VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Tuan Hoang Nguyen

INVESTIGATION, BUILDING, AND PILOTING OF A LOW-CODE SYSTEM FOR ENTERPRISE DEVELOPMENT

Technology and Communication
2020

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Tuan Hoang Nguyen
Title	Investigation, Building, and Piloting of a Low-code System for Enterprise Development
Year	2020
Language	English
Page	33
Name of Supervisor	Dr. Smail Menani

The thesis is made for Wärtsilä Oyj aiming to improve the internal business process. The purpose of this thesis is to create a low-code system for users to build webforms faster with minimum coding. The idea of this thesis and internal tools used in this project are provided by Wärtsilä Oyj.

React was used to build the frontend with the help of react-formio framework. Django and Django REST framework were used to create the backend and manage endpoints. The Waterfall methodology was utilized during the development process. Application design, programming implementation, testing, and debugging processes were included in the development logic.

The result of this thesis is a web service to allow users to use drag and drop components to create webforms. The final software product will be a subset of Wärtsilä's Enterprise Rules Engine and Decision Platform (EREP).

The project documented here is a proof of concept (POC), it does not reflect the final product implementation.

Keywords EREP, webform, form.io, and react-formio

CONTENTS

LIST OF FIGURES AND TABLES

LIST OF ABBREVIATIONS

RPA	Robotic Process Automation
EREP	Enterprise Rules Engine and Decision Platform
JSON	JavaScript Object Notation
APIs	Application Programming Interfaces
DOM	Document Object Model
NPM	Node Package Manager
ORM	Object Relation Mapper
CSS	Cascading Style Sheets
GCP	Google Cloud Platform
AWS	Amazon Web Services

1 INTRODUCTION

Robotic process automation is a form of business process automation technology based on metaphorical software robots or artificial intelligence workers. In traditional workflow automation tools, a software developer produces a list of actions to automate a task and interface to the back-end system using internal application programming interfaces (APIs) or dedicated scripting language. /2/

More and more systems are automated using robots, which means that the workload is getting bigger which is a concern for developers. As before, forms were designed by developers that have little or none experience in the business field. It takes quite a long time to understand the working method and business model to build an automated form. This thesis was proposed to investigate and solve this problem. The solution leads to the design and implementation of a low-code system for business users. This system allows users to create their own web forms faster with minimum coding. Based on a Data Driven Design method, this project was built around an existing database and APIs that is provided by Wärtsilä. The final software product will be a subset of Wärtsilä's Enterprise Rules Engine and Decision Platform (EREP). The result of this integrated solution will have a great impact on the workload for developers. Workload is estimated to be reduced by 70% for developers.

The thesis is composed of five chapters. The first chapter gives the reader a clear and comprehensive view about the project. All the information and knowledge on components used in the software are written in Chapter 2. The next chapter defines the software requirements. Enterprise Rules Engine and Decision Platform's architecture is also included in this chapter to show at what stage this thesis work is and what it can contribute to the whole system. Chapter 4 describes the implementation and demonstrates the validation of the solution. Conclusion, restriction, and future work are described in chapter 5.

1.1 Wärtsilä Oyj Abp

Wärtsilä is a global leader in smart technologies and complete lifecycle solutions for the marine and energy markets. By emphasising sustainable innovation, total efficiency, and data analytics, Wärtsilä maximises the environmental and economic performance of the vessels and power plants of its customers. In 2019, Wärtsilä's net sales totalled EUR 5.2 billion with approximately 19,000 employees. The company has operations in over 200 locations in more than 80 countries around the world. Wärtsilä is listed on Nasdaq Helsinki. /1/

1.2 Wärtsilä’s Forms and Search

Wärtsilä’s Form and Search are services that provide advanced automation tools and data visibility to end users and system developers. Form provides a platform for building forms used in creating, modifying, or deleting data.

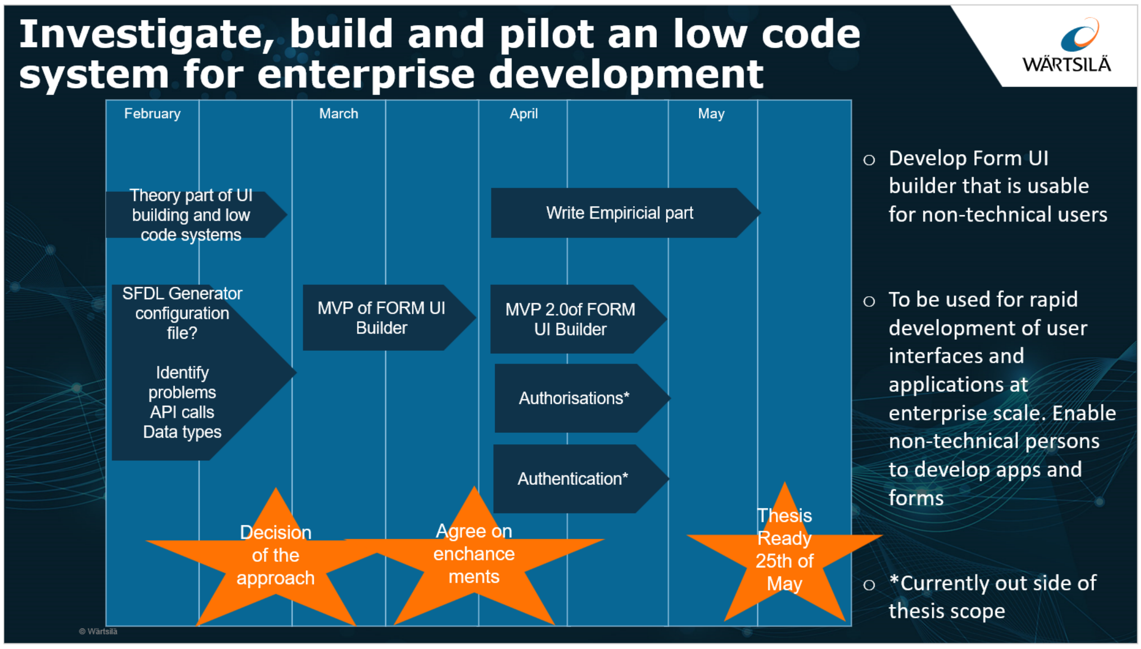
Search is an application that aims to provide generic search functionality into various internal and external datasets, and act as a jumping point to modify individual data objects.

1.3 Enterprise Rules Engine and Decision Platform

Enterprise Rules Engine and Decision Platform is a software used to define, deploy, execute, maintain, and monitor the business process decision logic that can be used by enterprise systems.

1.4 Thesis Plan and Milestones

The thesis plan and milestones are given in Figure 1. The first prototype of the form builder was finished in March 2020. After the first prototype, enhancements will be agreed with the company and the result will be ready by May 2020.



Investigate, build and pilot an low code system for enterprise development

- Develop Form UI builder that is usable for non-technical users
- To be used for rapid development of user interfaces and applications at enterprise scale. Enable non-technical persons to develop apps and forms
- *Currently out side of thesis scope

Figure 1. Thesis plan and milestones

2 THEORETICAL BACKGROUND

This chapter of the thesis explains the key concepts, models and software used in the project. It also answers the question why the approach and software are chosen to solve the problem.

2.1 Docker

Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating-system kernel and are thus more light-weight than virtual machines.

2.2 Front-end

2.2.1 React.js

React.js is a Javascript library, maintained by Facebook and community of individual developers or companies /4/.

The purpose of React is to show content (HTML) to users and handle user interaction. React can work by itself but it can also work with tremendous variety of other libraries, packages, servers, and data-bases. React is split into two separate libraries: ‘React’ knows what a component is and how to make components work together; ‘ReactDOM’ knows how to make a component and make it show up in the DOM.

2.2.2 Redux

Redux is an open source Javascript for managing application state. It is most commonly used with library sush as React for building user interfaces /5/.

With Redux, rather than authoring state or maintaining state inside the React component, it is instead extracted from the Redux library. Redux is handling data inside of the application.

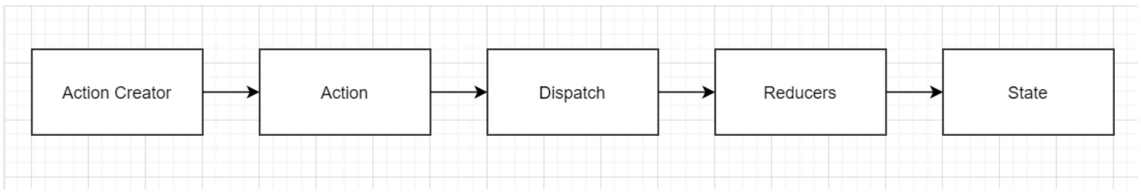


Figure 2. Redux Cycle

Action creator is a function that creates or returns a plain JavaScript object as an action.

An action has a type property and payload property. The type property describes some changes that we want to make inside data. The payload property describes some contexts around the changes that we want to make.

Dispatch function takes in an action, makes a copy of that object, and then passes it to different places inside the application.

Reducer is a function responsible for taking in an action and some existing amount of data. It processes that action and makes some changes to the data, then returns it so it can be centralized in other locations.

State property is a central property of all information that has been created by reducers. All the information is consolidated inside the state object. Because of that, react application can very easily reach into Redux application, the Redux side of the application, and get access to all the data of the application.

2.2.3 Document Object Model (DOM)

Document Object Model (DOM) is a cross-platform and language independent interface that treat XML or HTML document as a tree structure wherein each node is an object representing a part of the document. /6/

When a web page is loaded, the browser creates an object-oriented representation of an HTML document that acts as an interface between JavaScript and the document itself.

2.2.4 Form.io & React Formio

Form.io is a low code enterprise scale form and front-end development platform. /7/

React Formio is a UI framework library compatible with form.io for deploying forms' front-end. /8/

2.2.5 Bootstrap (Front-end Framework)

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS and Javascript based design templates for typography, forms, buttons, navigation and other interface components. /9/

2.2.6 Node.js

Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser. /10/

2.2.7 Node Package Manager (npm)

Node Package Manager is a package manager for the JavaScript programming language. It is the default package manager for the JavaScript runtime environment Node.js. /11/

To install a library using npm, the easiest way is to use the following command "npm install <library name>". Beside used to install libraries, npm is also used to run packages without downloading, restrict code to specific developers, manage multiple versions of code and code dependencies.

2.3 Back-end

2.3.1 Django (Web Framework)

Django is a popular Python framework used for building web applications. It comes with many features that help with building web applications rapidly. The main features that commonly used are Django Object Relation Mapper (ORM) and Django admin. The ORM provides an easy to use way for converting objects in APIs to rows in the database. The Django admin will provide an out of the box website that users can use for managing objects in the database.

Django provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models. /12/

2.3.2 Django REST Framework

Django REST framework is a powerful and flexible toolkit for building Web APIs. /13/

Django web applications group the code that handles each of these steps into separate files. Urls.py file will handle sending the request to the right view. Views.py file will handle the request and models.py file will define the data model.

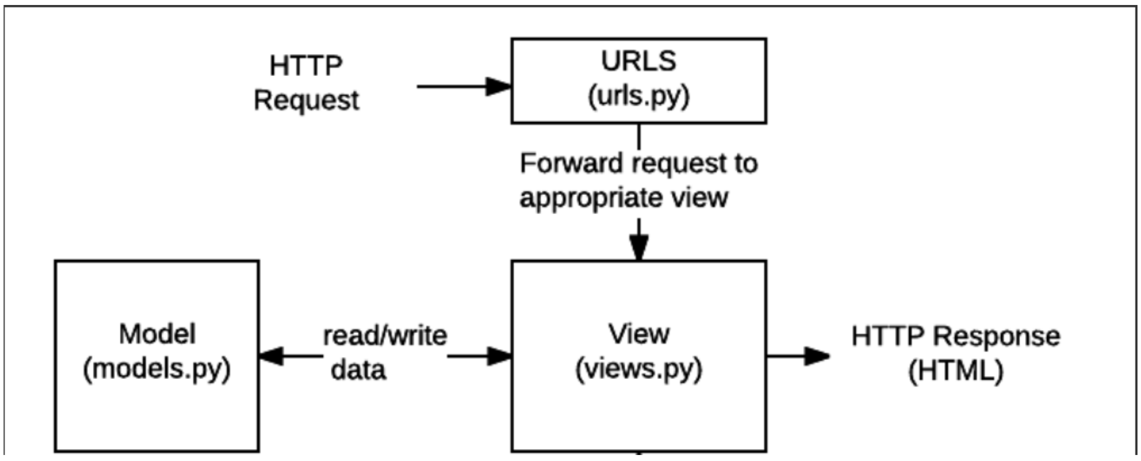


Figure 3. Steps to handle HTTP request in Django

2.3.3 PostgreSQL Database

PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. /14/

PostgreSQL includes features such as table inheritance and function overloading. Postgres is also known for protecting data integrity at the transaction level. This makes it less vulnerable to data corruption.

3 SOFTWARE SPECIFICATION AND ARCHITECTURE

3.1 Specific Requirement

The software requirements are description of features and functionalities of the target system. The requirements can be obvious or hidden, known or unknown, expected, or unexpected from client's point of view. /3/

3.1.1 Functional Requirements

Functional requirements are the basis for the system function. Firstly, the form builder should return data under the JSON format, which contains information about different fields and variables in the form and save that data to the database so that developers can use that to verify. Secondly, the form builder must render UI based on JSON data as well. The next requirement is that the software has ability to consume data from internal databases and APIs, which needed for form's functions such as autofill, and search. Lastly, the system must compatible with other systems in EREP. Login system, which authenticates user and authorization system are also needed but currently they are outside of the thesis scope.

Table 1 shortly demonstrates the list of functional requirements for the low-code system.

Table 1. Functional requirements for the system

No	Functional requirements
1	Able to render HTML web form with json config input
2	Can generate UI using JSON data
3	Consume data from databases and APIs to use in form's functions
4	Compatible with other systems in EREP

3.1.2 Nonfunctional Requirements

The nonfuntional requirements of the software are:

- Citizen Developer Driven development.
- Low code/Minimum coding required by citizen developers.
- Adapted to Wärtsilä's UI standards.
- API Driven.
- Easy to embed and use in other enterprise solution.
- Use Robust UI framework.

3.2 Software Architecture

As mentioned before, this thesis will be a part of Enterprise Rules Engine and Decision Platform.

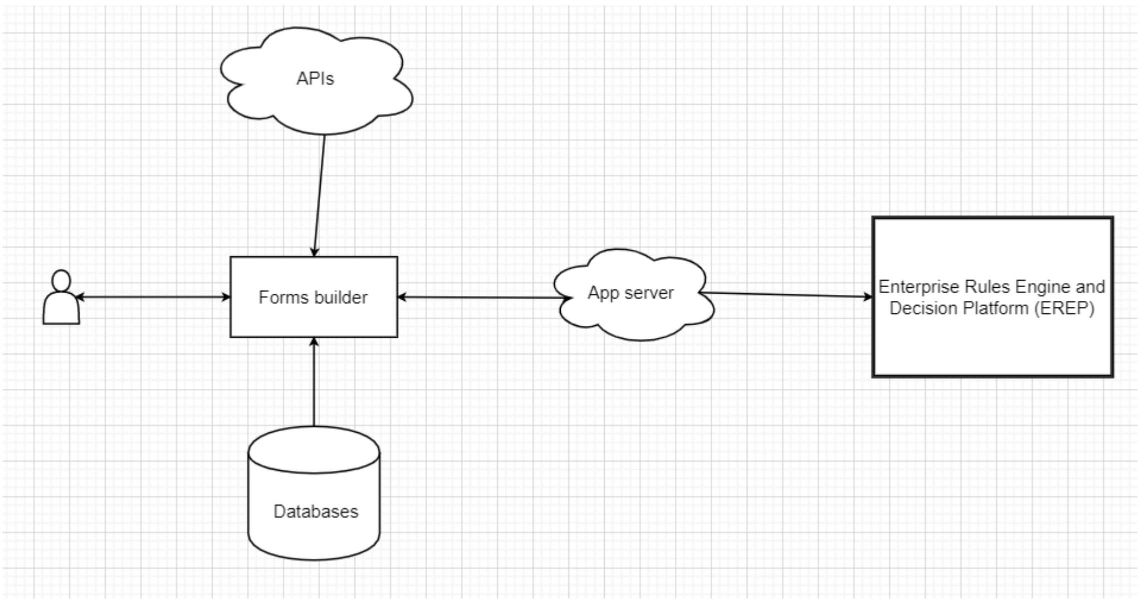


Figure 4. Form builder's architecture

Form builder will be an integral part of EREP. It enables citizen developers to build, modify, and deploy forms on their own. In the form builder's webpage, users can use different components that were created before to create their own forms with APIs. After citizen developers submit their forms to the system, forms will be saved to the App Server for real developers to verify and adjust if needed. Then forms will be deployed as a part of the business process executed on rules engine platform (EREP).

The following picture is the flow diagram to explain the system use flow.

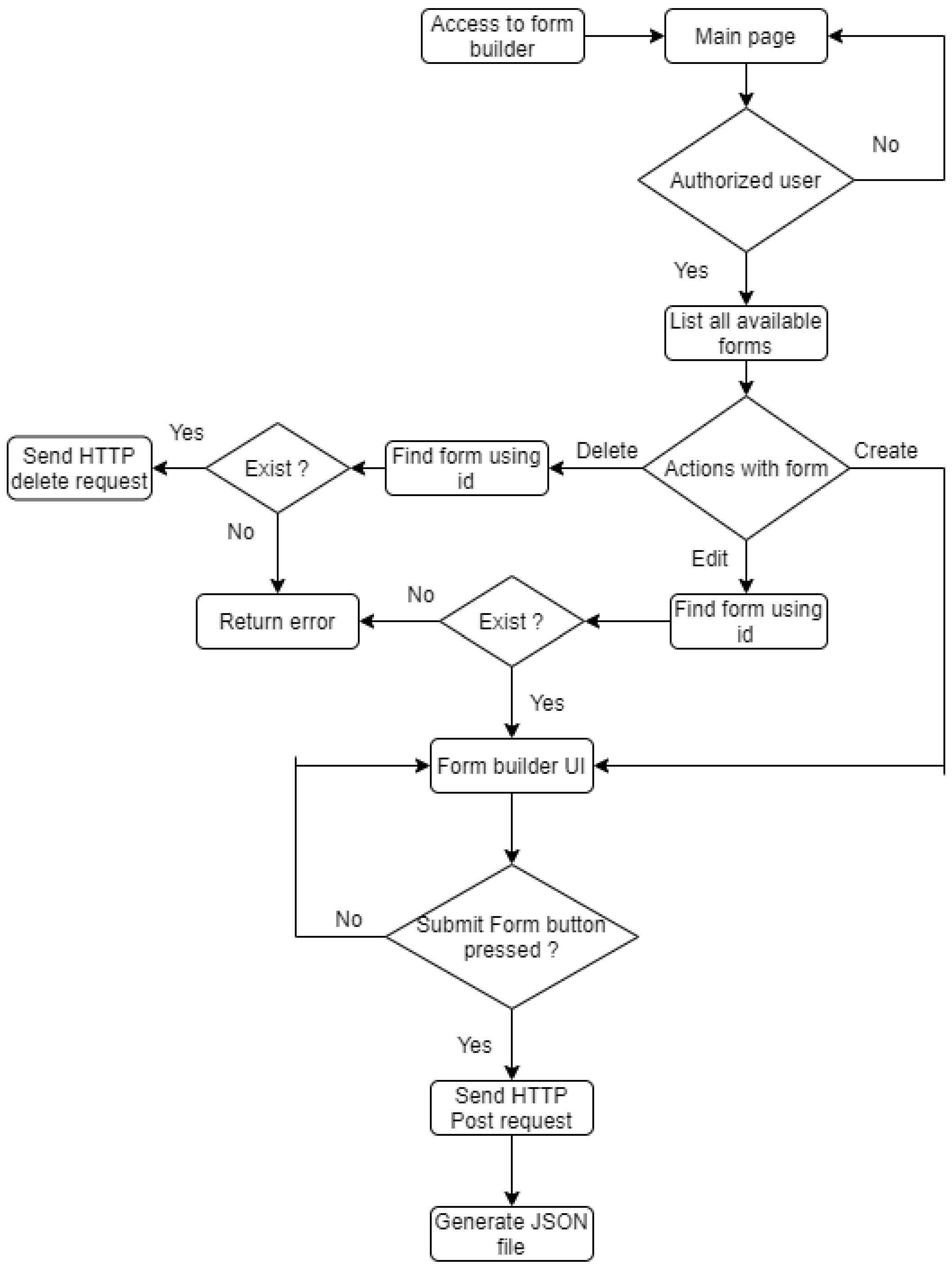


Figure 5. System use flow

When visiting the form builder webpage, if the user is authorized, all available forms are displayed. The user can use three main actions: delete, edit, and create new forms. When creating new a form, the user will be moved to the form builder's user interface. Citizen developers can build form by dragging and dropping components. After pushing the “Submit form” button, a Json file is generated and a HTTP Post request is sent to the server.

When in the edit mode, a form can be fetched from the server using id. If that id cannot be found in the database, an error is generated otherwise a form builder UI is generated. The same process as when creating a form is repeated.

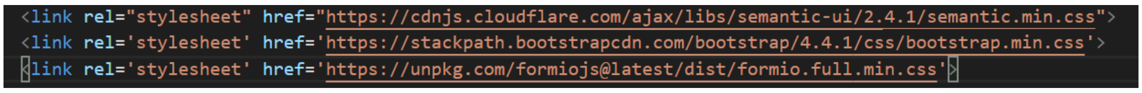
Deleting forms has almost the same process with editing forms. Forms can be retrieved using an id. If a form with a searched id does not exist, an error is generated. Otherwise, a HTTP delete request is sent to the server.

4 DETAIL IMPLEMENTATION

4.1 Front-end Implementation:

4.1.1 Styling and Resource

In this project, semantic-ui was used as a stylesheet to decorate, bootstrap, and formio’s stylesheet as requirements to be able to use formio. To be able to use those stylesheets, links to css sources were added to index.js file.



```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.4.1/semantic.min.css">
<link rel='stylesheet' href='https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css'>
<link rel='stylesheet' href='https://unpkg.com/formiojs@latest/dist/formio.full.min.css'>
```

Figure 6. Styling and its resource

4.1.2 Actions with Form

Form’s actions include create form, edit form, delete form, display form and list form.

a) Create form:

There are two ways to create a form using react-formio: using FormBuilder component and using FormEdit component. Because FormEdit component wraps the FormBuilder component and adds the title, display, name, and path fields at the top along with the save button so this component was chosen.

Table 2. FormEdit’s props

Name	**Type**	**Default**	**Description**
Form	object	{display:’form’}	Form definition object. It should at least have a display property set to form, wizard, or pdf
Options	object	{}	The options to be passed to FormBuilder

saveText	string	‘ ’	The string that will be displayed in the save button

As Table 2 shows, there are three props that are needed while using this component. Beside from these props, FormEdit has an event prop that triggers when the save button is pushed.

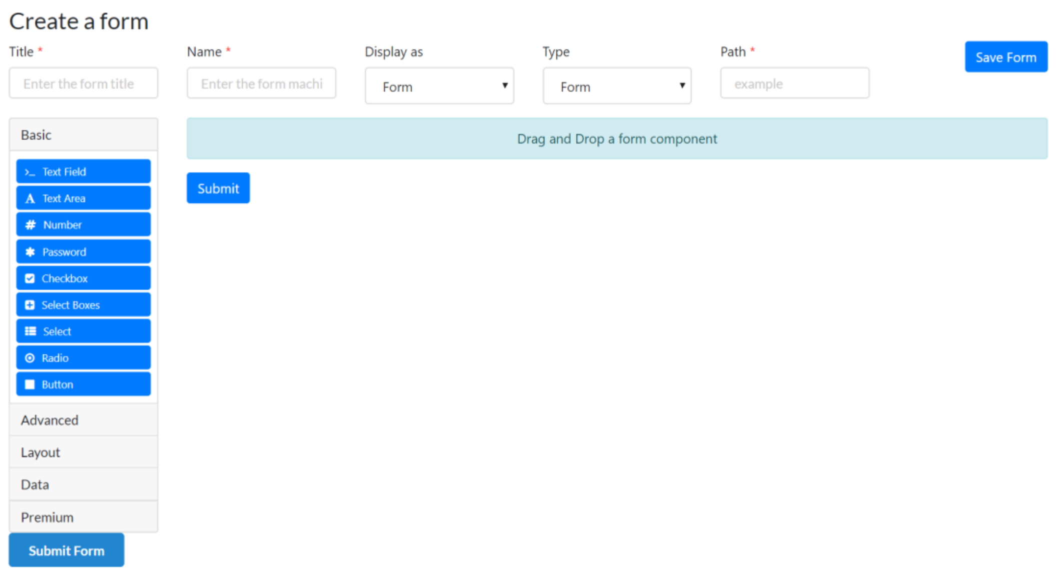
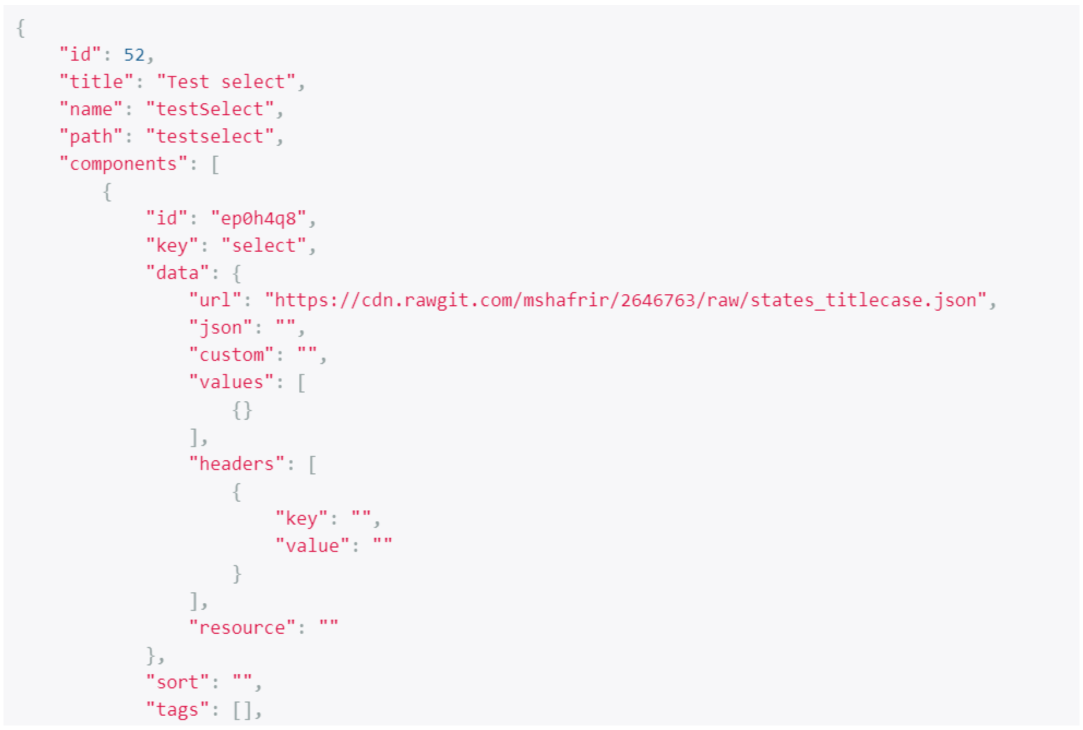


Figure 7. Example of a form builder generated by form.io

Figure 7 shows what a form builder looks like. On the left side are the components to create a form. Users can easily drag and drop those components and edit based on their choices.

When the “Save form” button is pushed, FormEdit component will return a json string containings all information about that form including styling, values, and position.



```
{
    "id": 52,
    "title": "Test select",
    "name": "testSelect",
    "path": "testselect",
    "components": [
        {
            "id": "ep0h4q8",
            "key": "select",
            "data": {
                "url": "https://cdn.rawgit.com/mshafrir/2646763/raw/states_titlecase.json",
                "json": "",
                "custom": "",
                "values": [
                    {}
                ],
                "headers": [
                    {
                        "key": "",
                        "value": ""
                    }
                ],
                "resource": ""
            },
            "sort": "",
            "tags": [],
```

Figure 8. Json string returned by form builder

The “Submit form” button is used to send the json string to the server and generate json file as in the thesis requirement. For this task, the following function was used:

```
const handleSaveFile = (jsonData, name) => {

   const fileData = JSON.stringify(jsonData);

   const blob = new Blob([fileData], {type: "application/json"});

   const url = URL.createObjectURL(blob);

   const link = document.createElement('a');

   link.download = `${name}.json`;

   link.href = url;

   link.click();

   URL.revokeObjectURL(url); }
```

This function will take json data and save it into an "application/json" type file then creates a download link for it. The file's name is set based on the title of the form.

b) Edit form:

The process to edit a form is almost like creating a form. First, the front-end will send a get request to the server to fetch the form's data based on its id. After finding and fetch the data, it will be saved into a prop and FormEdit component will use that prop to render the original form. Users can delete, edit, or create new elements of the form like when they create a new form.

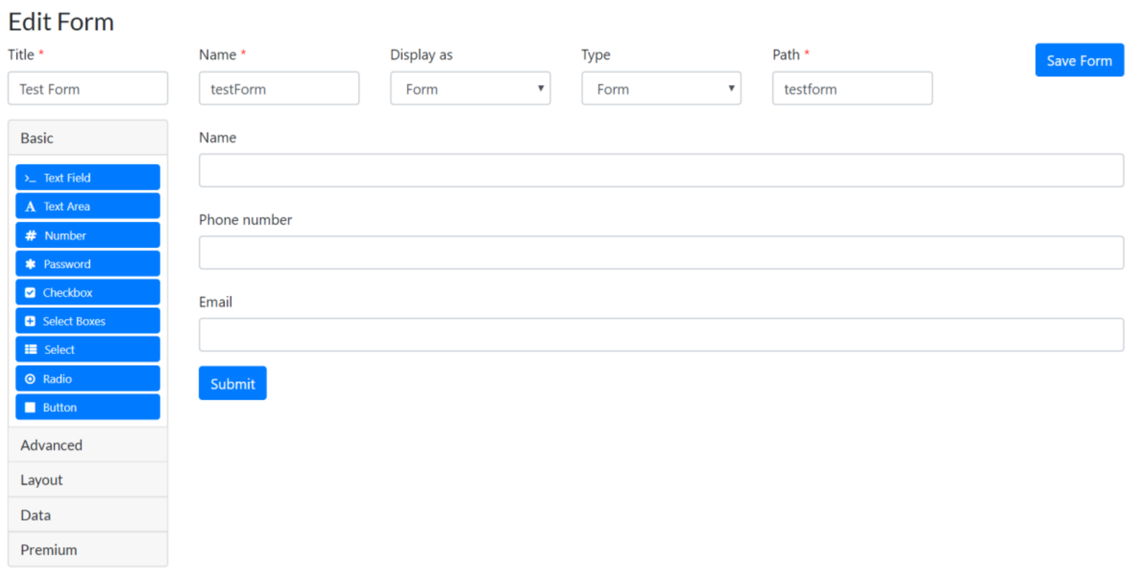


Figure 9. Edit form page

When triggered, the "Save Form" button will send a post request to the server to overwrite the existed form with the edited one.

c) Delete form:

The front end will send a delete request along with the form's id to the server to delete that form. For safety, there will be a prompt appear to ask the user again for the deleting decision.

d) Display form and list form:

Formio's Form component was used to display a form. Form can be generated using two types of resource: url of the form definition, the object that contains the form definition.

Table 3. Types of resource to load form definition

Name	Type	Description
src	url	The url of the form definition. When using src, the form will automatically submit the data to that url as well
form	object	Instead of loading a form from the url, user can preload form definition and pass it in with the form prop.

In this project, "form" resource was used to load form definition by sending a get request along with form's id to the server.

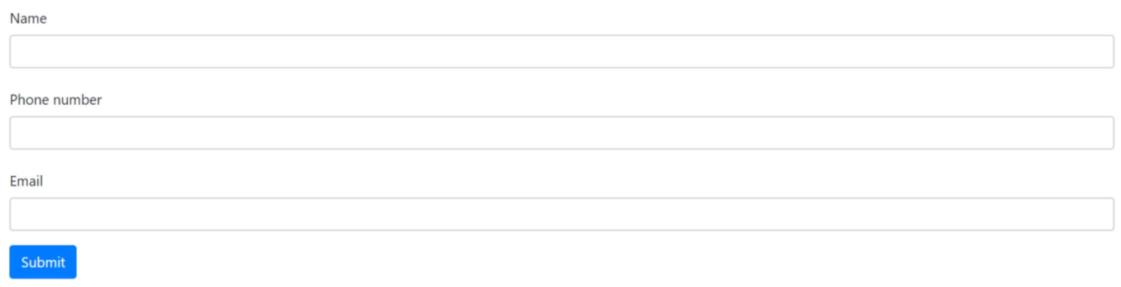


Figure 10. Display form page

To list forms, a get request is sent to the server and in return there is the list of all available forms in the server. Next to the title of the form there are two buttons used to edit and delete that form. Users can click the form's title to be able to display that form.

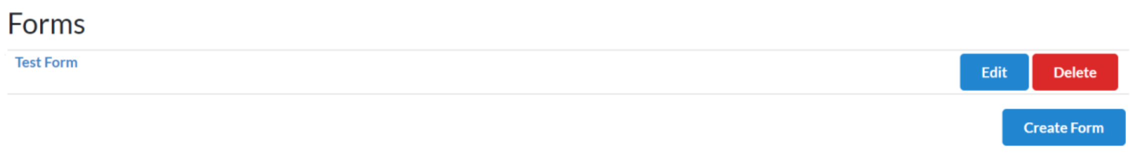


Figure 11. List forms page

4.1.3 Creating Path to Different Pages

React-router-dom has ready-made functions that help the developer to create path to components easily. The following function was used to map components to paths.

```
const App = () => {
    return (
        <div className="ui container">
            <Router history={history}>
                <div>
                    <Header/>
                    <Switch>
                        <Route path="/" exact component={FormList} />
                        <Route path="/login/" component={Login} />
                        <Route path="/signup/" component={Signup} />
                        <Route path="/forms/new/" exact component={FormCreate} />
                        <Route path="/forms/edit/:id/" exact component={FormEdit} />
                        <Route path="/forms/delete/:id/" exact component={FormDelete} />
                        <Route path="/forms/:id/" exact component={FormShow} />
                    </Switch>
                </div>
            </Router>
        </div>
    );
};
```

Switch element ensures that only one component is rendered at a time. Route tags link and render components and they need to be placed inside Switch component.

4.1.4 Communicate with Backend

As mentioned in the previous parts, to communicate with backend, front-end will send different HTTP requests needed for different actions:

```
export const createForm = (formValues) => {
    return async (dispatch, getState) => {
        const response = await forms.post('/formbuilder/forms/', {...formValues});
        dispatch({type: CREATE_FORM, payload: response.data});
        history.push('/');
    };
};
export const fetchForms = () => {
    return async (dispatch) => {
        const response = await forms.get('/formbuilder/forms/');
        dispatch({type: FETCH_FORMS, payload: response.data});
    };
};
export const fetchForm = (id) => {
    return async (dispatch) => {
        const response = await forms.get(`/formbuilder/forms/${id}/`);

        dispatch({type: FETCH_FORM, payload: response.data});

    };};
```

Action to edit the form is almost the same as creating a form action except for the form's id is added to fetch the specific form. A HTTP response is received after sending the request. Most usual response received are shown in Table 4.

Table 4. HTTP response

Code	**Detail description**
200	The request has succeeded
400	The server cannot understand the request due to invalid syntax
401	Client must authenticate itself to get the requested response
403	The client does not have access rights to the content
404	The server cannot find the requested resource

4.1.5 Formio Configuration

When installing react-formio, all the default settings and APIs will point to Form.io site so it is essential to change all of that to point to desired endpoints. Formio.baseUrl variable was set to https://api.form.io so that everything when submitted will be sent to that endpoint. To change this variable, users must reach Formio.js file.

When using the form builder, there is a select component. This component lets users to select variables from the database, API, or a user-defined variable. To be able to fetch data from external resources, we can define the endpoint with valid Authorization API token in url type resource. If the endpoint is fixed and users do not want to enter the link as well as the token, it can be changed in Select.edit.data.js and Select.js files.

4.2 Back-end Implementation

4.2.1 Docker Setup

In this thesis work, Docker was used to ease the setup work when deploying the entire system from local machine to a proper server such as Google Cloud Platform (GCP) or Amazon Web Services (AWS).

A Docker file was used to install all the required libraries and dependencies that needed to run the server:

```
FROM python:3.7-alpine

MAINTAINER Tuan Hoang Nguyen

ENV PYTHONUNBUFFERED 1

COPY ./requirements.txt /requirements.txt

RUN apk add --update --no-cache postgresql-client

RUN apk add --update --no-cache --virtual .tmp-build-deps gcc libc-dev linux-headers postgresql-dev

RUN pip install -r /requirements.txt

RUN pip install django-cors-headers

RUN apk del .tmp-build-deps

RUN mkdir /app

WORKDIR /app

COPY ./app /app
```

```
RUN adduser -D admin

USER admin
```

This server will run based on Python 3.7-alpine, a light-weight image that runs Python. This file will run command to install all the requirements listed requirements.txt file as well as the PostgreSQL client. Then the program will create a directory called app then change the current working directory to that directory. “docker build .” was used to build whatever is in the Docker file.

Another file required is Docker-compose file. It allows us to run a Docker image easily from the project location, manage the different services that make up the project. This .yaml file contains the configuration of all the services that make up the project.

For port configuration of the project, port 8000 on the host was mapped to port 8000 on the image. Volumes allows users to get the update that they made to the project into the docker image in real-time. Whenever users change something in the project, it will automatically update in the container.

```
version: "3"

services:

  app:

    build:

      context: .

    ports:

      - "8000:8000"

    volumes:

      - ./app:/app

    command: >
```

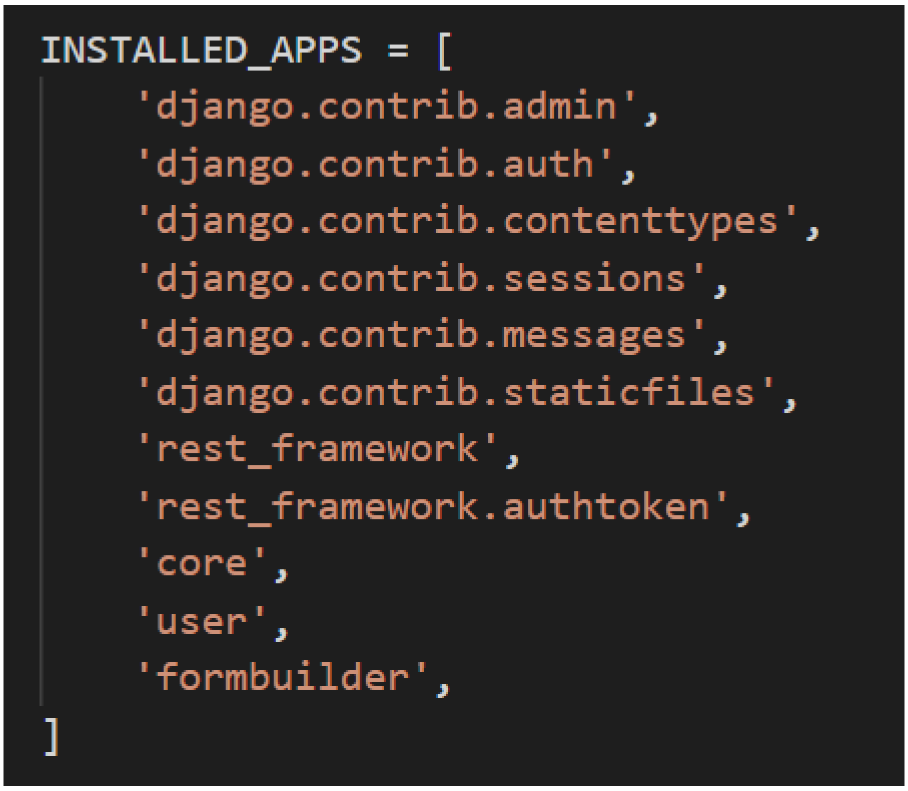
```
      sh -c "python manage.py wait_for_db &&
             python manage.py migrate &&
             python manage.py runserver 0.0.0.0:8000"
    environment:
      - DB_HOST=db
      - DB_NAME=app
      - DB_USER=postgres
      - DB_PASS=password
    depends_on:
      - db
  db:
    image: postgres:10-alpine
    environment:
      - POSTGRES_DB=app
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=password
```

Service with postgres:10-alpine was created called db. The environment variable contains the setting of the PostgreSQL database.

The command part will execute commands to wait for the database to set up, after that it will migrate all the tables that required for the application and then run the server on localhost port 8000. The “docker-compose build” command was used to build an image using the docker-compose configuration.

4.2.2 Creating Endpoints

There are three main applications in this project: core, users and formbuilder. We can easily create these applications using the command “docker-compose run app sh -c “python manage.py startapp app-name””. All the installed applications must be listed in the setting.py file.



```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework.authtoken',
    'core',
    'user',
    'formbuilder',
]
```

Figure 12. Installed apps

a) User endpoints:

The essential thing when creating user management endpoints is to create endpoints for users to create, update, and authenticate using token.

There are two objects for user model. The UserManager object was used to define the data model when creating a user and create superuser. The second object is User model that inherits from the previous object. This object will define elements that can be changed when editing user info such as email, and name.

When it comes to creating serializer classes, UserSerializer was created for the user object with create and update functions. AuthTokenSerializer was created for user authentication object. Whenever a new user is created, it will be assigned a unique token as an authentication credential. There are different endpoints for different actions. To create new users, users need to access to “/user/create” endpoint, “/user/token” endpoint to get token.

b) Form builder endpoint

As mentioned in the front-end implementation part, there should be views to create, edit, delete, and list forms. Unlike in building a user endpoint, form builder endpoint only has one object. This object contains a special type of field, json field, this is because of components field is a json string so it needs a special field to hold it. The serializer file of the form builder contains one class as serializer for form object. Front-end will send HTTP request to “/formbuilder/forms” to retrieve or create new forms, to “/formbuilder/forms/<:id>” to retrieve, edit or delete a specific form.

4.3 Version Controlling

To control the version of the software, Bitbucket was used. Bitbucket is a web-based version control repository hosting service that uses Git reversion control systems /15/.

For every new feature or update, Bitbucket was used to update the previous version. Commands listed in Table 5 were used to update new version or bug fix of the software:

Table 5. Git command and usage

Git task	Git commands
Create a new local reposi-tory	Git init
Check out a repository	Git clone /path/to/repository
Add files	Git add <filename>

	Git add *
Commit	Git commit -m “commit message”
Push	Git push
Status	Git status
Branch	Git checkout <branch name> Git branch
Update from the remote repository	Git pull Git merge Git diff
Undo local changes	Git checkout -- <file name> Git fetch origin Git reset –hard origin/master

To be able to push a directory to the repository, that directory must be a local repository first. Git command “git init” was used to create a new local repository on the local machine. To add files to an existing repository, “git add <filename>” was used to add a specific file or “git add *” was used to add all the files in the directory.

To make each commit easy to understand, “git commit -m” command must be followed by a commit message to commit changes to head. This message will shortly describe what is new or what has been changed in this commit.

After committing the changes, “git push” command was used to send changes to the master or current branch of the remote repository. In some cases, it does not know what file has been changed or committed, “git status” was used to check that.

To avoid conflict between different commits, branches should be created. By doing this, changes in the branch will not affect the master. To create a new branch and switch to it, “git checkout -b <branch name>” command is used. Usually, each bug fix or enhancement will have its own branch. “git branch” command will help to list all branches in the repository. It also indicates which branch is currently used.

“Git pull” was used to fetch and merge changes on the remote server to the local machine. After making sure all the code on the working branch, “git merge” was used to merge the current branch into the master branch. Conflict when merging branches together is not avoidable, “git diff” can be used to view all the conflicts against the base file.

In the worst scenario, if something fails, “git checkout -- <filename>” can replace the changes in the working tree with the last content in the head. There is also another way to fix the problem. Instead of dropping all local changes, we can fetch the latest history from the server and point the local master branch to it. We can do this by using “git fetch origin” and “git reset –hard origin/master” commands.

5 CONCLUSIONS AND DISCUSSION

The implementation of this thesis can be considered as a successful project. The design phase and implementation phase of the basic system were finished in a short period of time. Business productivity was increased due to an increase in production speed and lower error rate. Workload was also reduced by nearly 70% for developers.

The form builder comes with both advantages and disadvantages. The first advantage of this software is the user-friendly interface. Any user with an understanding of the business process can build a form using ready-made components. The second benefit of the form builder is form can be generated using JSON config. A form generated by this method will have the same function as a form generated by the drag and drop method. On the other hand, there are some drawbacks of the software. Firstly, this web application might be slowed down by adding more features on the client side. Secondly, the JSON file is generated on the user machine after submitting the form. There should be a function to send that file to the database automatically along with JSON data.

The thesis demonstrated here is only a proof of concept, a prototype so the implementation needs to be refactored before being put into the EREP production. For example, authentication and authorization should be added to restrict users from using actions on different forms. A new function to send JSON file generated by the software automatically to the database also needs to be added in the future. Finally, there are a limited number of components that can be used to build form. Creating new components can also be considered to put in future enhancements.

REFERENCES

/1/ This is Wartsila. Accessed 25 February 2020. https://www.wartsila.com/about

/2/ Robotic process automation. Accessed 25 February 2020. https://en.wikipedia.org/wiki/Ro-botic_process_automation

/3/ Software requirements. Accessed 25 February 2020. Available:https://www.tutori-alspoint.com/software_engineering/software_requirements.htm

/4/ React (web framework). Acceessed 13 March 2020.
https://en.wikipedia.org/wiki/React_(web_framework)

/5/ Redux (javascript library). Accessed 13 March 2020.
https://en.wikipedia.org/wiki/Redux_(JavaScript_library)

/6/ Document Object Model (DOM). Accessed 13 March 2020.
https://en.wikipedia.org/wiki/Document_Object_Model

/7/ Form.io. Accessed 13 March 2020. https://www.form.io/

/8/ React Formio. Access 13 March 2020. https://github.com/formio/react-formio

/9/ Bootstrap (front-end framework). Accessed 14 March 2020.
https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)

/10/ Node.js. Access 14 March 2020. https://en.wikipedia.org/wiki/Node.js

/11/ Node Package Manager (npm). Access 14 March 2020.
https://en.wikipedia.org/wiki/Npm_(software)

/12/ Django (web framework). Accessed 14 March 2020.
https://en.wikipedia.org/wiki/Django_(web_framework)

/13/ Django REST framework. Accessed 14 March 2020. https://www.django-rest-framework.org/

/14/ PostgreSQL. Accessed 1 April 2020. https://www.postgresql.org/

/15/ Bitbucket. Accessed 10 April 2020. https://en.wikipedia.org/wiki/Bitbucket