Contents

Chapter 1

Source Code

1.1 index.html

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
    <!--
      Notice the use of %PUBLIC_URL% in the tag above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head>
  <body>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start`.
      To create a production bundle, use `npm run build`.
    -->
  </body>
</html>
```

1.2 actionTypes.js

```
export { ANALYSE_NATURAL_TEXT } from './setup/analyseNaturalText.js';
export { UPDATE_MODEL_PREVIEW } from './setup/updateModelPreview.js';
export { GENERATE_WEBHOOK_URL } from './setup/generateWebhookURL.js';
export { NEW_SERVICE } from './setup/newService.js';
export { RECEIVE_WEBHOOK_URL } from './setup/receiveWebhookURL.js';
export { SELECT_DEVICE } from './setup/selectDevice.js';
export { SET_DEVICE_FLOW_DIRECTION } from './setup/setDeviceFlowDirection.js';
export { SET_SERVICE_CREATE_METHOD } from './setup/setServiceCreateMethod.js';
export { SET_SERVICE_NAME } from './setup/setServiceName.js';
export { SETUP_DEVICE_QUERY } from './setup/setupDeviceQuery.js';
export { NEXT_SCREEN } from './setup/nextScreen.js';
export { UPDATE_NATURAL_TEXT } from './setup/updateNaturalText.js';
export { AUTH_USER } from './auth/authUser.js';
export { UPDATE_USER } from './auth/updateUser.js';
export { AUTH_USER_RESULT } from './auth/authUserResult.js';
export { LOGOUT_USER } from './auth/logoutUser.js';
export { CHANGE_SIDEBAR_ITEM } from './dashboard/changeSidebarItem.js';
export { RECEIVE_SERVICE_LIST } from './auth/receiveServiceList.js';
export { SELECT_SERVICE } from './setup/selectService.js';
export { RECEIVE_SERVICE } from './setup/receiveService.js';
export { CHANGE_SELECTED_MODEL } from './dashboard/changeSelectedModel.js';
export { RECEIVE_ENTRY } from './dashboard/receiveEntry.js';
export { RECEIVE_MODEL } from './dashboard/receiveModel.js';
export { RECEIVE_ATTRIBUTE } from './dashboard/receiveAttribute.js';
export { DELETE_ENTRY_LOCALLY } from './dashboard/deleteEntryLocally.js';
export { UPDATE_VALUE_LOCALLY } from './dashboard/updateValueLocally.js';
export { UPDATE_SERVICE_LOCALLY } from './dashboard/updateServiceLocally.js';
export { UPDATE_MODEL_LOCALLY } from './dashboard/updateModelLocally.js';
export { UPDATE_ATTRIBUTE_LOCALLY } from './dashboard/updateAttributeLocally.js';
export { SELECT_ATTRIBUTE } from './dashboard/selectAttribute.js';
export { DELETE_MODEL_LOCALLY } from './dashboard/deleteModelLocally.js';
export { DELETE_ATTRIBUTE_LOCALLY } from './dashboard/deleteAttributeLocally.js';
```

1.3 authUser.js

```
import { authUserResult } from './authUserResult';
import { authenticateUser } from '../../utils/API';
import { saveToken } from '../../utils/Auth';
import { showError } from '../other/showError';

export function authUser(username, password) {
  return function (dispatch) {
    authenticateUser(username, password)
    .then(result => dispatch(authUserResult(result)))
    .then((result) => {
      if (result.success) {
        saveToken(result.token);
      }
    })
    .catch(e => showError(e.message));
  };
}
```

1.4 authUserResult.js

```
export const AUTH_USER_RESULT = 'AUTH_USER_RESULT';

export function authUserResult(result) {
  return {
    type: AUTH_USER_RESULT,
    success: result.success,
    errors: result.errors,
    token: result.token,
  };
}
```

1.5 getServiceList.js

```
import { receiveServiceList } from './receiveServiceList';
import { showError } from '../other/showError';
import * as API from '../../utils/API';

export function getServiceList() {
  return function (dispatch) {
    API.getServiceList()
    .then((result) => {
      dispatch(receiveServiceList(result));
    })
    .catch(e => dispatch(showError(e.message)));
  };
}
```

1.6 index.js

```
export { authUser } from './authUser.js';
export { updateUser } from './updateUser.js';
export { logoutUser } from './logoutUser.js';
```

1.7 logoutUser.js

```
import { removeToken } from '../../utils/Auth';

export const LOGOUT_USER = 'LOGOUT_USER';

export function logoutUser() {
  removeToken();
  return {
    type: LOGOUT_USER,
  };
}
```

1.8 receiveService.js

```
export const RECEIVE_SERVICE = 'RECEIVE_SERVICE';

export default function receiveService(data) {
  return {
    type: RECEIVE_SERVICE,
    data,
  };
}
```

1.9 receiveServiceList.js

```
export const RECEIVE_SERVICE_LIST = 'RECEIVE_SERVICE_LIST';

export function receiveServiceList(data) {
  return {
    type: RECEIVE_SERVICE_LIST,
    services: data.services,
  };
}
```

1.10 updateUser.js

```
export const UPDATE_USER = 'UPDATE_USER';

export function updateUser(username, password) {
  return {
    type: UPDATE_USER,
    username,
    password,
  };
}
```

1.11 changeDashboardPage.js

```
import { push } from 'react-router-redux';
import { changeSidebarItem } from './changeSidebarItem';

export const CHANGE_DASHBOARD_PAGE = 'CHANGE_DASHBOARD_PAGE';

export function changeDashboardPage(index, item) {
  return function (dispatch, getState) {
    dispatch(changeSidebarItem(index));
    dispatch(push(item.path));
  };
}
```

1.12 changeSelectedModel.js

```
export const CHANGE_SELECTED_MODEL = 'CHANGE_SELECTED_MODEL';

export const changeSelectedModel = id => ({
  type: CHANGE_SELECTED_MODEL,
  id,
});
```

1.13 changeSidebarItem.js

```
export const CHANGE_SIDEBAR_ITEM = 'CHANGE_SIDEBAR_ITEM';

export const changeSidebarItem = index => ({
  type: CHANGE_SIDEBAR_ITEM,
  index,
});
```

1.14 createAttribute.js

```
import { push } from 'react-router-redux';
import { postAttribute } from '../../utils/API';
import { showError } from '../other/showError';
import { receiveAttribute } from './receiveAttribute';


export function createAttribute(model) {
  return function (dispatch, getState) {
    postAttribute({
      model,
      name: '',
      type: 'string',
      required: false,
      multiple: false,
    })
    .then((result) => {
      if (result.success) {
        dispatch(receiveAttribute(result.attribute));
      } else {
        showError(result.error);
      }
    })
    .catch(e =>
    showError(e));
  };
}
```

1.15 createEntry.js

```
import { push } from 'react-router-redux';
import { changeSidebarItem } from './changeSidebarItem';
import { postEntry } from '../../utils/API';
import { showError } from '../other/showError';
import { receiveEntry } from './receiveEntry';


export function createEntry(index, item) {
  return function (dispatch, getState) {
    const state = getState().toJS();

    const model = state.dashboard.selectedModel ||
     state.serviceById[state.user.currentServiceId].Models[0];
    postEntry(model)
    .then((result) => {
      if (result.success) {
        dispatch(receiveEntry(result.entry));
      } else {
        showError(result.error);
      }
    })
    .catch(e =>
    showError(e));
  };
}
```

1.16 createModel.js

```
import { push } from 'react-router-redux';
import { postModel } from '../../utils/API';
import { showError } from '../other/showError';
import { receiveModel } from './receiveModel';


export function createModel() {
  return function (dispatch, getState) {
    const state = getState().toJS();

    postModel({
      service: state.user.currentServiceId,
      name: '',
    })
    .then((result) => {
      if (result.success) {
        dispatch(receiveModel(result.model));
      } else {
        showError(result.error);
      }
    })
    .catch(e =>
    showError(e));
  };
}
```

1.17 deleteAttribute.js

```
import * as API from '../../utils/API';
import { showError } from '../other/showError';
import { deleteAttributeLocally } from './deleteAttributeLocally';


export function deleteAttribute(id) {
  return function (dispatch) {
    dispatch(deleteAttributeLocally(id));
    API.deleteAttribute({
      id,
    })
    .then((result) => {
      if (!result.success) {
        showError(result.error);
      }
    })
    .catch(e =>
    showError(e));
  };
}
```

1.18 deleteAttributeLocally.js

```
export const DELETE_ATTRIBUTE_LOCALLY = 'DELETE_ATTRIBUTE_LOCALLY';

export const deleteAttributeLocally = id => ({
  type: DELETE_ATTRIBUTE_LOCALLY,
  id,
});
```

1.19 deleteEntry.js

```
import { push } from 'react-router-redux';
import { changeSidebarItem } from './changeSidebarItem';
import * as API from '../../utils/API';
import { showError } from '../other/showError';
import { deleteEntryLocally } from './deleteEntryLocally';


export function deleteEntry(id) {
  return function (dispatch, getState) {
    const entry = getState().get('entryById').toJS()[id];
    console.log('this is ', getState().get('entryById').toJS(), entry, id);
    API.deleteEntry(id)
    .then((result) => {
      if (result.success) {
        dispatch(deleteEntryLocally(entry));
      } else {
        showError(result.error);
      }
    })
    .catch(e =>
    showError(e));
  };
}
```

1.20 deleteEntryLocally.js

```
export const DELETE_ENTRY_LOCALLY = 'DELETE_ENTRY_LOCALLY';

export const deleteEntryLocally = entry => ({
  type: DELETE_ENTRY_LOCALLY,
  entry,
});
```

1.21 deleteModel.js

```
import * as API from '../../utils/API';
import { showError } from '../other/showError';
import { deleteModelLocally } from './deleteModelLocally';


export function deleteModel(id) {
  return function (dispatch) {
    dispatch(deleteModelLocally(id));
    API.deleteModel({
      id,
    })
    .then((result) => {
      if (!result.success) {
        showError(result.error);
      }
    })
    .catch(e =>
    showError(e));
  };
}
```

1.22 deleteModelLocally.js

```
export const DELETE_MODEL_LOCALLY = 'DELETE_MODEL_LOCALLY';

export const deleteModelLocally = id => ({
  type: DELETE_MODEL_LOCALLY,
  id,
});
```

1.23 receiveAttribute.js

```
export const RECEIVE_ATTRIBUTE = 'RECEIVE_ATTRIBUTE';

export const receiveAttribute = attribute => ({
  type: RECEIVE_ATTRIBUTE,
  attribute,
});
```

1.24 receiveEntry.js

```
export const RECEIVE_ENTRY = 'RECEIVE_ENTRY';

export const receiveEntry = entry => ({
  type: RECEIVE_ENTRY,
  entry,
});
```

1.25 receiveModel.js

```
export const RECEIVE_MODEL = 'RECEIVE_MODEL';

export const receiveModel = model => ({
  type: RECEIVE_MODEL,
  model,
});
```

1.26 selectAttribute.js

```
export const SELECT_ATTRIBUTE = 'SELECT_ATTRIBUTE';

export const selectAttribute = id => ({
  type: SELECT_ATTRIBUTE,
  id,
});
```

1.27 updateAttribute.js

```
import * as API from '../../utils/API';
import { showError } from '../other/showError';
import { updateAttributeLocally } from './updateAttributeLocally';

export function updateAttribute(id, changes) {
  return function (dispatch) {
    dispatch(updateAttributeLocally(id, changes));
    API.patchAttribute({ id, ...changes })
    .then((result) => {
      if (!result.success) {
        showError(result.error);
      }
    })
    .catch(e => showError(e));
  };
}
```

1.28 updateAttributeLocally.js

```
export const UPDATE_ATTRIBUTE_LOCALLY = 'UPDATE_ATTRIBUTE_LOCALLY';

export const updateAttributeLocally = (id, changes) => ({
  type: UPDATE_ATTRIBUTE_LOCALLY,
  id,
  changes,
});
```

1.29 updateModel.js

```
import * as API from '../../utils/API';
import { showError } from '../other/showError';
import { updateModelLocally } from './updateModelLocally';

export function updateModel(id, name) {
  return function (dispatch) {
    dispatch(updateModelLocally(id, name));
    API.patchModel({ id, name })
    .then((result) => {
      if (!result.success) {
        showError(result.error);
      }
    })
    .catch(e => showError(e));
  };
}
```

1.30 updateModelLocally.js

```
export const UPDATE_MODEL_LOCALLY = 'UPDATE_MODEL_LOCALLY';

export const updateModelLocally = (id, name) => ({
  type: UPDATE_MODEL_LOCALLY,
  id,
  name,
});
```

1.31 updateService.js

```
import { push } from 'react-router-redux';
import { changeSidebarItem } from './changeSidebarItem';
import * as API from '../../utils/API';
import { showError } from '../other/showError';
import { receiveEntry } from './receiveEntry';


export function updateService(changes) {
  return function (dispatch, getStore) {
    const id = getStore().toJS().user.currentServiceId;
    API.updateService(id, changes)
    .then((result) => {
      if (!result.success) {
        showError(result.error);
      }
    })
    .catch(e =>
    showError(e));
  };
}
```

1.32 updateServiceLocally.js

```
export const UPDATE_SERVICE_LOCALLY = 'UPDATE_SERVICE_LOCALLY';

export const updateServiceLocally = changes => ({
  type: UPDATE_SERVICE_LOCALLY,
  changes,
});
```

1.33 updateValue.js

```
import { push } from 'react-router-redux';
import { changeSidebarItem } from './changeSidebarItem';
import * as API from '../../utils/API';
import { showError } from '../other/showError';
import { receiveEntry } from './receiveEntry';


export function updateValue(entryId, attributeId, value) {
  return function (dispatch) {
    API.updateValue(entryId, attributeId, value)
    .then((result) => {
      if (!result.success) {
        showError(result.error);
      }
    })
    .catch(e =>
    showError(e));
  };
}
```

1.34 updateValueLocally.js

```
export const UPDATE_VALUE_LOCALLY = 'UPDATE_VALUE_LOCALLY';

export const updateValueLocally = (entry, id, value) => ({
  type: UPDATE_VALUE_LOCALLY,
  entry,
  id,
  value,
});
```

1.35 showError.js

```
export const SHOW_ERROR = 'SHOW_ERROR';

export function showError(message) {
  console.error(`${message}`);
  return {
    type: SHOW_ERROR,
    message,
  };
}
```

1.36 analyseNaturalText.js

```
import { updateModelPreview } from './updateModelPreview';
import { updateNaturalText } from './updateNaturalText';
import { extractModelFromText } from '../../utils/API';

export const ANALYSE_NATURAL_TEXT = 'ANALYSE_NATURAL_TEXT';


export function analyseNaturalText(text) {
  return function (dispatch) {
    dispatch(updateNaturalText(text));

    return extractModelFromText(text)
      .then(result => dispatch(updateModelPreview(result)))
      .catch(console.log);
  };
}
```

1.37 analyseSpreadsheet.js

```
// @flow

import { updateModelPreview } from './updateModelPreview';
import { showError } from '../other/showError';
import { postAnalyzeSpreadsheet } from '../../utils/API';

export function analyseSpreadsheet(file) {
  return function (dispatch) {
    console.log(postAnalyzeSpreadsheet);
    return postAnalyzeSpreadsheet(file)
      .then(result => dispatch(updateModelPreview(result)))
      .catch(showError);
  };
}
```

1.38 createService.js

```
import { push } from 'react-router-redux';
import { postService } from '../../utils/API';
import { nextScreen } from './nextScreen';
import { showError } from '../other/showError';
import { receiveService } from './receiveService';

export const CREATE_SERVICE = 'CREATE_SERVICE';

export function createService() {
  return function (dispatch, getState) {
    const state = getState();

    const setup = state.get('setup');
    return postService(setup.get('name'), setup.get('modelDefinitionPreview'))
      .then((result) => {
        if (result.success) {
          dispatch(receiveService(result.service));
          dispatch(push('/service/dashboard'));
        } else {
          dispatch(showError(result.error));
        }
      })
      .catch(e =>
      showError(e));
  };
}
```

1.39 generateWebhookURL.js

```
// @flow

export const GENERATE_WEBHOOK_URL = 'GENERATE_WEBHOOK_URL';

export function generateWebhookURL() {
  return {
    type: GENERATE_WEBHOOK_URL,
  };
}
```

1.40 index.js

```
export { analyseNaturalText } from './analyseNaturalText.js';
export { updateModelPreview } from './updateModelPreview.js';
export { generateWebhookURL } from './generateWebhookURL.js';
export { setServiceName } from './setServiceName.js';
export { setServiceCreateMethod } from './setServiceCreateMethod.js';
export { nextScreen } from './nextScreen.js';
export { newService } from './newService.js';
export { createService } from './createService.js';
export { selectService } from './selectService.js';
```

1.41 newService.js

```
import { push } from 'react-router-redux';

export const NEW_SERVICE = 'NEW_SERVICE';

export function newService() {
  return (dispatch) => {
    dispatch(push('/service/setup'));
  };
}
```

1.42 nextScreen.js

```
// @flow

export const NEXT_SCREEN = 'NEXT_SCREEN';

export function nextScreen() {
  return {
    type: NEXT_SCREEN,
  };
}
```

1.43 receiveService.js

```
// @flow

export const RECEIVE_SERVICE = 'RECEIVE_SERVICE';

export function receiveService(service) {
  return {
    type: RECEIVE_SERVICE,
    service,
  };
}
```

1.44 receiveWebhookURL.js

```
// @flow

export const RECEIVE_WEBHOOK_URL = 'RECEIVE_WEBHOOK_URL';

export function receiveWebhookURL(url) {
  return {
    type: RECEIVE_WEBHOOK_URL,
    url,
  };
}
```

1.45 selectDevice.js

```
// @flow

export const SELECT_DEVICE = 'SELECT_DEVICE';

export function selectDevice(device: number) {
  return {
    type: SELECT_DEVICE,
    device,
  };
}
```

1.46 selectService.js

```
import { push } from 'react-router-redux';
export const SELECT_SERVICE = 'SELECT_SERVICE';


export function selectService(id) {
  return (dispatch) => {
    dispatch({
      type: SELECT_SERVICE,
      id,
    });
    dispatch(push('/service/dashboard'));
  };
}
```

1.47 setDeviceFlowDirection.js

```
// @flow

export const SET_DEVICE_FLOW_DIRECTION = 'SET_DEVICE_FLOW_DIRECTION';

type deviceFlowDirection = 'query' | 'webhook';

export function setDeviceFlowDirection(direction: deviceFlowDirection) {
  return {
    type: SET_DEVICE_FLOW_DIRECTION,
    direction,
  };
}
```

1.48 setServiceCreateMethod.js

```
// @flow

export const SET_SERVICE_CREATE_METHOD = 'SET_SERVICE_CREATE_METHOD';

type methodString = 'strach' | 'spreadsheet' | 'device';

export function setServiceCreateMethod(method: methodString) {
  return {
    type: SET_SERVICE_CREATE_METHOD,
    method,
  };
}
```

1.49 setServiceName.js

```
// @flow

export const SET_SERVICE_NAME = 'SET_SERVICE_NAME';

export function setServiceName(name) {
  return {
    type: SET_SERVICE_NAME,
    name,
  };
}
```

1.50 setupDeviceQuery.js

```
// @flow

export const SETUP_DEVICE_QUERY = 'SETUP_DEVICE_QUERY';

export function setupDeviceQuery(
  url: string,
  method: string,
  attributes: [{ string: string }],
  interval: number,
) {
  return {
    type: SETUP_DEVICE_QUERY,
    url,
    method,
    attributes,
    interval,
  };
}
```

1.51 updateModelPreview.js

```

export const UPDATE_MODEL_PREVIEW = 'UPDATE_MODEL_PREVIEW';

export function updateModelPreview(preview) {
  return {
    type: UPDATE_MODEL_PREVIEW,
    preview,
  };
}
```

1.52 updateNaturalText.js

```

export const UPDATE_NATURAL_TEXT = 'UPDATE_NATURAL_TEXT';

export function updateNaturalText(text) {
  return {
    type: UPDATE_NATURAL_TEXT,
    text,
  };
}
```

1.53 AuthForm.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import TextInput from './TextInput';
import Button from './Button';
import { Color } from './StyleConstant';

const style = {
  width: 140,
  field: {
    marginBottom: 5,
  },
  error: {
    margin: 4,
    fontSize: 14,
    color: Color.red,
  }
}

const AuthForm = ({
  onSubmit,
  onChange,
  errors = {},
  username,
  password,
}) => (
  <div>
    <h1>Login</h1>
    <form action="/" onSubmit={(e) => {
      e.preventDefault();
      onSubmit({username, password});
    }} method="post">
      <div style={style.field}>
        <TextInput
          name="username"
          placeholder="Username"
          onChange={username => onChange({username})}
          text={username}
        />
        <p style={style.error}>{errors.username}</p>
      </div>
      <div style={style.field}>
        <TextInput
          name="password"
          placeholder="Password"
          onChange={password => onChange({password})}
          text={password}
        />
        <p style={style.error}>{errors.password}</p>
      </div>
      <div>
```

```
        <Button type="submit" text="Log In"/>
      </div>
    </form>
  </div>
)

AuthForm.propTypes = {
  onSubmit: PropTypes.func.isRequired,
  onChange: PropTypes.func.isRequired,
  errors: PropTypes.array,
  username: PropTypes.string.isRequired,
  password: PropTypes.string.isRequired
};

export default AuthForm;
```

1.54 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import AuthForm from './AuthForm';

describe('<AuthForm />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<AuthForm
      errors={[]}
      username="username"
      password="password"
    />);
  });
});
```

1.55 Button.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import { Color, Dimensions } from './StyleConstant';

const activeStyle = {
  backgroundColor: Color.greenDark,
  border: 'none',
  outline: 'none',
};

const style = {
  base: {
    backgroundColor: Color.green,
    minWidth: Dimensions.fieldWidth,
    height: Dimensions.fieldHeight,
    border: 'none',
    borderRadius: Dimensions.borderRadius,
    cursor: 'pointer',
    transition: `${Dimensions.transitionTime.normal} background-color`,
    fontSize: Dimensions.fontSize.normal,
    color: Color.whiteText,
    ':hover': {
      backgroundColor: Color.greenLight,
    },
    ':active': activeStyle,
    ':focus': activeStyle,
  },
  isDisabled: {
    pointerEvents: 'none',
    backgroundColor: Color.grey,
  },
};

const Button = ({ text, onClick, isDisabled, type }) => (
  <button type={type} onClick={onClick} style={[style.base, isDisabled ? style.isDisabled : {}]}>
    {text}
  </button>
);

Button.PropTypes = {
  text: PropTypes.string,
  onClick: PropTypes.func,
  isDisabled: PropTypes.bool,
  type: PropTypes.string,
};


export default Radium(Button);
```

1.56 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import Button from './Button';

describe('<Button />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<Button
      text="Next"
      isDisabled={false}
    />);
  });
});
```

1.57 About.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import TopBar from '../TopBar';
import { Color } from '../../StyleConstant';
import TextInput from '../../TextInput';
import Button from '../../Button';

const style = {
  base: {
    height: '100vh',
    overflowY: 'auto',
    padding: 30,
  },
  h3: {
    padding: 0,
    margin: 0,
  },
  label: {
    marginTop: 10,
  },
  field: {
    marginBottom: 10,
    marginTop: 4,
  },
};

const metaExample = {
  name: 'Pets',
  url: 'pets',
  author: 'Martin Hartt',
  isPublic: false,
};

const About = ({ name, meta = metaExample, onChange = () => {} }) => <div>
  <TopBar name={name} />
  <div style={style.base}>
    <h3 style={style.h3}>About</h3>
    {
      Object.keys(meta).map(key =>
        (typeof (meta[key].value) === 'boolean') ?
          <div>
            <input id={key} type="checkbox" checked={meta[key].value === true} onChange={e => onChange({ [key]: !!e.target.checked })} />
            <label htmlFor={key} style={style.label}>{key}</label>
          </div>
                 :
          <div>
            <label style={style.label} htmlFor={key}>{meta[key].label}</label>
            <div style={style.field}>
              <TextInput id={key} text={meta[key].value} onChange={value => onChange({ [key]: value })} />
            </div>
          </div>,

      )
    }

  </div>
</div>;

About.propTypes = {
  name: PropTypes.string,
  meta: PropTypes.shape({
    name: PropTypes.string,
    url: PropTypes.string,
    author: PropTypes.string,
    public: PropTypes.bool,
  }),
  onChange: PropTypes.func,
};

export default About;
```

1.58 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import About from './About';

describe('<About />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<About
      name="Example"
      meta={{
        name: 'Example',
        url: 'example',
        author: 'Martin Hartt',
        public: true,
      }}
    />);
  });
});
```

1.59 Dashboard.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Logo from '../Logo';
import SidebarContainer from '../../containers/dashboard/SidebarContainer';
import { Color, lightBorder } from '../StyleConstant';


const style = {
  base: {
    display: 'flex',
  },
  sidebar: {
    width: 230,
    borderRight: lightBorder,
    height: '100vh',
  },
  main: {
    flex: 1,
  },
  logo: {
    textAlign: 'center',
    padding: '20px 0',
    borderBottom: lightBorder,
  },
};

const Dashboard = ({ children }) => <div style={style.base}>
  <div style={style.sidebar}>
    <div style={style.logo}>
      <Logo />
    </div>
    <SidebarContainer />
  </div>
  <div style={style.main}>
    {children}
  </div>
</div>;

Dashboard.propTypes = {
  children: PropTypes.node,
};

export default Dashboard;
```

1.60 test.js

```


import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import Dashboard from './Dashboard';

describe('<Dashboard />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<Dashboard>
      <p>Hello</p>
    </Dashboard>);
  });
});
```

1.61 Column.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';

const style = {
  base: {
    minWidth: 70,
    // height: '80%',
    marginLeft: 0,
    textAlign: 'center',
    marginRight: 5,
    width: 190,
  },
  item: {
    borderRadius: 3,
    cursor: 'pointer',
    ':hover': {
      backgroundColor: '#EEE',
    },
    border: 'none',
    height: 45,
    fontSize: 18,
    ':focus': {
      outline: 0,
      border: 0,
    },
  },
  first: {
    width: 80,
  },
};

const Column = ({ value, isItem, onChange, first = false }) =>
  isItem ?
    <input style={[style.base, style.item, first && style.first]} value={value} onChange={onChange} /> :
    <div style={[style.base, first && style.first]}>{value}</div>;


Column.propTypes = {
  value: PropTypes.string.isRequired,
  isItem: PropTypes.bool.isRequired,
  onChange: PropTypes.func,
};

export default Radium(Column);
```

1.62 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import Column from './Column';

describe('<Column />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<Column
      value="Hello"
      isItem
    />);
  });
});
```

1.63 Entries.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import TopBar from '../TopBar';
import Tabs from './Tabs';
import RowHeader from './RowHeader';
import Column from './Column';
import Row from './Row';
import capitalizeString from '../../../utils/capitalizeString';
import { Color } from '../../StyleConstant';

const style = {
  base: {
    backgroundColor: Color.lighterGrey,
    overflowX: 'auto',
  },
  main: {
    height: 'calc(100vh - 77px)',
    overflowY: 'auto',
  },
};


function decode(string, type) {
  switch (type) {
    case 'integer':
      return parseInt(string, 10);
    case 'float':
      return parseFloat(string);
    default:
      return string;
  }
}

const Entries = ({ name, entries = [], attributes = [], headers = [], onSelected, onDelete, onCreate, onUpdate }) =>
  <div style={style.base}>
    <TopBar name={name} onNew={() => onCreate()} />
    <div style={style.main}>
      <Tabs headers={headers} onSelected={onSelected} />
      <RowHeader>
        <Column key="headerid" value="ID" first />
        {attributes.map(attr => <Column value={capitalizeString(attr.name)} key={attr.id} />)}
      </RowHeader>
      {entries.map(entry =>
        <Row key={entry.id} onDelete={() => onDelete(entry.realId)}>
          <Column key={`${entry.id}.id`} value={entry.id} first />

          {attributes.map(attr =>
            <Column
              key={`${entry.realId}.${attr.id}`}
              value={entry[attr.name] ? decode(entry[attr.name].value, attr.type) : ''}
              isItem
              onChange={e => onUpdate(entry.realId, attr.id, e.target.value, entry[attr.name].id)}
            />)}
        </Row>,
      )}
    </div>
  </div>;

Entries.propTypes = {
  name: PropTypes.string.isRequired,
  entries: PropTypes.array.isRequired,
  attributes: PropTypes.array.isRequired,
  headers: PropTypes.array.isRequired,
  onSelected: PropTypes.func,
  onDelete: PropTypes.func,
  onCreate: PropTypes.func,
  onUpdate: PropTypes.func,
};

export default Entries;
```

1.64 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import Entries from './Entries';

describe('<Entries />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<Entries
      name="Example"
      entries={[]}
      attributes={[]}
      headers={[]}
      onSelected={false}
    />);
  });
});
```

1.65 Row.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import rowStyle from './rowStyle';
import RoundButton from '../../RoundButton';
import { Color } from '../../StyleConstant';


const style = {
  base: rowStyle,
};

const Row = ({ children, onDelete }) => <div style={style.base}>
  {children}
  <RoundButton onClick={onDelete} text="remove" color={Color.red} />

</div>;

Row.propTypes = {
  children: PropTypes.node,
  onDelete: PropTypes.func,
};

export default Row;
```

1.66 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import Row from './Row';

describe('<Row />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<Row>
      <p>Hello</p>
    </Row>);
  });
});
```

1.67 RowHeader.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import rowStyle from './rowStyle';
import { Color } from '../../StyleConstant';

const style = {
  base: [
    rowStyle,
    {
      backgroundColor: Color.lighterGrey,
    },
  ],
};

const RowHeader = ({ children }) => <div style={style.base}>{children}</div>;

RowHeader.propTypes = {
  children: PropTypes.node,
};

export default Radium(RowHeader);
```

1.68 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import RowHeader from './RowHeader';

describe('<RowHeader />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<RowHeader>
      <p>Test</p>
    </RowHeader>);
  });
});
```

1.69 rowStyle.js

```
import { Color } from '../../StyleConstant';

const rowStyle = {
  display: 'flex',
  flexDirection: 'row',
  height: 57,
  alignItems: 'center',
  backgroundColor: Color.whiteText,
  borderBottom: `2px solid ${Color.lightGrey}`
};

export default rowStyle;
```

1.70 Tabs.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import { Color } from '../../StyleConstant';

const style = {
  base: {
    display: 'flex',
    flexDirection: 'row',
    backgroundColor: Color.whiteText,
    borderBottom: `2px solid ${Color.lightGrey}`,
  },
  tab: {
    display: 'inline-block',
    cursor: 'pointer',
    minWidth: 100,
    height: 56,
    display: 'flex',
    justifyContent: 'center',
    alignItems: 'center',
    borderTop: '3px solid transparent',
  },
  selected: {
    backgroundColor: Color.lighterGrey,
    borderTop: `3px solid ${Color.green}`,
  },
};

const Tabs = ({ headers, onSelected }) => <div style={style.base}>
  {headers.map(header =>
    <div
      key={header.text}
      style={[style.tab, header.selected && style.selected]}
      onClick={() => onSelected(header.id)}
    >
      {header.text}
    </div>)}
</div>;

Tabs.propTypes = {
  headers: PropTypes.array,
};

export default Radium(Tabs);
```

1.71 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import Tabs from './Tabs';

describe('<Tabs />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<Tabs
      headers={[]}
    />);
  });
});
```

1.72 Pages.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import TopBar from '../TopBar';
import { Color } from '../../StyleConstant';
import TextInput from '../../TextInput';
import Button from '../../Button';

const style = {
  base: {
    backgroundColor: Color.lighterGrey,
    height: '100vh',
    overflowY: 'auto',
  },
  page: {
    backgroundColor: Color.whiteText,
    margin: 20,
    padding: 15,
    borderRadius: 5,
    border: `2px solid ${Color.lightGrey}`,
  },
  title: {
    margin: 0,
    padding: 0,
    fontWeight: 600,
  },
  method: {
    color: Color.green,
  },
  description: {
    padding: 0,
    marginBottom: 0,
  },
  label: {
  },
  field: {
    marginBottom: 10,
    marginTop: 4,
  },
};


const pagesExamples = [
  {
    method: 'GET',
    path: '/owners',
    operation: 'find',
    model: 'owners',
  },
  {
    method: 'GET',
    path: '/pets/{id}',
    operation: 'findById',
    model: 'pets',
  },
];

function bind(model, action, onChange) {
  const name = `${model.name}`;
  const prop = `${action.prop}`;
  console.log('bind', name, prop);
  return console.log.bind(console, name, prop);// onChange(name, { [prop]: !!e.target.checked });
}

const Pages = ({ name, models = [], actions = [], onChange, urlPrefix }) => <div style={style.base}>
  <TopBar name={name} />
  {models.map((model, modelIndex) => <div style={style.page}>
    <h3 style={style.title}>{model.name}</h3>
    {actions.map(action => (
      <div>
        <input
          id={action}
          type="checkbox"
          checked={model[action.prop].value === true}
          onChange={bind(model, action)}
        />
        <label
          htmlFor={action}
          style={style.label}
        >
          {action.label} ({action.method} {urlPrefix}{model.name})
        </label>
      </div>
    ))}
  </div>)}
</div>;

Pages.propTypes = {
  name: PropTypes.string,
  models: PropTypes.array,
  actions: PropTypes.array,
  onChange: PropTypes.func,
  urlPrefix: PropTypes.string,
};

export default Pages;
```

1.73 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import Pages from './Pages';

describe('<Pages />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<Pages
      name="Example"
      models={[]}
      actions={[]}
      urlPrefix="example"
    />);
  });
});
```

1.74 Sidebar.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import SidebarItem from './SidebarItem';

const itemsExample = [
  { name: 'Structure', path: '/service/X/structure', selected: true },
  { name: 'Entries', path: '/service/X/entries' },
  { name: 'Pages', path: '/service/X/pages' },
  { name: 'About', path: '/service/X/about' },
  { name: 'Publish', path: '/service/X/publish' },
];

const Sidebar = ({ items = itemsExample, onSelect }) => <div>
  {items.map(
    (item, i) => <SidebarItem item={item} key={item.name} onClick={() => onSelect(i, items[i])} />,
  )}
</div>;

Sidebar.propTypes = {
  items: PropTypes.arrayOf(PropTypes.shape({
    name: PropTypes.string,
    path: PropTypes.string,
    selected: PropTypes.bool,
  })),
  onSelect: PropTypes.func,
};

export default Sidebar;
```

1.75 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import Sidebar from './Sidebar';

describe('<Sidebar />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<Sidebar
      items={[]}
    />);
  });
});
```

1.76 SidebarItem.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import { Link } from 'react-router';
import { Color } from '../StyleConstant';

const style = {
  base: {
    height: 57,
    display: 'flex',
    alignItems: 'center',
    paddingLeft: 20,
    textDecoration: 'none',
    color: Color.black,
    transition: '0.3s all',
    ':hover': {
      background: Color.lighterGrey,
    },
  },
  selected: {
    background: Color.lightGrey,
    ':hover': {
      background: Color.lightGrey,
    },
  },
};

const SidebarItem = ({ item, onClick }) =>
  <div
    style={{ textDecoration: 'none', cursor: 'pointer' }}
    onClick={onClick}
  >
    <div style={[style.base, item.selected && style.selected]}>
      {item.name}
    </div>
  </div>;

SidebarItem.propTypes = {
  item: PropTypes.shape({
    name: PropTypes.string,
    path: PropTypes.string,
    selected: PropTypes.bool,
  }),
  onClick: PropTypes.func,
};

export default Radium(SidebarItem);
```

1.77 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import SidebarItem from './SidebarItem';

describe('<SidebarItem />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<SidebarItem
      item={{
        name: 'Hello',
        path: 'hello',
        selected: true,
      }}
    />);
  });
});
```

1.78 Attribute.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import capitalizeString from '../../../utils/capitalizeString';
import { Color } from '../../StyleConstant';

const style = {
  base: {
    backgroundColor: Color.lighterGrey,
    marginBottom: 4,
    borderRadius: 3,
    padding: '5px 9px',
    cursor: 'pointer',
    minHeight: 25,
    transition: '0.3s all',
    ':hover': {
      backgroundColor: Color.lightGrey,
    },
  },
  noInteraction: {
    cursor: 'default',
  },
};

const prettify = string => string && string.replace(/_/g, ' ');

function formatAttribute(attribute) {
  console.log(attribute);
  const leftPar = attribute.multiple ? '[' : '';
  const rightPar = attribute.multiple ? ']' : '';
  return `${prettify(attribute.name)} (${leftPar}${attribute.type}${rightPar})`;
}

const Attribute = ({ attribute, onClick, enableInteractions }) => <div onClick={onClick} style={[style.base, !enableInteractions && style.
    noInteraction]}>
  {formatAttribute(attribute)}
</div>;

Attribute.propTypes = {
  attribute: PropTypes.shape({
    name: PropTypes.string,
    multiple: PropTypes.bool,
  }),
  enableInteractions: PropTypes.bool,
};

export default Radium(Attribute);
```

1.79 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import Attribute from './Attribute';

describe('<Attribute />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<Attribute
      attribute={{
        name: 'Attribute',
        multiple: true,
      }}
      enableInteractions
    />);
  });
});
```

1.80 DialogBox.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import { Color, lightBorder } from '../../StyleConstant';
import TextInput from '../../TextInput';
import capitalizeString from '../../../utils/capitalizeString';

const style = {
  base: {
    width: 400,
    height: 400,
    position: 'absolute',
    backgroundColor: Color.whiteText,
    top: '50%',
    left: '50%',
    transform: 'translate(-50%, -50%)',
    borderRadius: 3,
    padding: 15,
    zIndex: 50,
  },
  close: {
    width: 35,
    height: 35,
    backgroundImage: 'url("/img/cross.png")',
    backgroundSize: 'cover',
    opacity: 0.3,
    position: 'absolute',
    right: 15,
    top: 15,
    cursor: 'pointer',
  },
  title: {
    fontWeight: 400,
    margin: 0,
    marginBottom: 20,
  },
  label: {
    display: 'block',
    marginTop: 10,
    marginBottom: 5,
  },
  cover: {
    position: 'fixed',
    width: '100%',
    height: '100%',
    top: 0,
    left: 0,
    background: 'rgba(0, 0, 0, 0.6)',
    zIndex: 5,
  },
  delete: {
    width: 200,
    height: 42,
    backgroundColor: Color.red,
    borderRadius: 3,
    color: 'white',
    textAlign: 'center',
    lineHeight: `${42}px`,
    marginTop: 30,
    cursor: 'pointer',
  },
};

function field(attr, object, onChange) {
  switch (attr.type) {
    case 'string': {
      return (
        <div key={attr.value}>
          <label style={style.label} htmlFor={attr.value}>{attr.label}</label>
          <TextInput id={attr.value} text={object[attr.value]} onChange={val => onChange({ [attr.value]: val })} />
        </div>
      );
    }
    case 'integer': {
      return (
        <div key={attr.value}>
          <label style={style.label} htmlFor={attr.value}>{attr.label}</label>
          <TextInput id={attr.value} type="number" text={object[attr.value]} onChange={val => onChange({ [attr.value]: val })} />
        </div>
      );
    }
    case 'enum': {
      return (
        <div key={attr.value}>
          <label style={style.label} htmlFor={attr.value}>{attr.label}</label>
          <select value={object[attr.value]} onChange={e => onChange({ [attr.value]: e.target.value })}>
            {attr.options.map(option =>
              <option value={option}>{capitalizeString(option)}</option>,
            )}
          </select>
        </div>
      );
    }
    case 'boolean': {
      return (
        <div key={attr.value}>
          <label style={[style.label]} htmlFor={attr.value}><input type="checkbox" checked={object[attr.value]} onChange={e => onChange({ [
              attr.value]: e.target.checked })} />{attr.label}</label>
        </div>
      );
    }
  }
}
```

```
const DialogBox = ({ name, object, attributes, onChange, onClose, onDelete }) =>
  <div>
    <div style={style.cover} onClick={onClose} />
    <div style={style.base}>
      <div style={style.close} onClick={onClose} />
      <h3 style={style.title}>{name && capitalizeString(name)}</h3>
      {attributes.map(attr => field(attr, object, onChange))}
      <div onClick={onDelete} style={style.delete}>Delete</div>
    </div>
  </div>;

DialogBox.propTypes = {
  name: PropTypes.string,
  object: PropTypes.object,
  attributes: PropTypes.array,
  onChange: PropTypes.func,
  onClose: PropTypes.func,
  onDelete: PropTypes.func,
};

export default DialogBox;
```

1.81 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import DialogBox from './DialogBox';

describe('<DialogBox />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<DialogBox
      name="Example"
      object={{}}
      attributes={[]}
    />);
  });
});
```

1.82 Model.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import Attribute from './Attribute';
import { Color } from '../../StyleConstant';
import capitalizeString from '../../../utils/capitalizeString';
import RoundButton from '../../RoundButton';

const style = {
  base: {
    backgroundColor: Color.white,
    background: '#FFFFFF',
    border: `2px solid ${Color.grey}`,
    borderRadius: 3,
    marginBottom: 10,
    width: 250,
    padding: 5,
    position: 'relative',
    zIndex: 0,
  },
  title: {
    margin: 0,
    padding: '5px 0px',
    textAlign: 'center',
    borderRadius: 3,
    ':hover': {
      backgroundColor: '#EEE',
    },
    border: 'none',
    ':focus': {
      outline: 0,
      border: 0,
    },
    fontSize: 20,
    width: '100%',
  },
  close: {
    position: 'absolute',
    top: 8,
    right: 6,
  },
  attributes: {
    marginTop: 10,
  },
  newAttribute: {
    textAlign: 'center',
    backgroundColor: '#EEE',
    margin: '6px 0',
    borderRadius: 3,
    padding: '5px 9px',
    cursor: 'pointer',
    color: 'black',
    transition: '0.5s all',
    fontSize: 18,
    ':hover': {
      backgroundColor: '#DDD',
    },
  },
};

const Model = ({ model, onClickAttribute, onDelete, onChange, onAttributeCreate, enableInteractions = true }) => <div style={style.base}>
  <input disabled={!enableInteractions} style={[style.title]} value={capitalizeString(model.name)} onChange={e => onChange(model.id, e.target
      .value)} />
  <div style={style.close}>
    {enableInteractions && <RoundButton text="remove" onClick={() => onDelete(model.id)} color={Color.red} small />}
  </div>
  <div style={style.attributes}>
    {model.attributes && model.attributes.map(attribute =>
      <Attribute
        key={`attr-${attribute.name}-${attribute.id}`}
        onClick={() => enableInteractions && onClickAttribute(attribute.id)}
        attribute={attribute}
        enableInteractions={enableInteractions}
      />)}
    {enableInteractions && <div key="newAttribute" style={style.newAttribute} onClick={() => onAttributeCreate(model.id)}>+</div>}
  </div>
</div>;

Model.propTypes = {
  model: PropTypes.shape({
    name: PropTypes.string,
    id: PropTypes.number,
    attributes: PropTypes.array,
  }).isRequired,
  onClickAttribute: PropTypes.func,
  onDelete: PropTypes.func,
  onChange: PropTypes.func,
  onAttributeCreate: PropTypes.func,
  enableInteractions: PropTypes.bool,
};

export default Radium(Model);
```

1.83 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import Model from './Model';

describe('<Model />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<Model
      model={{
        name: 'Test',
        id: 3,
        attributes: [],
      }}
    />);
  });
});
```

1.84 Structure.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import Model from './Model';
import TopBar from '../TopBar';
import DialogBox from './DialogBox';
import { lightBorder, Color } from '../../StyleConstant';

const style = {
  main: {
    backgroundColor: Color.lighterGrey,
    height: 'calc(100vh - 97px)',
    padding: 10,
    overflowY: 'auto',
  },
  model: {

  },
  newModel: {
    textAlign: 'center',
    backgroundColor: 'white',
    border: '2px solid rgba(198, 198, 198, 0.34)',
    borderRadius: 3,
    width: 250,
    padding: 5,
    paddingBottom: 10,
    fontSize: 27,
    cursor: 'pointer',
    color: 'black',
    transition: '0.5s all',
    ':hover': {
      border: '2px solid rgba(198, 198, 198, 0.8)',
    },
  },
};

const attributes = [
  {
    value: 'name',
    label: 'Name',
    type: 'string',
  },
  {
    value: 'type',
    label: 'Type',
    type: 'enum',
    options: ['string', 'integer', 'float'],
  },
  {
    value: 'multiple',
    label: 'Multiple',
    type: 'boolean',
  },
  {
    value: 'required',
    label: 'Required',
    type: 'boolean',
  },
];

const Structure = ({
  name,
  models = [],
  selectedAttribute,
  onSelectAttribute,
  onModelCreate,
  onModelDelete,
  onModelChange,
  onAttributeCreate,
  onAttributeDelete,
  onAttributeChange,
}) => <div style={style.base}>
  <TopBar name={name} onNew={() => onModelCreate()} />
  {selectedAttribute && <DialogBox
    name={selectedAttribute.name}
    object={selectedAttribute}
    attributes={attributes}
    onChange={changes => onAttributeChange(selectedAttribute.id, changes)}
    onDelete={() => onAttributeDelete(selectedAttribute.id)}
    onClose={() => onSelectAttribute(undefined)}
  />}
  <div style={style.main}>
    {models.map(model =>
      <Model key={`model-${model.id}`} onChange={onModelChange} onDelete={onModelDelete} onAttributeCreate={onAttributeCreate}
          onClickAttribute={onSelectAttribute} model={model} />,
    )}
  </div>
</div>;

Structure.propTypes = {
  name: PropTypes.string,
  models: PropTypes.array,
  selectedAttribute: PropTypes.object,
  onSelectAttribute: PropTypes.func,
  onModelCreate: PropTypes.func,
  onModelDelete: PropTypes.func,
  onModelChange: PropTypes.func,
  onAttributeCreate: PropTypes.func,
  onAttributeDelete: PropTypes.func,
  onAttributeChange: PropTypes.func,
};

export default Radium(Structure);
```

1.85 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import Structure from './Structure';

describe('<Structure />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<Structure
      name="Example"
      models={[]}
    />);
  });
});
```

1.86 TopBar.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import { lightBorder, Color } from '../StyleConstant';
import RoundButton from '../RoundButton';

const style = {
  base: {
    height: 74.5,
    borderBottom: lightBorder,
    display: 'flex',
    alignItems: 'center',
    justifyContent: 'space-between',
    padding: '0 20px',
    backgroundColor: Color.whiteText,
  },
  h2: {
    margin: 0,
    padding: 0,
    fontWeight: 500,
    fontSize: 24,
  },
};

const TopBar = ({ name, onNew }) =>
  <div style={style.base}>
    <h2 style={style.h2}>{name}</h2>
    <RoundButton text="add" onClick={onNew} />
  </div>;

TopBar.propTypes = {
  name: PropTypes.string,
  onNew: PropTypes.func,
};

export default TopBar;
```

1.87 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import TopBar from './TopBar';

describe('<TopBar />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<TopBar
      name="Example"
    />);
  });
});
```

1.88 Frame.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Logo from './Logo';

const style = {
  width: '90%',
  maxWidth: 960,
  marginLeft: 'auto',
  marginRight: 'auto',
  marginTop: 40,
};

const Frame = ({ children }) => <div style={style}>
  <Logo />
  {children}
</div>;

Frame.propTypes = {
  children: PropTypes.node,
};

export default Frame;
```

1.89 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import Frame from './Frame';

describe('<Frame />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<Frame>
      <p>Hello</p>
    </Frame>);
  });
});
```

1.90 HomePage.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import { Router, Route, browserHistory } from 'react-router';
import AuthFormContainer from '../containers/AuthFormContainer';
import ServiceListContainer from '../containers/ServiceListContainer';
import Frame from './Frame';

const HomePage = ({ authenticated }) => (
  <Frame>
    {authenticated ?
      <ServiceListContainer />
      :
      <AuthFormContainer />
    }
  </Frame>
);

HomePage.propTypes = {
  authenticated: PropTypes.bool,
};

/* eslint-disable new-cap */
export default Radium(HomePage);
```

1.91 Logo.jsx

```
import React from 'react';

const style = {
  width: 140,
};

const Logo = () => (
  <a href="/">
    <img alt="EasyAPI" src="/img/logo.png" style={style} />
  </a>
);

export default Logo;
```

1.92 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import Logo from './Logo';

describe('<Logo />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<Logo />);

  });
});
```

1.93 MethodButton.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import { Color, Dimensions } from './StyleConstant';
import createMethods from '../utils/createMethods';

const {
  naturalLanguage,
  spreadsheet,
  device,
} = createMethods;

const activeStyle = {
  outline: 'none',
};

const style = {
  base: {
    minWidth: 150,
    height: 170,
    border: `${Dimensions.borderWidth}px solid ${Color.grey}`,
    borderRadius: 10,
    backgroundColor: Color.whiteText,
    cursor: 'pointer',
    margin: '0 13px',
    transition: `${Dimensions.transitionTime.normal} all`,
    fontSize: Dimensions.fontSize.normal,
    color: Color.black,
    ':hover': {
      border: `${Dimensions.borderWidth}px solid ${Color.greenLight}`,
    },
    ':active': activeStyle,
    ':focus': activeStyle,
  },
  selected: {
    border: `${Dimensions.borderWidth}px solid ${Color.greenDark}`,
    ':hover': {
      border: `${Dimensions.borderWidth}px solid ${Color.green}`,
    },
  },
  image: {
    scratch: {
      width: 83,
    },
    spreadsheet: {
      width: 81,
    },
    device: {
      width: 80,
    },
  },
  inner: {
    textAlign: 'center',
    marginBottom: 20,
  },
};

const MethodButton = ({ method, onClick, isSelected }) => {
  let text;
  let image;

  switch (method) {
    case naturalLanguage:
      text = 'Scratch';
      image = 'scratch';
      break;
    case spreadsheet:
      text = 'Dataset';
      image = 'spreadsheet';
      break;
    case device:
      text = 'Device';
      image = 'device';
      break;
  }

  return (
    <button
      onClick={onClick} style={[
        style.base,
        isSelected ? style.selected : {},
      ]}
    >
      <div style={style.inner}>
        <img src={`/img/${image}.png`} style={style.image[image]} alt={text} />
      </div>
      {text}
    </button>
  );
};

MethodButton.PropTypes = {
  method: PropTypes.string,
  onClick: PropTypes.func,
  isSelected: PropTypes.bool,
};

export default Radium(MethodButton);
```

1.94 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';
import createMethods from '../utils/createMethods';

const {
  naturalLanguage,
} = createMethods;

import MethodButton from './MethodButton';

describe('<MethodButton />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<MethodButton
      method={naturalLanguage}
      isSelected
    />);
  });
});
```

1.95 RoundButton.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import { Color, Dimensions } from './StyleConstant';

const activeStyle = {
  opacity: 1.0,
  border: 'none',
  outline: 'none',
};

const style = {
  base: {
    backgroundColor: Color.green,
    width: Dimensions.fieldHeight,
    height: Dimensions.fieldHeight,
    border: 'none',
    borderRadius: '50%',
    cursor: 'pointer',
    transition: `${Dimensions.transitionTime.normal} opacity`,
    fontSize: 30,
    backgroundSize: 'contain',
    color: Color.whiteText,
    ':hover': {
      opacity: 0.8,
    },
    ':active': activeStyle,
    ':focus': activeStyle,
  },
  isDisabled: {
    pointerEvents: 'none',
    backgroundColor: Color.grey,
  },
};

const RoundButton = ({ text, onClick, isDisabled, color = Color.green, small = false }) => (
  <button onClick={onClick} style={[style.base, isDisabled && style.isDisabled, { backgroundColor: color, backgroundImage: `url('/img/${text
      }.png')` }, small && { width: 25, height: 25 }]} />
);

RoundButton.PropTypes = {
  text: PropTypes.string,
  onClick: PropTypes.func,
  isDisabled: PropTypes.bool,
  color: PropTypes.string,
  small: PropTypes.bool,
};


export default Radium(RoundButton);
```

1.96 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import RoundButton from './RoundButton';

describe('<RoundButton />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<RoundButton
      text="add"
      color="red"
      small
    />);
  });
});
```

1.97 ServiceList.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import RoundButton from './RoundButton';
import ServiceListItem from './ServiceListItem';

const style = {
  p: {
    textAlign: 'center',
  },
  list: {
    display: 'flex',
    flexDirection: 'column',
    alignItems: 'center',
  },
};

class ServiceList extends React.Component {
  componentDidMount() {
    this.props.onReady();
  }

  render() {
    return (
      <div>
        <p style={style.p}>Which API would you like to work on?</p>
        <div style={style.list}>
          {this.props.services.map(service => <ServiceListItem key={`si-${service.id}`} onClick={() => this.props.onSelect(service.id)}
              service={service} />)}
          <RoundButton text="add" onClick={this.props.onCreate} />
        </div>
      </div>
    );
  }
}

ServiceList.propTypes = {
  services: PropTypes.array,
  onSelect: PropTypes.func,
  onCreate: PropTypes.func,
  onReady: PropTypes.func,
};

/* eslint-disable new-cap */
export default Radium(ServiceList);
```

1.98 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import ServiceList from './ServiceList';

describe('<ServiceList />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<ServiceList
      services={[]}
    />);
  });
});
```

1.99 ServiceListItem.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import { Color, Dimensions } from './StyleConstant';

const style = {
  marginBottom: 20,
  border: `${Dimensions.borderWidth}px solid ${Color.grey}`,
  borderRadius: 4,
  width: 270,
  height: 60,
  cursor: 'pointer',
  display: 'flex',
  alignItems: 'center',
  justifyContent: 'center',
  transition: `all ${Dimensions.transitionTime.normal}`,
  ':hover': {
    border: `${Dimensions.borderWidth}px solid ${Color.green}`,
  },
};

const ServiceListItem = ({ service, onClick }) => <div style={style} onClick={onClick}>
  {service && service.name}
</div>;

ServiceListItem.propTypes = {
  service: PropTypes.shape({
    name: PropTypes.string,
  }),
  onClick: PropTypes.func,
};

export default Radium(ServiceListItem);
```

1.100 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import ServiceListItem from './ServiceListItem';

describe('<ServiceListItem />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<ServiceListItem
      service={{
        name: 'Example',
      }}
    />);
  });
});
```

1.101 Setup.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import Frame from '../Frame';
import SetupNameContainer from '../../containers/setup/SetupNameContainer';
import SetupMethodContainer from '../../containers/setup/SetupMethodContainer';
import SetupNaturalContainer from '../../containers/setup/SetupNaturalContainer';
import SetupSpreadsheetContainer from '../../containers/setup/SetupSpreadsheetContainer';
import {
  SERVICE_SETUP_SCREEN_METHOD,
  SERVICE_SETUP_SCREEN_NAME,
  SERVICE_SETUP_SCREEN_NATURAL,
  SERVICE_SETUP_SCREEN_SPREADSHEET,
} from '../../utils/setupScreens';

const Setup = ({ screen }) => {
  let inner;

  switch (screen) {
    case SERVICE_SETUP_SCREEN_NAME:
      inner = (<SetupNameContainer />);
      break;
    case SERVICE_SETUP_SCREEN_METHOD:
      inner = (<SetupMethodContainer />);
      break;
    case SERVICE_SETUP_SCREEN_NATURAL:
      inner = (<SetupNaturalContainer />);
      break;
    case SERVICE_SETUP_SCREEN_SPREADSHEET:
      inner = (<SetupSpreadsheetContainer />);
      break;
    default:
      inner = (<p>{'404 Setup screen not found'}</p>);
  }

  return (
    <Frame>
      {inner}
    </Frame>
  );
};

Setup.propTypes = {
  screen: PropTypes.string,
};

/* eslint-disable new-cap */
export default Radium(Setup);
```

1.102 SetupMethod.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import MethodButton from '../MethodButton';
import Button from '../Button';
import createMethods from '../../utils/createMethods';

const {
  naturalLanguage,
  spreadsheet,
} = createMethods;


const styles = {
  nextButton: {
    marginTop: 100,
    float: 'right',
  },
  field: {
    width: 700,
    marginLeft: 'auto',
    marginRight: 'auto',
    textAlign: 'center',
    marginTop: 100,
  },
  methods: {
    display: 'flex',
    justifyContent: 'center',
  },
};

const SetupMethod = ({ method, onChange, onDone }) => (
  <div>
    <div style={styles.field}>
      <p>How do you want to create your API?</p>
      <div style={styles.methods}>
        <MethodButton
          method={naturalLanguage}
          isSelected={method === naturalLanguage}
          onClick={() => onChange(naturalLanguage)}
        />
        <MethodButton
          method={spreadsheet}
          isSelected={method === spreadsheet}
          onClick={() => onChange(spreadsheet)}
        />
      </div>
    </div>
    <div style={styles.nextButton} >
      <Button isDisabled={!method} onClick={onDone} text="Next" />
    </div>
  </div>
);

SetupMethod.propTypes = {
  method: PropTypes.string,
  onChange: PropTypes.func,
  onDone: PropTypes.func,
};

/* eslint-disable new-cap */
export default Radium(SetupMethod);
```

1.103 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';
import createMethods from '../../utils/createMethods';

const {
  naturalLanguage,
  spreadsheet,
} = createMethods;

import SetupMethod from './SetupMethod';

describe('<SetupMethod />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<SetupMethod
      method={naturalLanguage}
    />);
  });
});
```

1.104 SetupName.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import TextInput from '../TextInput';
import Button from '../Button';

const styles = {
  nextButton: {
    marginTop: 100,
    float: 'right',
  },
  field: {
    width: 500,
    margin: 'auto',
    textAlign: 'center',
    marginTop: 100,
  },
};

const SetupMethod = ({ name, onChange, onDone }) => (
  <div>
    <div style={styles.field}>
      <p>What is the name of your API?</p>
      <TextInput placeholder={'Name'} text={name} onChange={onChange} />
    </div>
    <div style={styles.nextButton}>
      <Button onClick={onDone} text="Next" isDisabled={!name || !name.length} />
    </div>
  </div>
);

SetupMethod.PropTypes = {
  name: PropTypes.string,
  onChange: PropTypes.func,
  onDone: PropTypes.func,
};

/* eslint-disable new-cap */
export default Radium(SetupMethod);
```

1.105 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import SetupName from './SetupName';

describe('<SetupName />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<SetupName
      name="Example"
    />);
  });
});
```

1.106 SetupNatural.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import Button from '../Button';
import TextInput from '../TextInput';
import capitalizeString from '../../utils/capitalizeString';
import Model from '../dashboard/structure/Model';

const styles = {
  nextButton: {
    marginTop: 100,
    float: 'right',
  },
  field: {
    width: 700,
    marginLeft: 'auto',
    marginRight: 'auto',
    textAlign: 'center',
    marginTop: 100,
  },
};


const SetupNatural = ({ text, onChange, onDone, preview, nextEnabled }) => (
  <div>
    <div style={styles.field}>
      <p>Please describe the various things and entities, <br />along with their properties and relationships</p>
      <div>
        <TextInput
          text={text}
          onChange={onChange}
          long
        />
      </div>
      {preview && preview.map(a => <Model enableInteractions={false} model={a} />)}
    </div>
    <div style={styles.nextButton} >
      <Button isDisabled={!nextEnabled} onClick={onDone} text="Next" />
    </div>
  </div>
);

SetupNatural.PropTypes = {
  text: PropTypes.string,
  onChange: PropTypes.func,
  onDone: PropTypes.func,
  preview: PropTypes.array,
  nextEnabled: PropTypes.bool,
};

```

```
/* eslint-disable new-cap */
export default Radium(SetupNatural);
```

1.107 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import SetupNatural from './SetupNatural';

describe('<SetupNatural />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<SetupNatural
      text="A dog has a bone."
      preview={[]}
      nextEnabled={false}
    />);
  });
});
```

1.108 SetupSpreadsheet.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import Button from '../Button';
import TextInput from '../TextInput';
import capitalizeString from '../../utils/capitalizeString';
import Dropzone from 'react-dropzone';
import Model from '../dashboard/structure/Model';

const styles = {
  nextButton: {
    marginTop: 100,
    float: 'right',
  },
  field: {
    width: 700,
    marginLeft: 'auto',
    marginRight: 'auto',
    textAlign: 'center',
    marginTop: 100,
  },
  sheet: {
    width: '100%',
    height: 300,
    border: '2px solid gray',
    borderRadius: 5,
    borderStyle: 'dashed',
    alignItems: 'center',
    justifyContent: 'center',
    display: 'flex',
  },
};

const SetupSpreadsheet = ({ onChange, onDone, preview, nextEnabled }) => (
  <div>
    <div style={styles.field}>
      <div>
        <Dropzone onDrop={onChange} style={styles.sheet}>
          Drop a spreadsheet file into this area
        </Dropzone>
      </div>
      {preview && preview.map(a => <Model enableInteractions={false} model={a} />)}
    </div>
    <div style={styles.nextButton} >
      <Button isDisabled={!nextEnabled} onClick={onDone} text="Next" />
    </div>
  </div>
);

SetupSpreadsheet.propTypes = {
  onChange: PropTypes.func,
  onDone: PropTypes.func,
  preview: PropTypes.array,
  nextEnabled: PropTypes.bool,
};

/* eslint-disable new-cap */
export default Radium(SetupSpreadsheet);
```

1.109 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import SetupSpreadsheet from './SetupSpreadsheet';

describe('<SetupSpreadsheet />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<SetupSpreadsheet
      preview={[]}
      nextEnabled={false}
    />);
  });
});
```

1.110 StyleConstant.js

```
export const Color = {
  green: '#50E39C',
  greenLight: '#54F0A5',
  greenDark: '#4BD793',
  red: '#FA6461',
  redLight: '#FA706E',
  redDark: '#EE5F5C',
  whiteText: '#FFFFFF',
  black: '#000000',
  grey: '#C6C6C6',
  lightGrey: '#E6E6E6',
  lighterGrey: '#F8F9FB',
};

export const Dimensions = {
  fieldHeight: 44,
  fieldWidth: 125,
  borderRadius: 3,
  borderWidth: 2.9,
  padding: 6,
  fontSize: {
    normal: 17,
  },
  transitionTime: {
    normal: '0.25s',
  }
}

export const lightBorder = `2px solid ${Color.lightGrey}`;
```

1.111 TextInput.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
import Radium from 'radium';
import { Color, Dimensions } from './StyleConstant';

const activeStyle = {
  outline: 'none',
  border: `${Dimensions.borderWidth}px solid ${Color.green}`,
};

const styles = {
  base: {
    border: `${Dimensions.borderWidth}px solid ${Color.grey}`,
    minWidth: Dimensions.fieldWidth,
    height: Dimensions.fieldHeight - Dimensions.borderWidth * 2,
    borderRadius: Dimensions.borderRadius,
    fontSize: Dimensions.fontSize.normal,
    padding: `0 ${Dimensions.padding}px`,
    transition: ` ${Dimensions.transitionTime.normal} all`,
    ':active': activeStyle,
    ':focus': activeStyle,
  },
  long: {
    width: 500,
    height: 130,
    padding: Dimensions.padding,
  },
};

const TextInput = ({ text, placeholder, onChange, long = false, name, type = 'text', id }) => (
  long ? (
    <textarea
      value={text}
      placeholder={placeholder}
      onChange={e => onChange(e.target.value)}
      style={[styles.base, styles.long]}
      name={name}
      id={id}
    />
  ) : (
    <input
      value={text}
      name={name}
      type={type}
      placeholder={placeholder}
      onChange={e => onChange(e.target.value)}
      style={styles.base}
      id={id}
    />
  )

);

TextInput.propTypes = {
  text: PropTypes.string.isRequired,
  placeholder: PropTypes.string,
  onChange: PropTypes.func,
  long: PropTypes.bool,
  name: PropTypes.string,
  type: PropTypes.string,
  id: PropTypes.any,
};

/* eslint-disable new-cap */
export default Radium(TextInput);
```

1.112 test.js

```

import React from 'react';
import { expect } from 'chai';
import { shallow } from 'enzyme';
import sinon from 'sinon';

import TextInput from './TextInput';

describe('<TextInput />', () => {
  it('renders the component correcty', () => {
    const wrapper = shallow(<TextInput
      text="Hello"
      placeholder="Example"
      long
      name="ok"
      type="text"
      id={4}
    />);
  });
});
```

1.113 AuthFormContainer.js

```
import { connect } from 'react-redux';
import {
  updateUser,
  authUser,
} from '../actions/auth';
import AuthForm from '../components/AuthForm';

const mapStateToProps = (state, ownProps) => ({
  username: state
      .getIn(['user', 'username']),
  password: state
      .getIn(['user', 'password']),
  errors: state
      .getIn(['user', 'errors']),
});

const mapDispatchToProps = (dispatch, ownProps) => ({
  onSubmit: ({ username, password }) => dispatch(authUser(username, password)),
  onChange: ({ username, password }) => dispatch(updateUser(username, password)),
});

const AuthFormContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(AuthForm);

export default AuthFormContainer;
```

1.114 AboutContainer.js

```
import { connect } from 'react-redux';
import { updateService } from '../../actions/dashboard/updateService';
import { updateServiceLocally } from '../../actions/dashboard/updateServiceLocally';
import About from '../../components/dashboard/about/About';


const mapStateToProps = (immutableState) => {
  const state = immutableState.toJS();

  const service = state.serviceById[state.user.currentServiceId];

  return {
    name: service.name,
    meta: {
      name: {
        value: service.name,
        label: 'Name',
      },
      shortName: {
        value: service.shortName,
        label: 'URL',
      },
      isPublic: {
        value: service.isPublic,
        label: 'Public?',
      },
    },
  };
};

const mapDispatchToProps = dispatch => ({
  onChange: (changes) => {
    dispatch(updateServiceLocally(changes));
    dispatch(updateService(changes));
  },
});

const AboutContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(About);

export default AboutContainer;
```

1.115 EntriesContainer.js

```
import { connect } from 'react-redux';
import { debounce, difference } from 'underscore';
import { changeSelectedModel } from '../../actions/dashboard/changeSelectedModel';
import { createEntry } from '../../actions/dashboard/createEntry';
import { deleteEntry } from '../../actions/dashboard/deleteEntry';
import { updateValue } from '../../actions/dashboard/updateValue';
import { updateValueLocally } from '../../actions/dashboard/updateValueLocally';
import Entries from '../../components/dashboard/entries/Entries';


const mapStateToProps = (immutableState) => {
  const state = immutableState.toJS();

  const service = state.serviceById[state.user.currentServiceId];

  const selectedModel = state.dashboard.selectedModel || service.Models[0];

  if (!service) {
    return {};
  }

  const model = state.modelById[selectedModel];

  if (!model) return {};

  model.attributes = model.Attributes ? model.Attributes.map(i => state.attributeById[i]) : [];
  model.entries = model.Entries ? model.Entries.map(i => state.entryById[i]) : [];

  const headers = service.Models
    .map(i => state.modelById[i])
    .map(m => ({ id: m.id, text: m.name, selected: m.id === selectedModel }));
  const attributes = model.attributes;

  const entries = [];
  for (const entry of model.entries) {
    const obj = { id: entry.index, realId: entry.id };

    const values = entry.Values ? entry.Values.map(i => state.valueById[i]) : [];

    const missing = difference(
      attributes.map(a => a.id),
      values.map(v => v.Attribute || v.AttributeId),
    ).map(id => state.attributeById[id]);

    for (const valueObj of values) {
      const value = valueObj.value;

      const attr = state.attributeById[valueObj.Attribute || valueObj.AttributeId];

      if (!attr) continue;
      obj[attr.name] = { value, id: valueObj.id };
    }

    entries.push(obj);
  }

  return {
    name: service.name,
    headers,
    attributes,
    entries,
  };
};

const mapDispatchToProps = (dispatch) => {
  const update = debounce((id, attr, value) => dispatch(updateValue(id, attr, value)), 1000);

  return {
    onSelected: id => dispatch(changeSelectedModel(id)),
    onCreate: () => dispatch(createEntry()),
    onDelete: id => dispatch(deleteEntry(id)),
    onUpdate: (id, attr, value, valueId) => {
      dispatch(updateValueLocally(id, valueId, value));
      update(id, attr, value);
    },
  };
};

const EntriesContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(Entries);

export default EntriesContainer;
```

1.116 PagesContainer.js

```
import { connect } from 'react-redux';
import { updateModel } from '../../actions/dashboard/updateModel';
import { updateModelLocally } from '../../actions/dashboard/updateModelLocally';
import Pages from '../../components/dashboard/pages/Pages';


const mapStateToProps = (immutableState) => {
  const state = immutableState.toJS();

  const service = state.serviceById[state.user.currentServiceId];
  const models = service.Models.map(id => state.modelById[id]);

  const actions = [
    {
      label: 'Find',
      prop: 'isFindEnabled',
      method: 'GET',
    },
    {
      label: 'Find One',
      prop: 'isFindOneEnabled',
      method: 'GET',
    },
    {
      label: 'Create',
      prop: 'isCreateEnabled',
      method: 'POST',
    },
    {
      label: 'Update',
      prop: 'isUpdateEnabled',
      method: 'PATCH',
    },
    {
      label: 'Delete',
      prop: 'isDeleteEnabled',
      method: 'DELETE',
    },
  ];

  const urlPrefix = `http://localhost:8000/api/${state.user.username}/${service.shortName}/`;

  return {
    name: service.name,
    actions,
    models,
    urlPrefix,
  };
};

```

```
const mapDispatchToProps = dispatch => ({
  onChange: (changes) => {
    dispatch(updateModelLocally(changes));
    dispatch(updateModel(changes));
  },
});

const PagesContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(Pages);

export default PagesContainer;
```

1.117 SidebarContainer.js

```
import { connect } from 'react-redux';
import {
  changeDashboardPage,
} from '../../actions/dashboard/changeDashboardPage';
import Sidebar from '../../components/dashboard/Sidebar';
import 'immutable';

const mapStateToProps = state => ({
  items: state.getIn(['dashboard', 'items']).toJS(),
});

const mapDispatchToProps = dispatch => ({
  onSelect: (index, item) => dispatch(changeDashboardPage(index, item)),
});


const SidebarContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(Sidebar);

export default SidebarContainer;
```

1.118 StructureContainer.js

```
import { connect } from 'react-redux';
import {
} from '../../actions/dashboard/changeSidebarItem';
import Structure from '../../components/dashboard/structure/Structure';
import { selectAttribute } from '../../actions/dashboard/selectAttribute';
import { createModel } from '../../actions/dashboard/createModel';
import { createAttribute } from '../../actions/dashboard/createAttribute';
import { deleteModel } from '../../actions/dashboard/deleteModel';
import { deleteAttribute } from '../../actions/dashboard/deleteAttribute';
import { updateModel } from '../../actions/dashboard/updateModel';
import { updateAttribute } from '../../actions/dashboard/updateAttribute';

const mapStateToProps = (immutableState) => {
  const state = immutableState.toJS();

  const service = state.serviceById[state.user.currentServiceId];

  if (!service) {
    return {};
  }

  const models = service.Models
    .map(i => state.modelById[i])
    .map(model => Object.assign(model, {
      attributes: model.Attributes && model.Attributes.map(i => state.attributeById[i]),
    }));

  const selectedAttribute = state.dashboard.selectedAttribute &&
   state.attributeById[state.dashboard.selectedAttribute];

  return {
    name: service.name,
    models,
    selectedAttribute,
  };
};

const mapDispatchToProps = dispatch => ({
  onSelectAttribute: id => dispatch(selectAttribute(id)),
  onModelCreate: () => dispatch(createModel()),
  onAttributeCreate: attribute => dispatch(createAttribute(attribute)),
  onModelDelete: id => dispatch(deleteModel(id)),
  onAttributeDelete: id => dispatch(deleteAttribute(id)),
  onModelChange: (id, name) => dispatch(updateModel(id, name)),
  onAttributeChange: (id, changes) => dispatch(updateAttribute(id, changes)),
});


const StructureContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(Structure);

export default StructureContainer;
```

1.119 HomePageContainer.js

```
import { connect } from 'react-redux';
import HomePage from '../components/HomePage';

const mapStateToProps = state => ({
  authenticated: state.getIn(['user', 'authenticated']),
});


const HomePageContainer = connect(
  mapStateToProps,
)(HomePage);

export default HomePageContainer;
```

1.120 NameInput.js

1.121 ServiceListContainer.js

```
import ServiceList from '../components/ServiceList';
import 'immutable';
import { connect } from 'react-redux';
import { push } from 'react-router-redux';
import {
  selectService,
  newService,
} from '../actions/setup';
import { getServiceList } from '../actions/auth/getServiceList';

const mapStateToProps = (state) => {
  const services = state.getIn(['user', 'services'])
      .map(id => state.getIn(['serviceById', `${id}`]))
      .filter(e => !!e)
      .toJS();

  return {
    services,
  };
};

const mapDispatchToProps = dispatch => ({
  onReady: () => dispatch(getServiceList()),
  onSelect: id =>
    dispatch(selectService(id)),
  onCreate: () => dispatch(newService()),
});


const ServiceListContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(ServiceList);

export default ServiceListContainer;
```

1.122 SetupContainer.js

```
import { connect } from 'react-redux';
import Setup from '../../components/setup/Setup';
import 'immutable';

const mapStateToProps = (state, ownProps) => ({
  screen: state.getIn(['setup', 'screen']),
});

const SetupContainer = connect(
  mapStateToProps,
)(Setup);

export default SetupContainer;
```

1.123 SetupMethodContainer.js

```
import { connect } from 'react-redux';
import {
  setServiceCreateMethod,
  nextScreen,
} from '../../actions/setup';
import SetupMethod from '../../components/setup/SetupMethod';

const mapStateToProps = state => ({
  method: state
      .get('setup')
      .get('method'),
});

const mapDispatchToProps = dispatch => ({
  onDone: () => dispatch(nextScreen()),
  onChange: method => dispatch(setServiceCreateMethod(method)),
});

const SetupMethodContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(SetupMethod);

export default SetupMethodContainer;
```

1.124 SetupNameContainer.js

```
import { connect } from 'react-redux';
import 'immutable';
import {
  setServiceName,
  nextScreen,
} from '../../actions/setup';
import ServiceSetupName from '../../components/setup/SetupName';

const mapStateToProps = state => ({
  name: state
      .get('setup')
      .get('name'),
});

const mapDispatchToProps = dispatch => ({
  onDone: () => dispatch(nextScreen()),
  onChange: name => dispatch(setServiceName(name)),

});

const SetupNameContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(ServiceSetupName);

export default SetupNameContainer;
```

1.125 SetupNaturalContainer.js

```
import { connect } from 'react-redux';
import 'immutable';
import {
  analyseNaturalText,
  createService,
} from '../../actions/setup';
import SetupNatural from '../../components/setup/SetupNatural';

const mapStateToProps = (state) => {
  const preview = state.getIn(['setup', 'modelDefinitionPreview']);
  console.log(preview, preview && preview.size);
  return {
    text: state.getIn(['setup', 'naturalText']),
    preview: preview,
    nextEnabled: preview && !!preview.length,
  };
};

const mapDispatchToProps = dispatch => ({
  onDone: () => dispatch(createService()),
  onChange: text => dispatch(analyseNaturalText(text)),
});

const SetupNaturalContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(SetupNatural);

export default SetupNaturalContainer;
```

1.126 SetupSpreadsheetContainer.js

```
import { connect } from 'react-redux';
import 'immutable';
import {
  createService,
} from '../../actions/setup';
import { analyseSpreadsheet } from '../../actions/setup/analyseSpreadsheet';
import SetupSpreadsheet from '../../components/setup/SetupSpreadsheet';

const mapStateToProps = (state) => {
  const preview = state.getIn(['setup', 'modelDefinitionPreview']);
  return {
    file: state.getIn(['setup', 'file']),
    preview: preview,
    nextEnabled: preview && !!preview.length,
  };
};


const mapDispatchToProps = dispatch => ({
  onDone: () => dispatch(createService()),
  onChange: ([file]) => dispatch(analyseSpreadsheet(file)),

});

const SetupSpreadsheetContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(SetupSpreadsheet);

export default SetupSpreadsheetContainer;
```

1.127 index.css

```css


/** Ultra Light */
@font-face {
  font-family: "San Francisco";
  font-weight: 100;
  src: url("https://applesocial.s3.amazonaws.com/assets/styles/fonts/sanfrancisco/sanfranciscodisplay-ultralight-webfont.woff");
}

/** Thin */
@font-face {
  font-family: "San Francisco";
  font-weight: 200;
  src: url("https://applesocial.s3.amazonaws.com/assets/styles/fonts/sanfrancisco/sanfranciscodisplay-thin-webfont.woff");
}

/** Regular */
@font-face {
  font-family: "San Francisco";
  font-weight: 400;
  src: url("https://applesocial.s3.amazonaws.com/assets/styles/fonts/sanfrancisco/sanfranciscodisplay-regular-webfont.woff");
}

/** Medium */
@font-face {
  font-family: "San Francisco";
  font-weight: 500;
  src: url("https://applesocial.s3.amazonaws.com/assets/styles/fonts/sanfrancisco/sanfranciscodisplay-medium-webfont.woff");
}

/** Semi Bold */
@font-face {
  font-family: "San Francisco";
  font-weight: 600;
  src: url("https://applesocial.s3.amazonaws.com/assets/styles/fonts/sanfrancisco/sanfranciscodisplay-semibold-webfont.woff");
}

/** Bold */
@font-face {
  font-family: "San Francisco";
  font-weight: 700;
  src: url("https://applesocial.s3.amazonaws.com/assets/styles/fonts/sanfrancisco/sanfranciscodisplay-bold-webfont.woff");
}

body {
  /*font-family: "San Francisco", sans-serif;*/
  font-family: sans-serif;
  font-weight: 500;
  letter-spacing: 0.4px;
  font-size: 18px;
  margin: 0;

    -webkit-font-smoothing: antialiased;
}
```

1.128 index.js

```
import React from 'react';
import { render } from 'react-dom';
import { Provider } from 'react-redux';
import { createStore, applyMiddleware } from 'redux';
import { Router, IndexRedirect, Route, browserHistory } from 'react-router';
import thunk from 'redux-thunk';
import { syncHistoryWithStore, routerMiddleware } from 'react-router-redux';
import easyAPI from './reducers';
import './index.css';
import SetupContainer from './containers/setup/SetupContainer';
import Dashboard from './components/dashboard/Dashboard';
import StructureContainer from './containers/dashboard/StructureContainer';
import EntriesContainer from './containers/dashboard/EntriesContainer';
import AboutContainer from './containers/dashboard/AboutContainer';
import PagesContainer from './containers/dashboard/PagesContainer';
import ServiceListContainer from './containers/ServiceListContainer';
import HomePageContainer from './containers/HomePageContainer';
import { isAuthenticated } from './utils/Auth';

const middleware = routerMiddleware(browserHistory);
const store = createStore(easyAPI,
   window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__(),
   applyMiddleware(thunk, middleware),
);

const history = syncHistoryWithStore(browserHistory, store, {
  selectLocationState(state) {
    return state.get('routing').toJS();
  },
});

function requireAuth(nextState, replace) {
  console.log(nextState);
  const isStuck = !store.getState().getIn(['user', 'currentServiceId']) && (nextState.location.pathname !== '/service/setup');

  if (!isAuthenticated() || isStuck) {
    return replace({
      pathname: '/',
    });
  }
}

const r = () => render(
  <Provider store={store}>
    <Router history={history}>
      <Route
        path="/"
        component={HomePageContainer}
      />
      <Route
```

```
        path="/services"
        component={ServiceListContainer}
      />
      <Route
        path="/service/setup"
        component={SetupContainer}
        onEnter={requireAuth}
      />
      <Route
        path="/service/dashboard"
        component={Dashboard}
        onEnter={requireAuth}
      >
        <Route
          path="structure"
          component={StructureContainer}
        />
        <Route
          path="entries"
          component={EntriesContainer}
        />
        <Route
          path="pages"
          component={PagesContainer}
        />
        <Route
          path="about"
          component={AboutContainer}
        />
        <IndexRedirect to="structure" />
      </Route>

    </Router>
  </Provider>,
  document.getElementById('root'),
);

r();
store.subscribe(r);
```

1.129 index.js

```
// @flow
import { List, Map, fromJS } from 'immutable';
import {
  UPDATE_MODEL_PREVIEW,
  NEW_SERVICE,
  RECEIVE_WEBHOOK_URL,
  SELECT_DEVICE,
  SET_DEVICE_FLOW_DIRECTION,
  SET_SERVICE_CREATE_METHOD,
  SET_SERVICE_NAME,
  SETUP_DEVICE_QUERY,
  NEXT_SCREEN,
  UPDATE_NATURAL_TEXT,
  UPDATE_USER,
  AUTH_USER_RESULT,
  LOGOUT_USER,
  CHANGE_SIDEBAR_ITEM,
  RECEIVE_SERVICE_LIST,
  SELECT_SERVICE,
  RECEIVE_SERVICE,
  CHANGE_SELECTED_MODEL,
  RECEIVE_ENTRY,
  DELETE_ENTRY_LOCALLY,
  UPDATE_VALUE_LOCALLY,
  UPDATE_SERVICE_LOCALLY,
  SELECT_ATTRIBUTE,
  RECEIVE_MODEL,
  RECEIVE_ATTRIBUTE,
  DELETE_MODEL_LOCALLY,
  DELETE_ATTRIBUTE_LOCALLY,
  UPDATE_ATTRIBUTE_LOCALLY,
  UPDATE_MODEL_LOCALLY,
} from '../actions/actionTypes';
import capitalizeString from '../utils/capitalizeString';
import formatSentences from '../utils/formatSentences';
import createMethods from '../utils/createMethods';
import { isAuthenticated, getToken } from '../utils/Auth';
import { normalizeServices, normalizeService, normalizeEntry, normalizeModel, normalizeAttribute } from '../utils/normalizr';
import {
  LOCATION_CHANGE,
} from 'react-router-redux';
import {
  SERVICE_SETUP_SCREEN_METHOD,
  SERVICE_SETUP_SCREEN_NAME,
  SERVICE_SETUP_SCREEN_NATURAL,
  SERVICE_SETUP_SCREEN_SPREADSHEET,
  SERVICE_SETUP_SCREEN_DEVICE,
} from '../utils/setupScreens';


```

```
const {
  naturalLanguage,
  spreadsheet,
  device,
} = createMethods;


const NEW_ID = '-1';


const defaultState = fromJS({
  routing: {
    locationBeforeTransitions: null,
  },
  user: {
    currentServiceId: null,
    username: '',
    password: '',
    authenticated: isAuthenticated(),
    services: [],
    token: getToken(),
  },
  dashboard: {
    items: [
      {
        name: 'Structure',
        path: '/service/dashboard/structure',
        selected: true,
      },
      {
        name: 'Entries',
        path: '/service/dashboard/entries',
      },
      {
        name: 'Pages',
        path: '/service/dashboard/pages',
      },
      {
        name: 'About',
        path: '/service/dashboard/about',
      },
    ],
    selectedAttribute: null,
    selectedModel: null,
  },
  setup: {
    name: '',
    screen: 'SERVICE_SETUP_SCREEN_NAME',
    method: 'CREATE_METHOD_NATURAL_LANGUAGE',
  },
  serviceById: {
  },
  modelById: {
  },
  attributeById: {
  },
  entryById: {},
  valueById: {},
  endpointById: {},
});

function easyAPI(state = defaultState, action) {
  switch (action.type) {
    case LOCATION_CHANGE: {
      return state.setIn(['routing', 'locationBeforeTransitions'], action.payload);
    }
    case NEW_SERVICE: {
      return state
        .setIn(['user', 'currentServiceId'], NEW_ID)
        .setIn(['serviceById', NEW_ID], Map({
          id: NEW_ID,
          name: null,
          author: state.getIn(['user', 'name']),
          models: List(),
          endpoints: List(),
        }));
    }
    case NEXT_SCREEN: {
      switch (state.getIn(['setup', 'screen'])) {
        case SERVICE_SETUP_SCREEN_NAME:
          return state.setIn(['setup', 'screen'], SERVICE_SETUP_SCREEN_METHOD);
          break;
        case SERVICE_SETUP_SCREEN_METHOD:
          const method = state.getIn(['setup', 'method']);
          switch (method) {
            case naturalLanguage:
              return state.setIn(['setup', 'screen'], SERVICE_SETUP_SCREEN_NATURAL);
              break;
            case spreadsheet:
              return state.setIn(['setup', 'screen'], SERVICE_SETUP_SCREEN_SPREADSHEET);
              break;
            case device:
              return state.setIn(['setup', 'screen'], SERVICE_SETUP_SCREEN_NAME);
              break;
            default:
              return state;
          }
          break;
      }

      return state;
    }
    case SET_SERVICE_NAME: {
      console.log('capitalizeString', capitalizeString('ok'));
      return state
        .setIn([
          'setup',
          'name',
        ], capitalizeString(action.name));
    }
    case SET_SERVICE_CREATE_METHOD: {
      const newState = state
        .setIn([
          'setup',
          'method',
        ], action.method);
      switch (action.method) {
        case naturalLanguage: {
          return newState
            .setIn(['setup', 'naturalText'], '');
        }
        case spreadsheet: {
          return newState
            .setIn(['setup', 'spreadsheet'], '');
        }
        case device: {
          return newState
            .setIn(['setup', 'device'], Map({
              selectedDevice: '',
              flowDirection: null,
            }));
        }
        default: {
          return newState;
        }
      }
    }
    case UPDATE_NATURAL_TEXT: {
      return state.setIn(['setup', 'naturalText'], formatSentences(action.text));
    }
    case UPDATE_MODEL_PREVIEW: {
      return state
        .setIn(['setup', 'modelDefinitionPreview'], action.preview);
    }
    case SELECT_DEVICE: {
      return state
        .setIn(['setup', 'selectedDevice'], state.device);
    }
    case SET_DEVICE_FLOW_DIRECTION: {
      return state
        .setIn(['setup', 'flowDirection'], state.direction);
    }
    case RECEIVE_WEBHOOK_URL: {
      return state
        .setIn(['setup', 'webhookURL'], state.url);
    }
    case SETUP_DEVICE_QUERY: {
      return state
        .setIn(['setup', 'query'], Map({
          url: action.url,
          method: action.method,
          attributes: action.attributes,
          interval: action.interval,
        }));
    }
    case UPDATE_USER: {
      const { username, password } = action;

      if (username && password) {
        return state
          .setIn(['user', 'username'], username)
          .setIn(['user', 'password'], password);
      } else if (username) {
        return state
          .setIn(['user', 'username'], username);
      }
      return state
          .setIn(['user', 'password'], password);
    }
    case AUTH_USER_RESULT: {
      return state
        .setIn(['user', 'authenticated'], action.success)
        .setIn(['user', 'errors'], action.errors)
        .setIn(['user', 'token'], action.token);
    }
    case LOGOUT_USER: {
      return state
        .setIn(['user', 'authenticated'], false)
        .setIn(['user', 'errors'], null)
        .setIn(['user', 'token'], null);
    }
    case CHANGE_SIDEBAR_ITEM: {
      // console.log('CHANGE_SIDEBAR_ITEM', state.getIn(['dashboard', 'items']).map((item, i) => item.set('selected', i === action.index)).
          toJS());
      return state
        .setIn(
          ['dashboard', 'items'],
          state.getIn(['dashboard', 'items']).map((item, i) => item.set('selected', i === action.index)),
      );
    }
    case RECEIVE_SERVICE_LIST: {
      const services = action.services;

      const entities = normalizeServices({ services }).entities;

      const serviceIds = entities.services.undefined.services;

      const serviceById = entities.service || {};
      const modelById = entities.model || {};
      const attributeById = entities.attribute || {};
      const entryById = entities.entry || {};
      const valueById = entities.value || {};

      return state
        .setIn(['user', 'services'], fromJS(serviceIds))
        .set('serviceById', fromJS(serviceById).merge(state.get('serviceById')))
        .set('modelById', fromJS(modelById).merge(state.get('modelById')))
        .set('attributeById', fromJS(attributeById).merge(state.get('attributeById')))
        .set('entryById', fromJS(entryById).merge(state.get('entryById')))
        .set('valueById', fromJS(valueById).merge(state.get('valueById')));
    }
    case RECEIVE_SERVICE: {
      // TODO
      console.log(JSON.stringify(action.service));

      const entities = normalizeService(action.service).entities;

      console.log(entities);
      const serviceById = entities.service || {};
      const modelById = entities.model || {};
      const attributeById = entities.attribute || {};
      const entryById = entities.entry || {};
      const valueById = entities.value || {};

      return state
        .setIn(['user', 'currentServiceId'], action.service.id)
        .set('serviceById', fromJS(serviceById).merge(state.get('serviceById')))
        .set('modelById', fromJS(modelById).merge(state.get('modelById')))
        .set('attributeById', fromJS(attributeById).merge(state.get('attributeById')))
        .set('entryById', fromJS(entryById).merge(state.get('entryById')))
        .set('valueById', fromJS(valueById).merge(state.get('valueById')));
    }
    case RECEIVE_ENTRY: {
      const entities = normalizeEntry(action.entry).entities;

      const model = action.entry.ModelId;

      const valueById = entities.value || {};
      const entryById = entities.entry || {};

      const entryIdsPath = ['modelById', `${model}`, 'Entries'];

      return state
        .setIn(entryIdsPath, state.getIn(entryIdsPath).push(action.entry.id))
        .set('entryById', fromJS(entryById).merge(state.get('entryById')))
        .set('valueById', fromJS(valueById).merge(state.get('valueById')));
    }
    case RECEIVE_MODEL: {
      const entities = normalizeModel(action.model).entities;

      const modelById = entities.model || {};

      const modelIdsPath = ['serviceById', `${state.getIn(['user', 'currentServiceId'])}`, 'Models'];

      return state
```

```
        .setIn(modelIdsPath, state.getIn(modelIdsPath).push(action.model.id))
        .set('modelById', fromJS(modelById).merge(state.get('modelById')));
    }
    case RECEIVE_ATTRIBUTE: {
      const entities = normalizeAttribute(action.attribute).entities;

      const attributeById = entities.attribute || {};

      const attributeIdsPath = ['modelById', `${action.attribute.ModelId}`, 'Attributes'];

      return state
        .setIn(attributeIdsPath, (state.getIn(attributeIdsPath) || fromJS([])).push(action.attribute.id))
        .set('attributeById', fromJS(attributeById).merge(state.get('attributeById')));
    }
    case SELECT_SERVICE: {
      return state.setIn(
        [
          'user', 'currentServiceId',
        ],
        action.id,
      );
    }
    case CHANGE_SELECTED_MODEL: {
      return state.setIn(
        [
          'dashboard', 'selectedModel',
        ],
        action.id,
      );
    }
    case DELETE_ENTRY_LOCALLY: {
      const entries = ['modelById', `${action.entry.ModelId}`, 'Entries'];
      return state
        .deleteIn(['entryById', `${action.entry.id}`])
        .setIn(entries, state.getIn(entries).filter(i => i !== action.entry.id));
    }
    case DELETE_MODEL_LOCALLY: {
      const models = ['serviceById', `${state.getIn(['user', 'currentServiceId'])}`, 'Models'];
      return state
        .deleteIn(['modelById', `${action.id}`])
        .setIn(models, state.getIn(models).filter(i => i !== action.id));
    }
    case DELETE_ATTRIBUTE_LOCALLY: {
      const modelId = state.getIn(['attributeById', `${action.id}`, 'ModelId']);
      const attributes = ['modelById', `${modelId}`, 'Attributes'];
      return state
        .deleteIn(['attributeById', `${action.id}`])
        .setIn(attributes, state.getIn(attributes).filter(i => i !== action.id));
    }
    case UPDATE_VALUE_LOCALLY: {
      const valueIdsPath = ['valueById', `${action.value}`, 'Entries'];

      return state
```

```
        .setIn(['valueById', `${action.id}`, 'value'], action.value);
    }
    case UPDATE_SERVICE_LOCALLY: {
      const servicePath = ['serviceById', `${state.getIn(['user', 'currentServiceId'])}`];

      return state
        .setIn(servicePath, state.getIn(servicePath).merge(fromJS(action.changes)));
    }
    case SELECT_ATTRIBUTE: {
      return state
        .setIn(['dashboard', 'selectedAttribute'], action.id);
    }
    case UPDATE_MODEL_LOCALLY: {
      return state
        .setIn(['modelById', `${action.id}`, 'name'], action.name);
    }
    case UPDATE_ATTRIBUTE_LOCALLY: {
      const path = ['attributeById', `${action.id}`];
      return state
        .setIn(path, state.getIn(path).merge(fromJS(action.changes)));
    }
    default:
      return state;
  }
}

export default easyAPI;
```

1.130 test.js

```
import reducer from './index';
import { List, Map, fromJS } from 'immutable';
import * as types from '../actions/actionTypes';

const defaultState = fromJS({
  routing: {
    locationBeforeTransitions: null,
  },
  user: {
    currentServiceId: '1',
    username: '',
    password: '',
    authenticated: undefined,
    services: [],
    token: undefined,
  },
  dashboard: {
    items: [
      {
        name: 'Structure',
        path: '/service/dashboard/structure',
        selected: true,
      },
      {
        name: 'Entries',
        path: '/service/dashboard/entries',
      },
      {
        name: 'Pages',
        path: '/service/dashboard/pages',
      },
      {
        name: 'About',
        path: '/service/dashboard/about',
      },
    ],
    selectedAttribute: null,
    selectedModel: null,
  },
  setup: {
    name: '',
    screen: 'SERVICE_SETUP_SCREEN_NAME',
    method: 'CREATE_METHOD_NATURAL_LANGUAGE',
  },
  serviceById: {
  },
  modelById: {
  },
  attributeById: {
  },
  entryById: {},
  valueById: {},
  endpointById: {},
});

describe('easyAPI reducer', () => {
  it('should return the initial state', () => {
    expect(
      reducer(undefined, {}),
    ).toEqual(defaultState);
  });
});
```

1.131 annotateText.js

```
export default function annotateText(text = '', preview = []) {
  let html = text;
  let adj = 0;
  preview.forEach(a => {
    html = html.substr(0, a.start + adj) + '<b>' + html.substr(a.start + adj);
    adj += 3;
    html = html.substr(0, a.end + adj) + '</b>' + html.substr(a.end + adj);
    adj += 4;
  });
  return html;
}
```

1.132 API.js

```
import { getToken } from './Auth';

function curryReq(path, useToken = true, method = 'POST') {
  return async (params) => {
    const headers = {
      'Content-Type': 'application/json',
    };

    if (useToken) {
      headers.Authorization = `bearer ${getToken()}`;
    }

    const response = await fetch(`/api${path}`, {
      method,
      headers,
      body: JSON.stringify(params),
    });

    const json = await response.json();
    console.log(`API response ${method} ${path}`, params, json);
    return json;
  };
}

export const req = (path, params) => curryReq(path)(params);

export const extractModelFromText = text => curryReq('/service/parseText')({ text });

export const authenticateUser = (username, password) => curryReq('/auth/login', false)({ username, password });

export const getService = id => curryReq(`/service/${id}`, true, 'GET')({});

export const getServiceList = () => curryReq('/service', true, 'GET')();

export const postService = (name, models) => curryReq('/service', true, 'POST')({ name, models });

export const postEntry = model => curryReq('/entry', true, 'POST')({ model });
export const deleteEntry = id => curryReq('/entry', true, 'DELETE')({ id });
export const updateValue = (entry, attribute, value) => curryReq('/value', true, 'PATCH')({ entry, attribute, value });
export const updateService = (id, changes) => curryReq(`/service/${id}`, true, 'PATCH')(changes);

export const postModel = curryReq('/model', true, 'POST');
export const deleteModel = curryReq('/model', true, 'DELETE');
export const patchModel = obj => curryReq(`/model/${obj.id}`, true, 'PATCH')(obj);

export const postAttribute = curryReq('/attribute', true, 'POST');
export const patchAttribute = obj => curryReq(`/attribute/${obj.id}`, true, 'PATCH')(obj);
export const deleteAttribute = curryReq('/attribute', true, 'DELETE');


export async function postAnalyzeSpreadsheet(file) {
  const formData = new FormData();
  formData.append('spreadsheet', file);

  const headers = {
  };

  headers.Authorization = `bearer ${getToken()}`;

  const response = await fetch('/api/service/parseSpreadsheet', {
    method: 'POST',
    headers,
    body: formData,
  });

  const json = await response.json();
  console.log('FILE API response', json);
  return json;
}
```

1.133 Auth.js

```
// let savedToken;

const localStorage = window.localStorage || null;


export function saveToken(token) {
  // savedToken = token;
  if (!localStorage) return;
  localStorage.setItem('token', token);
}
export function isAuthenticated() {
  // return savedToken != null;
  if (!localStorage) return;
  return localStorage.getItem('token') != null;
}
export function removeToken() {
  // savedToken = undefined;
  if (!localStorage) return;
  localStorage.removeItem('token');
}
export function getToken() {
  // return savedToken;
  if (!localStorage) return;
  return localStorage.getItem('token');
}
```

1.134 capitalizeString.js

```
const capitalizeWord = (str) => str.charAt(0).toUpperCase() + str.slice(1);

function capitalizeString(str) {
  return str.split(/\s+/).map(capitalizeWord).join(' ');
}

export default capitalizeString;
```

1.135 createMethods.js

```
const naturalLanguage = 'CREATE_METHOD_NATURAL_LANGUAGE';
const spreadsheet = 'CREATE_METHOD_SPREADSHEET';
const device = 'CREATE_METHOD_DEVICE';

const createMethods = {
  naturalLanguage,
  spreadsheet,
  device,
}

export default createMethods;
```

1.136 formatSentences.js

```
const capitalize = (str) => str.charAt(0).toUpperCase() + str.slice(1);

function formatSentences(str) {
  return  str.split(/\.\s+/)
             .map(capitalize)
             .join('. ')
            //  .replace(/\s+/g, ' ')
            //  .replace(/\s+\./, '.')
            //  .replace(/([,.])[,.]+/, '$1');
 // TODO Improve sentence splitting

}

export default formatSentences;
```

1.137 normalizr.js

```
import { normalize, schema } from 'normalizr';

const attribute = new schema.Entity('attribute');

const value = new schema.Entity('value', {
  Attribute: attribute,
});

const entry = new schema.Entity('entry', {
  Values: [value],
});

const model = new schema.Entity('model', {
  Attributes: [attribute],
  Entries: [entry],
});

const endpoint = new schema.Entity('endpoint');

const service = new schema.Entity('service', {
  Endpoints: [endpoint],
  Models: [model],
});

const services = new schema.Entity('services', {
  services: [service],
});

export const normalizeServices = data => normalize(data, services);
export const normalizeService = data => normalize(data, service);
export const normalizeEntry = data => normalize(data, entry);
export const normalizeModel = data => normalize(data, model);
export const normalizeAttribute = data => normalize(data, attribute);
```

1.138 setupScreens.js

```
export const SERVICE_SETUP_SCREEN_NAME = 'SERVICE_SETUP_SCREEN_NAME';
export const SERVICE_SETUP_SCREEN_METHOD = 'SERVICE_SETUP_SCREEN_METHOD';
export const SERVICE_SETUP_SCREEN_NATURAL = 'SERVICE_SETUP_SCREEN_NATURAL';
export const SERVICE_SETUP_SCREEN_SPREADSHEET = 'SERVICE_SETUP_SCREEN_SPREADSHEET';
export const SERVICE_SETUP_SCREEN_DEVICE = 'SERVICE_SETUP_SCREEN_DEVICE';
```

1.139 natural.js

```
import request from 'request-promise';
import compromise from 'nlp_compromise';

/**
 * Natural Service: A service for extracting information from natural speech.
 */
import { sentences as seperateSentences } from 'sbd';

// Uses spacy to deconstruct text into a dependancy parse tree
function parse(text) {
  return request.post('http://localhost:5000/parse', {
    form: {
      text: seperateSentences(text).join('<#SENT_SEPERATOR#>'),
    },
  })
  .then(res => JSON.parse(res));
}

// In the dependency parse tree it finds first object which satisfies the condition
function find(object, condition) {
  if (condition(object)) return object;

  if (!object || !object.modifiers || object.modifiers.length === 0) return null;
  for (const child of object.modifiers) {
    const result = find(child, condition);
    if (result) return result;
  }
  return null;
}

// In the dependency parse tree it finds all objects which satisfy the condition
function findAll(object, condition) {
  let found = [];
  if (condition(object)) found.push(object);

  if (!object || !object.modifiers || object.modifiers.length === 0) return found;

  for (const child of object.modifiers) {
    const result = findAll(child, condition);
    if (result.length) found = [...result, ...found];
  }
  return found;
}

// From an array of booleans decide the final value
function decide(values) {
  if (values.length === 0) return null;

  let sum = 0;
  for (const value of values) {
    sum += Number(value);
  }
  return sum / values.length >= 0.5;
}

// Finds the existance of property. Returns string of 'required', 'optional', 'unknown'
function findIfPropertyIsRequired(prop, context) {
  //  https://en.wikipedia.org/wiki/Auxiliary_verb
  const optionalKeywords = ['may', 'might', 'could', 'should', 'maybe', 'possible', 'possibly', 'optionally', 'optional', 'ought'];
  const requiredKeywords = ['must', 'needs', 'need', 'shall', 'will'];

  const allRequiredInformation = [];

  // Find if the relationship has monads attached
  if (!context.modifiers || !context.modifiers.length) return false;
  const monads = context.modifiers.filter(o => o.arc === 'aux');

  for (const monad of monads) {
    if (optionalKeywords.find(k => k === monad.lemma)) {
      allRequiredInformation.push(false);
    } else if (requiredKeywords.find(k => k === monad.lemma)) {
      allRequiredInformation.push(true);
    }
  }

  return decide(allRequiredInformation) || false;
}

// Finds if a property has multiple instances
function findIfPropertyHasMultiple(prop) {
  const determiners = prop.modifiers ? prop.modifiers.filter(o => o.arc === 'det') : [];
  const adjModifiers = prop.modifiers ? prop.modifiers.filter(o => o.arc === 'amod') : [];
  const numModifiers = prop.modifiers ? prop.modifiers.filter(o => o.arc === 'nummod') : [];

  const combined = determiners.concat(adjModifiers).concat(numModifiers);
  // console.log(prop.lemma, ' findupper ', combined);

  // If the noun is plural then it will be multiple
  if (prop.POS_fine === 'NNS') {
    return true;
  }

  if (combined.length === 0) return false;

  // Find all information related to upper bound
  const allCardinalityInfo = [];
  for (const modifier of combined) {
    const singleKeywords = ['a', 'single', 'one'];
    const multipleKeywords = ['many', 'multiple', 'several'];
    // const singleNumbers = ['one', 'zero'];

    if (modifier.arc === 'nummod') {
      // Parse value of number
      allCardinalityInfo.push(compromise.value(modifier.lemma).number > 1);
    }

    if (singleKeywords.find(k => k === modifier.lemma)) {
      allCardinalityInfo.push(false);
    } else if (multipleKeywords.find(k => k === modifier.lemma)) {
      allCardinalityInfo.push(true);
    }
  }

  return decide(allCardinalityInfo) || false;
}

function isContainment(relationship) {
  const containmentWords = [
    'have',
    'include',
    'incorporate',
    'consist',
    'comprise',
    'contain',
  ];

  return containmentWords.find(w => w == relationship.lemma);
}

function buildPhrase(tree, transform = w => w, space = '_') {
  const othersInPhrase = tree.othersInPhrase;

  if (othersInPhrase.length) {
    return [tree, ...othersInPhrase].sort((a, b) => a.start - b.start).map(o => o.word).map(transform).join(space);
  }
  return tree.word;
}

function propertyName(prop, relationship, multiple) {
  let entity = '';

  const correctedNoun = multiple ?
    compromise.noun(prop.lemma).pluralize() :
    compromise.noun(prop.lemma).singularize();


  const othersInPhrase = prop.othersInPhrase;

  entity = buildPhrase(prop);

  if (isContainment(relationship)) {
    return entity;
  }

  const presentVerb = compromise.verb(relationship.word).to_present();

  return `${presentVerb}_${entity}`;
}

const capitalizeWord = str => str.charAt(0).toUpperCase() + str.slice(1);

function propertyType(prop, entities = []) {
  for (const entity of entities) {
    if (entity.raw === prop.raw ||
        entity.lemma === prop.lemma) {
      return capitalizeWord(entity.lemma);
    }
  }

  // Check criteria for date.
  const dateKeywords = [
    'date',
    'day', // TODO Add more keywords
  ];

  // if prop.raw.toLowerCase.includes()

  // Check criteria for number.
  const numberKeywords = [
    'number',
    'integer',
    'float',
    'double',
  ];

  // Check for integer or float

  // TODO

  return 'string';
}

function categoriseProp(prop, context, relationship, entities) {
  const multiple = findIfPropertyHasMultiple(prop);

  const type = propertyType(prop, entities);
  const name = propertyName(prop, relationship, multiple);
  const required = findIfPropertyIsRequired(prop, context);

  return {
    type,
    name,
    raw: prop.word,
    lemma: prop.lemma,
    required,
    multiple,
  };
}

```

```
function getConjuctions(object) {
  return followModifiers(object, o => o.arc === 'conj');
}

function followModifiers(tree, condition) {
  if (!tree || !tree.modifiers || tree.modifiers.length === 0) return [];

  const [modifier] = tree.modifiers.filter(condition);
  const deeperConjuctions = followModifiers(modifier, condition);

  if (deeperConjuctions.length) {
    return [
      modifier,
      ...deeperConjuctions,
    ];
  }
  if (modifier) {
    return [modifier];
  }
  return [];
}

function postprocess(modelStructure, entities) {
  for (const models of modelStructure) {
    for (const prop of models.attributes) {
      prop.type = propertyType(prop, entities);
    }
  }
}

function flatMap(array, lambda) {
  if (!array) return [];
  return Array.prototype.concat.apply([], array.map(lambda));
}

function flatten(array) {
  if (!array) return [];
  return Array.prototype.concat.apply([], array);
}

function filterTree(tree, condition, depth = 0) {
  if (!tree) return;
  if (depth === 0) tree = JSON.parse(JSON.stringify(tree)); // Clone the tree

  const modifiers = flatMap(
    tree.modifiers,
    m => filterTree(m, e => condition(e, depth, tree), depth + 1),
  );

  if (condition(tree)) {
    if (modifiers.length < 1) {
      // delete tree.modifiers;
      return Object.assign(tree, {
        modifiers: undefined,
      });
    }
    return Object.assign(tree, {
      modifiers,
    });
  }
  return modifiers;
}

function assignNounPhrase(p) {
  const preps = findAll(p, o => o.arc === 'prep');
  const prepPhrases = preps.map(
    o => [o, ...(o.modifiers.filter(m => m.arc === 'pobj'))],
  );

  const tags = ['compound', 'amod'];
  const more = findAll(p, m => tags.includes(m.arc));

  p.othersInPhrase = [...flatten(prepPhrases), ...more].sort((a, b) => a.start - b.start);

  return p;
}

async function generateModelStructure(text) {
  // Annotate raw text with POS and get dependency structure
  const parseResult = await parse(text);
  const modelStructure = [];
  let allEntities = [];

  // Useful transformations
  // Remove oxford comma!

  for (const sentenceResult of parseResult.data) {
    // Find potential entities
    // const potentialEntities = sentenceResult.parse_list
      // .filter(word => word.POS_coarse === 'NOUN');

    // Find relationships
    const potentialRelationships = sentenceResult.parse_list
      .filter(word => word.POS_fine.startsWith('V'));

    // Build up tree of words to their place in parse tree
    const tokens = sentenceResult.parse_list;
    const cleanTree = filterTree(sentenceResult.parse_tree[0], m => m.POS_fine.startsWith('V') || m.POS_fine.startsWith('N') || m.POS_fine
        === 'PRP');

    const treeIndex = {};
    const cleanTreeIndex = {};
    tokens.forEach((token) => {
      treeIndex[token.id] = find(sentenceResult.parse_tree[0], obj => obj.id === token.id);
      cleanTreeIndex[token.id] = find(cleanTree, obj => obj.id === token.id);
    });


    // console.log(cleanTree, cleanTreeIndex)

    for (const relationship of potentialRelationships) {
      // First containment
      let inTree = cleanTreeIndex[relationship.id];

      const nounTree = filterTree(inTree, m => m.POS_fine.startsWith('N') || m.POS_fine === 'PRP');
      const compareDepth = (a, b) => a.depth - b.depth;

      if (!nounTree || nounTree.length < 1) continue;
      // Find subject and object
      // console.log('\n\n\nOK ',inTree, nounTree);
      const [subject] = nounTree.filter(o => o.arc.includes('subj')).sort(compareDepth);
      const [object] = nounTree.filter(o => o.arc.includes('obj')).sort(compareDepth);

      let attributes = [];
      if (object) {
        // This is the attributes
        const fullObject = treeIndex[object.id];
        attributes = [fullObject, ...getConjuctions(fullObject)];

        attributes = attributes.map(assignNounPhrase);
      }
      let entities = [];
      if (subject) {
        // This is entities
        const fullSubject = treeIndex[subject.id];
        entities = [fullSubject, ...getConjuctions(fullSubject)];
        allEntities = [...allEntities, ...entities];
        allEntities = allEntities.map(assignNounPhrase);
      }


      inTree = treeIndex[relationship.id];

      const attributesWithTypes = [];
      for (const property of attributes) {
        attributesWithTypes.push(categoriseProp(property, inTree, relationship, entities));
      }

      for (const entity of entities) {
        const existingEntity = modelStructure.find(s => s.name === entity.lemma);

        if (existingEntity) {
          existingEntity.attributes = existingEntity.attributes.concat(attributesWithTypes);
        } else {
          modelStructure.push({
            name: entity.lemma, // buildPhrase(entity, w => capitalizeWord(w), ' '),
            raw: entity.word,
            attributes: attributesWithTypes,
          });
        }
      }
    }
  }

  postprocess(modelStructure, allEntities);
  return modelStructure;
}


const Natural = {
  _find: find,
  _findAll: findAll,
  _findIfPropertyIsRequired: findIfPropertyIsRequired,
  _findIfPropertyHasMultiple: findIfPropertyHasMultiple,
  _filterTree: filterTree,
  seperateSentences,
  generateModelStructure,
  parse,
};

export default Natural;
```

1.140 parse.js

```
import XLSX from 'xlsx';
import Natural from '../components/natural';
import { object } from 'underscore';

export function parseSpreadsheet(file) {
  // Assume spreadsheet is array of csv's
  const workbook = XLSX.readFile(file.path);

  const sheetNames = workbook.SheetNames;

  const csvs = sheetNames
    .map(name => workbook.Sheets[name])
    .map(sheet => XLSX.utils.sheet_to_csv(sheet));

  const sheetByName = object(sheetNames, csvs);

  const allModelDefinitions = [];

  for (const name of sheetNames) {
    const csv = sheetByName[name];
    const modelDefinition = {};
    const [headingLine, ...rowLines] = csv.split('\n');
    const headings = headingLine.split(',');
    const rows = rowLines
      .map(r => r.split(','))
      .filter(r => r.join('').trim().length > 0);

    console.log(rows);

    modelDefinition.name = name;

    const attributes = [];
    const entries = [];

    for (let i = 0; i < headings.length; i++) {
      const headingName = headings[i].toLowerCase();

       // Get first 20 rows for sample data
      const types = determineType(new Set(rows.slice(0, 20).map(row => findType(row[i]))));
      attributes.push(Object.assign({ name: headingName }, types));
    }

    rows.forEach((row) => {
      const entry = {};
      attributes.forEach((attribute, i) => {
        entry[attribute.name] = row[i];
      });
      entries.push(entry);
    });

    modelDefinition.entries = entries;
    modelDefinition.attributes = attributes;
    allModelDefinitions.push(modelDefinition);
  }
  return Promise.resolve(allModelDefinitions);
}

// Given a array of type information, determines the type which encompases all values
export function determineType(information) {
  let type;
  let multiple = false;
  let required = true;

  for (const value of information) {
    if (value === null || value === undefined) {
      required = false;
      continue;
    }

    if (value.type === 'string') {
      type = 'string';
    } else if (value.type === 'float') {
      if (type !== 'string') {
        type = 'float';
      }
    } else if (value.type === 'integer') {
      if (type !== 'float' || type !== 'string') {
        type = 'integer';
      }
    }

    if (value.multiple == true) {
      multiple = true;
    }
  }

  return {
    type,
    multiple,
    required,
  };
}

// Given a string, finds the most likely type
export function findType(raw) {
  // If there is no value assume null
  if ((raw === null) || (raw === undefined)) return null;

  const string = raw.trim();
  if (string.length === 0) return null;

  const object = safeJSONParse(string);
  const multiple = Array.isArray(object);
  let type;

  if (multiple) {
    type = 'string';
    type = determineType(object.map(findType)).type;
    console.log(type);
    if (type.multiple) {
      throw new Error('Multidimensional arrays are not supported!');
    }
  } else {
    // Check for floats
    if (/^-?((\d+\.\d*)|(\d+\.\d*))$/.test(string)) {
      type = 'float';
    } else if (/^-?(\d+)$/.test(string)) {
      type = 'integer';
    } else {
      type = 'string';
    }
  }

  console.log(string, type);
  return {
    type,
    multiple,
    example: string,
  };
}

function safeJSONParse(string) {
  try {
    return JSON.parse(string);
  } catch (e) {
    return null;
  }
}

export function parseNaturalLanguage(text) {
  return Natural.generateModelStructure(text);
}
```

1.141 service.js

```
import databaseModels from '../models';
import { stringToShortName } from './utils';

const { Service, Model, Attribute, Entry, Value } = databaseModels;

/* Model definition format

{
  name: string,
  modelDefinitions: [
    {
      name: string,
      attributes: [
        {
          name: string,
          type: string,
          required: boolean,
          multiple: boolean,
        }
      ],
      entries: [
        {
          [key]: value,
        }
      ]
    }
  ]
}
*/

export async function createService(name, modelDefinitions, userId) {
  let service = await Service.create({
    name,
    isPublic: false,
    shortName: stringToShortName(name),
    UserId: userId,
  });

  service = service.toJSON();

  await Model.bulkCreate(modelDefinitions.map(def => ({
    name: def.name,
    ServiceId: service.id,
    shortName: stringToShortName(def.name),
  })));

  let models = await Model.findAll({
    where: {
      ServiceId: service.id,
    },
  });

  const attributesToCreate = [];
  const entriesToCreate = [];
  const entryByIndexByModel = {};

  let i = 0;
  for (const modelDefinition of modelDefinitions) {
    const model = models[i];
    i++;
    // Create attributes
    for (const attributeDefinition of modelDefinition.attributes) {
      attributesToCreate.push({
        name: attributeDefinition.name,
        type: attributeDefinition.type,
        required: attributeDefinition.required,
        multiple: attributeDefinition.multiple,
        ModelId: model.id,
      });
    }

    if (!modelDefinition.entries || modelDefinition.entries.length === 0) {
      continue;
    }

    const entryByIndex = {};
    // Create entries
    let index = 1;
    for (const entriesDefinition of modelDefinition.entries) {
      entriesToCreate.push({
        index,
        ModelId: model.id,
      });
      index++;
      entryByIndex[index] = entriesDefinition;
    }
    entryByIndexByModel[modelDefinition.name] = entryByIndex;
  }

  await Attribute.bulkCreate(attributesToCreate);
  await Entry.bulkCreate(entriesToCreate);

  models = await Model.findAll({
    where: {
      ServiceId: service.id,
    },
    include: [{ all: true, nested: true }],
  });

  const valuesToCreate = [];

  // Index: model > entry > attribute > value

  console.log(entryByIndexByModel);

  for (const model of models) {
    for (const entry of model.Entries) {
      for (const attribute of model.Attributes) {
        console.log(model.name, entry.index, attribute.name);
        const entryDefinition = entryByIndexByModel[model.name][entry.index];
        valuesToCreate.push({
          AttributeId: attribute.id,
          EntryId: entry.id,
          value: entryDefinition && entryDefinition[attribute.name],
        });
      }
    }
  }

  await Value.bulkCreate(valuesToCreate);

  service = await Service.findOne({
    where: {
      id: service.id,
    },
    include: [{ all: true, nested: true }],
  });

  return service;
}
```

1.142 utils.js

```


export function stringToShortName(string) {
  return string.toLowerCase().replace(/\W/g, '');
}

export function encode(value, type) {
  return `${value}`;
}

export function decode(string, type) {
  switch (type) {
    case 'integer':
      return parseInt(string, 10);
    case 'float':
      return parseFloat(string);
    default:
      return string;
  }
}
```

1.143 bootstrap.js

```
/**
 * Bootstrap: All scripts that should be executed before server starts running
 */

export default function bootstrap() {
  return Promise.resolve();
}
```

1.144 connections.js

```
const connections = {
  development: {
    username: 'martinkubat',
    password: '',
    database: 'martinkubat',
    host: 'localhost',
    dialect: 'postgres',
  },
  test: {
    username: 'root',
    password: null,
    database: 'database_test',
    host: '127.0.0.1',
    dialect: 'mysql',
  },
  production: {
    username: 'root',
    password: null,
    database: 'database_production',
    host: '127.0.0.1',
    dialect: 'mysql',
  },
};

export default connections;
```

1.145 passport.js

```
import passport from 'passport';
import { Strategy as LocalStrategy } from 'passport-local';
import models from '../models';
import jwt from 'jsonwebtoken';

const { User } = models;

passport.use(new LocalStrategy({
  usernameField: 'username',
  passwordField: 'password',
  session: false,
  passReqToCallback: true,
}, (req, username, password, done) => User.findOne({
  where: {
    username,
  },
})
    .then(async (foundUser) => {
      let user;
      if (foundUser) {
        // User exists
        if (!(await foundUser.validPassword(password))) {
          console.log('Invalid password');
          return done(null, false, {
            message: 'Incorrect password.',
          });
        }
        user = foundUser;
      } else {
        // New user
        user = await User.create({
          username,
          passwordHash: User.generateHash(password),
        });
      }

      const payload = {
        user: user.id,
      };

      const token = jwt.sign(payload, 'secret');

      return done(null, {
        user: {
          username: user.username,
        },
        token,
      });
    })
    .catch(err => done(err)),
));

passport.serializeUser((user, done) => {
  done(null, user.id);
});

passport.deserializeUser((id, done) => {
  User.find({
    where: { id },
  }, (err, [user]) => {
    done(err, user);
  });
});

export default passport;
```

1.146 index.js

```
import Express from 'express';
import bodyParser from 'body-parser';
import passport from './config/passport';
import index from './routes/index';
import auth from './routes/auth';
import service from './routes/service';
import model from './routes/model';
import entry from './routes/entry';
import attribute from './routes/attribute';
import value from './routes/value';
import api from './routes/api';
import bootstrap from './config/bootstrap';
import models from './models';
import authentication from './middleware/authentication';


bootstrap().then(async () => {
  /* eslint-disable new-cap */
  const app = Express();
  const port = 9001;

  await models.sequelize.sync();

  app.use(bodyParser.json());

  app.use(passport.initialize());
  app.use(authentication);

  app.use('/api', index);
  app.use('/api/service', service);
  app.use('/api/auth', auth);
  app.use('/api/model', model);
  app.use('/api/attribute', attribute);
  app.use('/api/entry', entry);
  app.use('/api/value', value);
  app.use('/api/api/', api);

  // catch 404 and forward to error handler
  app.use((req, res, next) => {
    const err = new Error('Not Found!');
    err.status = 404;
    next(err);
  });

  // error handler
  app.use((err, req, res) => {
    // set locals, only providing error in development
    /* eslint-disable no-param-reassign */
    res.locals.message = err.message;
    res.locals.error = req.app.get('env') === 'development' ? err : {};

    // render the error page
    res.status(err.status || 500);
    res.render('error');
  });

  app.listen(port);
  console.log(`Server is running on port ${port}`);
})
.catch(err => console.error(err));
```

1.147 authentication.js

```
import jwt from 'jsonwebtoken';
import models from '../models';

const { User } = models;

export default function(req, res, next) {
  if (req.originalUrl.startsWith('/api/auth/')) {
    return next();
  }

  if (!req.headers.authorization) {
    return res.status(401).end();
  }

  const token = req.headers.authorization.split(' ')[1];
  return jwt.verify(token, 'secret', (err, decoded) => {
    if (err) return res.status(401).end();

    const userId = decoded.user;

    return User.findById(userId)
      .then(user => {
        if (user) {
          req.user = user;
          return next();
        }
        return res.status(401).end();
      })
      .catch(err => res.status(401).end());
  });
};
```

1.148 attribute.js

```
export default function (sequelize, DataTypes) {
  const Attribute = sequelize.define('Attribute', {
    name: DataTypes.STRING,
    type: DataTypes.STRING,
    multiple: DataTypes.BOOLEAN,
    required: DataTypes.BOOLEAN,
  }, {
    classMethods: {
      associate(models) {
        Attribute.belongsTo(models.Model, {
          onDelete: 'CASCADE',
          foreignKey: {
            allowNull: false,
          },
        });
      },
    },
  });

  return Attribute;
}
```

1.149 entry.js

```
export default function (sequelize, DataTypes) {
  const Entry = sequelize.define('Entry', {
    index: DataTypes.INTEGER,
  }, {
    classMethods: {
      associate(models) {
        Entry.belongsTo(models.Model, {
          onDelete: 'CASCADE',
          foreignKey: {
            allowNull: false,
          },
        });
        Entry.hasMany(models.Value);
      },
    },
  });

  return Entry;
}
```

1.150 index.js

```
import fs from 'fs';
import path from 'path';
import Sequelize from 'sequelize';
import connections from '../config/connections';

const basename = path.basename(__filename);
const env = process.env.NODE_ENV || 'development';
const db = {};

const config = connections[env];

let sequelize;
if (config.use_env_variable) {
  sequelize = new Sequelize(process.env[config.use_env_variable]);
} else {
  sequelize = new Sequelize(config.database, config.username, config.password, config);
}

import attribute from './attribute';
import entry from './entry';
import model from './model';
import service from './service';
import user from './user';
import value from './value';

const models = {
  attribute,
  entry,
  model,
  service,
  user,
  value,
};

const capitalizeString = str => str.charAt(0).toUpperCase() + str.slice(1);

for (const modelName in models) {
  if (!models.hasOwnProperty(modelName)) continue;

  db[capitalizeString(modelName)] = models[modelName](sequelize, Sequelize);
}

console.log(db);

Object.keys(db).forEach((modelName) => {
  if (db[modelName].associate) {
    db[modelName].associate(db);
  }
});

```

```
db.sequelize = sequelize;
db.Sequelize = Sequelize;

export default db;
```

1.151 model.js

```
export default function (sequelize, DataTypes) {
  const Model = sequelize.define('Model', {
    name: DataTypes.STRING,
    shortName: DataTypes.STRING,
    isFindEnabled: {
      type: DataTypes.BOOLEAN,
      defaultValue: false,
    },
    isFindOneEnabled: {
      type: DataTypes.BOOLEAN,
      defaultValue: false,
    },
    isCreateEnabled: {
      type: DataTypes.BOOLEAN,
      defaultValue: false,
    },
    isUpdateEnabled: {
      type: DataTypes.BOOLEAN,
      defaultValue: false,
    },
    isDeleteEnabled: {
      type: DataTypes.BOOLEAN,
      defaultValue: false,
    },
  }, {
    classMethods: {
      associate(models) {
        Model.belongsTo(models.Service, {
          onDelete: 'CASCADE',
          foreignKey: {
            allowNull: false,
          },
        });
        Model.hasMany(models.Attribute);
        Model.hasMany(models.Entry);
      },
    },
  });

  return Model;
}
```

1.152 service.js

```
export default function (sequelize, DataTypes) {
  const Service = sequelize.define('Service', {
    name: DataTypes.STRING,
    shortName: DataTypes.STRING,
    isPublic: DataTypes.BOOLEAN,
  }, {
    classMethods: {
      associate(models) {
        Service.belongsTo(models.User, {
          onDelete: 'CASCADE',
          foreignKey: {
            allowNull: false,
          },
        });
        Service.hasMany(models.Model);
      },
    },
  });

  return Service;
}
```

1.153 user.js

```
import bcrypt from 'bcrypt';

export default function (sequelize, DataTypes) {
  const User = sequelize.define('User', {
    username: {
      type: DataTypes.STRING,
      unique: true,
    },
    passwordHash: DataTypes.STRING,
  }, {
    classMethods: {
      associate(models) {
        User.hasMany(models.Service);
      },
      generateHash: password => bcrypt.hashSync(password, bcrypt.genSaltSync(8), null),
    },
    instanceMethods: {
      generateHash: password => bcrypt.hashSync(password, bcrypt.genSaltSync(8), null),
      validPassword(password) {
        console.log(password, this.passwordHash);
        return bcrypt.compare(password, this.passwordHash);
      },
      toJSON() {
        const response = this.get();
        response.passwordHash = undefined;
        return response;
      },
    },
  });

  return User;
}
```

1.154 value.js

```
export default function (sequelize, DataTypes) {
  const Value = sequelize.define('Value', {
    value: DataTypes.STRING,
  }, {
    classMethods: {
      associate(models) {
        Value.belongsTo(models.Entry, {
          onDelete: 'CASCADE',
          foreignKey: {
            allowNull: false,
          },
        });
        Value.belongsTo(models.Attribute);
      },
    },
  });

  return Value;
}
```

1.155 index.py

```
from flask import Flask, request, jsonify
app = Flask(__name__)
from spacyparse import parse

@app.route("/")
def index():
    return "Hello World!"

@app.route("/parse", methods=['POST'])
def dependency():
    text = request.form.get('text')

    print(text)
    result = parse(text)

    return jsonify(data=result)

if __name__ == "__main__":
    app.run()
```

1.156 spacyparse.py

```
# Credit: https://github.com/kengz/spacy-nlp/blob/master/src/py/nlp.py

# MIT License
#
# Copyright (c) 2016 Wah Loon Keng
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.

from collections import OrderedDict
from spacy.en import English
nlp = English()

# Helper methods
##########################################

def merge_ents(doc):
    '''Helper: merge adjacent entities into single tokens; modifies the doc.'''
    for ent in doc.ents:
        ent.merge(ent.root.tag_, ent.text, ent.label_)
    return doc


def format_POS(token, light=False, flat=False, depth=0):
    '''helper: form the POS output for a token'''
    subtree = OrderedDict([
        ("word", token.text),
        ("lemma", token.lemma_),  # trigger
        ("NE", token.ent_type_),  # trigger
        ("POS_fine", token.tag_),
        ("POS_coarse", token.pos_),
        ("arc", token.dep_),
        ("id", token.i),
        ("start", token.idx),
        ("depth", depth),
        ("modifiers", [])
    ])
    if light:
        subtree.pop("lemma")
        subtree.pop("NE")
    if flat:
        subtree.pop("arc")
        subtree.pop("modifiers")
    return subtree


def POS_tree_(root, light=False, depth=0):
    '''
    Helper: generate a POS tree for a root token.
    The doc must have merge_ents(doc) ran on it.
    '''
    subtree = format_POS(root, light=light, depth=depth)
    for c in root.children:
        subtree["modifiers"].append(POS_tree_(c, light=False, depth=depth+1))
    return subtree


def parse_tree(doc, light=False):
    '''generate the POS tree for all sentences in a doc'''
    merge_ents(doc)  # merge the entities into single tokens first
    return [POS_tree_(sent.root, light=light) for sent in doc.sents]


def parse_list(doc, light=False):
    '''tag the doc first by NER (merged as tokens) then
    POS. Can be seen as the flat version of parse_tree'''
    merge_ents(doc)  # merge the entities into single tokens first
    return [format_POS(token, light=light, flat=True) for token in doc]

# s = "find me flights from New York to London next month"
# doc = nlp(s)
# parse_list(doc)


# Primary methods
##########################################

def parse_sentence(sentence):
    '''
    Main method: parse an input sentence and return the nlp properties.
    '''

    doc = nlp(sentence)
    reply = OrderedDict([
        ("text", doc.text),
        ("len", len(doc)),
        ("tokens", [token.text for token in doc]),
        ("noun_phrases", [token.text for token in doc.noun_chunks]),
        ("parse_tree", parse_tree(doc)),
        ("parse_list", parse_list(doc))
    ])
    return reply

# res = parse_sentence("find me flights from New York to London next month.")


def parse(input):
    '''
    parse for multi-sentences; split and apply parse in a list.
    '''
    return [parse_sentence(sent) for sent in input.split("<#SENT_SEPERATOR#>")]
```

1.157 api.js

```
import { Router } from 'express';
import { object } from 'underscore';
import databaseModels from '../models';
import { decode } from '../components/utils';

const { Service, Model, Attribute, Entry, Value, User } = databaseModels;


/* eslint-disable new-cap */
const router = Router();

router.all('/:user/:service/:model/:id?', async (req, res) => {
  const username = req.param('user');
  const serviceShortName = req.param('service');
  const modelShortName = req.param('model');
  const id = req.param('id');
  const method = req.method;
  const input = req.body;

  let data;

  try {
    const user = await User.findOne({
      where: {
        username,
      },
    });

    const service = await Service.findOne({
      where: {
        shortName: serviceShortName,
        UserId: user.id,
      },
    });

    if (!service.isPublic) {
      return res.status(403).send({ success: false });
    }

    const model = await Model.findOne({
      where: {
        shortName: modelShortName,
      },
    });

    const attributes = await Attribute.findAll({
      where: {
        ModelId: model.id,
      },
    });

    data = { user, service, model };

    switch (method) {
      case 'GET': {
        if (id) {
          // Find One
          if (!model.isFindOneEnabled) {
            return res.status(403).send({ success: false });
          }

          const entry = await Entry.findOne({
            where: {
              index: id,
              ModelId: model.id,
            },
          });
          const values = await Value.findAll({
            where: {
              EntryId: entry.id,
            },
          });

          const valueByAttributeId = object(
            values.map(v => v.AttributeId), values.map(v => v.value),
          );
          const obj = {};
          obj.id = entry.index;
          for (const attribute of attributes) {
            obj[attribute.name] = decode(valueByAttributeId[attribute.id], attribute.type);
          }

          data = obj;
        } else {
          // Find All
          if (!model.isFindEnabled) {
            return res.status(403).send({ success: false });
          }

          const entries = await Entry.findAll({
            where: {
              ModelId: model.id,
            },
          });
          const values = await Value.findAll({
            where: {
              EntryId: entries.map(a => a.id),
            },
          });
          data = { values, attributes, entries };

          const objects = [];
          for (const entry of entries) {
            const obj = {};

            const localValues = values.filter(v => v.EntryId === entry.id);
            const valueByAttributeId = object(
              localValues.map(v => v.AttributeId), localValues.map(v => v.value),
            );
            obj.id = entry.index;
            for (const attribute of attributes) {
              obj[attribute.name] = decode(valueByAttributeId[attribute.id], attribute.type);
            }

            objects.push(obj);
          }

          data = objects;
        }
        break;
      }
      case 'POST': {
        // Create
        if (!model.isCreateEnabled) {
          return res.status(403).send({ success: false });
        }
        const newestEntry = await Entry.findOne({
          where: {
            ModelId: model.id,
          },
          order: 'index DESC',
        });

        const index = (newestEntry ? newestEntry.index : 0) + 1;

        const entry = await Entry.create({
          index,
          ModelId: model.id,
        });

        const obj = {};
        obj.id = entry.index;

        const valuePromises = [];
        for (const attribute of attributes) {
          valuePromises.push(
            Value.create({
              EntryId: entry.id,
              AttributeId: attribute.id,
              value: input[attribute.name],
            }),
          );
          obj[attribute.name] = decode(input[attribute.name], attribute.type) || null;
        }
        await Promise.all(valuePromises);
        data = obj;
        break;
      }
      case 'PATCH': {
        // Update
        if (!model.isUpdateEnabled) {
          return res.status(403).send({ success: false });
        }

        const entry = await Entry.findOne({
          where: {
            index: id,
            ModelId: model.id,
          },
        });
        const values = await Value.findAll({
          where: {
            EntryId: entry.id,
          },
        });

        const valuePromises = [];
        const valueByAttributeId = object(values.map(v => v.AttributeId), values.map(v => v));


        const obj = {};
        obj.id = entry.id;
        for (const attribute of attributes) {
          const newValue = input[attribute.name];
          if (newValue) {
            const oldValue = valueByAttributeId[attribute.id];
            if (newValue !== oldValue.value) {
              if (oldValue) {
                // Update
                valuePromises.push(
                  Value.update(
                    { value: newValue },
                    { where: { id: oldValue.id } },
                  ),
                );
              } else {
                // Create
                valuePromises.push(
                  Value.create({
                    EntryId: entry.id,
                    AttributeId: attribute.id,
                    value: newValue,
                  }),
                );
              }
            }
            obj[attribute.name] = newValue;
          } else {
            obj[attribute.name] = valueByAttributeId[attribute.id].value;
          }
        }

        await Promise.all(valuePromises);
        data = obj;

        break;
      }
      case 'DELETE': {
        // Delete
        if (!model.isDeleteEnabled) {
          return res.status(403).send({ success: false });
        }

        const entry = await Entry.findOne({
          where: {
            index: id,
            ModelId: model.id,
          },
        });

        await Value.destroy({
          where: {
            EntryId: entry.id,
          },
        });

        const result = await Entry.destroy({
          where: {
            index: id,
            ModelId: model.id,
          },
        });
        data = Boolean(result);

        break;
      }
      default: {
        return res.status(400).send({ success: false });
      }
    }
  } catch (e) {
    return res.status(500).send({ success: false, error: e });
  }

  res.send({ success: true, data });
  });


  export default router;
```

1.158 attribute.js

```
import { Router } from 'express';
import databaseModels from '../models';

const { Service, Model, Attribute, Entry, Value } = databaseModels;

/* eslint-disable new-cap */
const router = Router();

/* POST scratch. */
router.post('/', async (req, res) => {
  const modelId = req.param('model');
  const name = req.param('name');
  const type = req.param('type');
  const required = req.param('required');
  const multiple = req.param('multiple');

  try {
    const attribute = await Attribute.create({
      name,
      type,
      required,
      multiple,
      ModelId: modelId,
    });

    const response = {
      attribute,
      success: true,
    };
    return res.json(response);
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});

router.patch('/:id', async (req, res) => {
  const attributeId = req.param('id');

  const toUpdate = {};

  if (req.param('name')) {
    toUpdate.name = req.param('name');
  }
  if (req.param('type')) {
    toUpdate.type = req.param('type');
  }

  try {
    const attribute = await Attribute.update(
      toUpdate,
      { where: { id: attributeId } },
    );

    return res.json({
      attribute,
      success: true,
    });
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});

router.get('/', async (req, res) => {
  try {
    const modelId = req.param('model');
    const attributes = await Attribute.findAll({
      where: {
        ModelId: modelId,
      },
      include: [{ all: true }],
    });
    return res.json({
      attributes,
      success: true,
    });
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});

router.delete('/', async (req, res) => {
  try {
    const id = req.param('id');
    const result = await Attribute.destroy({
      where: {
        id,
      },
    });
    return res.json({
      result,
      success: true,
    });
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});

export default router;
```

1.159 auth.js

```
import { Router } from 'express';
import passport from '../config/passport';

/* eslint-disable new-cap */
const router = Router();

function validate(form) {
  const errors = {};
  let success = true;

  if (!form || !form.username || form.username.length < 5) {
    success = false;
    errors.username = 'This is not a valid username.';
  }

  if (!form || !form.password || form.password.length < 5) {
    success = false;
    errors.password = 'This password is too short.';
  }

  return {
    success,
    errors,
  };
}

/* GET index. */
router.post('/login', (req, res, next) => {
  const validation = validate({
    username: req.param('username'),
    password: req.param('password'),
  });

  if (!validation.success) {
    return res.status(400).json({
      success: false,
      errors: validation.errors,
    });
  }

  return passport.authenticate('local', (err, user) => {
    console.log(err, user);
    if (err || !user) {
      return res.status(400).json({
        success: false,
        message: 'Incorrect details',
      });
    }

    return res.status(200).json(Object.assign({
      success: true,
      errors: {},
    }, user));
  })(req, res, next);
});


export default router;
```

1.160 entry.js

```
import { Router } from 'express';
import databaseModels from '../models';

const { Service, Model, Attribute, Entry, Value } = databaseModels;

/* eslint-disable new-cap */
const router = Router();

/* POST scratch. */
router.post('/', async (req, res) => {
  const modelId = req.param('model');

  try {
    const newestEntry = await Entry.findOne({
      where: {
        ModelId: modelId,
      },
      order: 'index DESC',
    });

    const index = (newestEntry ? newestEntry.index : 0) + 1;

    const attributes = await Attribute.findAll({
      where: {
        ModelId: modelId,
      },
    });

    let entry = await Entry.create({
      index,
      ModelId: modelId,
    });

    const valuePromises = [];
    for (const attribute of attributes) {
      valuePromises.push(
        Value.create({
          EntryId: entry.id,
          AttributeId: attribute.id,
          value: '',
        }),
      );
    }
    await Promise.all(valuePromises);

    entry = await Entry.findOne({
      where: {
        id: entry.id,
      },
      include: [{ all: true }],
    });

    const response = {
      entry,
      success: true,
    };
    return res.json(response);
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});

router.get('/', async (req, res) => {
  try {
    const modelId = req.param('model');
    const entries = await Entry.findAll({
      where: {
        ModelId: modelId,
      },
      include: [{ all: true }],
    });
    return res.json({
      entries,
      success: true,
    });
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});

router.delete('/', async (req, res) => {
  try {
    const id = req.param('id');

    await Value.destroy({
      where: {
        EntryId: id,
      },
    });

    await Entry.destroy({
      where: {
        id,
      },
    });

    return res.json({
      success: true,
    });
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});

export default router;
```

1.161 index.js

```
import { Router } from 'express';
import { User, Service } from '../models';

/* eslint-disable new-cap */
const router = Router();

router.get('/models', (req, res) => {
  User.findAll({
    include: [Service],
  }).then((users) => {
    res.send(users);
  });
});

export default router;
```

1.162 model.js

```
import { Router } from 'express';
import databaseModels from '../models';
import { stringToShortName } from '../components/utils';

const { Service, Model, Attribute, Entry, Value } = databaseModels;

/* eslint-disable new-cap */
const router = Router();

/* POST scratch. */
router.post('/', async (req, res) => {
  const serviceId = req.param('service');
  const name = req.param('name');

  try {
    const model = await Model.create({
      name,
      shortName: stringToShortName(name),
      ServiceId: serviceId,
    });

    const response = {
      model,
      success: true,
    };
    return res.json(response);
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});

router.patch('/:id', async (req, res) => {
  const newName = req.param('name');
  const modelId = req.param('id');

  try {
    const model = await Model.update(
      {
        name: newName,
        shortName: stringToShortName(newName),
      },
      { where: { id: modelId } },
    );

    return res.json({
      model,
      success: true,
    });
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});

router.get('/', async (req, res) => {
  try {
    const serviceId = req.param('service');
    const model = await Model.findAll({
      where: {
        ServiceId: serviceId,
      },
      include: [{ all: true }],
    });
    return res.json({
      model,
      success: true,
    });
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});

router.delete('/', async (req, res) => {
  try {
    const id = req.param('id');
    const result = await Model.destroy({
      where: {
        id,
      },
    });
    return res.json({
      result,
      success: true,
    });
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});


export default router;
```

1.163 service.js

```
import { Router } from 'express';
import multer from 'multer';
import { parseSpreadsheet, parseNaturalLanguage } from '../components/parse';
import { createService, findServices } from '../components/service';
import databaseModels from '../models';

const { Service, Model, Attribute, Entry, Value } = databaseModels;

const upload = multer({ dest: 'upload/' });

/* eslint-disable new-cap */
const router = Router();

router.post('/parseText', (req, res) => {
  const text = req.param('text');
  return parseNaturalLanguage(text)
  .then(result => res.send(result));
});

router.post('/parseSpreadsheet', upload.single('spreadsheet'), (req, res) => {
  console.log(req.file);
  return parseSpreadsheet(req.file)
  .then(result => res.send(result));
});

/* POST scratch. */
router.post('/', async (req, res) => {
  const name = req.param('name');
  const modelDefinitions = req.param('models');

  try {
    const service = await createService(
      name,
      modelDefinitions,
      req.user.id,
    );

    const response = {
      service,
      success: true,
    };
    return res.json(response);
  } catch (e) {
    console.error(e);
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});
```

```

router.get('/', async (req, res) => {
  try {
    const services = await Service.findAll({
      where: {
        UserId: req.user.id,
      },
      include: [{ all: true, nested: true }],
    });
    return res.json({
      services,
      success: true,
    });
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});


router.get('/:id', async (req, res) => {
  try {
    const serviceId = req.param('id');
    const service = await Service.findOne({
      where: {
        id: serviceId,
        UserId: req.user.id,
      },
      include: [{ all: true, nested: true }],
    });
    return res.json({
      service,
      success: true,
    });
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});

router.patch('/:id', async (req, res) => {
  try {
    const serviceId = req.param('id');
    const toUpdate = {};

    if (req.param('name')) {
      toUpdate.name = req.param('name');
    }
    if (req.body.isPublic !== undefined) {
      toUpdate.isPublic = req.body.isPublic;
    }
    if (req.param('shortName')) {
      toUpdate.shortName = req.param('shortName');
    }

    const service = await Service.update(
      toUpdate,
      { where: { id: serviceId } },
    );
    return res.json({
      service,
      success: true,
    });
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});


export default router;
```

1.164 value.js

```
import { Router } from 'express';
import databaseModels from '../models';

const { Service, Model, Attribute, Entry, Value } = databaseModels;

/* eslint-disable new-cap */
const router = Router();

router.patch('/', async (req, res) => {
  const entryId = req.param('entry');
  const attributeId = req.param('attribute');
  const newValue = req.param('value');

  try {
    const [foundValue] = await Value.findOrCreate({
      where: {
        EntryId: entryId,
        AttributeId: attributeId,
      },
      include: [{ all: true }],
    });

    // TODO Validate new value

    const [value] = await Value.update(
      { value: newValue },
      { where: { id: foundValue.id } },
    );

    const response = {
      value,
      success: true,
    };
    return res.json(response);
  } catch (e) {
    return res.status(501).json({
      error: e,
      success: false,
    });
  }
});


export default router;
```

1.165 natural_test.js

```
import { expect } from 'chai';
import { describe, it } from 'mocha';
import Natural from '../src/components/natural';

describe('Natural Service', () => {
  it('should exist', () => {
    /* eslint-disable no-unused-expressions */
    expect(Natural).to.exist;
  });

  describe('seperateSentences', () => {
    it('should correctly seperate a string into different sentences', () => {
      const text = `On Jan. 20, former Sen. Barack Obama became the 44th
      President of the U.S. Millions attended the Inauguration.`;

      const expected = [
        'On Jan. 20, former Sen. Barack Obama became the 44th \n President of the U.S.',
        'Millions attended the Inauguration.',
      ];

      expect(Natural.seperateSentences(text)).to.deep.equal(expected);
    });
  });

  describe('parse', () => {
    it('should deconstruct a sentence and annotate recognisable entities.', async () => {
      const text = 'Bob brought the pizza to Alice.';

      const result = await Natural.parse(text);

      expect(result).to.exist;
      expect(result.data[0].parse_list.length).to.equal(7);
      expect(result.data[0].noun_phrases.length).to.equal(3);
      expect(result.data[0].text).to.equal('Bob brought the pizza to Alice.');
    });
  });

  describe('find', () => {
    it('should find first modifier in tree which satisfies condition', () => {
      const tree = {
        lemma: 'runs',
        pos: 'VERB',
        modifiers: [
          {
            lemma: 'duck',
            pos: 'NOUN',
            modifiers: [
              {
                lemma: 'yellow',
                pos: 'ADJ',
                modifiers: [],
              },
            ],
          },
        ],
      };

      const expected = {
        lemma: 'yellow',
        pos: 'ADJ',
        modifiers: [],
      };

      const result = Natural._find(tree, o => o.lemma === 'yellow');
      expect(result).to.deep.equal(expected);
    });
  });

  describe('filterTree', () => {
    it('should remove nodes which don\'t match a condition', () => {
      const tree = {
        pos: 'VBZ',
        modifiers: [
          {
            pos: 'JJ',
          },
          {
            pos: 'NN',
          },
        ],
      };

      const result = Natural._filterTree(tree, o => o.pos != 'JJ');
      console.log(result);
      const expected = {
        pos: 'VBZ',
        modifiers: [
          {
            modifiers: undefined,
            pos: 'NN',
          },
        ],
      };
      expect(result).to.deep.equal(expected);
    });

    it('should keep child nodes which match the condition', () => {
      const tree = {
        pos: 'VBZ',
        word: 'store',
        modifiers: [
          {
            pos: 'IN',
            word: 'about',
            modifiers: [
              {
                pos: 'NN',
                word: 'movies',
              },
            ],
          },
          {
            pos: 'NN',
            word: 'information',
          },
        ],
      };

      const result = Natural._filterTree(tree, o => o.pos != 'IN');
      const expected = {
        pos: 'VBZ',
        word: 'store',
        modifiers: [
          {
            pos: 'NN',
            word: 'movies',
            modifiers: undefined,
          },
          {
            pos: 'NN',
            word: 'information',
            modifiers: undefined,
          },
        ],
      };
      expect(result).to.deep.equal(expected);
    });

    it('should not alter the original tree', () => {
      const tree = {
        pos: 'VBZ',
        word: 'store',
        modifiers: [
          {
            pos: 'IN',
            word: 'about',
            modifiers: [
              {
                pos: 'NN',
                word: 'movies',
              },
            ],
          },
          {
            pos: 'NN',
            word: 'information',
          },
        ],
      };

      const copy = JSON.parse(JSON.stringify(tree));
      const result = Natural._filterTree(tree, o => o.pos != 'IN');
      expect(tree).to.deep.equal(copy);
    });
  });

  describe('findAll', () => {
    it('should find all modifiers in tree which satisfy a condition', () => {
      const tree = {
        lemma: 'runs',
        pos: 'VERB',
        modifiers: [
          {
            lemma: 'duck',
            pos: 'NOUN',
            modifiers: [
              {
                lemma: 'yellow',
                pos: 'ADJ',
                modifiers: [],
              },
              {
                lemma: 'happy',
                pos: 'ADJ',
                modifiers: [],
              },
            ],
          },
        ],
      };

      const expected = [
        {
          lemma: 'happy',
          pos: 'ADJ',
          modifiers: [],
        },
        {
          lemma: 'yellow',
          pos: 'ADJ',
          modifiers: [],
        },
      ];

      const result = Natural._findAll(tree, o => o.pos === 'ADJ');
      expect(result).to.deep.equal(expected);
    });
  });

  describe('findIfPropertyIsRequired', () => {
    it('should deduce a property is not required when no information is given', () => {
      const prop = {
        lemma: 'cat',
        modifiers: [],
      };

      const context = {
        lemma: 'play',
        modifiers: [],
      };

      const result = Natural._findIfPropertyIsRequired(prop, context);
      expect(result).to.equal(false);
    });

    it('should deduce a property is required when there is only required keywords', () => {
      const prop = {
        lemma: 'cat',
        modifiers: [],
      };

      const context = {
        lemma: 'play',
        modifiers: [
          { lemma: 'must', arc: 'aux' },
        ],
      };

      const result = Natural._findIfPropertyIsRequired(prop, context);
      expect(result).to.equal(true);
    });

    it('should deduce a property is not required when there are only optional keywords', () => {
      const prop = {
        lemma: 'cat',
        modifiers: [],
      };

      const context = {
        lemma: 'play',
        modifiers: [
          { lemma: 'might', arc: 'aux' },
        ],
      };

      const result = Natural._findIfPropertyIsRequired(prop, context);
      expect(result).to.equal(false);
    });

    it('should deduce a property is required when there are more required keywords than optional keywords', () => {
      const prop = {
        lemma: 'cat',
        modifiers: [],
      };

      const context = {
        lemma: 'play',
        modifiers: [
          { lemma: 'might', arc: 'aux' },
          { lemma: 'needs', arc: 'aux' },
          { lemma: 'must', arc: 'aux' },
        ],
      };

      const result = Natural._findIfPropertyIsRequired(prop, context);
      expect(result).to.equal(true);
    });

    it('should deduce a property is not required when there are more optional keywords than required keywords', () => {
      const prop = {
        lemma: 'cat',
        modifiers: [],
      };

      const context = {
        lemma: 'play',
        modifiers: [
          { lemma: 'might', arc: 'aux' },
          { lemma: 'may', arc: 'aux' },
          { lemma: 'could', arc: 'aux' },
          { lemma: 'needs', arc: 'aux' },
          { lemma: 'must', arc: 'aux' },
        ],
      };

      const result = Natural._findIfPropertyIsRequired(prop, context);
      expect(result).to.equal(false);
    });
  });

  describe('findIfPropertyHasMultiple', () => {
    it('should determine its singular if no information is given', () => {
      const prop = {
        lemma: 'cat',
        modifiers: [],
      };

      const result = Natural._findIfPropertyHasMultiple(prop);
      expect(result).to.equal(false);
    });

    it('should determine its multiple if word is plural', () => {
      const prop = {
        lemma: 'cats',
        POS_fine: 'NNS',
        modifiers: [],
      };

      const result = Natural._findIfPropertyHasMultiple(prop);
      expect(result).to.equal(true);
    });

    it('should determine its multiple if prop has modifiers with plural keywords', () => {
      ['det', 'amod'].forEach((arc) => {
        const prop = {
          lemma: 'cats',
          POS_fine: 'NN',
          modifiers: [
            { arc, lemma: 'many' },
          ],
        };

        const result = Natural._findIfPropertyHasMultiple(prop);
        expect(result).to.equal(true);
      });
    });

    it('should determine its singular if prop has modifiers with singular keywords', () => {
      ['det', 'amod'].forEach((arc) => {
        const prop = {
          lemma: 'cats',
          POS_fine: 'NN',
          modifiers: [
            { arc, lemma: 'single' },
          ],
        };

        const result = Natural._findIfPropertyHasMultiple(prop);
        expect(result).to.equal(false);
      });
    });

    it('should determine its singular if prop has modifiers with singular keywords', () => {
      ['det', 'amod'].forEach((arc) => {
        const prop = {
          lemma: 'cats',
          POS_fine: 'NN',
          modifiers: [
            { arc, lemma: 'single' },
          ],
        };

        const result = Natural._findIfPropertyHasMultiple(prop);
        expect(result).to.equal(false);
      });
    });

    it('should determine its singular if prop has singular number', () => {
```

```
    ['zero', 'one'].forEach((lemma) => {
      const prop = {
        lemma: 'cats',
        POS_fine: 'NN',
        modifiers: [
          { arc: 'nummod', lemma },
        ],
      };

      const result = Natural._findIfPropertyHasMultiple(prop);
      expect(result).to.equal(false);
    });
  });

  it('should determine its singular if prop has singular number', () => {
    ['twenty two', 'nine', 'fifty', 'ten thousand'].forEach((lemma) => {
      const prop = {
        lemma: 'cats',
        POS_fine: 'NN',
        modifiers: [
          { arc: 'nummod', lemma },
        ],
      };

      const result = Natural._findIfPropertyHasMultiple(prop);
      expect(result).to.equal(true);
    });
  });
});

describe('generateModelStructure', () => {
  it('should correctly analyse basic Pet model structure', async () => {
    const text = 'A pet has a name, breed and owner. The Owner has a name. The owner owns a pet. Toy has a name. Pet likes a toy.';

    const modelStructure = await Natural.generateModelStructure(text);

    const expected = [
      {
        name: 'pet',
        raw: 'pet',
        properties: [
          {
            type: 'string',
            name: 'name',
            raw: 'name',
            lemma: 'name',
            required: false,
            multiple: false,
          },
          {
            type: 'string',
            name: 'breed',
            raw: 'breed',
            lemma: 'breed',
            required: false,
            multiple: false,
          },
          {
            type: 'Owner',
            name: 'owner',
            raw: 'owner',
            lemma: 'owner',
            required: false,
            multiple: false,
          },
          {
            type: 'Toy',
            name: 'likes_toy',
            raw: 'toy',
            lemma: 'toy',
            required: false,
            multiple: false,
          },
        ],
      },
      {
        name: 'owner',
        raw: 'Owner',
        properties: [
          {
            type: 'string',
            name: 'name',
            raw: 'name',
            lemma: 'name',
            required: false,
            multiple: false,
          },
          {
            type: 'Pet',
            name: 'owns_pet',
            raw: 'pet',
            lemma: 'pet',
            required: false,
            multiple: false,
          },
        ],
      },
      {
        name: 'toy',
        raw: 'Toy',
        properties: [
          {
            type: 'string',
            name: 'name',
            raw: 'name',
            lemma: 'name',
              required: false,
              multiple: false,
            },
          ],
        },
      ];

      expect(modelStructure).to.deep.equal(expected);
    });
  });
});
```

1.166 parse_test.js

```
import { expect } from 'chai';
import { describe, it } from 'mocha';
import { parseSpreadsheet, findType, determineType } from '../src/components/parse';

describe('Parse Service', () => {
  it('should exist', () => {
    expect(parseSpreadsheet).to.exist;
  });

  describe('parseSpreadsheet', () => {
    const input = `policyID,statecode,county,eq_site_limit,hu_site_limit,fl_site_limit,fr_site_limit,tiv_2011,tiv_2012,eq_site_deductible,
        hu_site_deductible,fl_site_deductible,fr_site_deductible,point_latitude,point_longitude,line,construction,point_granularity
119736,FL,CLAY COUNTY,498960,498960,498960,498960,498960,792148.9,0,9979.2,0,0,30.102261,-81.711777,Residential,Masonry,1
448094,FL,CLAY COUNTY,1322376.3,1322376.3,1322376.3,1322376.3,1322376.3,1438163.57,0,0,0,0,30.063936,-81.707664,Residential,Masonry,3
206893,FL,CLAY COUNTY,190724.4,190724.4,190724.4,190724.4,190724.4,192476.78,0,0,0,0,30.089579,-81.700455,Residential,Wood,1
333743,FL,CLAY COUNTY,0,79520.76,0,0,79520.76,86854.48,0,0,0,0,30.063236,-81.707703,Residential,Wood,3
172534,FL,CLAY COUNTY,0,254281.5,0,254281.5,254281.5,246144.49,0,0,0,0,30.060614,-81.702675,Residential,Wood,1
785275,FL,CLAY COUNTY,0,515035.62,0,0,515035.62,884419.17,0,0,0,0,30.063236,-81.707703,Residential,Masonry,3
995932,FL,CLAY COUNTY,0,19260000,0,0,19260000,20610000,0,0,0,0,30.102226,-81.713882,Commercial,Reinforced Concrete,1
223488,FL,CLAY COUNTY,328500,328500,328500,328500,328500,348374.25,0,16425,0,0,30.102217,-81.707146,Residential,Wood,1
433512,FL,CLAY COUNTY,315000,315000,315000,315000,315000,265821.57,0,15750,0,0,30.118774,-81.704613,Residential,Wood,1
142071,FL,CLAY COUNTY,705600,705600,705600,705600,705600,1010842.56,14112,35280,0,0,30.100628,-81.703751,Residential,Masonry,1
253816,FL,CLAY COUNTY,831498.3,831498.3,831498.3,831498.3,831498.3,1117791.48,0,0,0,0,30.10216,-81.719444,Residential,Masonry,1
894922,FL,CLAY COUNTY,0,24059.09,0,0,24059.09,33952.19,0,0,0,0,30.095957,-81.695099,Residential,Wood,1
422834,FL,CLAY COUNTY,0,48115.94,0,0,48115.94,66755.39,0,0,0,0,30.100073,-81.739822,Residential,Wood,1
582721,FL,CLAY COUNTY,0,28869.12,0,0,28869.12,42826.99,0,0,0,0,30.09248,-81.725167,Residential,Wood,1
842700,FL,CLAY COUNTY,0,56135.64,0,0,56135.64,50656.8,0,0,0,0,30.101356,-81.726248,Residential,Wood,1
874333,FL,CLAY COUNTY,0,48115.94,0,0,48115.94,67905.07,0,0,0,0,30.113743,-81.727463,Residential,Wood,1
580146,FL,CLAY COUNTY,0,48115.94,0,0,48115.94,66938.9,0,0,0,0,30.121655,-81.732391,Residential,Wood,3
456149,FL,CLAY COUNTY,0,80192.49,0,0,80192.49,86421.04,0,0,0,0,30.109537,-81.741661,Residential,Wood,1
767862,FL,CLAY COUNTY,0,48115.94,0,0,48115.94,73798.5,0,0,0,0,30.11824,-81.745335,Residential,Wood,3
353022,FL,CLAY COUNTY,0,60946.79,0,0,60946.79,62467.29,0,0,0,0,30.065799,-81.717416,Residential,Wood,1
367814,FL,CLAY COUNTY,0,28869.12,0,0,28869.12,42727.74,0,0,0,0,30.082993,-81.710581,Residential,Wood,1
671392,FL,CLAY COUNTY,0,13410000,0,0,13410000,11700000,0,0,0,0,30.091921,-81.711929,Commercial,Reinforced Concrete,3
772887,FL,CLAY COUNTY,0,1669113.93,0,0,1669113.93,2099127.76,0,0,0,0,30.117352,-81.711884,Residential,Masonry,1
983122,FL,CLAY COUNTY,0,179562.23,0,0,179562.23,211372.57,0,0,0,0,30.095783,-81.713181,Residential,Wood,3
934215,FL,CLAY COUNTY,0,177744.16,0,0,177744.16,157171.16,0,0,0,0,30.110518,-81.727478,Residential,Wood,1
385951,FL,CLAY COUNTY,0,17757.58,0,0,17757.58,16948.72,0,0,0,0,30.10288,-81.705719,Residential,Wood,1
716332,FL,CLAY COUNTY,0,130129.87,0,0,130129.87,101758.43,0,0,0,0,30.068468,-81.71624,Residential,Wood,1
751262,FL,CLAY COUNTY,0,42854.77,0,0,42854.77,63592.88,0,0,0,0,30.068468,-81.71624,Residential,Wood,1
633663,FL,CLAY COUNTY,0,785.58,0,0,785.58,662.18,0,0,0,0,30.068468,-81.71624,Residential,Wood,1
105851,FL,CLAY COUNTY,0,170361.91,0,0,170361.91,177176.38,0,0,0,0,30.068468,-81.71624,Residential,Wood,1
`;
    it('should find the correct model definition', () => {
      // const result = parseSpreadsheet([input]);
      // expect(result).to.equal([
      //   {
      //     "name": "",
      //     "raw": "",
      //     "properties": [
      //        {
      //          "type": "integer",
      //          "name": "policyID",
      //          "raw": "policyID",
      //          "lemma": "policyID",
      //          "required": true,
      //          "multiple": false
      //        },
      //        {
      //          "type": "string",
      //          "name": "breed",
      //          "raw": "breed",
      //          "lemma": "breed",
      //          "required": false,
      //          "multiple": false
      //        },
      //        {
      //          "type": "Owner",
      //          "name": "owner",
      //          "raw": "owner",
      //          "lemma": "owner",
      //          "required": false,
      //          "multiple": false
      //        },
      //        {
      //          "type": "Toy",
      //          "name": "likes_toy",
      //          "raw": "toy",
      //          "lemma": "toy",
      //          "required": false,
      //          "multiple": false
      //        }
      //      ]
      //    },
      //    {
      //      "name": "owner",
      //      "raw": "Owner",
      //      "properties": [
      //        {
      //          "type": "string",
      //          "name": "name",
      //          "raw": "name",
      //          "lemma": "name",
      //          "required": false,
      //          "multiple": false
      //        },
      //        {
      //          "type": "Pet",
      //          "name": "owns_pet",
      //          "raw": "pet",
      //          "lemma": "pet",
      //          "required": false,
      //          "multiple": false
```

```
    //       }
    //     ]
    //   },
    //   {
    //     "name": "toy",
    //     "raw": "Toy",
    //     "properties": [
    //       {
    //         "type": "string",
    //         "name": "name",
    //         "raw": "name",
    //         "lemma": "name",
    //         "required": false,
    //         "multiple": false
    //       }
    //     ]
    //   }
    // ]);
  });
});

describe('findType', () => {
  it('should return null if no value is supplied', () => {
    const result = findType();
    expect(result).to.equal(null);
  });

  it('should return float if string contains one dot', () => {
    const result = findType('5.3');
    expect(result.type).to.equal('float');
  });

  it('should return string if string contains more than one dot', () => {
    const result = findType('5.3.3');
    expect(result.type).to.equal('string');
  });

  it('should return integer if string is only digits', () => {
    const result = findType('432');
    expect(result.type).to.equal('integer');
  });

  it('should return string otherwise', () => {
    const result = findType('This is a sentence.');
    expect(result.type).to.equal('string');
  });

  it('should detect arrays and find the type of elements', () => {
    const result = findType('[5.5,3.2,2.3]');

    expect(result.multiple).to.equal(true);
    expect(result.type).to.equal('float');
  });

  it('should detect arrays but throw if they are multidimensional', () => {
    const result = findType('[[5.5],[3.2],[2.3]]');

    // expect(findType).to.throw(Error); TODO Check for throwing error
  });
});

describe('determineType', () => {
  return;
  it('should return string if one of the types is string', () => {
    const result = determineType([
      {
        type: 'string',
        multiple: 'false',
      },
      {
        type: 'float',
        multiple: 'false',
      },
      {
        type: 'integer',
        multiple: 'false',
      },
    ]);

    expect(result.type).to.equal('string');
    expect(result.required).to.equal(true);
  });

  it('should return float if one of the types is float and there is no string', () => {
    const result = determineType([
      {
        type: 'float',
        multiple: 'false',
      },
      {
        type: 'integer',
        multiple: 'false',
      },
    ]);

    expect(result.type).to.equal('float');
    expect(result.required).to.equal(true);
  });

  it('should return integer if one of the types is float and there is no string', () => {
    const result = determineType([
      {
        type: 'integer',
        multiple: 'false',
      },
      {
        type: 'integer',
        multiple: 'false',
      },
    ]);

    expect(result.type).to.equal('float');
    expect(result.required).to.equal(true);
  });

  it('should not be required if one of the types is not required', () => {
    const result = determineType([
      {
        type: 'string',
        multiple: 'false',
      },
      null,
    ]);

    expect(result.type).to.equal('string');
    expect(result.required).to.equal(false);
  });
 });
});
```

1.167 service_test.js

```
import { describe, it } from 'mocha';
// import { createService } from '../src/components/service';

describe('createService', () => {
  it('should correctly create a service', () => {
  //   createService(
  //     'Cats',
  //     [
  //       {
  //         name: 'guy',
  //         attributes: [
  //           {
  //             type: 'string',
  //             name: 'name',
  //           },
  //           {
  //             type: 'integer',
  //             name: 'age',
  //           },
  //         ],
  //         entries: [
  //           {
  //             name: 'Tom',
  //             age: 50,
  //           },
  //           {
  //             name: 'Jack',
  //             age: 20,
  //           },
  //         ],
  //       },
  //     ],
  //    1,
  //   );
  });
});
```