react-supermodel

Supercharged REST-api wrapper for React.

Features

- Works out of the box
- Backend-agnostic
- Cache control
- Optimistic/Pessimistic strategies for UI/data updating
- Immutable state

Demo

- Gif
- https://codesandbox.io/s/04poy3y2kp

Installation

Through yarn

```
yarn add react-supermodel
```

Through NPM

```
npm install react-supermodel --save
```

Get started

1. Setup

The first thing you need to do is to init main config. Typically, your app's top-level component or main file like `index.js` will probably contains this config.

```
import { setConfig } from 'react-supermodel'
setConfig( options )
```

options.tree – *required*

Baobab instance. Make sure you have an `$api` cursor in your tree – it's required.

```
import Baobab from 'baobab'
import { setConfig } from 'react-supermodel'
const tree = new Baobab({
  $api: {}, //
  whateverYouNeedThere: {},
})
setConfig({ tree })
```

options.accept

Accept header for request. Default is `json` See – https://visionmedia.github.io/superagent/#setting-accept

options.auth

`Authorization` header. Default is empty string. Can be `string` or `function`. For example:

```
{
  auth: `Bearer: USER_TOKEN`,
  // Or using dynamic token
  auth: () => `Bearer: ${window.ComputeUserToken()}`,
}
```

options.prefix

Base URL prefix. Can be `string` or `function`. All model's requests will be prefixed with it. If you are going to use custom domain as prefix, make sure you know about CORS and credentials (see below).

```
setConfig({ prefix: '/api' })
// Or custom domain
setConfig({ prefix: 'http://customdomain.com/api' })
// Or custom function
setConfig({ prefix: () => `/api/${window.API_VERSION_CONFIG}` })
```

options.withCredentials

This option enables the ability to send cookies from the origin, however only when Access-Control-Allow-Origin is not a wildcard ("*"), and Access-Control-Allow-Credentials is "true". See – https://visionmedia.github.io/superagent/#cors

2. Create model

Once you've setuped supermodel's config, you'll need to create model. Basically model describes how to store/import/export and sync local data with an API provided data.

```
import { Model } from 'react-supermodel'
const UserModel = new Model({

  name: 'User', // This will be the name in props for connected component
  api: {
    get: '/users/:id', // :id will be replaced for real user is by
connector
    list: '/users',
    create: 'POST /users',  // Also you can speciafy request's method with
first-word prefix like GET, POST. DELETE, PUT
    delete: 'DELETE /users/:id',
    update: 'PUT /users/:id',
  }
})
```

modelOptions.name – required

`name` key contains the name of the model. It'll be passed to `Component` props via `connect` function.

modelOptions.idKey

`idKey` is the name for unique key of your objects. By default it is equal to `id`. For example, if your API has users collection contains an objects like `{user_id: 1, user_name: 'admin'}`, you should set up `idKey` as `user_id`.

modelOptions.dataItemKey

Default is `data`. Name of the key from your API response when requesting a single object.

modelOptions.dataListkey

Default is `data`. Name of the key from your API response when requesting a list.

modelOptions.api – required

The most important option. `api` is an object that describes how to work with your API. `api` has several predefined special keys which have a mapped dataflow methods like `get`, `list`, `create`, `delete`. You can also create your own methods. Working with connectors

Each property can be `string` contains an url pattern or an `object`. If you need to manipulate with response data, you should and object condiguration.

Here is an example how to add an extra property `full_name` to user object.

```
{
  api: {
    get: {
      url: '/user/:id', // the same if it was a string
      import( user ) {
        return {
          ...user,
          full_name: `${user.last_name} ${user.first_name}`
        }
      }
    }
  }
}
```

modelOptions.optimistic

– WIP: UI/data updating strategy

3. Create connection

```
import connect from 'react-supermodel'
import UserModel from './models/UserModel'
@connect(UserModel)
class App extends Component {}
```

That's it.

Working with connectors

Once you've added connector to your component, you a ready to use it.

Getting connector

```
@connect(UserModel)
class App extends Component {
  render(){
    const { User } = this.props // Get UserModel's connector from props
    // ...
  }
}
```

Model's connector provides some predefined methods like `get`, `list`, 'create', 'delete' and 'update' using own dataflow inside.

.get(id)

```
const user = User.get(1)
    return (
      <pre>{JSON.stringify(user)}</pre>
    )
```

Dataflow concepts

Examples

```
import connect from 'react-supermodel'
import UserModel from './models/UserModel'
@connect(UserModel)
class App extends Component {
  render(){
    // Get UserModel's connector from props
    const { User } = this.props
    // Getting users list from an API
    const users = User.list()
    // Showing progress indicator while users are loading
    if( users.isLoading ) {
      return 'Loading ...'
    }
    return (
      <ul>
        {users.data.map({data} =>
          <li key={data.id}>{data.name}</li>
        )}
      </ul>
    )
  }
}
```

Using Baobab as application's store

– WIP

Using with redux / etc

– WIP

Using without React

– WIP

Development & test

```
git clone https://github.com/ekorzun/react-supermodel.git
cd react-supermodel
yarn install
yarn test
```

Licence

MIT.