eMDs: E-Forms in Angular

Team Members:
"Tommy" Jibril Abdul
"Jason" Hyeoncheol Kim
Adam Nguyen
Tien Quang
Yvon Randolph Tsaju
Andres Daniel Uriegas

Abstract:
By converting eMDs' assessment forms into Angular, eMDs will be able to phase out the use of outdated third party software to generate assessment forms and, instead, use Angular to dynamically generate assessment forms that can be opened in a web browser and will have better performance. A database and web service were created to support the assessment forms.

Table of Contents

Executive Summary	**2**
Introduction	**3**
Discussion	**3**
Resources	**3**
Key Roles	**3**
Communication Plan	**4**
Risk Analysis	**4**
Costs	**4**
Timetable - Deliverables	**4**
Evaluations	**5**
Conclusion	**5**
Contact information	**5**
Implementation Details	**5**
Sources	**5**
Appendix	**6**
Impact & Security	**6**
Individual Assessment	**7**
Issues & Lessons Learnt	**7**
Future Work	**7**
Ethics Discussion	**7**
Print Name/ Signatures/ Date	**7**

Executive Summary

Generating medical assessment forms is a core aspect of eMDs business. eMDs current method of generating medical assessment forms has become outdated. This method has become outdated due to general business adoption, eMDs client's adoption, and widespread integration of modern web technologies. In order to assist eMDs in keeping on track with its clients' needs, it is a business benefit to transition eMDs current method of generating medical assessment forms to web-based forms. To transition eMDs method of generating medical assessment forms to a web-based model, our team proposes we build a web application with a Node js backend and an Angular 7 frontend. It is recommended that in order to meet the business needs of eMDs and transition eMDs current method of form generation, building a web application with a Node js backend and Angular 7 frontend should be implemented.

- **Introduction**
  - "Assessment Form Conversion to Web Form"
  - Project entails updating an existing EMR assessment forms solution from a built-in Windows popup to a form easily accessible as a web form
  - Functionality Overview
    - Database that holds data forms in differing formats (XML, HTML, JSON, etc.)
    - Endpoint that goes from processing XML form from database to creating and outputting HTML form in web-based environment
    - Front-End client that sends request to endpoint for HTML viewing

- **Discussion**
  - There aren't any open source solutions or libraries that do this conversion. Our solution is to parse the currently existing XML files into a JSON file. Then use that to dynamically generate a HTML file which is the web form. Doing so, we are modernizing the current eMDs' method of patient data collection.

- **Resources**
  - Node JS
    - EJS
    - Xml2js
    - Express.js
  - MySQL
  - Angular 7
  - Visual Studio Code
  - Notepad ++
  - GitHub
  - Google Chrome

- **Key Roles**
  - Point-of-Contact: Andres Daniel Uriegas, Adam Nguyen
  - Meeting Organizer: Andres Daniel Uriegas
  - Note taker: "Tommy" Jibril Abdul
  - Front end: "Tommy" Jibril Abdul, "Jason" Hyeoncheol Kim, Tien Quang
  - Back end: Yvon Randolph Tsaju, Adam Nguyen, Andres Daniel Uriegas

- **Communication Plan**
    - Tools: Slack, GroupMe, Trello, Google Calendar, email, Discord
    - Meeting Schedule
        - Tuesdays w/ team @7PM
        - Before March 6: Thursdays w/ sponsor @9AM
        - March 6 and onward: Fridays w/ sponsor @4PM

- **Risk Analysis**
    - In the event of a sick team member, healthy members will continue work, and sick member may rest until ready to work again
    - To refrain from data loss, all current project data and files will be uploaded to Github and/or Slack for redundancy

- **Costs**
    - Our project does not foresee any monetary requirements for our implementation

- **Timetable - Deliverables**
    - Phase I: due 2/6:
        - Open and Parse the XML file with Node.js
    - Phase II: due 2/13:
        - Create MySQL Data table
        - Create HTML dynamically
        - Convert XML to JSON
    - Phase III: due 2/20:
        - Populate Data Table with existing data
    - Phase IV: due 2/27:
        - Client to retrieve data from database
        - Build Generic class to Parse forms
        - Build classes for each unique controls
    - Phase V: due 3/6:
        - Modified classes for generic purposes
        - Updated web service to have variable URL form input
    - Phase VI: due 3/13:
        - Modularized some code
        - Added Controls module
    - Phase VII: due 4/3:
        - Completed project with assessment forms displayed in browser
        - Modularized all classes and functions
        - Added README for running test example w/o database

        - Used EJS to parse JSON to HTML
    - Phase VIII: due 4/21:
        - Final Report finished
        - Poster Complete
        - Single Slide
    - Phase IX: due 4/24:
        - Sponsor Approval form signed
    - Phase X: due 5/1:
        - Single Slide Video presentation
    - Phase XI: due 5/7:
        - Individual Contribution videos

- **Evaluations**
    - Weekly assessments from the project sponsor will be used to gauge progress.
    - The program will be tested by entering test data to ensure that the system functions properly.

- **Conclusion**
    - Being able to produce electronic medical forms is extremely useful in our current society. By bringing these medical forms to an even more accessible realm - web based HTML - eMDs can find compatibility across all electronic devices. We did not have enough time to achieve converting the forms in angular, but finished with an MVP that still consisted of a fully functional product from XML to HTML.

- **Contact information**

    - "Tommy" Jibril Abdul: jaa161230@utdallas.edu
    - "Jason" Hyeoncheol Kim: hxk173930@utdallas.edu
    - Adam Nguyen: apn170330@utdallas.edu
    - Tien Quang: txq170130@utdallas.edu
    - Yvon Randolph Tsaju: yst170030@utdallas.edu
    - Andres Daniel Uriegas adu170030@utdallas.edu

- **Implementation Details**

1. Parsing XML
   To follow Top-Down methodology and to work with javascript objects, parsing XML provided by the sponsor was our first mission. The problem was solved with an open source library called xml2js from node package manager(NPM).

JS XMLParser.js

emds-angular > controls > JS XMLParser.js > <unknown> > module

```
const parseString = require('xml2js').parseString;

const config = {
  explicitArray: false,
  charkey: "innerXML",
  explicitCharkey: true,
  mergeAttrs: true
}

module.exports = (xml, callback) => {
  parseString(xml, config, (err, result) => {
    if (err) {
      console.log(err);
    } else {
      if (typeof callback == "function") {
        callback(result);
      }
    }
  });
}
```

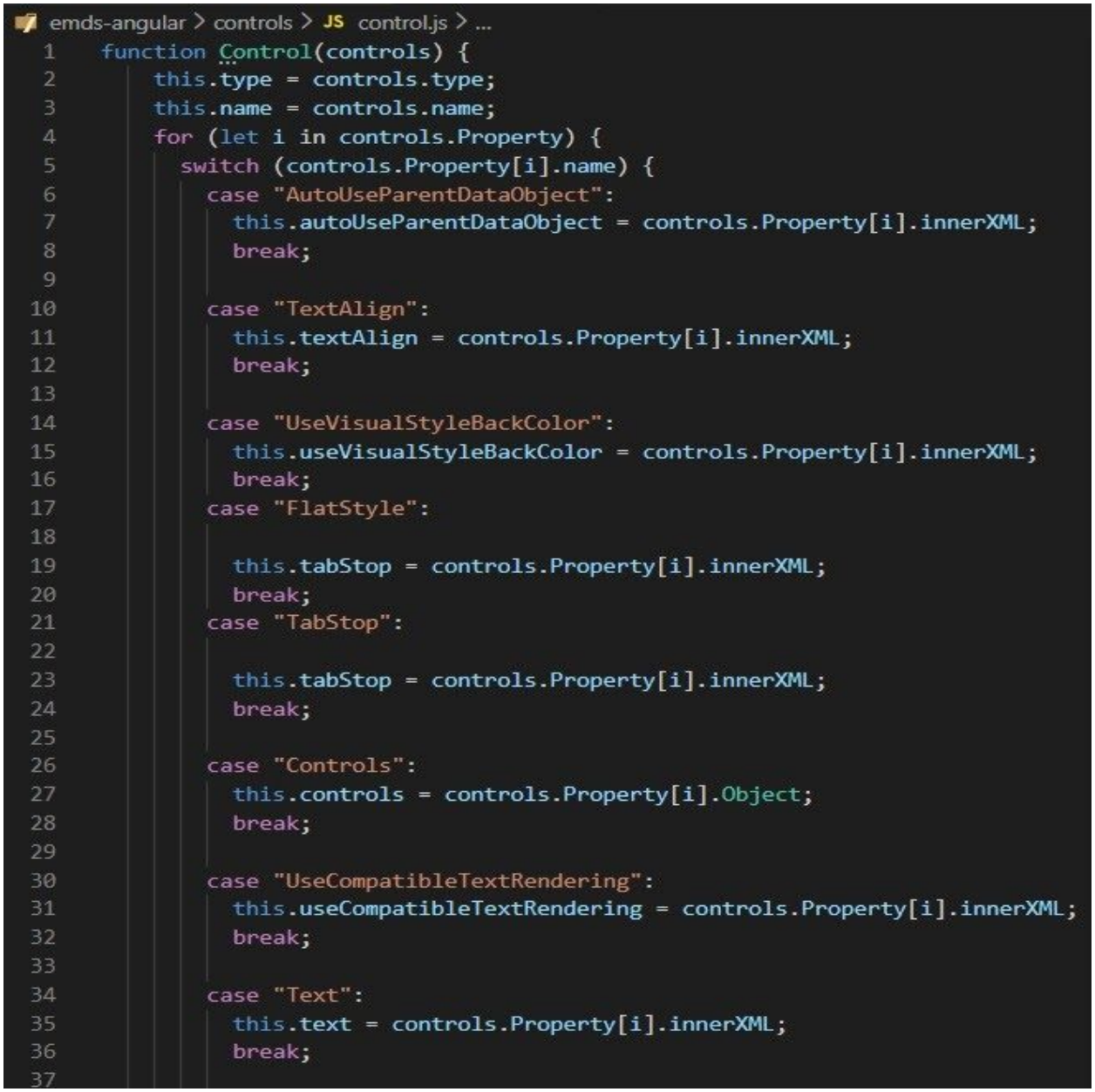
API ‘parseString’ requires two arguments which are XML and configurable options to change its default values.

- explicitArray (default:true) : Always put child nodes in an array if true, otherwise an array is created only if there is more than one.
- Charkey (default: _): Prefix that is used to access the character content in XML.
- explicitCharkey (default: false): To enable explicit charkey.
- mergeAttrs (default: false): Merge attributes and child elements as properties of the parent, instead of keying attributes off a child attribute object.

```
<Viklele.FormDesigner Version="1.1">
  <Object type="Viklele.Win.Designer.FormDesigner, Vikle
Version=2.1.5233.21178, Culture=neutral, PublicKeyToken=
name="formDesigner" ObservationTypeUid="00000000-0000-00
    <Property name="ErrorCaption">Form Designer</Propert
    <Property name="AutoAlignAid">SnapToGrid</Property>
    <Property name="DesignerContextMenu">True</Property>
    <Property name="AutoScroll">True</Property>
    <Property name="AutoScrollMargin">0, 0</Property>
    <Property name="AutoScrollMinSize">0, 0</Property>
    <Property name="AccessibleDescription" />
    <Property name="AccessibleName" />
    <Property name="AccessibleRole">Default</Property>
    <Property name="AllowDrop">False</Property>
    <Property name="AutoScrollOffset">0, 0</Property>
    <Property name="BackColor">Control</Property>
    <Property name="BackgroundImageLayout">Tile</Propert
    <Property name="CausesValidation">True</Property>
    <Property name="ContextMenu" />
    <Property name="ContextMenuStrip" />
    <Property name="Controls">
      <Object type="iMedica.Prm.Client.UI.BaseControls.P
iMedica.Prm.Client.UI.BaseControls, Version=14.0.1404.26
PublicKeyToken=null" name="PrmGroupBox6">
        <Property name="TabStop">False</Property>
```

```
{
  "Viklele.FormDesigner": {
    "Version": "1.1",
    "Object": {
      "type": "Viklele.Win.Designer.FormDesigner, Viklele
      "name": "formDesigner",
      "ObservationTypeUid": "00000000-0000-0000-0000-000
      "Property": [
        {
          "innerXML": "Form Designer",
          "name": "ErrorCaption"
        },
        {
          "innerXML": "SnapToGrid",
          "name": "AutoAlignAid"
        },
        {
          "innerXML": "True",
          "name": "DesignerContextMenu"
```

2. Creating generic class
   2.1. Control class
   - Major components of this form reside inside of the property node called ‘Controls’ containing some common properties of DOM and css styles. To deserialize and consume converted JSON after parsing, start with creating a javascript class that represents what is in the JSON and whose instance members map to the keys in the JSON.



- The switch statement in this constructor was used to feed different properties to each control by looping through the array object passed by the argument at runtime.

- Controls are divided into two ways: Outer control and inner control. Outer controls are mainly used for specifying boundaries between components in the whole layout. Inner controls have properties of actual functionality within the component.

The red rectangle is outer control and the black circle is inner control.

2.2. Sorting controls

- Sort.js is a module containing helper methods to correct the order of controls since the order in the XML file is not consistent with the original form.

```
const Control = require('./control');

//Object containing sorted outercontrols.
const sortedControls = {};

//Sorting by TabIndex.
module.exports = function sortControls(property) {
  let controls = findControls(property) // locate control property in the layout and returns it.
  const outerControls = [];
  for (let controlNumber = 0; controlNumber <= maxTabIndex(controls); controlNumber++) {
    for (let i in controls) {
      if (findTabIndex(controls[i].Property) == controlNumber) { //To match with tabIndex from 0.

        let outerControl = new Control(controls[i]) // create outercontrol

        if (outerControl.hasOwnProperty('controls')) { //if outercontrol has inner control

          //sort and add inner controls inside of each outercontrol .
          outerControl["inner"] = getInnerControl(outerControl.controls);
        }
        //push sorted outercontrol into array.
        outerControls.push(outerControl);
      }
    }
  }
  sortedControls.outerControls = outerControls;
  return sortedControls;
}
```

- 'findControls' finds 'Controls' in the form and returns its unsorted control objects.

```
//find control property in the form
function findControls(property) {
  let control = property.find((x) => x.name == 'Controls');
  return control.Object;
}
```

- In each control, they have a property called tabIndex which is the number indicating the correct order of each control. We used this number to sort out controls.

```
$ node app.js
server is running on port 4000
{ innerXML: '12', name: 'TabIndex' }
{ innerXML: '11', name: 'TabIndex' }
{ innerXML: '1', name: 'TabIndex' }
{ innerXML: '9', name: 'TabIndex' }
{ innerXML: '8', name: 'TabIndex' }
{ innerXML: '7', name: 'TabIndex' }
{ innerXML: '6', name: 'TabIndex' }
{ innerXML: '5', name: 'TabIndex' }
{ innerXML: '4', name: 'TabIndex' }
{ innerXML: '3', name: 'TabIndex' }
{ innerXML: '2', name: 'TabIndex' }
{ innerXML: '12', name: 'TabIndex' }
{ innerXML: '11', name: 'TabIndex' }
{ innerXML: '1', name: 'TabIndex' }
{ innerXML: '9', name: 'TabIndex' }
{ innerXML: '8', name: 'TabIndex' }
```

- After finding the index, we calculate the max number for the range of loops while we sort. We find the tabindex and push it into an array. Javascript built-in object Math.max.apply returns the maximum number in an array.

```
//Find TabIndex of controls.
function findTabIndex(property) {
  let tabIndex = property.find((x) => x.name == 'TabIndex');
  console.log(tabIndex);
  return tabIndex.innerXML;
}
//To find out how many controls are in the form
function maxTabIndex(controls) {
  let arr = [];
  for (let i in controls) {
    arr.push(findTabIndex(controls[i].Property));
  }
  return Math.max.apply(null, arr);
}
```

- Lastly, if outer control has inner control, ‘getInnerControl’ will find and sort its inner controls.

```
//Find inner controls
function getInnerControl(controls) {
  let innerControls = [];
  for (let controlNumber = 0; controlNumber < Object.keys(controls).length; controlNumber++) {
    for (let i in controls) {
      if (findTabIndex(controls[i].Property) == controlNumber) {
        let innerControl = new Control(controls[i]);
        innerControls.push(innerControl);
      }
    }
  }
  return innerControls;
}
```

- As a result, you can see the sorted controls.

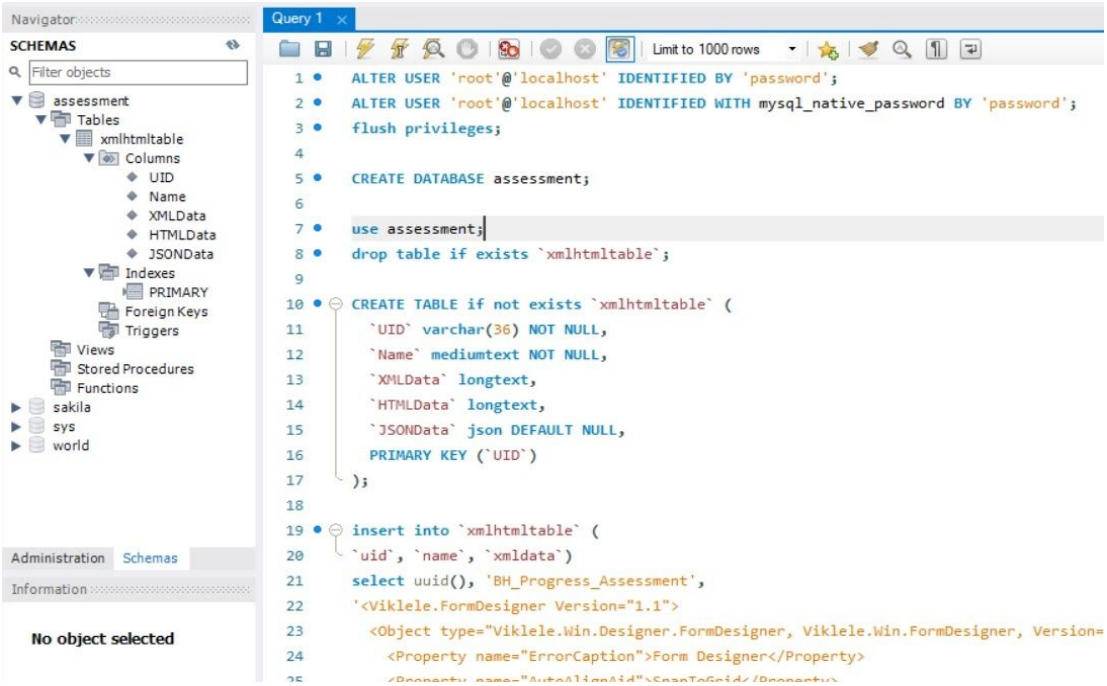
```
$ node app.js
server is running on port 4000
{
  outerControls: [
    Control {
      type: 'iMedica.Prm.Client.UI.BaseControls.PrmGroupBox, iMedica.Prm.Client.UI.Base
Controls, Version=14.0.1404.2612, Culture=neutral, PublicKeyToken=null',
      name: 'PrmGroupBox1',
      tabStop: 'False',
      text: 'Direct Service',
      useCompatibleTextRendering: 'True',
      controls: [Array],
      dataBindings: [Object],
      font: 'Microsoft Sans Serif, 8.25pt',
      location: '16, 8',
      xCoordinate: '16',
      yCoordinate: '8',
      size: '456, 80',
      tabIndex: '1',
      useWaitCursor: 'True',
      inner: [Array]
    },
    Control {
      type: 'iMedica.Prm.Client.UI.BaseControls.PrmGroupBox, iMedica.Prm.Client.UI.Base
Controls, Version=14.0.1404.2612, Culture=neutral, PublicKeyToken=null',
      name: 'PrmGroupBox2',
      tabStop: 'False',
      text: 'Individual(s) present',
      useCompatibleTextRendering: 'True',
      controls: [Array],
      dataBindings: [Object],
      location: '16, 112',
      xCoordinate: '16',
      yCoordinate: '112',
      size: '456, 112',
      tabIndex: '2',
      useWaitCursor: 'True',
      inner: [Array]
    },
    Control {
      type: 'iMedica.Prm.Client.UI.BaseControls.PrmLabel, iMedica.Prm.Client.UI.BaseCon
```

3. MVC pattern in node.js with express.js

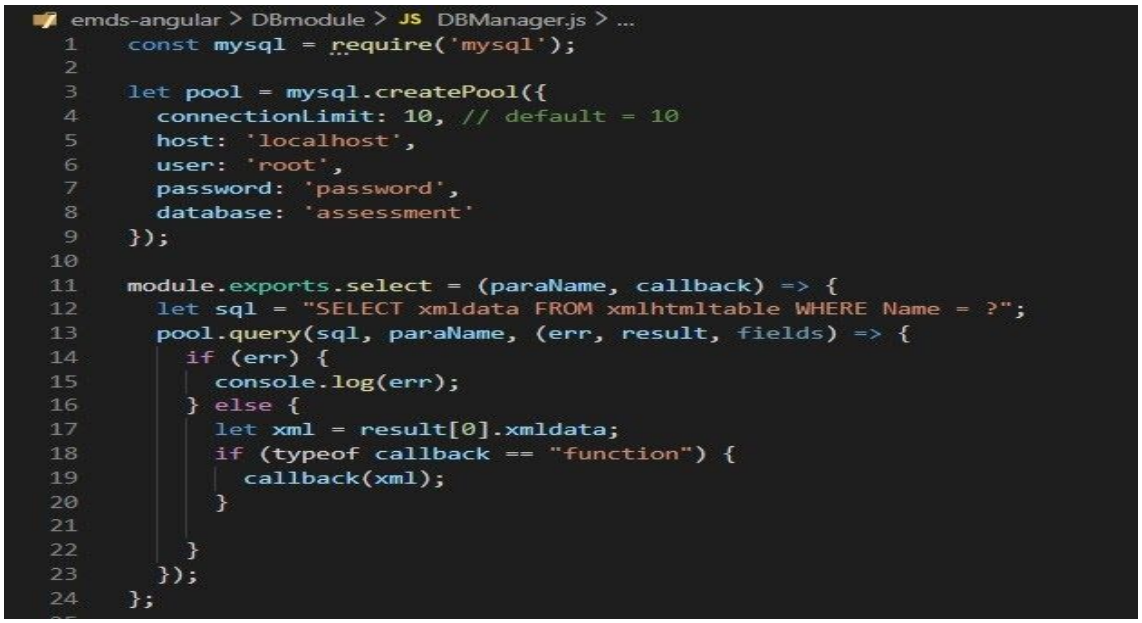
   Model-View-Controller is a software design pattern commonly used for developing user interfaces which divides the related program logic into three interconnected elements. This pattern has become popular for designing web applications.

   3.1. Model(MySQL)

   - In this project, we used MySQL as our model to store XML for dynamic HTML.

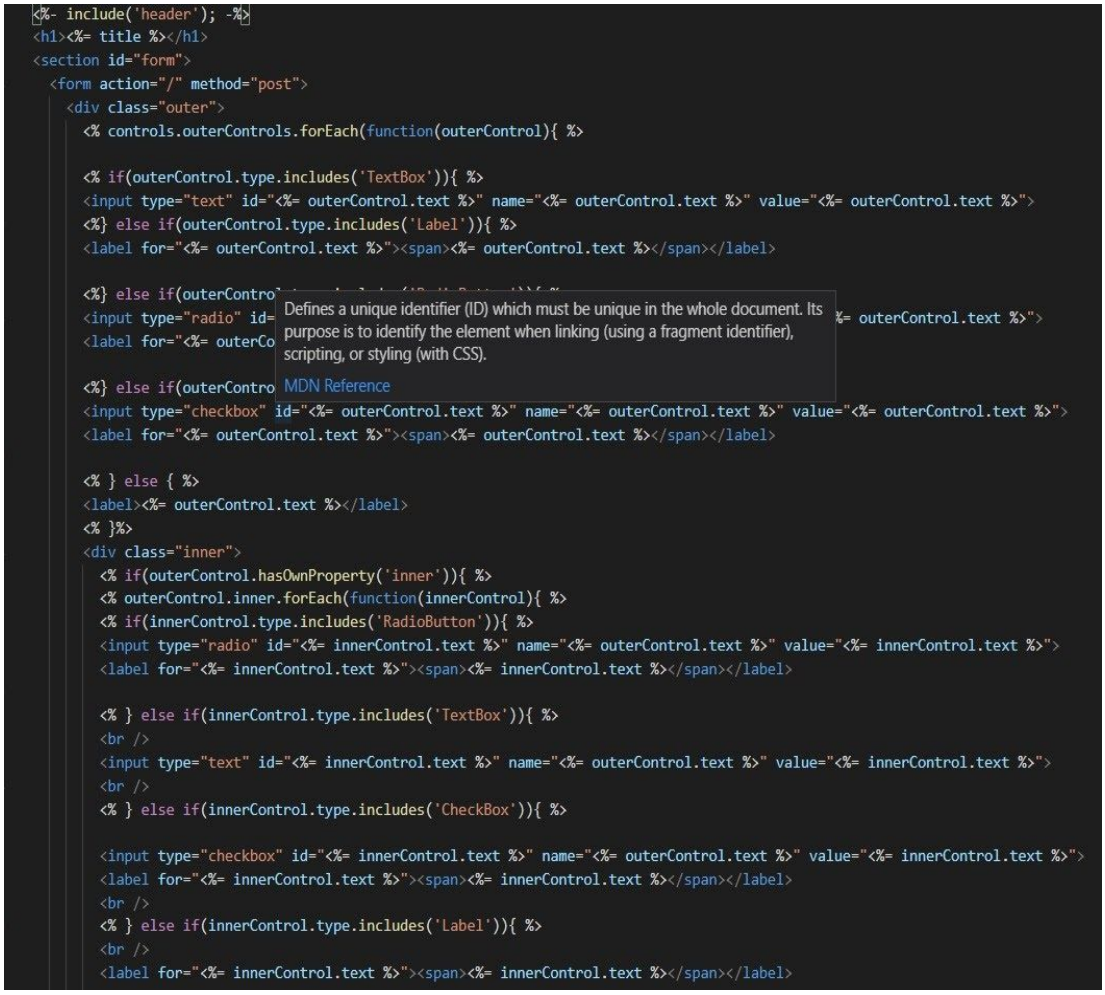


   - Below screenshot is creating a connection to database and query xml data.



3.2. View(EJS)

- To render dynamic HTML on the server side, we used a template engine called Embedded Javascript(EJS) for our template engine. EJS is a simple templating language that lets you generate HTML markup with plain javascript.
- Once the controller sends sorted control object to view, we can inject and manipulate this data in this form.




```
<%- include('header'); -%>
<h1><%= title %></h1>
<section id="form">
  <form action="/" method="post">
    <div class="outer">
      <% controls.outerControls.forEach(function(outerControl){ %>

      <% if(outerControl.type.includes('TextBox')){ %>
      <input type="text" id="<%= outerControl.text %>" name="<%= outerControl.text %>" value="<%= outerControl.text %>">
      <%} else if(outerControl.type.includes('Label')){ %>
      <label for="<%= outerControl.text %>"><span><%= outerControl.text %></span></label>

      <%} else if(outerContro
      <input type="radio" id=                                                    %= outerControl.text %>">
      <label for="<%= outerCo

      <%} else if(outerContro
      <input type="checkbox" id="<%= outerControl.text %>" name="<%= outerControl.text %>" value="<%= outerControl.text %>">
      <label for="<%= outerControl.text %>"><span><%= outerControl.text %></span></label>

      <% } else { %>
      <label><%= outerControl.text %></label>
      <% }%>
      <div class="inner">
        <% if(outerControl.hasOwnProperty('inner')){ %>
        <% outerControl.inner.forEach(function(innerControl){ %>
        <% if(innerControl.type.includes('RadioButton')){ %>
        <input type="radio" id="<%= innerControl.text %>" name="<%= outerControl.text %>" value="<%= innerControl.text %>">
        <label for="<%= innerControl.text %>"><span><%= innerControl.text %></span></label>

        <% } else if(innerControl.type.includes('TextBox')){ %>
        <br />
        <input type="text" id="<%= innerControl.text %>" name="<%= outerControl.text %>" value="<%= innerControl.text %>">
        <br />
        <% } else if(innerControl.type.includes('CheckBox')){ %>

        <input type="checkbox" id="<%= innerControl.text %>" name="<%= outerControl.text %>" value="<%= innerControl.text %>">
        <label for="<%= innerControl.text %>"><span><%= innerControl.text %></span></label>
        <br />
        <% } else if(innerControl.type.includes('Label')){ %>
        <br />
        <label for="<%= innerControl.text %>"><span><%= innerControl.text %></span></label>
```

- By using foreach function, it loops through the outer control array first and checks if this outer control has inner controls. If it has it generates every inner control.
- This form can be used to apply for any other forms that our sponsor might have. As long as their XMLs are in the similar layout, it will work.

3.3. Controller(express)

- The Controller is our main server that groups routes and manages middleware functions.
- The following screenshot shows the elements of a middleware function call:

```
const express = require('express');
const app = express();

// By adding semicolon, it takes user's query string.
app.get('/:custom', (req, res, next) => {
  let paraName = req.params.custom; // parsed quesry string.
  dbAPI.select(paraName, (xml) => {
    xmlParser(xml, (json) => {
      //  path for controls.
      let property = json['Viklele.FormDesigner'].Object.Proper
      //  sorted outercontrols
      const controls = sortControl(property);

      const options = {
        title: paraName,
        controls: controls
      }

      res.render('form', options);

    });
  });
});

app.listen(process.env.PORT || 4000, () => {
  console.log('server is running on port 4000');
});
```

- 'Get': is HTTP method for which the middleware function applies.
- '/:custom': Path(route) for which the middleware function applies.
- (req, res, nex) is the callback argument to the middleware function.
- Req is the HTTP request argument and Res is the response.
- Because of the asynchronous feature of node.js, we used callback functions for dbaccess and parsing xml which will be executed in sequence.
- After dbaccess, parsing and sorting, 'res.render' will pass sorted controls to view.

- **Sources**

  https://www.geeksforgeeks.org/angular-7-installation/

  https://www.thepolyglotdeveloper.com/2015/01/parse-xml-response-nodejs/

https://www.udemy.com/

https://stackoverflow.com/

https://developer.okta.com/blog/2019/08/16/angular-mysql-express

- **Appendix**

  https://nodejs.org/en/

  https://ejs.co/

  https://www.npmjs.com/package/xml2js

  https://expressjs.com/

  https://www.mysql.com/

  https://angular.io/

  https://code.visualstudio.com/

  https://notepad-plus-plus.org/

  https://github.com/

  https://www.google.com/chrome/

- **Impact & Security**
  - By creating a new method to generate e-forms, eMDs will no longer have the need to use third-party software to generate e-forms, which would provide more flexibility for users by giving them more options and tools to create the e-forms they need. Furthermore, by utilizing this newer technology, eMDs will be able to support mobile and web platforms. In terms of security, reducing the use of third-party software would lower the chances of a data breach since it would lower the amount of individuals with access to sensitive information.

- **Individual Assessment**
  - "Jason" Hyeoncheol Kim:
    - As a part of the Front-end team, I worked on parsing xml files, creating a generic class for controls and sorting the nested controls. I helped my team to create a view for rendering html dynamically on the server side with EJS. And I also helped the back-end team to set up our server, route and database connection. To avoid monolithic application, I modularized dbaccess and XML parser.
  - "Tommy" Jibril Abdul:
    - As a member of the frontend team, I assisted in designing and building the frontend. This included helping design the methods that parsed JSON to a format that could be output by EJS templating. I also assisted in helping create a GitHub repository structure that helped our team collaborate effectively. I also served as the team 'scribe' keeping the minutes for our meetings.
  - Andres Daniel Uriegas:
    - I worked as the team lead for this project. Being the team lead, I was the main point of contact to our sponsor and faculty. I helped organize weekly meetings, even after the stay-in-place order was placed. I organized tasks for each team member to do through both Trello and in-person meetings. Aside from all that, I also helped in the programming of the project. I primarily worked on creating a database in MySQL for storing our XML and processed HTML forms. I also created the web service that runs on a localhost to run our Node program in a browser.
  - Tien Quang:
    - On this project I was part of the front-end team. I worked on creating the classes within the controls folder and helped with the HTML aspect of the project. Specifically, I helped in identifying and setting up the "controls" of the JSON file, which enables us to create the HTML dynamically and using EJS templating engine to create the web form.
  - Yvon Randolph Tsaju:
    - On this project, I worked with the back-end team. I helped them with the creation of the SQL table. I also built classes for each method by refactoring a server.js file and then, I helped the team to create modules by assigning the methods to the exports property of the module object.
  - Adam Nguyen:
    - As a part of the back-end team, I led the creation of the SQL table. I helped the team the database to the front-end by helping to create the web service, and I refactored the app.js and dbmanager.js files to ensure that the program follows proper URL syntax.

- **Issues & Lessons Learnt**
    - Teamwork
        - Working as a team was a great experience for everyone. Our sponsor helped us during the whole process by giving us many directives on how we should carry out this project.
    - Dynamically creating HTML
        - Because the JSON were so nested, we had to create and revise classes that will identify the specific things we were looking and organize the JSON in a way that will make the HTML creation easier
    - Custom variable in URL using Express
        - By utilizing Express.js, we were able to add a custom URL variable to our web service that allowed the program to potentially search for any form based off of the URL you go to
    - Version Control in Github
        - Despite most of us having used Github before, it was still a big struggle to keep versioning synced across all our users. As we quickly learned, to best fix this problem was to create individual branches of our development branch. Then we could just call for pull requests to merge potential conflicts before causing any problems
    - Getting started
        - Most of us were unfamiliar working with MySQL and XML so we had a slow start due to the amount of research required. But with the guidance of our sponsor we were able to get in the flow of things.

- **Future Work**
    - Features and updates to improve the project:
        - Saving the filled out web form into a database
        - Create JSON API for single page application with Angular
        - Stylizing with CSS

- **Ethics Discussion**
    - We did not encounter situations where we had to make ethical decisions, but should we continue this project, we would be dealing with private information of patients. So having a secure database and making sure everyone doesn't share any sensitive information would be our top priority.

- **Print Name/ Signatures/ Date**

Print Names	Signatures/Date
Yvon Randolph Tsaju	5/7/20
Andres Daniel Uriegas	Andres Uriegas 5/6/20
Hyeoncheol Kim	Hyeoncheol Kim 5/7/2020
Jibril Abdul	Jibril Abdul 05/06/2020
Tien Quang	5/7/2020
Adam Nguyen	Adam Nguyen 5.7.2020
Divya Eluri (Faculty Advisor)	5.7.2020
Paul Shinohara (Company Sponsor)	Paul Shinohara 05/06/2020