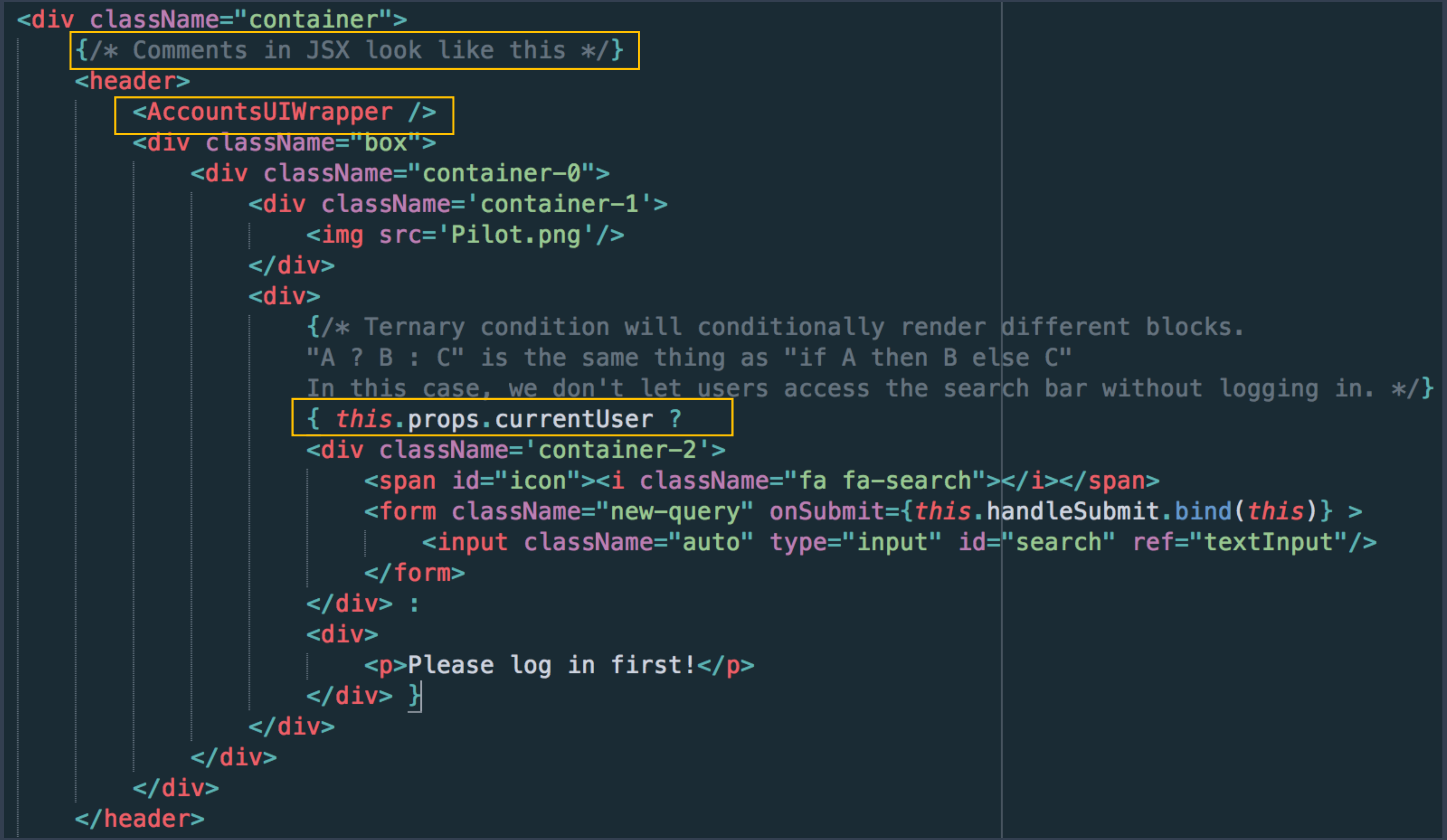
http://localhost:3000



Please log in first!

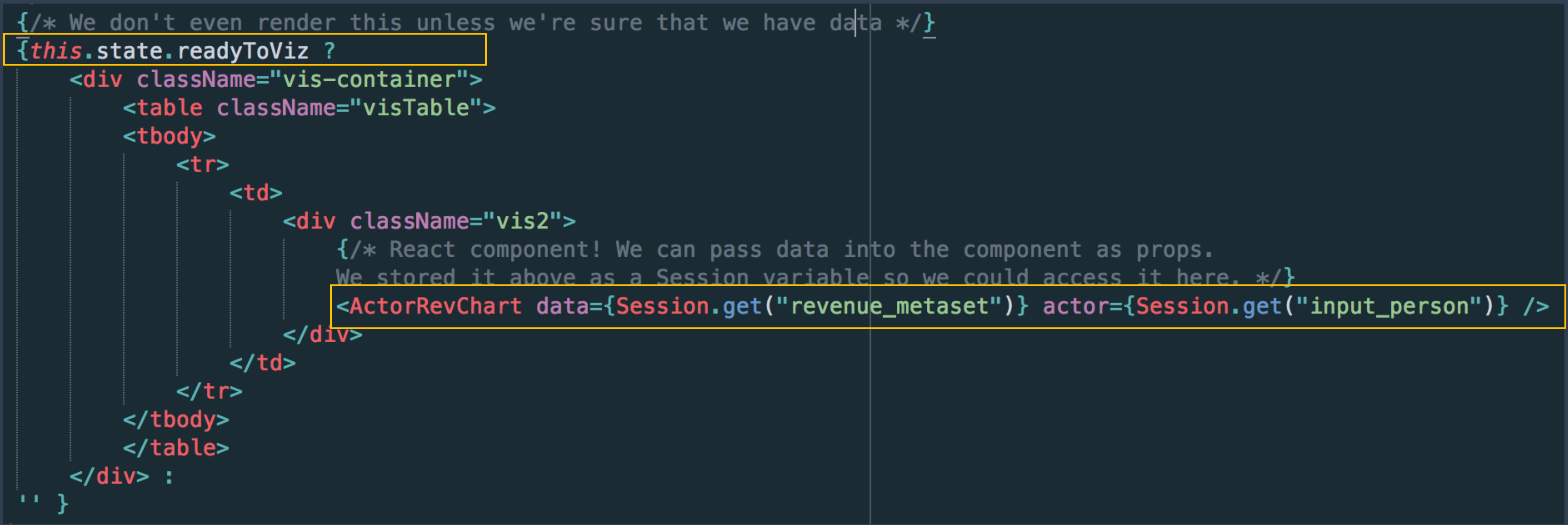

Pilot
localhost:3000
Alan
alan
pilot
Chris Evans Revenue
Revenue ($M)
600
550
500
450
400
350
300
250
200
150
100
50

Let’s look at code.



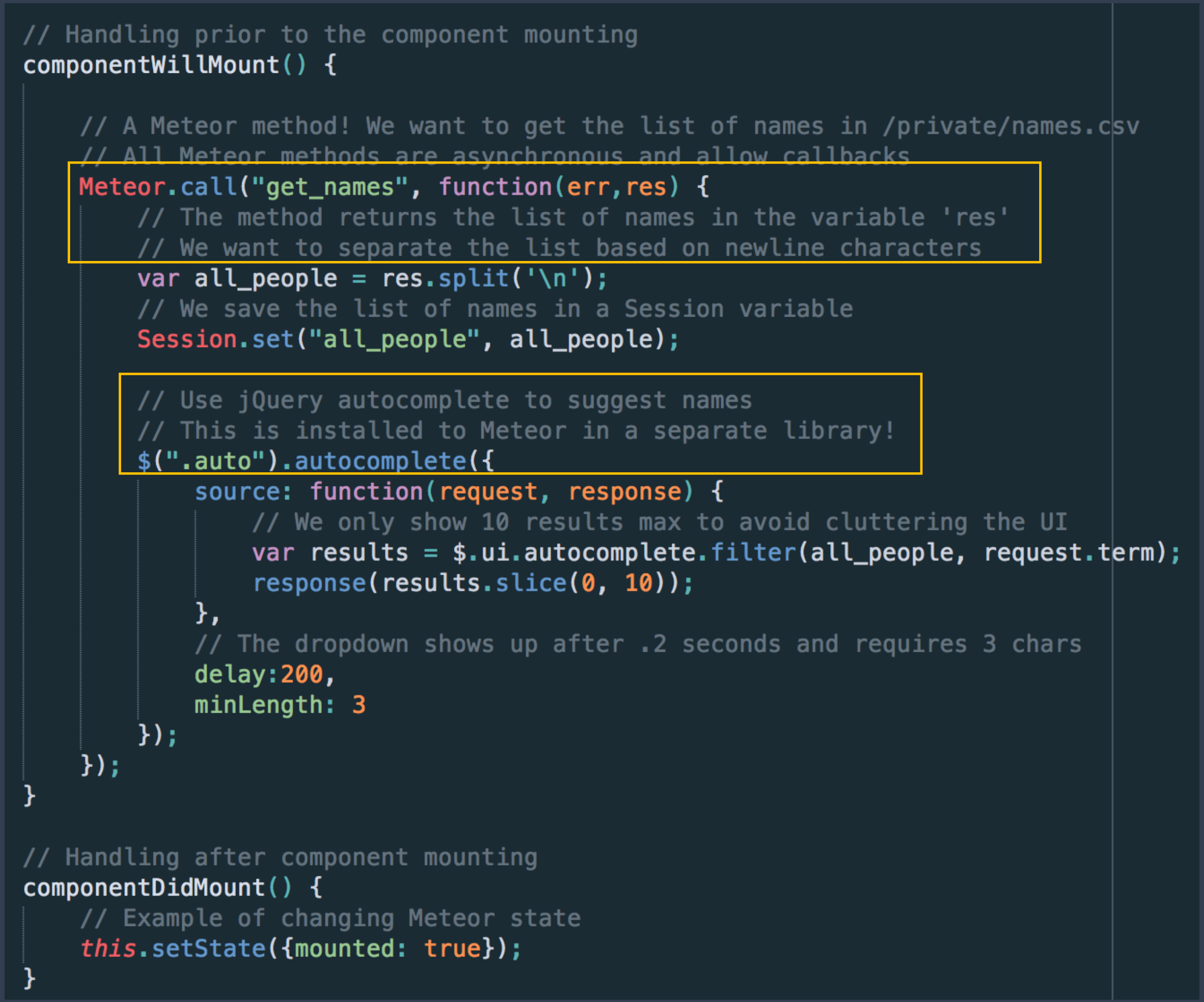
```
<div className="container">
    {/* Comments in JSX look like this */}
    <header>
        <AccountsUIWrapper />
        <div className="box">
            <div className="container-0">
                <div className='container-1'>
                    <img src='Pilot.png'/>
                </div>
                <div>
                    {/* Ternary condition will conditionally render different blocks.
                    "A ? B : C" is the same thing as "if A then B else C"
                    In this case, we don't let users access the search bar without logging in. */}
                    { this.props.currentUser ?
                    <div className='container-2'>
                        <span id="icon"><i className="fa fa-search"></i></span>
                        <form className="new-query" onSubmit={this.handleSubmit.bind(this)} >
                            <input className="auto" type="input" id="search" ref="textInput"/>
                        </form>
                    </div> :
                    <div>
                        <p>Please log in first!</p>
                    </div> }
                </div>
            </div>
        </div>
    </header>
```

/imports/ui/layouts/App.jsx



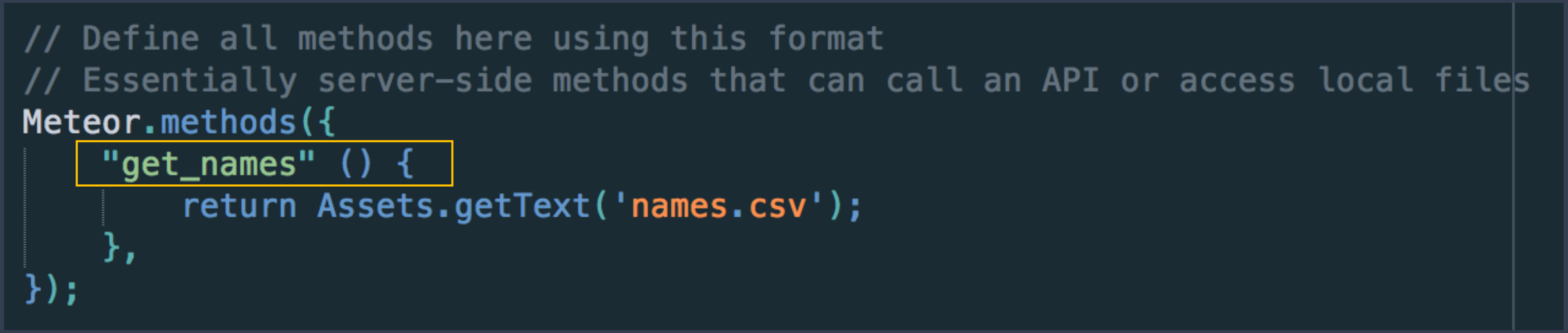
```
{/* We don't even render this unless we're sure that we have data */}
{this.state.readyToViz ?
    <div className="vis-container">
        <table className="visTable">
        <tbody>
            <tr>
                <td>
                    <div className="vis2">
                        {/* React component! We can pass data into the component as props.
                        We stored it above as a Session variable so we could access it here. */}
                        <ActorRevChart data={Session.get("revenue_metaset")} actor={Session.get("input_person")} />
                    </div>
                </td>
            </tr>
        </tbody>
        </table>
    </div> :
'' }
```

/imports/ui/layouts/App.jsx



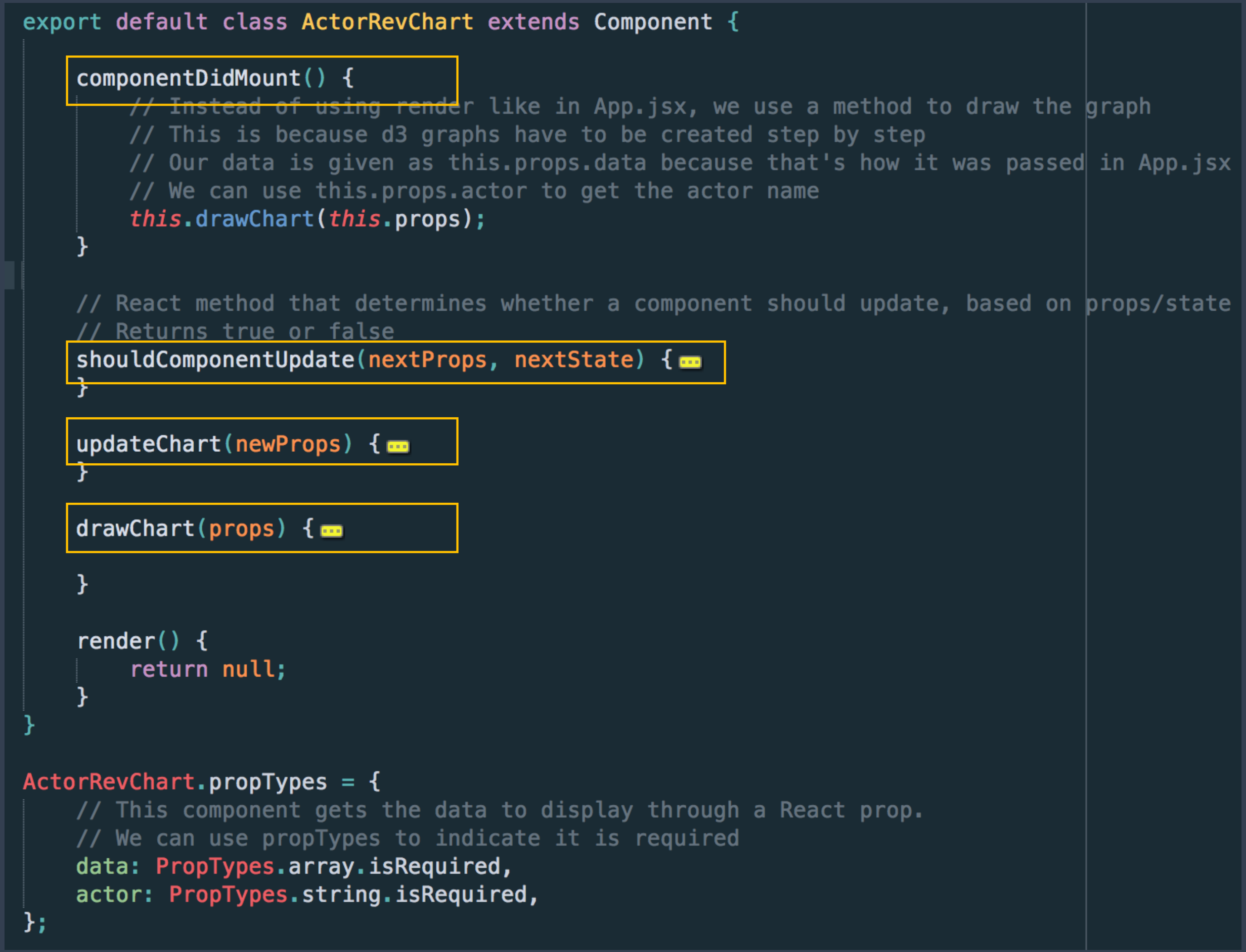
```
// Handling prior to the component mounting
componentWillMount() {

    // A Meteor method! We want to get the list of names in /private/names.csv
    // All Meteor methods are asynchronous and allow callbacks
    Meteor.call("get_names", function(err,res) {
        // The method returns the list of names in the variable 'res'
        // We want to separate the list based on newline characters
        var all_people = res.split('\n');
        // We save the list of names in a Session variable
        Session.set("all_people", all_people);

        // Use jQuery autocomplete to suggest names
        // This is installed to Meteor in a separate library!
        $(".auto").autocomplete({
            source: function(request, response) {
                // We only show 10 results max to avoid cluttering the UI
                var results = $.ui.autocomplete.filter(all_people, request.term);
                response(results.slice(0, 10));
            },
            // The dropdown shows up after .2 seconds and requires 3 chars
            delay:200,
            minLength: 3
        });
    });
}

// Handling after component mounting
componentDidMount() {
    // Example of changing Meteor state
    this.setState({mounted: true});
}
```

/imports/ui/layouts/App.jsx



```
// Define all methods here using this format
// Essentially server-side methods that can call an API or access local files
Meteor.methods({
    "get_names" () {
        return Assets.getText('names.csv');
    },
});
```

/imports/startup/server/films.js



```
export default class ActorRevChart extends Component {

    componentDidMount() {
        // Instead of using render like in App.jsx, we use a method to draw the graph
        // This is because d3 graphs have to be created step by step
        // Our data is given as this.props.data because that's how it was passed in App.jsx
        // We can use this.props.actor to get the actor name
        this.drawChart(this.props);
    }

    // React method that determines whether a component should update, based on props/state
    // Returns true or false
    shouldComponentUpdate(nextProps, nextState) {
    }

    updateChart(newProps) {
    }

    drawChart(props) {

    }

    render() {
        return null;
    }
}

ActorRevChart.propTypes = {
    // This component gets the data to display through a React prop.
    // We can use propTypes to indicate it is required
    data: PropTypes.array.isRequired,
    actor: PropTypes.string.isRequired,
};
```

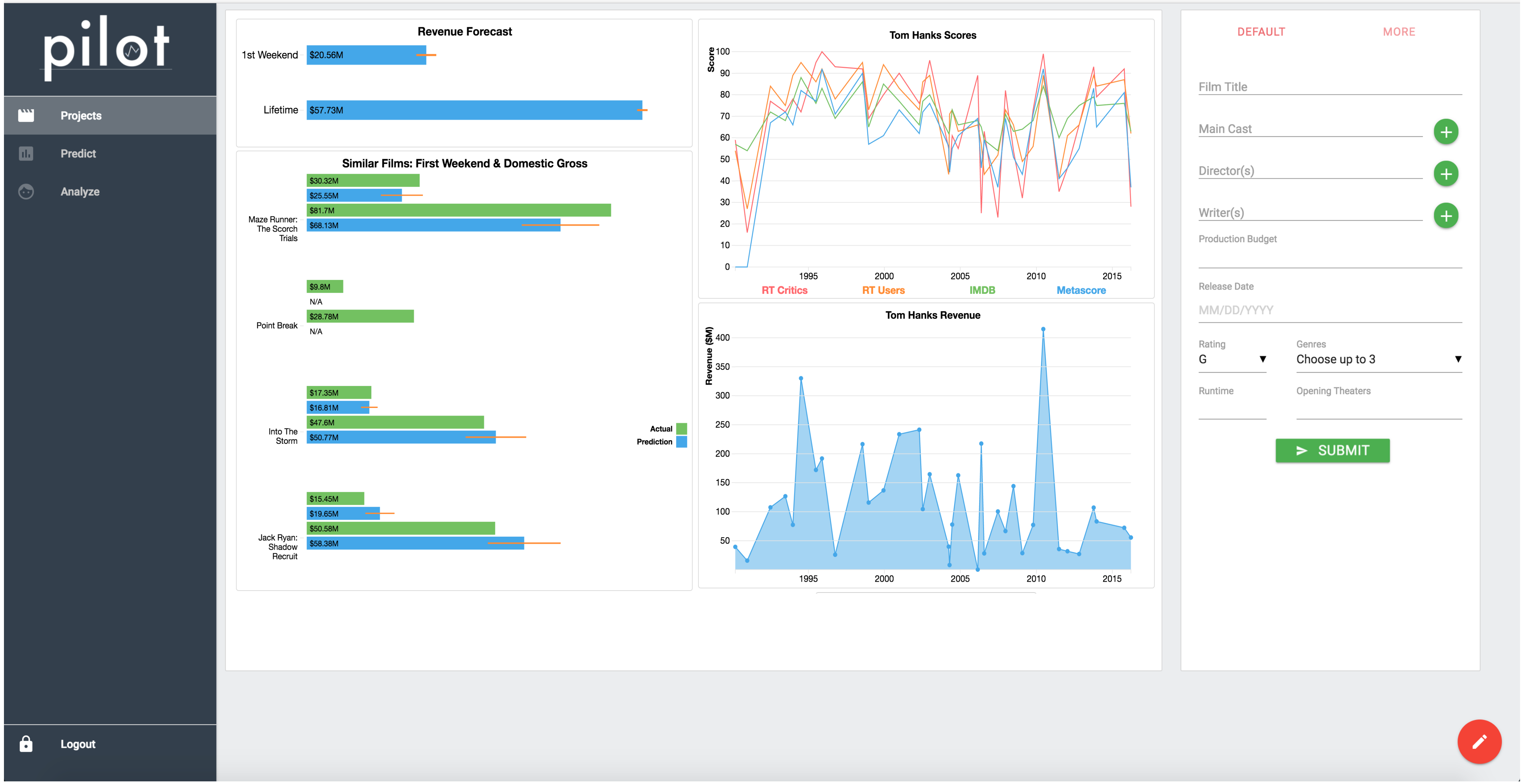
/imports/ui/components/Actor_RevChart.jsx

Next Steps

Jupyter and the numpy/scipy/pandas/scikit-learn stack allow us to perform rigorous data processing and machine learning in Python

Flask allows us to serve our machine learning as an API that interacts with Meteor via a server-side Meteor.method

AWS allows us to easily deploy our Flask API and Meteor app (as well as our MongoDB instance) to appropriately provisioned servers with the required libraries



Film Title

Main Cast

Director(s)

Writer(s)

Production Budget

Release Date
MM/DD/YYYY

Rating
G

Genres
Choose up to 3

Runtime

Opening Theaters

Questions?