Contents

A Extra Information

A.1 List of third party libraries and tools . . . . . .

A.2 Example of a model definition . . . . . .

B User Guide

B.1 Instructions . . . . . .

C Source code

C.1 ./frontend/src/actions/actionTypes.js . . . . . .

C.2 ./frontend/src/actions/auth/authUser.js . . . . . .

C.3 ./frontend/src/actions/auth/authUserResult.js . . . . . .

C.4 ./frontend/src/actions/auth/getServiceList.js . . . . . .

C.5 ./frontend/src/actions/auth/getUser.js . . . . . .

C.6 ./frontend/src/actions/auth/index.js . . . . . .

C.7 ./frontend/src/actions/auth/logoutUser.js . . . . . .

C.8 ./frontend/src/actions/auth/receiveService.js . . . . . .

C.9 ./frontend/src/actions/auth/receiveServiceList.js . . . . . .

C.10 ./frontend/src/actions/auth/updateUser.js . . . . . .

C.11 ./frontend/src/actions/dashboard/changeDashboardPage.js . . . . . .

C.12 ./frontend/src/actions/dashboard/changeSelectedModel.js . . . . . .

C.13 ./frontend/src/actions/dashboard/changeSidebarItem.js . . . . . .

C.14 ./frontend/src/actions/dashboard/createAttribute.js . . . . . .

C.15 ./frontend/src/actions/dashboard/createEntry.js . . . . . .

C.16 ./frontend/src/actions/dashboard/createModel.js . . . . . .

C.17 ./frontend/src/actions/dashboard/deleteAttribute.js . . . . . .

C.18 ./frontend/src/actions/dashboard/deleteAttributeLocally.js . . . . . .

C.19 ./frontend/src/actions/dashboard/deleteEntry.js . . . . . .

C.20 ./frontend/src/actions/dashboard/deleteEntryLocally.js . . . . . .

C.21 ./frontend/src/actions/dashboard/deleteModel.js . . . . . .

C.22 ./frontend/src/actions/dashboard/deleteModelLocally.js . . . . . .

C.23 ./frontend/src/actions/dashboard/receiveAttribute.js . . . . . .
C.24 ./frontend/src/actions/dashboard/receiveEntry.js . . . . . .
C.25 ./frontend/src/actions/dashboard/receiveModel.js . . . . . .
C.26 ./frontend/src/actions/dashboard/selectAttribute.js . . . . . .
C.27 ./frontend/src/actions/dashboard/updateAttribute.js . . . . . .
C.28 ./frontend/src/actions/dashboard/updateAttributeLocally.js . . . . . .
C.29 ./frontend/src/actions/dashboard/updateModel.js . . . . . .
C.30 ./frontend/src/actions/dashboard/updateModelLocally.js . . . . . .
C.31 ./frontend/src/actions/dashboard/updateService.js . . . . . .
C.32 ./frontend/src/actions/dashboard/updateServiceLocally.js . . . . . .
C.33 ./frontend/src/actions/dashboard/updateValue.js . . . . . .
C.34 ./frontend/src/actions/dashboard/updateValueLocally.js . . . . . .
C.35 ./frontend/src/actions/other/showError.js . . . . . .
C.36 ./frontend/src/actions/setup/analyseNaturalText.js . . . . . .
C.37 ./frontend/src/actions/setup/analyseSpreadsheet.js . . . . . .
C.38 ./frontend/src/actions/setup/createService.js . . . . . .
C.39 ./frontend/src/actions/setup/index.js . . . . . .
C.40 ./frontend/src/actions/setup/newService.js . . . . . .
C.41 ./frontend/src/actions/setup/nextScreen.js . . . . . .
C.42 ./frontend/src/actions/setup/receiveService.js . . . . . .
C.43 ./frontend/src/actions/setup/selectService.js . . . . . .
C.44 ./frontend/src/actions/setup/setServiceCreateMethod.js . . . . . .
C.45 ./frontend/src/actions/setup/setServiceName.js . . . . . .
C.46 ./frontend/src/actions/setup/updateModelPreview.js . . . . . .
C.47 ./frontend/src/actions/setup/updateNaturalText.js . . . . . .
C.48 ./frontend/src/components/AuthForm.jsx . . . . . .
C.49 ./frontend/src/components/Button.jsx . . . . . .
C.50 ./frontend/src/components/dashboard/about/About.jsx . . . . . .
C.51 ./frontend/src/components/dashboard/Dashboard.jsx . . . . . .
C.52 ./frontend/src/components/dashboard/entries/Column.jsx . . . . . .
C.53 ./frontend/src/components/dashboard/entries/Entries.jsx . . . . . .
C.54 ./frontend/src/components/dashboard/entries/Row.jsx . . . . . .
C.55 ./frontend/src/components/dashboard/entries/RowHeader.jsx . . . . . .
C.56 ./frontend/src/components/dashboard/entries/rowStyle.js . . . . . .
C.57 ./frontend/src/components/dashboard/entries/Tabs.jsx . . . . . .
C.58 ./frontend/src/components/dashboard/pages/Pages.jsx . . . . . .
C.59 ./frontend/src/components/dashboard/Sidebar.jsx . . . . . .
C.60 ./frontend/src/components/dashboard/SidebarItem.jsx . . . . . .

C.61 ./frontend/src/components/dashboard/structure/Attribute.jsx . . . . . .
C.62 ./frontend/src/components/dashboard/structure/DialogBox.jsx . . . . . .
C.63 ./frontend/src/components/dashboard/structure/Model.jsx . . . . . .
C.64 ./frontend/src/components/dashboard/structure/Structure.jsx . . . . . .
C.65 ./frontend/src/components/dashboard/TopBar.jsx . . . . . .
C.66 ./frontend/src/components/Frame.jsx . . . . . .
C.67 ./frontend/src/components/HomePage.jsx . . . . . .
C.68 ./frontend/src/components/Logo.jsx . . . . . .
C.69 ./frontend/src/components/MethodButton.jsx . . . . . .
C.70 ./frontend/src/components/RoundButton.jsx . . . . . .
C.71 ./frontend/src/components/ServiceList.jsx . . . . . .
C.72 ./frontend/src/components/ServiceListItem.jsx . . . . . .
C.73 ./frontend/src/components/setup/Setup.jsx . . . . . .
C.74 ./frontend/src/components/setup/SetupMethod.jsx . . . . . .
C.75 ./frontend/src/components/setup/SetupName.jsx . . . . . .
C.76 ./frontend/src/components/setup/SetupNatural.jsx . . . . . .
C.77 ./frontend/src/components/setup/SetupSpreadsheet.jsx . . . . . .
C.78 ./frontend/src/components/StyleConstant.js . . . . . .
C.79 ./frontend/src/components/TextInput.jsx . . . . . .
C.80 ./frontend/src/containers/AuthFormContainer.js . . . . . .
C.81 ./frontend/src/containers/dashboard/AboutContainer.js . . . . . .
C.82 ./frontend/src/containers/dashboard/EntriesContainer.js . . . . . .
C.83 ./frontend/src/containers/dashboard/PagesContainer.js . . . . . .
C.84 ./frontend/src/containers/dashboard/SidebarContainer.js . . . . . .
C.85 ./frontend/src/containers/dashboard/StructureContainer.js . . . . . .
C.86 ./frontend/src/containers/HomePageContainer.js . . . . . .
C.87 ./frontend/src/containers/ServiceListContainer.js . . . . . .
C.88 ./frontend/src/containers/setup/SetupContainer.js . . . . . .
C.89 ./frontend/src/containers/setup/SetupMethodContainer.js . . . . . .
C.90 ./frontend/src/containers/setup/SetupNameContainer.js . . . . . .
C.91 ./frontend/src/containers/setup/SetupNaturalContainer.js . . . . . .
C.92 ./frontend/src/containers/setup/SetupSpreadsheetContainer.js . . . . . .
C.93 ./frontend/src/index.js . . . . . .
C.94 ./frontend/src/reducers/index.js . . . . . .
C.95 ./frontend/src/utils/API.js . . . . . .
C.96 ./frontend/src/utils/Auth.js . . . . . .
C.97 ./frontend/src/utils/capitalizeString.js . . . . . .
C.98 ./frontend/src/utils/createMethods.js . . . . . .

C.99 ./frontend/src/utils/normalizr.js . . . . . .
C.100./frontend/src/utils/setupScreens.js . . . . . .
C.101./frontend/src/index.css . . . . . .
C.102./frontend/package.json . . . . . .
C.103./backend/package.json . . . . . .
C.104./backend/src/components/natural.js . . . . . .
C.105./backend/src/components/parse.js . . . . . .
C.106./backend/src/components/service.js . . . . . .
C.107./backend/src/components/utils.js . . . . . .
C.108./backend/src/config/bootstrap.js . . . . . .
C.109./backend/src/config/connections.js . . . . . .
C.110./backend/src/config/passport.js . . . . . .
C.111./backend/src/nlp/index.py . . . . . .
C.112./backend/src/nlp/spacyparse.py . . . . . .
C.113./backend/src/index.js . . . . . .
C.114./backend/src/middleware/authentication.js . . . . . .
C.115./backend/src/models/attribute.js . . . . . .
C.116./backend/src/models/entry.js . . . . . .
C.117./backend/src/models/index.js . . . . . .
C.118./backend/src/models/model.js . . . . . .
C.119./backend/src/models/service.js . . . . . .
C.120./backend/src/models/user.js . . . . . .
C.121./backend/src/models/value.js . . . . . .
C.122./backend/src/routes/api.js . . . . . .
C.123./backend/src/routes/attribute.js . . . . . .
C.124./backend/src/routes/auth.js . . . . . .
C.125./backend/src/routes/entry.js . . . . . .
C.126./backend/src/routes/model.js . . . . . .
C.127./backend/src/routes/service.js . . . . . .
C.128./backend/src/routes/value.js . . . . . .
C.129./backend/test/natural_test.js . . . . . .
C.130./backend/test/parse_test.js . . . . . .

Appendix A

Extra Information

A.1 List of third party libraries and tools

- **ESLint**: A linting utility for Javascript
- **eslint-plugin-async-await**: ESLint plugin which enables async/await functionality
- **eslint-plugin-react**: React linting rules for ESLint
- **neutrino**: A companion tool for setting up the development environment
- **bcrypt**: A library for hashing passwords
- **body-parser**:A Node.js body parsing middle for Express.js
- **chai**: A test assertion library for Node.js
- **compromise**: A Natural Language Processing module
- **express**: Minimalistic web framework for Node.js
- **immutable**: A library which provides immutable data structures
- **isomorphic-fetch**: A Javascript polyfill for fetch
- **jsonwebtoken**: Support for JSON Web Tokens
- **multer**: Node.js middleware for multipart uploads
- **natural**: Natural language facilities for node
- **passport**: Authentication middleware for Node.js
- **passport-local**: A username and password strategy for passport
- **pg**: PostgreSQL client for Node.js
- **radium**: A toolchain for React styling

- **react**: A Javascript for building user interfaces
- **react-dom**: React bindings for the browser
- **react-redux**: React bindings for Redux
- **redux**: State container for Javascript applications
- **redux-immutable**: Support for Immutable in Redux
- **request-promise**: A HTTP request client with Promise support
- **sbd**: Sentence boundary detection
- **sequelize**: A promise-based ORM
- **sequelize-cli**: A Sequelize command line interface
- **spacy**: NLP processing library with dependency parsing
- **xlsx**: Spreadsheet parser and writer in Javasript
- **normalizr**: Library for normalizing nested JSON according to a schema
- **prop-types**: Runtime type checking for React
- **react-dropzone**: A HTML5 drag-drop zone for React
- **react-router**: Declarative routing for React
- **react-router-redux**: Bindings for react-router and Redux
- **react-scripts**: A development environment setup tool
- **underscore**: A library with utility functions
- **mocha**: Test framework for Javascript
- **redux-thunk**: Thunk middleware for Redux
- **spacy-nlp**: Exposes main Spacy functionality

A.2 Example of a model definition

‘

```
[
  {
    "name": "pet",
    "properties": [
      {
```

```
      "type": "string",
      "name": "name",
      "required": false,
      "multiple": false
    },
    {
      "type": "string",
      "name": "breed",
      "required": false,
      "multiple": false
    },
    {
      "type": "Owner",
      "name": "owner",
      "required": false,
      "multiple": false
    },
    {
      "type": "Toy",
      "name": "likes_toy",
      "required": false,
      "multiple": false
    }
  ]
},
{
  "name": "owner",
  "properties": [
    {
      "type": "string",
      "name": "name",
      "required": false,
      "multiple": false
    },
    {
      "type": "Pet",
      "name": "owns_pet",
```

```
        "required": false,
        "multiple": false
      }
    ]
  },
  {
    "name": "toy",
    "properties": [
      {
        "type": "string",
        "name": "name",
        "required": false,
        "multiple": false
      }
    ]
  }
];
```

Appendix B

User Guide

B.1 Instructions

The following guide provides instructions on how to use EasyAPI to create an API.

B.1.1 Setting up the environment

In order to run this project on your machine, follow these instructions:

1. Ensure you have Python 2.6+/3.3+ installed
2. If you haven't got pip installed, follow the instructions on the official pip installation page
3. Run the command "pip install –ignore-installed -U spacy" in Terminal/Command Line to install Spacy
4. Run the command "python -m spacy download en" to download the English corpus in Spacy. This may take over 30 minutes depending on the internet connection
5. If you have any issues following the two above instructions, please refer to the official instructions on the Spacy website
6. Install PostgreSQL by following the steps on the official website
7. Install Node.js from the download links in the official website (this will also install NPM)
8. Install Flask with the command "pip install flask"
9. Install the Node.js environment runners with the command "npm install -g neutrino react-scripts"
10. Run the PostgreSQL server and note the connection details
11. In the file "backend/src/config/connections.js", enter your connection details
12. Install the dependencies for the back-end with the command "npm install" while in the "./backend" directory
13. Install the dependencies for the front-end with the command "npm install" while in the "./frontend" directory

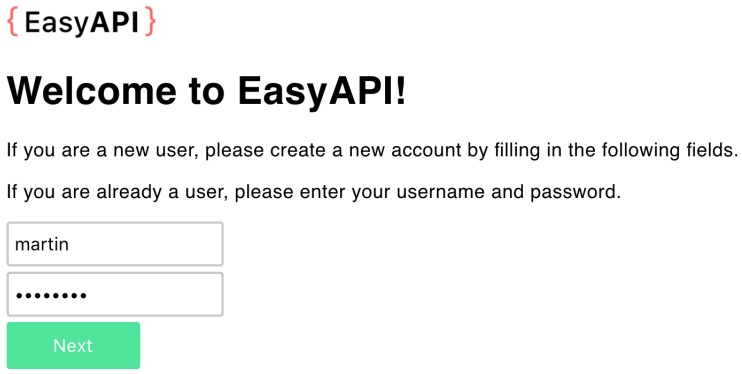

Figure B.1: The authentication screen

14. Run the python server with the command ”python backend/src/nlp/index.py”
15. In a seperate Terminal/Command Line process, run the back-end server with the command ”npm start” while in the “./backend” directory
16. In a seperate Terminal/Command Line process, run the front-end server with the command ”npm start” while in the “./frontend” directory
17. Open ”http://localhost:3000/” in a web browser and the application will appear

If you have issues running any of the commands above, try prefixing them with the “sudo” command. This will run them in an administrative mode.

B.1.2 Creating an account

When you first open the EasyAPI web application, you will be presented with an authentication screen. Here you will be asked to either create a new account or login with an existing one.

Since this is the first time, enter a username and a secret password into the text fields. Afterwards, click on the “Next” bellow.

If you receive any errors, simply adjust either your username or password to match the criteria.

B.1.3 Logging in with an account

If you are returning to the application with an existing account, you can use the authentication screen to login. Simply enter the email and password you used previously into the text-fields. The system will detect that these details were used before and will authenticate you.

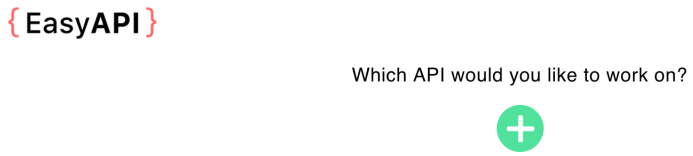

Figure B.2: List of APIs screen

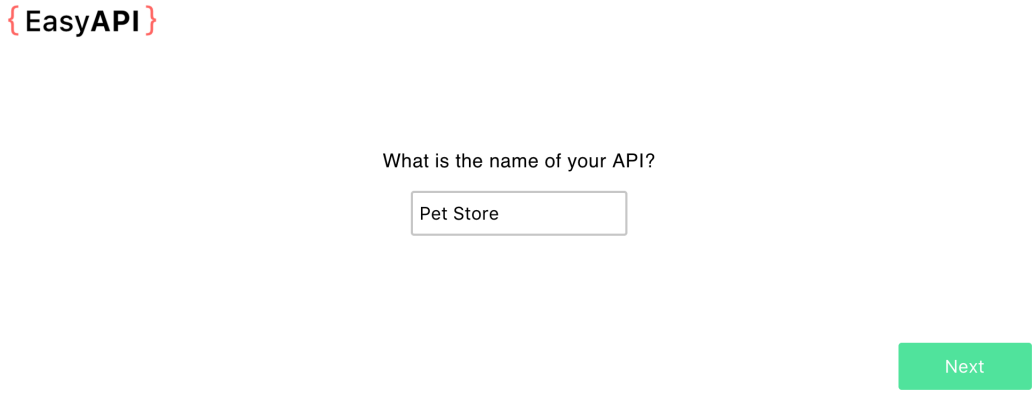

Figure B.3: Name selection screen

B.1.4 Creating a new API

After successful authentication you should be transferred to the API List screen. Here you can view and modify existing APIs and also create new ones.

To create a new API, click on the green plus button. This will transfer you to the setup screen where you can continue creating your API.

B.1.5 Creating a new API with a domain description

Once you are on the setup screen, you can follow the instructions to create your API.

We will now build an API from a domain description. In the first screen, you are asked to enter a name for the API. Think of a 1-3 word title which effectively sums up your domain. For example, if we want to create an API for a Pet store, we can use the name “Pet Store”. After entering the name, click on the “Next” button.

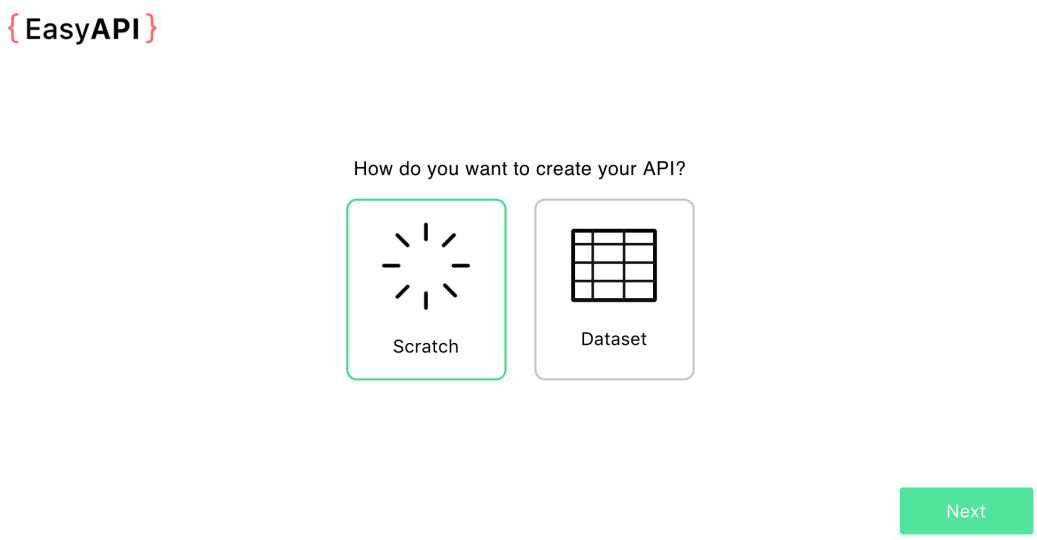

Figure B.4: Method selection screen

In the second screen, you will be asked to select how you want to create your API. Here we have two options: from a domain description or spreadsheet. In this case we want to create it from a domain description, therefore select the option by clicking on the "From scratch" option. Once it is selected, you can click on the next button.

Finally, we will be presented with a textbox where we can enter our domain description. In order for EasyAPI to extract the correct definition, it is best to describe the domain directly. One-by-one describe all the various entities that are in your domain, with their respective attributes and relationships.

For our pet store example, we may want to model a Pet store. We want to store information about our customers, employees, and our pets. This would be an appropriate domain description for those entities:

"A customer has a name. A customer owns a pet. A customer also has a phone number. An employee has a name, salary and an age. Pets have a type, a name and an customer."

When you type, the system will preview the model definition that was extracted from your description. If you think that something has been incorrectly parsed, try to rephrase it in simpler, clearer terms. If it still fails to correctly infer your domain description, in the next screen you will be able to modify it manually.

B.1.6 Creating a new API with a spreadsheet

If you already have a spreadsheet containing data which you would like to convert to the API, you can do so directly in the setup.

After creating a new API in the service list screen and typing the name of the API, select the spreadsheet from the creation methods. Afterwards, click on the "Next" button, which will transfer you to the spreadsheet setup screen.

In this screen you will see an area where you can upload your spreadsheet. Simply open File Explorer on Windows or Finder on MacOS and find the file. Once ready, simply drag and drop it onto the upload area on the screen.

It will take a few seconds to upload the file. It will attempt to extract the model definition from the spreadsheet by analysing the various pages, headings, and values. Once finished, EasyAPI will present a preview of the model

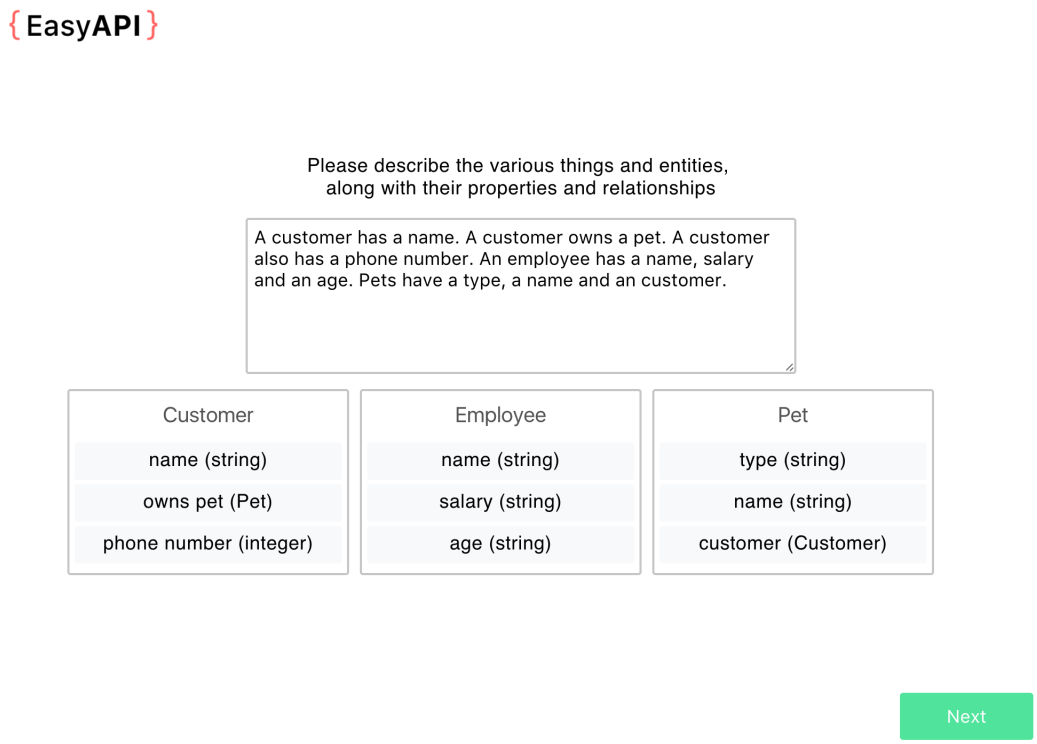

Figure B.5: Setup from scratch screen

definition.

If you find that the preview doesn't match your spreadsheet content, you can try one of these steps to help the parser:

- Make sure each spreadsheet page has its first row used for column headings.
- For each column, make sure the values are of the same type (string, integer, etc.)

If these steps did not help, you will be able to modify the model definition in the next screen.

B.1.7 Dashboard screen

Once you have finished creating your API, you will be transferred to the dashboard screen. Through this screen you will be able to modify various aspects of your API.

On the right of the screen there is a sidebar through which you can navigate to other dashboard screens. In the centre-right of the screen you will see the content of the page you have currently selected.

B.1.8 Modifying the database structure

We will first look at how we can change our database structure. Here you can modify the underlying model definition, which defines how the database is structured.

Initially you will see boxes representing the various existing entities in your database. Each box will have a list of attributes for that entity.

If you decide to create a new entity to store in the database, click on the green plus button on the top of the screen.

Figure B.6: Spreadsheet upload screen

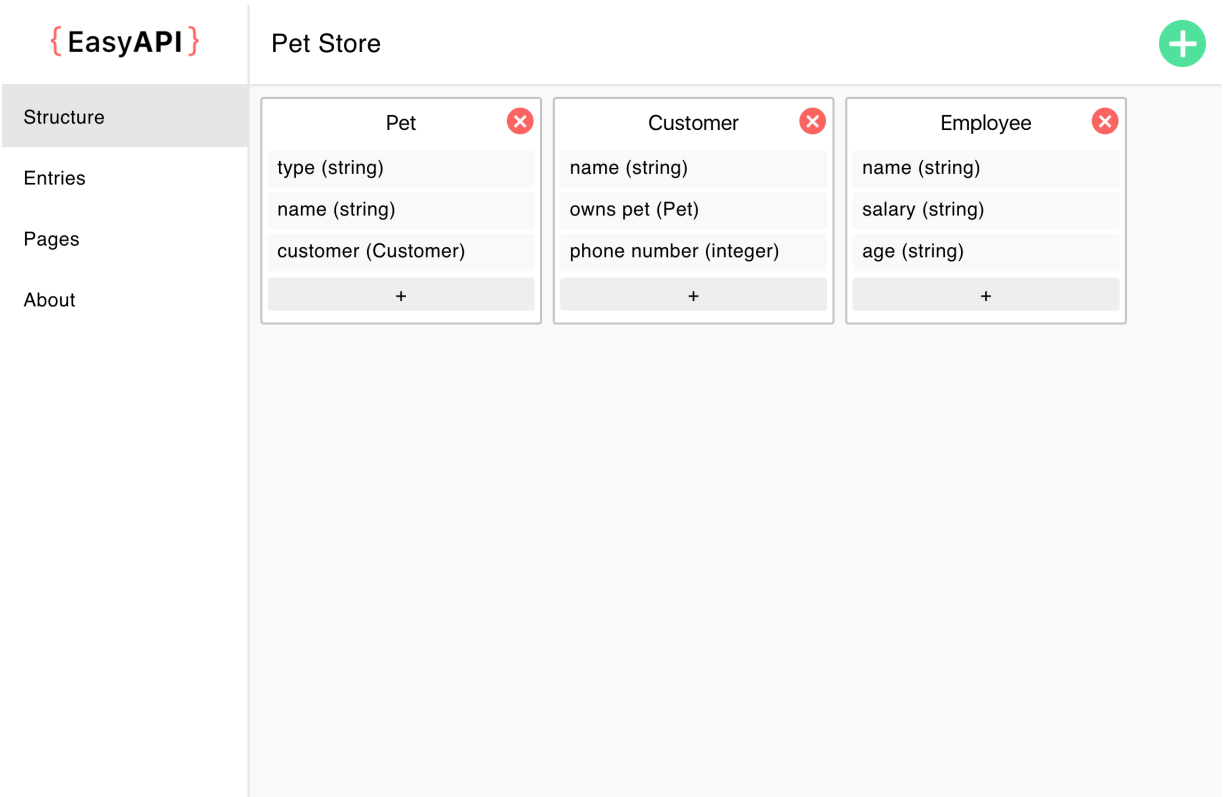

Figure B.7: Dashboard structure screen

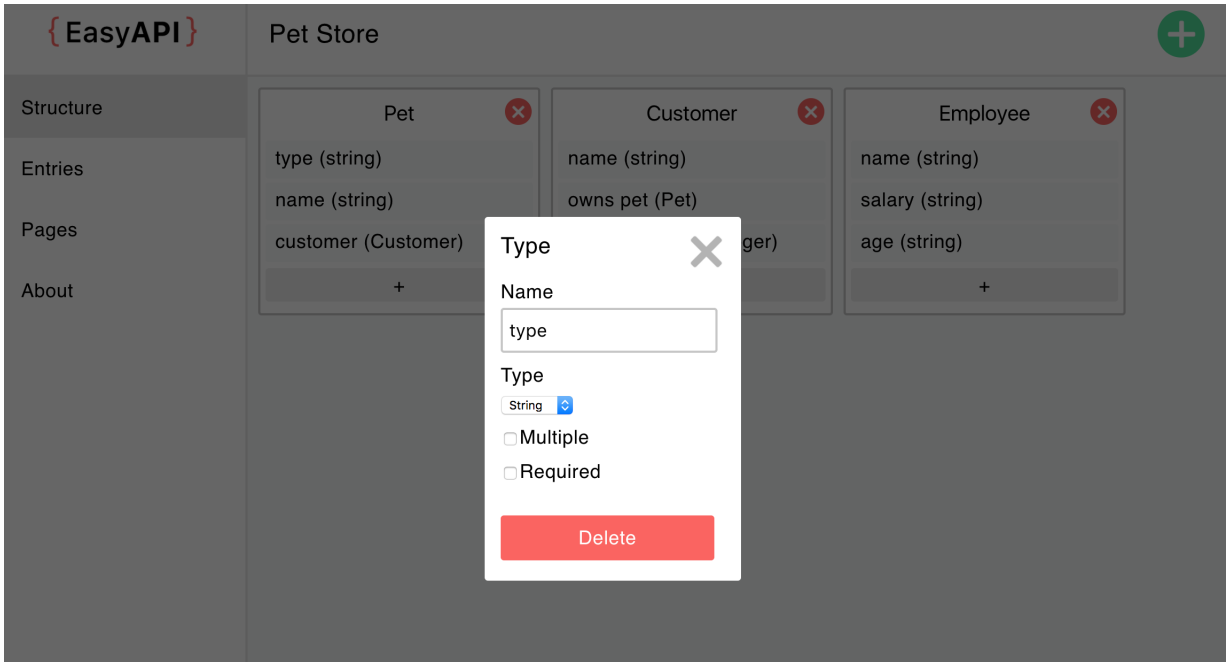

Figure B.8: Modify attribute dialog box

To modify the name of an entity, select the title of an entity-box and type in the new entity name.

To modify an attribute of an entity, click on the specific attribute you want to edit. This will open a dialogue box which will present a number of fields for that attribute and let you modify it.

To delete an entity, click on the red button in a model box.

B.1.9 Modifying the database entries

The dashboard screen also allows you to manually modify the data in your database. Start by clicking on the ”Entities” list item in the sidebar.

Your database is presented through a spreadsheet-like format. On the top of the screen you will see tabs for your corresponding entities. By clicking on each tab, the corresponding table will be shown.

To edit a value, simply click on it and type the new value. Be aware that it needs to conform to the defined type.

To delete an entry, click on red button which appears on the right to delete the row.

To create a new entry, click on the green button on the top of the screen and fill in the values in the row.

B.1.10 Modifying your public pages

The ”Pages” screen in the dashboard screen gives you control over which actions will be available to the public. Upon opening this page, you will see a list of your entities, each with a list of actions with check-boxes.

There are five optional actions for each entity:

- **Find One**: This action is used to retrieve data of one resource on the server. It corresponds to the HTTP method GET.
- **Find**: This action is similar to the one above, except it is used to retrieve a list of resources.

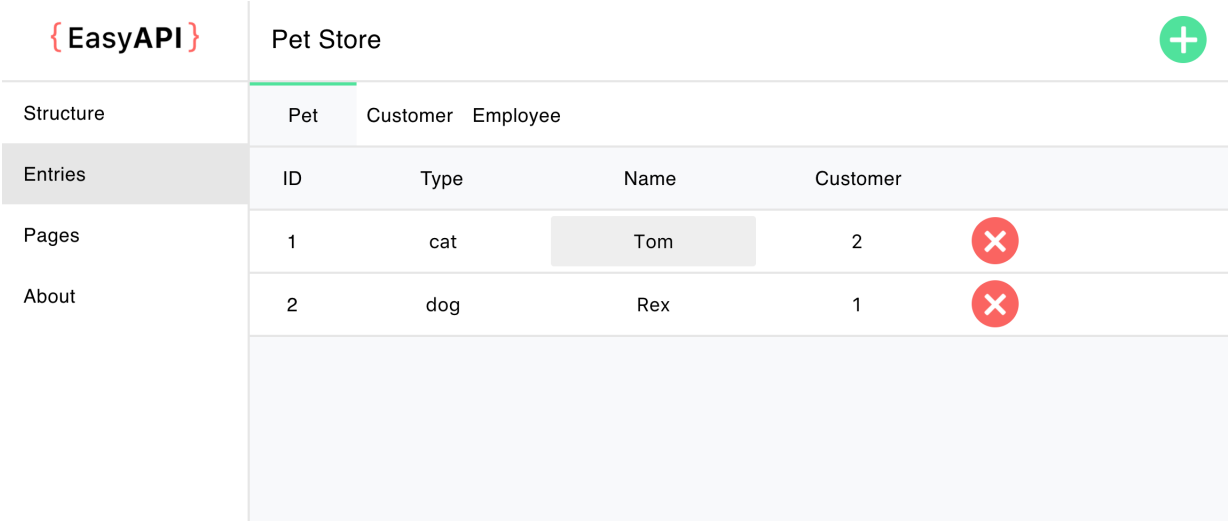

Figure B.9: Entries screen

- **Create**: This action is used to add a new entry to the database. It corresponds the the "POST" HTTP method.
- **Update**: This action allows users to modify existing resources and it corresponds to the PATCH HTTP action. Note that it does not correspond to the PUT HTTP action, as the request only requires the attributes to be updated.
- **Delete**: This action is used to delete an entry from the database.

To make a page available to the public, simply tick the checkbox next to an action. The URL of the endpoint will appear below.

B.1.11 Modifying metadata and publishing

Finally, you can also edit the metadata of the API and publish it.

You will find this screen by clicking on "About" in the sidebar.

To modify the name of the API, select the text field with the "Name" label and type your new name.

To modify the URL of the API, select the text field with the "URL" label and type in the new identifier for the API.

To publish your API, tick the checkbox next to the label "Public". You can unpublish it by un-ticking this checkbox.

B.1.12 Accessing your API

After you publish your API, the first thing you might want to do is to test it out. You can do so by returning to the "Pages" screen, where you will see URLs under every enabled action. These are in the form of URL templates, where variables in curly brackets should be replaced by a value. For instance if a URL is the following:

http://localhost:9001/api/api/petstore/cats/id

You should replace "id" with an entry id. For certain actions where a request body is required, you will need to use a tool such as Postman to make a request.

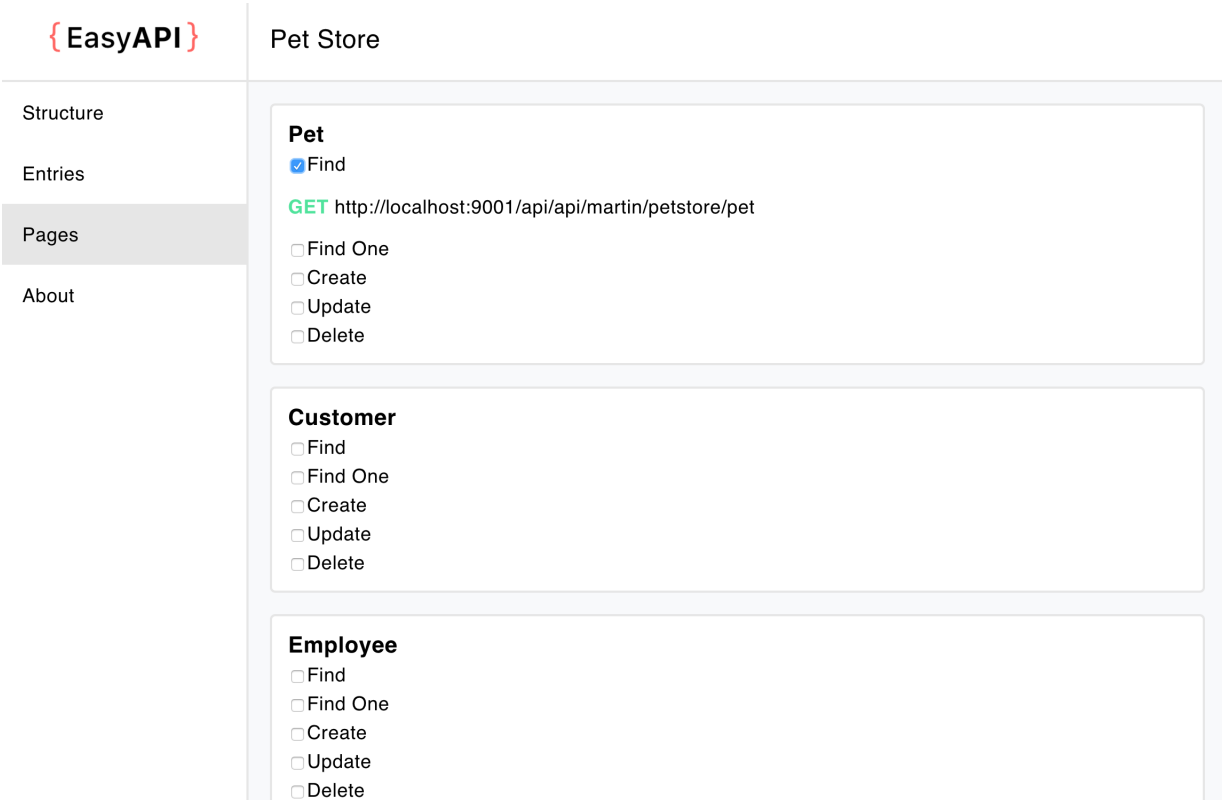

Figure B.10: Pages screen

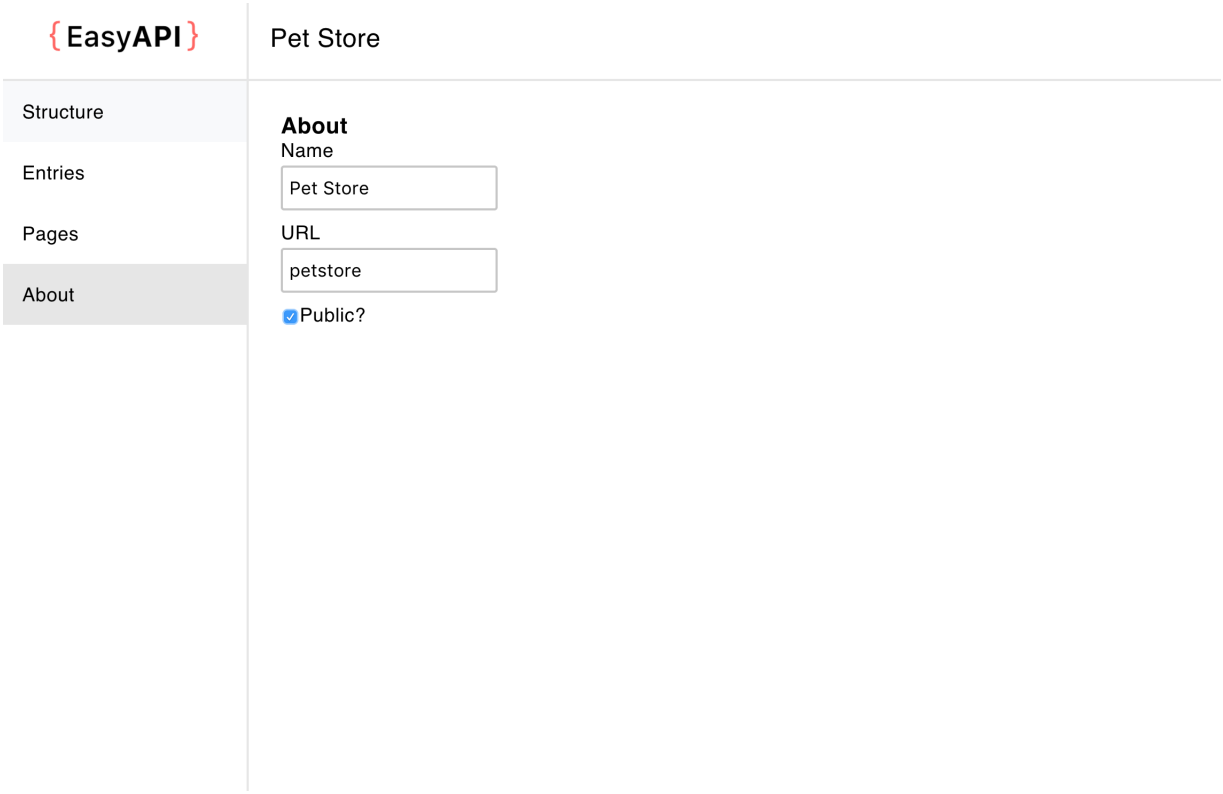

Figure B.11: About screen

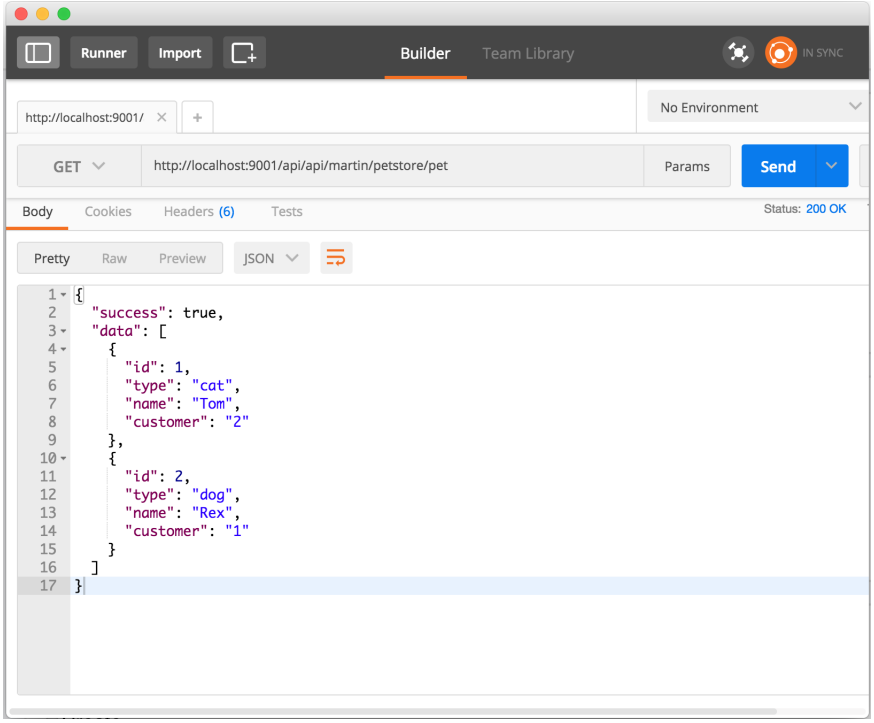

Figure B.12: Accessing API through Postman

By making a request to the specified URL with the correct HTTP action and body, you should see the expected results. For the POST and PATCH actions, the body should be specified in JSON format.

Appendix C

Source code

NOTE: Import statements from absolute paths (e.g. ”import redux from ’redux’”) indicate the use of an external library. A list of all libraries used is in the section A.1. This does not apply to relative paths (e.g. ”import getUser from ’./getUser’”).

C.1 ./frontend/src/actions/actionTypes.js

```
export { ANALYSE_NATURAL_TEXT } from ’./setup/analyseNaturalText.js’;
export { UPDATE_MODEL_PREVIEW } from ’./setup/updateModelPreview.js’;
export { NEW_SERVICE } from ’./setup/newService.js’;
export { SET_SERVICE_CREATE_METHOD } from ’./setup/setServiceCreateMethod.js’;
export { SET_SERVICE_NAME } from ’./setup/setServiceName.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’;
```

C.2 ./frontend/src/actions/auth/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));
      if (result.success) {
        saveToken(result.token);
      }
    })
    .catch(e => showError(e.message));
  };
}
```

C.3 ./frontend/src/actions/auth/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,
  };
}
```

C.4 ./frontend/src/actions/auth/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)));
  };
}
```

C.5 ./frontend/src/actions/auth/getUser.js

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

export function getUser() {
  return function (dispatch) {
    API.getUserInfo()
    .then((result) => {
      dispatch(updateUser(result.username, ''));
    })
    .catch(e => dispatch(showError(e.message)));
  };
}
```

C.6 ./frontend/src/actions/auth/index.js

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

C.7 ./frontend/src/actions/auth/logoutUser.js

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

export const LOGOUT_USER = 'LOGOUT_USER';

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

C.8 ./frontend/src/actions/auth/receiveService.js

```
export const RECEIVE_SERVICE = 'RECEIVE_SERVICE';

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

C.9 ./frontend/src/actions/auth/receiveServiceList.js

```
export const RECEIVE_SERVICE_LIST = 'RECEIVE_SERVICE_LIST';

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

C.10 ./frontend/src/actions/auth/updateUser.js

```
export const UPDATE_USER = 'UPDATE_USER';

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

C.11 ./frontend/src/actions/dashboard/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) {
    dispatch(changeSidebarItem(index));
    dispatch(push(item.path));
  };
}
```

C.12 ./frontend/src/actions/dashboard/changeSelectedModel.js

```
export const CHANGE_SELECTED_MODEL = 'CHANGE_SELECTED_MODEL';

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

C.13 ./frontend/src/actions/dashboard/changeSidebarItem.js

```
export const CHANGE_SIDEBAR_ITEM = 'CHANGE_SIDEBAR_ITEM';

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

C.14 ./frontend/src/actions/dashboard/createAttribute.js

```
import { postAttribute } from '../../utils/API';
import { showError } from '../other/showError';
import { receiveAttribute } from './receiveAttribute';
import { receiveEntry } from './receiveEntry';


export function createAttribute(model) {
  return function (dispatch, getState) {
    const state = getState().toJS();
    const newId = (state.modelById[model].Attributes ? (state.modelById[model].Attributes.
        length + 1) : 1);
    postAttribute({
      model,
      name: `attribute ${newId}`,
      type: 'string',
      required: false,
      multiple: false,
    })
    .then((result) => {
      if (result.success) {
        result.entries.map(e => dispatch(receiveEntry(e)));
        dispatch(receiveAttribute(result.attribute));
      } else {
        showError(result.error);
      }
    })
    .catch(e =>
    showError(e));
  };
}
```

C.15 ./frontend/src/actions/dashboard/createEntry.js

```
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));
  };
}
```

C.16 ./frontend/src/actions/dashboard/createModel.js

```
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();
    const newId = state.serviceById[state.user.currentServiceId].Models &&
      (state.serviceById[state.user.currentServiceId].Models.length + 1);

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

C.17 ./frontend/src/actions/dashboard/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));
  };
}
```

C.18 ./frontend/src/actions/dashboard/deleteAttributeLocally.js

```
export const DELETE_ATTRIBUTE_LOCALLY = 'DELETE_ATTRIBUTE_LOCALLY';

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

C.19 ./frontend/src/actions/dashboard/deleteEntry.js

```
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];
    API.deleteEntry(id)
    .then((result) => {
      if (result.success) {
        dispatch(deleteEntryLocally(entry));
      } else {
        showError(result.error);
      }
    })
    .catch(e =>
    showError(e));
  };
}
```

C.20 ./frontend/src/actions/dashboard/deleteEntryLocally.js

```
export const DELETE_ENTRY_LOCALLY = 'DELETE_ENTRY_LOCALLY';

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

C.21 ./frontend/src/actions/dashboard/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));
  };
}
```

C.22 ./frontend/src/actions/dashboard/deleteModelLocally.js

```
export const DELETE_MODEL_LOCALLY = 'DELETE_MODEL_LOCALLY';

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

C.23 ./frontend/src/actions/dashboard/receiveAttribute.js

```
export const RECEIVE_ATTRIBUTE = 'RECEIVE_ATTRIBUTE';

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

C.24 ./frontend/src/actions/dashboard/receiveEntry.js

```
export const RECEIVE_ENTRY = 'RECEIVE_ENTRY';

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

C.25 ./frontend/src/actions/dashboard/receiveModel.js

```
export const RECEIVE_MODEL = 'RECEIVE_MODEL';

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

C.26 ./frontend/src/actions/dashboard/selectAttribute.js

```
export const SELECT_ATTRIBUTE = 'SELECT_ATTRIBUTE';

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

C.27 ./frontend/src/actions/dashboard/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));
  };
}
```

C.28 ./frontend/src/actions/dashboard/updateAttributeLocally.js

```
export const UPDATE_ATTRIBUTE_LOCALLY = 'UPDATE_ATTRIBUTE_LOCALLY';

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

C.29 ./frontend/src/actions/dashboard/updateModel.js

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

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

C.30 ./frontend/src/actions/dashboard/updateModelLocally.js

```
export const UPDATE_MODEL_LOCALLY = 'UPDATE_MODEL_LOCALLY';

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

C.31 ./frontend/src/actions/dashboard/updateService.js

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


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));
  };
}
```

C.32 ./frontend/src/actions/dashboard/updateServiceLocally.js

```
export const UPDATE_SERVICE_LOCALLY = 'UPDATE_SERVICE_LOCALLY';

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

C.33 ./frontend/src/actions/dashboard/updateValue.js

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


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));
  };
}
```

C.34 ./frontend/src/actions/dashboard/updateValueLocally.js

```
export const UPDATE_VALUE_LOCALLY = 'UPDATE_VALUE_LOCALLY';

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

C.35 ./frontend/src/actions/other/showError.js

```
export const SHOW_ERROR = 'SHOW_ERROR';

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

C.36 ./frontend/src/actions/setup/analyseNaturalText.js

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

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(e => dispatch(showError(e.message)));
  };
}
```

C.37 ./frontend/src/actions/setup/analyseSpreadsheet.js

```


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

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

C.38 ./frontend/src/actions/setup/createService.js

```
import { push } from 'react-router-redux';
import { postService } from '../../utils/API';
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));
  };
}
```

C.39 ./frontend/src/actions/setup/index.js

```
export { analyseNaturalText } from ’./analyseNaturalText.js’;
export { updateModelPreview } from ’./updateModelPreview.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’;
```

C.40 ./frontend/src/actions/setup/newService.js

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

export const NEW_SERVICE = 'NEW_SERVICE';

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

C.41 ./frontend/src/actions/setup/nextScreen.js

```


export const NEXT_SCREEN = 'NEXT_SCREEN';

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

C.42 ./frontend/src/actions/setup/receiveService.js

```


export const RECEIVE_SERVICE = 'RECEIVE_SERVICE';

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

C.43 ./frontend/src/actions/setup/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'));
  };
}
```

C.44 ./frontend/src/actions/setup/setServiceCreateMethod.js

```


export const SET_SERVICE_CREATE_METHOD = 'SET_SERVICE_CREATE_METHOD';

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

C.45 ./frontend/src/actions/setup/setServiceName.js

```


export const SET_SERVICE_NAME = 'SET_SERVICE_NAME';

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

C.46 ./frontend/src/actions/setup/updateModelPreview.js

```

export const UPDATE_MODEL_PREVIEW = 'UPDATE_MODEL_PREVIEW';

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

C.47 ./frontend/src/actions/setup/updateNaturalText.js

```

export const UPDATE_NATURAL_TEXT = 'UPDATE_NATURAL_TEXT';

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

C.48 ./frontend/src/components/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>Welcome to EasyAPI!</h1>
    <p>If you are a new user, please create a new account by filling in the following fields.</
        p><p> If you are already a user, please enter your username and password.</p>
    <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"
          type="password"
          onChange={password => onChange({ password })}
          text={password}
        />
        <p style={style.error}>{errors.password}</p>
      </div>
      <div>
        <Button type="submit" text="Next" />
      </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;
```

C.49 ./frontend/src/components/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);
```

C.50 ./frontend/src/components/dashboard/about/About.jsx

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

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

const About = ({ name, meta, 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}>{meta[key].label}</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;
```

C.51 ./frontend/src/components/dashboard/Dashboard.jsx

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


const style = {
  base: {
    display: 'flex',
  },
  sidebar: {
    minWidth: 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;
```

C.52 ./frontend/src/components/dashboard/entries/Column.jsx

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

const style = {
  base: {
    minWidth: 70,
    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, type, isItem, onChange, first = false }) =>
  isItem ?
    <input style={[style.base, style.item, first && style.first]} type={(type === 'string' ? 'text' : 'number')} value={value || ''} onChange={onChange} /> :
    <div style={[style.base, first && style.first]}>{value}</div>;


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

```

```
export default Radium(Column);
```

C.53 ./frontend/src/components/dashboard/entries/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',
  },
};

const Entries = ({ name, entries = [], attributes = [], headers = [], onSelected, onDelete,
    onCreate, onUpdate }) =>
  <div style={style.base}>
    <TopBar name={name} enableNew 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`} />)}
      </RowHeader>

      {entries.map(entry =>
        <Row key={`${entry.id}row`} onDelete={() => onDelete(entry.realId)}>
          <Column key={`${entry.id}column`} value={entry.id} first />

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

```
44          </Row>,
45        )}
46      </div>
47    </div>;
48
49  Entries.propTypes = {
50    name: PropTypes.string.isRequired,
51    entries: PropTypes.array.isRequired,
52    attributes: PropTypes.array.isRequired,
53    headers: PropTypes.array.isRequired,
54    onSelected: PropTypes.func,
55    onDelete: PropTypes.func,
56    onCreate: PropTypes.func,
57    onUpdate: PropTypes.func,
58  };
59
60  export default Entries;
```

C.54 ./frontend/src/components/dashboard/entries/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;
```

C.55 ./frontend/src/components/dashboard/entries/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);
```

C.56 ./frontend/src/components/dashboard/entries/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;
```

C.57 ./frontend/src/components/dashboard/entries/Tabs.jsx

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

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)}
    >
      {capitalizeString(header.text)}
    </div>)}
</div>;

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

export default Radium(Tabs);
```

C.58 ./frontend/src/components/dashboard/pages/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';
import capitalizeString from '../../../utils/capitalizeString';

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,
  },
  action: {
    marginTop: 5,
  },
};


const Pages = ({ name, models = [], actions = [], onChange, urlPrefix }) => <div style={style.
```

```
      base}>
  <TopBar name={name} />
  {models.map((model, modelIndex) => <div style={style.page} key={`${modelIndex}model`}>
    <h3 style={style.title}>{capitalizeString(model.name)}</h3>
    {actions.map(action => (
      <div key={`${action.label} ${model.id}`} style={style.action}>
        <input
          id={`${action.label}-${model.id}`}
          type="checkbox"
          checked={model[action.prop]}
          onChange={e => onChange(model.id, { [action.prop]: e.target.checked })}
        />
        <label
          htmlFor={`${action.label}-${model.id}`}
          style={style.label}
        >
          {action.label}
        </label>
        {model[action.prop] && <p><b style={style.method}>{action.method}</b> {urlPrefix}{model
            .shortName}{action.suffix}</p>}
      </div>
    ))}
  </div>)}
</div>;

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

export default Pages;
```

C.59 ./frontend/src/components/dashboard/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;
```

C.60 ./frontend/src/components/dashboard/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);
```

C.61 ./frontend/src/components/dashboard/structure/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) {
  const leftPar = attribute.multiple ? '[' : '';
  const rightPar = attribute.multiple ? ']' : '';
  return `${prettify(attribute.name)}${attribute.required ? '*' : ''} (${leftPar}${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,
    required: PropTypes.bool,
  }),
  enableInteractions: PropTypes.bool,
};
```

```

export default Radium(Attribute);
```

C.62 ./frontend/src/components/dashboard/structure/DialogBox.jsx

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

const style = {
  base: {
    width: 210,
    height: 310,
    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,
    marginRight: 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',
  },
  field: {
    marginTop: 10,
  },
};

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} style={style.field}>
          <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} style={style.field}>
          <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 =>
```

```
91                <option value={option}>{capitalizeString(option)}</option>,
92              )}
93            </select>
94          </div>
95        );
96      }
97      case 'boolean': {
98        return (
99          <div key={attr.value} style={style.field}>
100           <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>
101         </div>
102       );
103     }
104   }
105   return null;
106 }
107
108 const DialogBox = ({ name, object, attributes, onChange, onClose, onDelete }) =>
109   <div>
110     <div style={style.cover} onClick={onClose} />
111     <div style={style.base}>
112       <div style={style.close} onClick={onClose} />
113       <h3 style={style.title}>{name && capitalizeString(name)}</h3>
114       {attributes.map(attr => field(attr, object, onChange))}
115       <div onClick={onDelete} style={style.delete}>Delete</div>
116     </div>
117   </div>;
118
119 DialogBox.propTypes = {
120   name: PropTypes.string,
121   object: PropTypes.object,
122   attributes: PropTypes.array,
123   onChange: PropTypes.func,
124   onClose: PropTypes.func,
125   onDelete: PropTypes.func,
126 };
127
128 export default DialogBox;
```

C.63 ./frontend/src/components/dashboard/structure/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,
    marginRight: 10,
  },
  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);
```

C.64 ./frontend/src/components/dashboard/structure/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',
    display: 'flex',
    flexFlow: 'row wrap',
    alignItems: 'flex-start',
  },
  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()} enableNew />
  {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);
```

C.65 ./frontend/src/components/dashboard/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, enableNew }) =>
  <div style={style.base}>
    <h2 style={style.h2}>{name}</h2>
    {enableNew && <RoundButton text="add" onClick={onNew} />}
  </div>;

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

export default TopBar;
```

C.66 ./frontend/src/components/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;
```

C.67 ./frontend/src/components/HomePage.jsx

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

C.68 ./frontend/src/components/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;
```

C.69 ./frontend/src/components/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);
```

C.70 ./frontend/src/components/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);
```

C.71 ./frontend/src/components/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);
```

C.72 ./frontend/src/components/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);
```

C.73 ./frontend/src/components/setup/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);
```

C.74 ./frontend/src/components/setup/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);
```

C.75 ./frontend/src/components/setup/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);
```

C.76 ./frontend/src/components/setup/SetupNatural.jsx

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

const styles = {
  nextButton: {
    marginTop: 100,
    float: 'right',
  },
  field: {
    width: 850,
    marginLeft: 'auto',
    marginRight: 'auto',
    textAlign: 'center',
    marginTop: 100,
  },
  preview: {
    display: 'flex',
    flexFlow: 'row wrap',
    alignItems: 'flex-start',
    marginTop: 10,
  },
};


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>
      <div style={styles.preview}>
        {preview && preview.map(a => <Model enableInteractions={false} model={a} />)}
      </div>
    </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);
```

C.77 ./frontend/src/components/setup/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: 850,
    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',
  },
  preview: {
    display: 'flex',
    flexFlow: 'row wrap',
    alignItems: 'flex-start',
    marginTop: 10,
  },
};

const SetupSpreadsheet = ({ onChange, onDone, preview, nextEnabled }) => (
  <div>
    <div style={styles.field}>
      <div>
        <Dropzone onDrop={onChange} style={styles.sheet}>
          Drop a spreadsheet into this area
        </Dropzone>
```

```
      </div>
      <div style={styles.preview}>
        {preview && preview.map(a => <Model enableInteractions={false} model={a} />)}
      </div>
    </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);
```

C.78 ./frontend/src/components/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}`;
```

C.79 ./frontend/src/components/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);
```

C.80 ./frontend/src/containers/AuthFormContainer.js

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

const mapStateToProps = state => ({
  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;
```

C.81 ./frontend/src/containers/dashboard/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;
```

C.82 ./frontend/src/containers/dashboard/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;
```

```
47
48        const attr = state.attributeById[valueObj.Attribute || valueObj.AttributeId];
49
50        if (!attr) continue;
51        obj[attr.name] = { value, id: valueObj.id };
52      }
53
54      entries.push(obj);
55    }
56
57    entries.sort((a, b) => a.realId - b.realId);
58    attributes.sort((a, b) => a.id - b.id);
59
60    return {
61      name: service.name,
62      headers,
63      attributes,
64      entries,
65    };
66  };
67
68  const mapDispatchToProps = (dispatch) => {
69    const update = debounce((id, attr, value) => dispatch(updateValue(id, attr, value)), 1000);
70
71    return {
72      onSelected: id => dispatch(changeSelectedModel(id)),
73      onCreate: () => dispatch(createEntry()),
74      onDelete: id => dispatch(deleteEntry(id)),
75      onUpdate: (id, attr, value, valueId) => {
76        valueId && dispatch(updateValueLocally(id, valueId, value));
77        update(id, attr, value);
78      },
79    };
80  };
81
82  const EntriesContainer = connect(
83    mapStateToProps,
84    mapDispatchToProps,
85  )(Entries);
86
87  export default EntriesContainer;
```

C.83 ./frontend/src/containers/dashboard/PagesContainer.js

```
import { connect } from 'react-redux';
import { updateModel } from '../../actions/dashboard/updateModel';
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',
      suffix: '',
    },
    {
      label: 'Find One',
      prop: 'isFindOneEnabled',
      method: 'GET',
      suffix: '/:id',
    },
    {
      label: 'Create',
      prop: 'isCreateEnabled',
      method: 'POST',
      suffix: '',
    },
    {
      label: 'Update',
      prop: 'isUpdateEnabled',
      method: 'PATCH',
      suffix: '/:id',
    },
    {
      label: 'Delete',
      prop: 'isDeleteEnabled',
      method: 'DELETE',
      suffix: '/:id',
    },
  ];

  const urlPrefix = `http://localhost:9001/api/api/${state.user.username}/${service.shortName
      }/`;
```

```
46
47    return {
48      name: service.name,
49      actions,
50      models,
51      urlPrefix,
52    };
53  };
54
55  const mapDispatchToProps = dispatch => ({
56    onChange: (id, changes) => {
57      dispatch(updateModel(id, changes));
58    },
59  });
60
61  const PagesContainer = connect(
62    mapStateToProps,
63    mapDispatchToProps,
64  )(Pages);
65
66  export default PagesContainer;
```

C.84 ./frontend/src/containers/dashboard/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;
```

C.85 ./frontend/src/containers/dashboard/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)),
});
```

```
47
48
49  const StructureContainer = connect(
50    mapStateToProps,
51    mapDispatchToProps,
52  )(Structure);
53
54  export default StructureContainer;
```

C.86 ./frontend/src/containers/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;
```

C.87 ./frontend/src/containers/ServiceListContainer.js

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

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()); dispatch(getUser()); },
  onSelect: id =>
    dispatch(selectService(id)),
  onCreate: () => dispatch(newService()),
});


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

export default ServiceListContainer;
```

C.88 ./frontend/src/containers/setup/SetupContainer.js

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

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

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

export default SetupContainer;
```

C.89 ./frontend/src/containers/setup/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;
```

C.90 ./frontend/src/containers/setup/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;
```

C.91 ./frontend/src/containers/setup/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']);
  return {
    text: state.getIn(['setup', 'naturalText']),
    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;
```

C.92 ./frontend/src/containers/setup/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,
    nextEnabled: preview && !!preview.length,
  };
};


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

});

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

export default SetupSpreadsheetContainer;
```

C.93 ./frontend/src/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) {
  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
```

```
46          path="/"
47          component={HomePageContainer}
48        />
49        <Route
50          path="/services"
51          component={ServiceListContainer}
52        />
53        <Route
54          path="/service/setup"
55          component={SetupContainer}
56          onEnter={requireAuth}
57        />
58        <Route
59          path="/service/dashboard"
60          component={Dashboard}
61          onEnter={requireAuth}
62        >
63          <Route
64            path="structure"
65            component={StructureContainer}
66          />
67          <Route
68            path="entries"
69            component={EntriesContainer}
70          />
71          <Route
72            path="pages"
73            component={PagesContainer}
74          />
75          <Route
76            path="about"
77            component={AboutContainer}
78          />
79          <IndexRedirect to="structure" />
80        </Route>
81
82      </Router>
83    </Provider>,
84    document.getElementById('root'),
85  );
86
87  r();
88  store.subscribe(r);
```

C.94 ./frontend/src/reducers/index.js

```

import { List, Map, fromJS } from 'immutable';
import {
  LOCATION_CHANGE,
} from 'react-router-redux';
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 createMethods from '../utils/createMethods';
import { isAuthenticated, getToken } from '../utils/Auth';
import { normalizeServices, normalizeService, normalizeEntry, normalizeModel,
    normalizeAttribute } from '../utils/normalizr';
import {
  SERVICE_SETUP_SCREEN_METHOD,
  SERVICE_SETUP_SCREEN_NAME,
  SERVICE_SETUP_SCREEN_NATURAL,
  SERVICE_SETUP_SCREEN_SPREADSHEET,
```

```
} 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);
        case SERVICE_SETUP_SCREEN_METHOD:
          const method = state.getIn(['setup', 'method']);
          switch (method) {
            case naturalLanguage:
              return state.setIn(['setup', 'screen'], SERVICE_SETUP_SCREEN_NATURAL);
            case spreadsheet:
              return state.setIn(['setup', 'screen'], SERVICE_SETUP_SCREEN_SPREADSHEET);
            case device:
              return state.setIn(['setup', 'screen'], SERVICE_SETUP_SCREEN_NAME);
            default:
```

```
140                 return state;
141             }
142         }
143
144         return state;
145       }
146       case SET_SERVICE_NAME: {
147         return state
148           .setIn([
149             'setup',
150             'name',
151           ], capitalizeString(action.name));
152       }
153       case SET_SERVICE_CREATE_METHOD: {
154         const newState = state
155           .setIn([
156             'setup',
157             'method',
158           ], action.method);
159         switch (action.method) {
160           case naturalLanguage: {
161             return newState
162               .setIn(['setup', 'naturalText'], '');
163           }
164           case spreadsheet: {
165             return newState
166               .setIn(['setup', 'spreadsheet'], '');
167           }
168           case device: {
169             return newState
170               .setIn(['setup', 'device'], Map({
171                 selectedDevice: '',
172                 flowDirection: null,
173               }));
174           }
175           default: {
176             return newState;
177           }
178         }
179       }
180       case UPDATE_NATURAL_TEXT: {
181         return state.setIn(['setup', 'naturalText'], action.text);
182       }
183       case UPDATE_MODEL_PREVIEW: {
184         return state
185           .setIn(['setup', 'modelDefinitionPreview'], action.preview);
186       }
```

```
    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: {
      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: {
      const entities = normalizeService(action.service).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: {
```

```
280         const entities = normalizeEntry(action.entry).entities;
281
282         const model = action.entry.ModelId;
283
284         const valueById = entities.value || {};
285         const entryById = entities.entry || {};
286
287         const entryIdsPath = ['modelById', `${model}`, 'Entries'];
288
289         const existingEntries = state
290           .getIn(entryIdsPath);
291         const newEntries = existingEntries ?
292           existingEntries.toSet()
293           .union(fromJS([action.entry.id]).toSet())
294           .toList() :
295           fromJS([action.entry.id]);
296
297         return state
298           .setIn(entryIdsPath, newEntries)
299           .set('entryById', fromJS(state.get('entryById')).merge(entryById))
300           .set('valueById', fromJS(state.get('valueById').merge(valueById)));
301       }
302       case RECEIVE_MODEL: {
303         const entities = normalizeModel(action.model).entities;
304
305         const modelById = entities.model || {};
306
307         const modelIdsPath = ['serviceById', `${state.getIn(['user', 'currentServiceId'])}`, '
              Models'];
308
309         return state
310           .setIn(modelIdsPath, state.getIn(modelIdsPath).push(action.model.id))
311           .set('modelById', fromJS(modelById).merge(state.get('modelById')));
312       }
313       case RECEIVE_ATTRIBUTE: {
314         const entities = normalizeAttribute(action.attribute).entities;
315
316         const attributeById = entities.attribute || {};
317
318         const attributeIdsPath = ['modelById', `${action.attribute.ModelId}`, 'Attributes'];
319
320         return state
321           .setIn(attributeIdsPath, (state.getIn(attributeIdsPath) || fromJS([])).push(action.
              attribute.id))
322           .set('attributeById', fromJS(attributeById).merge(state.get('attributeById')));
323       }
324       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', `${action.id}`])
        .setIn(attributes, state.getIn(attributes).filter(i => i !== action.id));
    }
    case UPDATE_VALUE_LOCALLY: {
      return state
        .setIn(['valueById', `${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: {
      const modelPath = ['modelById', `${action.id}`];
      return state
        .setIn(modelPath, state.getIn(modelPath).merge(fromJS(action.changes)));
    }
    case UPDATE_ATTRIBUTE_LOCALLY: {
      const path = ['attributeById', `${action.id}`];
      return state
        .setIn(path, state.getIn(path).merge(fromJS(action.changes)));
    }
    default:
      return state;
  }
}

export default easyAPI;
```

C.95 ./frontend/src/utils/API.js

```
import { getToken, saveToken } 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();

    if (json.token) {
      saveToken(json.token);
    }

    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 getUserInfo = () => curryReq('/auth/profile', true)();

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 = (id, obj) => curryReq(`/model/${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();

  return json;
}
```

C.96 ./frontend/src/utils/Auth.js

```
const localStorage = window.localStorage || null;


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

C.97 ./frontend/src/utils/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;
```

C.98 ./frontend/src/utils/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;
```

C.99 ./frontend/src/utils/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);
```

C.100 ./frontend/src/utils/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';
```

C.101 ./frontend/src/index.css

```

body {
  font-family: sans-serif;
  font-weight: 500;
  letter-spacing: 0.4px;
  font-size: 18px;
  margin: 0;
}
```

C.102 ./frontend/package.json

```
{
  "name": "easyapi",
  "version": "0.1.0",
  "private": true,
  "devDependencies": {
    "eslint": "^3.18.0",
    "eslint-config-airbnb": "^14.1.0",
    "eslint-plugin-import": "^2.2.0",
    "eslint-plugin-jsx-a11y": "^4.0.0",
    "eslint-plugin-react": "^6.10.3",
    "react-scripts": "0.8.5"
  },
  "dependencies": {
    "chai": "^3.5.0",
    "enzyme": "^2.8.1",
    "immutable": "^3.8.1",
    "normalizr": "^3.2.2",
    "prop-types": "^15.5.8",
    "radium": "^0.18.1",
    "react": "^15.4.2",
    "react-addons-test-utils": "^15.5.1",
    "react-dom": "^15.4.2",
    "react-dropzone": "^3.12.3",
    "react-redux": "^5.0.2",
    "react-router": "^3.0.2",
    "react-router-redux": "^4.0.8",
    "redux": "^3.6.0",
    "redux-immutable": "^3.1.0",
    "redux-thunk": "^2.2.0",
    "underscore": "^1.8.3"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  },
  "proxy": "http://localhost:9001"
}
```

C.103 ./backend/package.json

```
{
  "devDependencies": {
    "eslint": "^3.18.0",
    "eslint-config-airbnb": "^14.1.0",
    "eslint-plugin-import": "^2.2.0",
    "eslint-plugin-jsx-a11y": "^4.0.0",
    "eslint-plugin-react": "^6.10.3",
    "neutrino": "^4.2.0",
    "neutrino-preset-airbnb-base": "^4.2.0",
    "neutrino-preset-mocha": "^4.2.0",
    "neutrino-preset-node": "^4.0.1"
  },
  "dependencies": {
    "bcrypt": "^1.0.2",
    "body-parser": "^1.16.0",
    "chai": "^3.5.0",
    "compromise": "^7.0.28",
    "eslint-plugin-async-await": "^0.0.0",
    "express": "^4.14.0",
    "immutable": "^3.8.1",
    "isomorphic-fetch": "^2.2.1",
    "jsonwebtoken": "^7.3.0",
    "multer": "^1.3.0",
    "natural": "^0.4.0",
    "nlp_compromise": "^6.5.3",
    "passport": "^0.3.2",
    "passport-local": "^1.0.0",
    "pg": "^6.1.2",
    "radium": "^0.18.1",
    "react": "^15.4.2",
    "react-dom": "^15.4.2",
    "react-redux": "^5.0.2",
    "redux": "^3.6.0",
    "redux-immutable": "^3.0.10",
    "request-promise": "^4.1.1",
    "sbd": "^1.0.12",
    "sequelize": "^3.30.2",
    "sequelize-cli": "^2.5.1",
    "spacy-nlp": "^1.0.7",
    "xlsx": "^0.9.10",
    "underscore": "^1.8.3",
    "mocha": "^3.2.0"
  },
  "scripts": {
    "start": "neutrino start --presets neutrino-preset-node && node build/index.js",
    "build": "neutrino build --presets neutrino-preset-node",
```

```
    "test": "neutrino test --presets neutrino-preset-node neutrino-preset-mocha"
  },
  "config": {}
}
```

C.104 ./backend/src/components/natural.js

```
import { intersection } from 'underscore';
import request from 'request-promise';
import compromise from 'nlp_compromise';
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);

  // 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'];

    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) {
  let entity = '';

  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);
    }
  }

  if (!prop || !prop.name) {
    return 'string';
  }

  const propWords = prop.name.split(' ');

  // Check criteria for number.
  const integerKeywords = [
    'number',
    'integer',
    'digit',
    'digits',
    'numbers',
    'integers',
  ];

  const floatKeywords = [
    'float',
    'double',
    'floats',
    'doubles',
    'decimal',
    'decimals',
    'amount',
  ];

  if (intersection(propWords, integerKeywords).length > 0) {
    return 'integer';
```

```
  }

  if (intersection(propWords, floatKeywords).length > 0) {
    return 'float';
  }

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

```
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 = [];

  for (const sentenceResult of parseResult.data) {
    // 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);
    });

    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
      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) {
```

```
325         // This is the attributes
326         const fullObject = treeIndex[object.id];
327         attributes = [fullObject, ...getConjuctions(fullObject)];
328
329         attributes = attributes.map(assignNounPhrase);
330       }
331       let entities = [];
332       if (subject) {
333         // This is entities
334         const fullSubject = treeIndex[subject.id];
335         entities = [fullSubject, ...getConjuctions(fullSubject)];
336         allEntities = [...allEntities, ...entities];
337         allEntities = allEntities.map(assignNounPhrase);
338       }
339
340
341       inTree = treeIndex[relationship.id];
342
343       const attributesWithTypes = [];
344       for (const property of attributes) {
345         attributesWithTypes.push(categoriseProp(property, inTree, relationship, entities));
346       }
347
348       for (const entity of entities) {
349         const existingEntity = modelStructure.find(s => s.name === entity.lemma);
350
351         if (existingEntity) {
352           existingEntity.attributes = existingEntity.attributes.concat(attributesWithTypes);
353         } else {
354           modelStructure.push({
355             name: entity.lemma,
356             raw: entity.word,
357             attributes: attributesWithTypes,
358           });
359         }
360       }
361     }
362   }
363
364   postprocess(modelStructure, allEntities);
365   return modelStructure;
366 }
367
368
369 const Natural = {
370   _find: find,
371   _findAll: findAll,
```

```
  _findIfPropertyIsRequired: findIfPropertyIsRequired,
  _findIfPropertyHasMultiple: findIfPropertyHasMultiple,
  _filterTree: filterTree,
  seperateSentences,
  generateModelStructure,
  parse,
};

export default Natural;
```

C.105 ./backend/src/components/parse.js

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

export function parseSpreadsheet(file) {
  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);

    modelDefinition.name = name;

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

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

       // Use 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;

    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';
    }
  }

  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);
}
```

C.106 ./backend/src/components/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),
  })));

```

```
47    let models = await Model.findAll({
48      where: {
49        ServiceId: service.id,
50      },
51    });
52
53    const attributesToCreate = [];
54    const entriesToCreate = [];
55    const entryByIndexByModel = {};
56
57    let i = 0;
58    for (const modelDefinition of modelDefinitions) {
59      const model = models[i];
60      i++;
61      // Create attributes
62      for (const attributeDefinition of modelDefinition.attributes) {
63        attributesToCreate.push({
64          name: attributeDefinition.name,
65          type: attributeDefinition.type,
66          required: attributeDefinition.required,
67          multiple: attributeDefinition.multiple,
68          ModelId: model.id,
69        });
70      }
71
72      if (!modelDefinition.entries || modelDefinition.entries.length === 0) {
73        continue;
74      }
75
76      const entryByIndex = {};
77      // Create entries
78      let index = 1;
79      for (const entriesDefinition of modelDefinition.entries) {
80        entriesToCreate.push({
81          index,
82          ModelId: model.id,
83        });
84        entryByIndex[index] = entriesDefinition;
85        index++;
86      }
87      entryByIndexByModel[modelDefinition.name] = entryByIndex;
88    }
89
90    await Attribute.bulkCreate(attributesToCreate);
91    await Entry.bulkCreate(entriesToCreate);
92
93    models = await Model.findAll({
```

```
    where: {
      ServiceId: service.id,
    },
    include: [{ all: true, nested: true }],
  });

  const valuesToCreate = [];

  // Index: model > entry > attribute > value
  for (const model of models) {
    for (const entry of model.Entries) {
      for (const attribute of model.Attributes) {
        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;
}
```

C.107 ./backend/src/components/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;
  }
}
```

C.108 ./backend/src/config/bootstrap.js

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

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

C.109 ./backend/src/config/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;
```

C.110 ./backend/src/config/passport.js

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

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))) {
          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,
      };

      // Change secret in production
      const token = jwt.sign(payload, 'secret');

      return done(null, {
        user: {
          username: user.username,
        },
```

```
47          token,
48        });
49      })
50      .catch(err => done(err)),
51  ));
52
53  passport.serializeUser((user, done) => {
54    done(null, user.id);
55  });
56
57  passport.deserializeUser((id, done) => {
58    User.find({
59      where: { id },
60    }, (err, [user]) => {
61      done(err, user);
62    });
63  });
64
65  export default passport;
```

C.111 ./backend/src/nlp/index.py

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

@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()
```

C.112 ./backend/src/nlp/spacyparse.py

NOTE: This file is copied from https://github.com/kengz/spacy-nlp/blob/master/src/py/nlp.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]


# 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


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#>")]
```

C.113 ./backend/src/index.js

```
import Express from 'express';
import bodyParser from 'body-parser';
import passport from './config/passport';
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/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);

  app.use((req, res, next) => {
    const err = new Error('This page is not found!');
    err.status = 404;
    next(err);
  });

  app.use((err, req, res) => {
    /* 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 now running on port ${port}`);
})
.catch(err => console.error(err));
```

C.114 ./backend/src/middleware/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/login')) {
    return next();
  }
  if (req.originalUrl.startsWith('/api/api')) {
    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(() => res.status(401).end());
  });
}
```

C.115 ./backend/src/models/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;
}
```

C.116 ./backend/src/models/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;
}
```

C.117 ./backend/src/models/index.js

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

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 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);
}

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);
}

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

db.sequelize = sequelize;
```

```
db.Sequelize = Sequelize;

export default db;
```

C.118 ./backend/src/models/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;
}
```

C.119 ./backend/src/models/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;
}
```

C.120 ./backend/src/models/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) {
        return bcrypt.compare(password, this.passwordHash);
      },
    },
  });

  return User;
}
```

C.121 ./backend/src/models/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;
}
```

C.122 ./backend/src/routes/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,
      },
    });

    if (!user) {
      return res.status(404).send({ success: false, message: `This user (${username}) was not found!` });
    }

    const service = await Service.findOne({
      where: {
        shortName: serviceShortName,
        UserId: user.id,
      },
    });
    if (!service) {
      return res.status(404).send({ success: false, message: `This service (${serviceShortName}) was not found!` });
    }

    if (!service.isPublic) {
      return res.status(403).send({ success: false, message: 'This service is not public!' });
```

```
45      }
46
47      const model = await Model.findOne({
48        where: {
49          shortName: modelShortName,
50          ServiceId: service.id,
51        },
52      });
53
54      if (!model) {
55        return res.status(404).send({ success: false, message: `This model (${modelShortName})
            was not found!` });
56      }
57
58      const attributes = await Attribute.findAll({
59        where: {
60          ModelId: model.id,
61        },
62      });
63
64      data = { user, service, model };
65
66      const forbiddenResponse = { success: false, message: 'This action is not public!' };
67
68      switch (method) {
69        case 'GET': {
70          if (id) {
71            // Find One
72            if (!model.isFindOneEnabled) {
73              return res.status(403).send(forbiddenResponse);
74            }
75
76            const entry = await Entry.findOne({
77              where: {
78                index: id,
79                ModelId: model.id,
80              },
81            });
82
83            if (!entry) {
84              return res.status(404).send({ success: false, message: 'This resource doesn\'t
                  exist!' });
85            }
86
87            const values = await Value.findAll({
88              where: {
89                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(forbiddenResponse);
          }

          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(forbiddenResponse);
        }
        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(forbiddenResponse);
        }

```

```
184         const entry = await Entry.findOne({
185           where: {
186             index: id,
187             ModelId: model.id,
188           },
189         });
190
191
192         if (!entry) {
193           return res.status(404).send({ success: false, message: 'This resource doesn\'t exist
                !' });
194         }
195
196         const values = await Value.findAll({
197           where: {
198             EntryId: entry.id,
199           },
200         });
201
202         const valuePromises = [];
203         const valueByAttributeId = object(values.map(v => v.AttributeId), values.map(v => v));
204
205
206         const obj = {};
207         obj.id = entry.id;
208         for (const attribute of attributes) {
209           const newValue = input[attribute.name];
210           if (newValue) {
211             const oldValue = valueByAttributeId[attribute.id];
212             if (newValue !== oldValue.value) {
213               if (oldValue) {
214                 // Update
215                 valuePromises.push(
216                   Value.update(
217                     { value: newValue },
218                     { where: { id: oldValue.id } },
219                   ),
220                 );
221               } else {
222                 // Create
223                 valuePromises.push(
224                   Value.create({
225                     EntryId: entry.id,
226                     AttributeId: attribute.id,
227                     value: newValue,
228                   }),
229                 );
```

```
            }
          }
          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(forbiddenResponse);
      }

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


      if (!entry) {
        return res.status(404).send({ success: false, message: 'This resource doesn\'t exist
            !' });
      }

      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, message: 'This action is not supported by
            EasyAPI!' });
      }
    }
  } catch (e) {
    return res.status(500).send({ success: false, error: e });
  }

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


export default router;
```

C.123 ./backend/src/routes/attribute.js

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

const { 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 entries = await Entry.findAll({
      where: {
        ModelId: modelId,
      },
    });

    await Value.bulkCreate(entries.map(e => ({
      EntryId: e.id,
      AttributeId: attribute.id,
    })));

    const newEntries = await Entry.findAll({
      where: {
        ModelId: modelId,
      },
      include: [{ all: true }],
    });

    const response = {
      attribute,
      entries: newEntries,
```

```
      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;
```

C.124 ./backend/src/routes/auth.js

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

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,
  };
}

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) => {
    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);
});

router.post('/profile', (req, res) => {
  res.status(200).json(Object.assign({
    success: true,
    errors: {},
    username: req.user.username,
  }));
});


export default router;
```

C.125 ./backend/src/routes/entry.js

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

const { 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;
```

C.126 ./backend/src/routes/model.js

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

const { Model } = 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 modelId = req.param('id');

  const toUpdate = {};

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

  if (req.param('isFindEnabled')) {
    toUpdate.isFindEnabled = req.param('isFindEnabled');
```

```
  }

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

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

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

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

  try {
    const model = await Model.update(
      toUpdate,
      { 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;
```

C.127 ./backend/src/routes/service.js

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

const { Service } = 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))
  .catch(e => console.log(e));
});

router.post('/parseSpreadsheet', upload.single('spreadsheet'), (req, res) => 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) {
    return res.status(501).json({
      error: e,
      success: false,
    });
```

```
46    }
47  });
48
49  router.get('/', async (req, res) => {
50    try {
51      const services = await Service.findAll({
52        where: {
53          UserId: req.user.id,
54        },
55        include: [{ all: true, nested: true }],
56      });
57      return res.json({
58        services,
59        success: true,
60      });
61    } catch (e) {
62      return res.status(501).json({
63        error: e,
64        success: false,
65      });
66    }
67  });
68
69
70  router.get('/:id', async (req, res) => {
71    try {
72      const serviceId = req.param('id');
73      const service = await Service.findOne({
74        where: {
75          id: serviceId,
76          UserId: req.user.id,
77        },
78        include: [{ all: true, nested: true }],
79      });
80      return res.json({
81        service,
82        success: true,
83      });
84    } catch (e) {
85      return res.status(501).json({
86        error: e,
87        success: false,
88      });
89    }
90  });
91
92  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;
```

C.128 ./backend/src/routes/value.js

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

const { 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;
```

C.129 ./backend/test/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',
```

```
47            modifiers: [
48              {
49                lemma: 'yellow',
50                pos: 'ADJ',
51                modifiers: [],
52              },
53            ],
54          },
55        ],
56      };
57
58      const expected = {
59        lemma: 'yellow',
60        pos: 'ADJ',
61        modifiers: [],
62      };
63
64      const result = Natural._find(tree, o => o.lemma === 'yellow');
65      expect(result).to.deep.equal(expected);
66    });
67  });
68
69  describe('filterTree', () => {
70    it('should remove nodes which don\'t match a condition', () => {
71      const tree = {
72        pos: 'VBZ',
73        modifiers: [
74          {
75            pos: 'JJ',
76          },
77          {
78            pos: 'NN',
79          },
80        ],
81      };
82
83      const result = Natural._filterTree(tree, o => o.pos != 'JJ');
84
85      const expected = {
86        pos: 'VBZ',
87        modifiers: [
88          {
89            modifiers: undefined,
90            pos: 'NN',
91          },
92        ],
93      };
```

```
      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: [
```

```
235           { lemma: 'must', arc: 'aux' },
236         ],
237       };
238
239       const result = Natural._findIfPropertyIsRequired(prop, context);
240       expect(result).to.equal(true);
241     });
242
243     it('should deduce a property is not required when there are only optional keywords', () =>
            {
244       const prop = {
245         lemma: 'cat',
246         modifiers: [],
247       };
248
249       const context = {
250         lemma: 'play',
251         modifiers: [
252           { lemma: 'might', arc: 'aux' },
253         ],
254       };
255
256       const result = Natural._findIfPropertyIsRequired(prop, context);
257       expect(result).to.equal(false);
258     });
259
260     it('should deduce a property is required when there are more required keywords than
            optional keywords', () => {
261       const prop = {
262         lemma: 'cat',
263         modifiers: [],
264       };
265
266       const context = {
267         lemma: 'play',
268         modifiers: [
269           { lemma: 'might', arc: 'aux' },
270           { lemma: 'needs', arc: 'aux' },
271           { lemma: 'must', arc: 'aux' },
272         ],
273       };
274
275       const result = Natural._findIfPropertyIsRequired(prop, context);
276       expect(result).to.equal(true);
277     });
278
279     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 = {
```

```
326           lemma: 'cats',
327           POS_fine: 'NN',
328           modifiers: [
329             { arc, lemma: 'many' },
330           ],
331         };
332
333         const result = Natural._findIfPropertyHasMultiple(prop);
334         expect(result).to.equal(true);
335       });
336     });
337
338     it('should determine its singular if prop has modifiers with singular keywords', () => {
339       ['det', 'amod'].forEach((arc) => {
340         const prop = {
341           lemma: 'cats',
342           POS_fine: 'NN',
343           modifiers: [
344             { arc, lemma: 'single' },
345           ],
346         };
347
348         const result = Natural._findIfPropertyHasMultiple(prop);
349         expect(result).to.equal(false);
350       });
351     });
352
353     it('should determine its singular if prop has modifiers with singular keywords', () => {
354       ['det', 'amod'].forEach((arc) => {
355         const prop = {
356           lemma: 'cats',
357           POS_fine: 'NN',
358           modifiers: [
359             { arc, lemma: 'single' },
360           ],
361         };
362
363         const result = Natural._findIfPropertyHasMultiple(prop);
364         expect(result).to.equal(false);
365       });
366     });
367
368     it('should determine its singular if prop has singular number', () => {
369       ['zero', 'one'].forEach((lemma) => {
370         const prop = {
371           lemma: 'cats',
372           POS_fine: 'NN',
```

```
373           modifiers: [
374             { arc: 'nummod', lemma },
375           ],
376         };
377
378         const result = Natural._findIfPropertyHasMultiple(prop);
379         expect(result).to.equal(false);
380       });
381     });
382
383     it('should determine its singular if prop has singular number', () => {
384       ['twenty two', 'nine', 'fifty', 'ten thousand'].forEach((lemma) => {
385         const prop = {
386           lemma: 'cats',
387           POS_fine: 'NN',
388           modifiers: [
389             { arc: 'nummod', lemma },
390           ],
391         };
392
393         const result = Natural._findIfPropertyHasMultiple(prop);
394         expect(result).to.equal(true);
395       });
396     });
397   });
398
399   describe('generateModelStructure', () => {
400     it('should correctly analyse basic Pet model structure', async () => {
401       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.';
402
403       const modelStructure = await Natural.generateModelStructure(text);
404
405       const expected = [
406         {
407           name: 'pet',
408           raw: 'pet',
409           attributes: [
410             {
411               type: 'string',
412               name: 'name',
413               raw: 'name',
414               lemma: 'name',
415               required: false,
416               multiple: false,
417             },
418             {
```

```
419                 type: 'string',
420                 name: 'breed',
421                 raw: 'breed',
422                 lemma: 'breed',
423                 required: false,
424                 multiple: false,
425               },
426               {
427                 type: 'Owner',
428                 name: 'owner',
429                 raw: 'owner',
430                 lemma: 'owner',
431                 required: false,
432                 multiple: false,
433               },
434               {
435                 type: 'Toy',
436                 name: 'likes toy',
437                 raw: 'toy',
438                 lemma: 'toy',
439                 required: false,
440                 multiple: false,
441               },
442             ],
443           },
444           {
445             name: 'owner',
446             raw: 'Owner',
447             attributes: [
448               {
449                 type: 'string',
450                 name: 'name',
451                 raw: 'name',
452                 lemma: 'name',
453                 required: false,
454                 multiple: false,
455               },
456               {
457                 type: 'Pet',
458                 name: 'owns pet',
459                 raw: 'pet',
460                 lemma: 'pet',
461                 required: false,
462                 multiple: false,
463               },
464             ],
465           },
```

```
        {
          name: 'toy',
          raw: 'Toy',
          attributes: [
            {
              type: 'string',
              name: 'name',
              raw: 'name',
              lemma: 'name',
              required: false,
              multiple: false,
            },
          ],
        },
      ];

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

C.130 ./backend/test/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('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');
    });
  });

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

```
47          multiple: 'false',
48        },
49        {
50          type: 'integer',
51          multiple: 'false',
52        },
53      ]);
54
55      expect(result.type).to.equal('string');
56      expect(result.required).to.equal(true);
57    });
58
59    it('should return float if one of the types is float and there is no string', () => {
60      const result = determineType([
61        {
62          type: 'float',
63          multiple: 'false',
64        },
65        {
66          type: 'integer',
67          multiple: 'false',
68        },
69      ]);
70
71      expect(result.type).to.equal('float');
72      expect(result.required).to.equal(true);
73    });
74
75    it('should return integer if one of the types is integer and there is no string', () => {
76      const result = determineType([
77        {
78          type: 'integer',
79          multiple: 'false',
80        },
81        {
82          type: 'integer',
83          multiple: 'false',
84        },
85      ]);
86
87      expect(result.type).to.equal('integer');
88      expect(result.required).to.equal(true);
89    });
90
91    it('should not be required if one of the types is not required', () => {
92      const result = determineType([
93        {
```

```
            type: 'string',
            multiple: 'false',
          },
          null,
        ]);

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