Oral Roberts University

Senior Project Preparation

Project Documentation

BiBimBox Vision-Based Learning Platform

Prepared by

Taeyoun Won McKinzy

Submitted as partial fulfillment of the requirements for the degree of

Bachelor of Science, Computer Information Technology

Contents

BiBimBox Application **3**

Introduction 3

Project Background (Stage 1) 3

Project Background (Stage 2) 3

Final Project Outline 3

User Guide 5

Front-end Documentation 5

How to start 5

UI/UX Design 5

Technologies Used 5

Middleware Documentation 6

Technologies Used 6

Back-end Documentation 10

Developers Journal 11

Why Web Application 11

Why Node.Js Application 11

Why MongoDB 11

Database Development 11

User Authentication 11

Creating Content 12

Deleting Content 13

Current Project Status 14

Future Functionality 15

BiBimFact (A Small Research) 15

Meaning of BiBimBox 15

The Future of Education 15

The Current Trend 15

Conclusion 16

References **17**

BiBimBox Application

Introduction

Project Background (Stage 1)

After the EON internship Dr.Schniter at ORU had approached me with an idea to use virtual reality to assist people in Zimbabwe. At the time, ORU Healing Team had collected data from Zimbabwe and was planning for another missions trip in the summer of 2019. However, with about a month of analyzing the collected data and the clients needs the project shifted direction. The Original purpose was to build a virtual reality program for wicking bed training in Zimbabwe, but eventually, the purposed changed to building an online resource platform for missionaries and ORU students.

There were several factors that lead to this decision. The first factor was that Zimbabwe's internet connection is slow and not sufficient. according to TechZim and other resources collected by the ORU Healing Team and the Engineering Department at ORU, Zimbabwe does not have good internet connections due to its high network cost. This made implementing virtual reality very difficult due to internet access. The second factor was that many of the phone available in the targeted area is not compatible with virtual reality technology. I tried modeling the application to be accessible offline, but without smartphones, it will be impossible for regular people in the area to have access to the VR training program.

Due to these two factors, I have changed the application to target missionaries and ORU Students. The application will allow ORU students to prepare for the missions trip and other missionaries to have a resource on the missions field. Please refer to page ..... for database design and UI design.

Project Background (Stage 2)

About two weeks later I received information that the needs of the department have changed due to changes happening in the department. After this news, I had two more meetings with the department/client and had to change parts of my project to fit with the new department. This was challenging since I only had a month and a half left of the project deadline. I had to make sure I change enough to fit the new department, yet be able to meet my project deadline.

As a result, I came up with the application idea that focused on the student's vision and assisting them through effective learning. The platform's functionality was very similar, but I needed to tweak one of the section.

Final Project Outline

Purpose: To build a mobile-friendly web platform that assists students to manage and collaborate projects through providing easy access to resources and management systems.

Audience: University students and faculties.

User Guide

Front-end Documentation

How to start

- Download the project from my GitHub or D2L
- In order to test this prototype, you need npm and nodemon installed on your computer.
- In the directory where app.js is type npm run dev to run the server
- then go to localhost:5000 to see the application

UI/UX Design

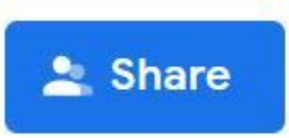
The focus of this project is to make it mobile-friendly and encourage students to grow through learning and building projects. Implementing the modern design trend of the clean and simplistic theme will make the users feel more comfortable and familiar with the application. Along with the modern design, using a color pallet that correlates with learning will assist the users to understand the purpose of the application.

Technologies Used

Bootstrap4: Bootstrap is one of the most popular opensource CSS frameworks. In this project, Bootstrap version 4 and a bootstrap theme "SB Admin 2" from startbootstrap.com was used. Bootstrap is a CSS framework, meaning most of the classes used on the HTML pages are predefined by Bootstrap.

example:

if you want to style a button to look like a blue button like the following, you can add a button tag with class= "btn-primary" attribute.



about 90% of CSS used in this project is from bootstrap. If further information on this topic is needed or need more examples, please read the documentation on at Bootstraps.com

Note: Bootstrap and the "SB Admin 2" Them used in this project is built using HTML5 and CSS and require a basic understanding of HTML5, CSS, and Saas concepts.

JavaScript: This project uses JQuary as a requirement for Bootstrap and also for custom ajax calls. JQuary is not a software, but an additional JavaScript library/repository to extend the functionality of JavaScript. The custom JavaScript file "globalScript.js" under public/js directory is used on buttons that need a specific ajax call to the rest API (app.js or it's sub-routers). Please refer to the comments in the file for more information.

Font Awesome: Font Awesome is one of the most popular icon set providers. Font Awesome provided a variety of SVG Icons to assist in mobile-friendly icons. In this project, only the free icons are used. Please visit fontawesome.com for more information and helpful documentation.

Google Fonts: Google Fonts is a, commonly used typography provider. The theme installed in this project uses Google Fonts for its typography. If you would like to use any other fonts/typography please feel free to use them. Please visit fonts.google.com for more information about Google Fonts and all the fonts/typography provided.

Middleware Documentation

Technologies Used

Embedded JavaScript Templates(EJS): EJS is one of the simpler templating engines that allows JS to run on HTML file. It makes it easier for the data to be referred in the HTML code or the whole HTML can be written in ejs. In this project, all the .html files were converted to .ejs for the convenience of data reference. At the time of this document, the main ejs website is not working. As a result please refer to nmpjs.com/package/ejs for more information.

examples: Any value within <% %> will be handled by ejs.

```
<% for(var i = 0; i < user.projects.length; i++) { %>
  <div class="col-xl-3 col-lg-4 d-flex">
    <div class="card shadow mb-3 h-90">
      <img
        src= "https://thenakedlistener.files.wordpress.com/2014/02/wall-koi-1280x960.jpg"
        class= "card-img-top"
        alt="programing langualge image"
      >
      <!-- Card Body -->
      <div class="card-body flex-fill">
        <div class= "text-center">
          <a href="/my-projects/<%=user.projects[i]._id%>" class="font-weight-bold text-primary"><%=user.projects[i].projectName%></a>
        </div>
        <p class="card-text"><%= user.projects[i].description.slice(0, 120) + " ..." %></p>
        <a href="#" value="<%=user.projects[i]._id%>" class="btn btn-sm btn-block btn-danger shadow-sm float-left" id="removeProjectBtn"> remove </a>
        <a href="#" class="btn btn-sm btn-block btn-primary shadow-sm float-right"> edit </a>
      </div>
    </div>
  </div>
  <% } %>
```

Image 1: an image of project-projects.ejs

The current code is looping through all the projects under the current user account and creating a card with the project name and description with char limit of 120 (model 2.2). The “user” value was sent from the rest API when the project page was rendered.

```
// User my project page
router.get("/", ensureAuthenticated, function(req, res) {
  User.findOne(req.user._id)
    .populate({ path: "projects", model: "Project" })
    .exec(function(err, user) {
      if (err) {
        res.status(500).send("Could not fetch user data");
      } else {
        curUser = user;
        res.render("project-projects", {
          user: user
        });
      }
    });
});
```

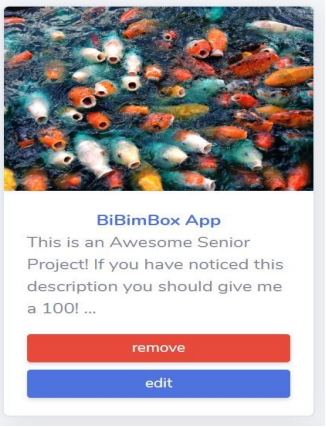


Image 2.1 image of projectRoute.js *Image 2.2: an image of an instance of a project card*

When there is a get request to a project page, the project page will find the logged in user using req.user._id and populates all its projects data with the associated project information. Then it will render the project-projects.ejs file with a parameter of user. That is how “user” was able to refer to the user’s information in *Image 1*. (If need more information on node syntax, please refer to page …)

Mongoose: Mongoose is Node.js package that handles object modeling for data going through Node.js to MongoDB. Since MongoDB is not a relational database, the data stored inside can become very messy and hard to track. by using Mongoose, I am able to limit what data is stored in MongoDB and keep the data format unified. Please visit mongoosejs.com for more information.

example: All project stored in MongoDB follows this model.

```
const mongoose = require("mongoose");
const FolderId = mongoose.Schema.Types.ObjectId;

const ProjectSchema = new mongoose.Schema({
  projectName: {
    type: String,
    required: true
  },
  folders: [
    {
      type: FolderId,
      ref: "Folder"
    }
  ],
  description: {
    type: String,
    required: true
  }
});

module.exports = mongoose.model("Project", ProjectSchema);
```

Image 3: an image of the project Model/Schema

The line `const FolderId = mongoose.Schema.Types.ObjectId;` makes `FolderId` a type of objectID. This allows the folders to hold other object's primary id in the MongoDB database. As a result, all of the projects in MongoDB must have projectName, description, and folders. However, "folders" is an optional key and can be empty because it does not have `required: true` (please refer to *Image 4).*

Image 4: an image of the data of the project in the database

Express: Express is a web framework for Node.js. Express work as a communicator between Node.js and the browser and translates all the request for C.R.U.D operation to Node.js. Express contains many packages to send and receive data from a browser such as JSON bodyParser. Please visit expressjs.com for more information and additional documentation on express.js.

example: how a post request is handled.

```
<form class="collapse show" id= "youtube-and-text" data-parent="#collapseArea" action="/my-projects/folders/add/<%=folder.parentProject%>/<%=folder._id%>"  method="POST">
    <div class="form-group">
        <label for="projectTitle" class="col-form-label">Content Name:</label>
        <input type="text" class="form-control" id="projectTitle" name="mTitle" placeholder="What is the title of your content?">
    </div>
    <div class="form-group">
        <label for="projectTitle" class="col-form-label">YouTube Link (Optional):</label>
        <input type="text" class="form-control" id="YouTubeLink" name="yLink" placeholder="Link a YouTube Video?">
    </div>
    <div class="form-group">
        <label for="ProjectDescription" class="col-form-label">Description:</label>
        <textarea type="text" class="form-control" id="projectDescription" name="mDescription" placeholder="Please describe your content here"></textarea>
    </div>

    <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-dismiss="modal">Cancel</button>
        <button type="submit" class="btn btn-primary">Create</button>
    </div>
</form>
<!--File uploader Tab-->
<form class="collapse" id= "file-uploader" data-parent="#collapseArea" action="/my-projects/folders/add/<%=folder._id%>"  method="POST">
    <div class="form-group">
        <label for="projectTitle" class="col-form-label">Content Name:</label>
        <input type="text" class="form-control" id="projectTitle" name="mTitle" placeholder="What is the title of your content?">
    </div>

    <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-dismiss="modal">Cancel</button>
        <button type="submit" class="btn btn-primary">Create</button>
    </div>
</form>
```

route

type

Image 5: an image of a form in project-contents

First, the form is filled out and is sent when a button with type = submit is pressed or through a ajax call. Then the form will send a request with the type of `"POST"` to the route:

`"/my-projects/folders/add/<%=folder.parentProject%>/<%=folder._id%>"`

```
//#region Routes
app.use("/", require("./routes/dashboard.js"));
app.use("/users", require("./routes/users.js"));
app.use("/my-courses", require("./routes/courseRouter.js"));
app.use("/my-projects", require("./routes/projectRouter"));
app.use("/my-projects/folders", require("./routes/folderRouter.js"));
app.use("/join-projects", require("./routes/joinProjectRouter"));
app.use("/sections", require("./routes/sectionRouter.js"));
app.use("/materials", require("./routes/materialRouter.js"));
app.use("/contents", require("./routes/contentRouter"));
//app.use("/uploads", require("./routes/uploadRouter"));
//#endregion end of routes
```

Image 6: an image of routing in app.js

Second, any data with `"/my-projects/folders”` will be rerouted by express to `"folderRouter.js”` file, since its route started with the `"/my-projects/folders”`.

```
//add content
router.post("/add/:projectId/:folderId", function(req, res) {
  let materialBody = new Material();
  (materialBody.parentProject = req.params.projectId),
    (materialBody.parentFolder = req.params.folderId),
    (materialBody.title = req.body.mTitle),
    (materialBody.url = req.body.yLink),
    (materialBody.description = req.body.mDescription);
  materialBody.save(function(err, savedMaterial) {
    if (err) {
      res.status(500).send({ error: "Could not save Project" + err });
    } else {
      Folder.findByIdAndUpdate(
        { _id: req.params.folderId },
        { $addToSet: { materials: savedMaterial._id } },
        function(err, UpdatedFolder) {
          if (err) {
            res.status(500).send({ error: "Could Not Add The Project" + err });
          } else {
            //res.send(Updatedproject);
            res.redirect("/my-projects/folders/" + req.params.folderId);
          }
        }
      );
    }
  });
});
```

Image 7: an image of a post request handler in folderRouter.js

Third, in the `folderRouter.js` file the data will be sent to a .post request that has the rest of the routing address "`/add/<%=folder.parentProject%>/<%=folder._id%>`" Now the post request will be handled by the framework.

In this case, the framework is saving the material information to the database and storing the reference to the folder. During this process, express will use the values from the route parameters and parsing the req.body by using bodyParser (Please refer to app.js line 50 - 52 for declaration of bodyParser).

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

Image 8: an image of an express reference in app.js

Note: The "`express`" is declared in app.js line 2 and "`app`" is defined in line 16. Also, if you look at the code app refers to express. When rerouted to a different file such as `folderRouter.js`, express will be referred by using a router. That is why *image 7* has `router.post` instead of `app.post`.

```
const router = express.Router();
```

Image 9: an image of an express reference in folderRouter.js

Back-end Documentation

Node.Js: Node.Js is a runtime environment for executing JavaScript. In this project, Node.js is used as a physical server. There really isn't much to understand about node beside the fact that it allows a JavaScript server to run on a physical machine (like a laptop). Please visit nodejs.org for more information and additional documentation.

One the other hand node package manager (npm) is used throughout the project to import different packages to the project. Please visit npmjs.com for more information and additional documentation.

MongoDB: MongoDB is a NoSQL database, meaning it does not have to follow all the relational database rules. Please visit mongodb.com for more information and documentation.

In this project, the database is connected using mongoose to organize the incoming data (please refer to the Middleware section for details on Mongoose).

```
//This is the connection point to MongoDB Atlas
dbPassword =
  "mongodb+srv://tmck4658:tmck4658@cluster0-wt1ga.mongodb.net/bibimbox";
```

Image 10: an image of the access point for MongoDB Atlas

The key/access point stored in *image 10* will be called in app.js to connect the MongoDB Atlas to Node.js Application. The address can be a local host if needed.

```
//#region Connecting to database (MongoDB Atlas)
// Grabbing key
const db = require("./config/keys").mongoURI;

// Setting the refference to the connection
const conn = mongoose.connection;

// Rerouting the database so that we can access MongoFB to user instead admin
// "mongodb+srv" protocol the default to admin and Admin is not allowed in MongoDB Atlas
conn.on("connected", function() {
  if (conn.client.s.url.startsWith("mongodb+srv")) {
    conn.db = conn.client.db("bibimbox");
  }
  console.log("Rerouting Atlas collection: Complete.");
});

// Connect to MongoDB using mongoose
mongoose
  .connect(db)
  .then(() => console.log("MongoDB Atlas: Connected"))
  .catch(err => console.log(err));
```

Image 11: an image of the connection process in app.js

Image 11 shows the connecting process of the MongoDB Atlas and Node.js using Mongoose. Please refer to the comments for more information.

Developers Journal

Why Web Application

Originally, this application was going to be built as a mobile application, but there have been specific problems I ran into that changed my mind. First, Even though I had taken mobile application development, I was not proficient enough to create a full-scale application. My option was to use React-Native to build a mobile application, but the new stack was very difficult to understand. Second, The admin managing everything on the application without web application will go against the idea of the human center design method. My initial goal was to create a full functioning application including administrative functionalities and client functionality. As a result, I thought it was best for me to create a web application first and let others work on the mobile application after me (if the project continues onward).

Why Node.Js Application

I knew I was not strong in JavaScript and wanted to challenge myself to learn this fundamental programming language for web development. I knew that Node.Js allowed me to use JavaScript for front and back-end development. This unified the front and back-end programming language and made it easier for me to focus on understanding the workflow of web application development. Also, there were enough tutorials online for me to learn and problem solve.

Why MongoDB

My project was constantly changing and MongoDB allowed me to be flexible with my database. Being new to technology will cause a lot of first-time errors and I needed a database that was flexible yet easy to learn. Thankfully, I was able to change a lot of my design mistakes due to a flexible database.

Note: Even though MongoDB is flexible, each object have a size limit of 16Mb. So, multer and multer-gridfs-storage and gridfs-streams packages from npm are needed for implementing a file upload.

Database Development

User Authentication

I based this project around the User Authentication because this is a key component in this project. I followed Traversy Media's tutorial on how to achieve user authentication with Node.Js and changed some things around to match this project's front-end and back-end. I had to add several things to the user model such as privileges, projects, and classes. Besides these few things and minor changes, the tutorial should be able to explain most of the process. If you would like more information or compare my code with Travery's original code, visit Travery's Github repository.

Note: The hard part is actually understanding someone else's code and merging it with your current code. Please be careful when pulling other repositories.

Creating Content

Creating contents were easy at first but when I started to implement nested content, it got difficult. Like shown in *image 7,* I originally just need to pull the data from the request body (req.body), but as the contents became sub-content of each other I needed to work with parent object's _Id.

example: Process of Creating a folder

```
//create folders
router.post("/new-folder/:projectId", function(req, res) {
  let folderBody = new Folder();
  (folderBody.parentProject = req.params.projectId),
    (folderBody.folderName = req.body.fTitle),
    (folderBody.description = req.body.fDescription);
  folderBody.save(function(err, savedFolder) {
    if (err) {
      res.status(500).send({ error: "Could not save Project" + err });
    } else {
      Project.findByIdAndUpdate(
        { _id: req.params.projectId },
        { $addToSet: { folders: savedFolder._id } },
        function(err, Updatedproject) {
          if (err) {
            res.status(500).send({ error: "Could Not Add The Project" + err });
          } else {
            //res.send(Updatedproject);
            res.redirect("/my-projects/" + req.params.projectId);
          }
        }
      );
    }
  });
});
```

Image 12: an image of the creating folder process in projectRouter.js

First, fill all the Model/Schema requirement by pulling values from requested data from the req.body (Line 4 -6 in *Image 12*).

Using the following format: `folderBody.parentProject = req.params.projectId`

Second, Save the data to the database (Line 7 in *Image 12)*. Third, check for error. If there was an error, send status 500 along with error code (Line 8 - 9 in *Image 12*). Fourth, If successful find the project with the same projectId and add new folderId to its folders field (Line 10 - 13 in *Image 12*). Fifth, repeat the third step. Sixth, If everything was successful refresh the page.

Note: Currently, creating folder is using nested callback functions, but for the sake of organization it is better to create different functions. Implementing multiple functions is used in Deleting Content if need a reference.

Deleting Content

Deleting a top-level data, such as project, is very difficult. When deleting a project, you much delete all the references inside the project. Without this process, there will ghost data that does not have any reference calls in the database. As a result, I decided to send a parent objects _Id to all of its subfolders when creating them (please refer to *image 7)*. This allowed me to use a delete function to delete all the object with projectId as a reference.

example: Process of Deleting a Project

```
//#region Project Delete Requests
//Delete Project
router.delete("/delete/:projectId", function(req, res) {
  Project.deleteOne({ _id: req.params.projectId }, function(
    err,
    deletedProject
  ) {
    if (err) {
      res.status(500).send("Could not remove project: " + err);
    } else {
      //when project is deleted, delete all folders inside the projects
      deleteFolder(req.params.projectId);
      res.send(req.params.projectId);
    }
  });
});

//Delete Folders
function deleteFolder(projectId) {
  //When each folder is deleted, delete all materials inside
  Folder.deleteMany({ parentProject: projectId }).then(
    deleteMaterial(projectId)
  );
}

//Delete Materials
function deleteMaterial(projectId) {
  //Console Log that the process is done
  Material.deleteMany({ parentProject: projectId }).then(console.log("Done"));
}
```

Image 13: an image of the deleting project process in projectRouter.js

When a delete project request is made, it deletes the project then calls the `deleteFolder` function. `deleteFolder` function will find all folders with `parentProject: projectId` and delete them. Then the `deleteFolder` function will call `deleteMaterial` function which repeats the process till all data with `parentProject: projectId` is deleted.

Note: OverView

Creating: When a sub-data is created, pass it's all of its parent's _Id.

Deleting: When a data is deleted, remove all data that holds deleted objects _Id. All data related to the deleted data should have the deleted data's _Id because of the creating step above.

Current Project Status

Done: Complete, Half: Halfway done, Fourth: Forthway done ..., Future: Future Implementation

Green: Critical Impact & done, Yellow: Critical Impact & working on it, Red: Critical impact & not done

Functionality	Status	Impact on Launching
User Authentication	Done	Critical
Create Project	Done	Critical
Join Project	Half	Critical
Delete Project	Done	Critical
Edit Project	Future	Critical
Create folder	Done	Critical
Delete folder	Half	Critical
Edit folder	Future	Critical
Create Project Material	Done	Critical
Delete Project Material	Half	Critical
Edit Project Material	Future	Critical
Create Course	Done	Critical
Edit Course	Future	Critical
Delete Course	Half	Critical
Join Course	Future	Critical
Create Section	Done	Critical
Edit Section	Future	Critical
Delete Section	Half	Critical
Create Content	Done	Critical
Edit Content	Future	Critical
Delete Content	Half	Critical
Admin Page	Eighth	Critical

Future Functionality

Along with finishing all the features required the following feature may be beneficial to the project:

- Search function
- Home/Welcome Page
- Data Analytics
- Additional UI/UX design
- Implementing Color Pallet
- Messaging functionality
- Email verification
- File Uploader
- Cooperative environment

BiBimFact (A Small Research)

Meaning of BiBimBox

BiBimBop is a Korean food that contains rice, vegetables, meat, and spicy sauce. However, the word "BiBim" in this case means to mix, hance BiBimBox means Mixed Box. This represents the concept of people's ideas growing by getting mixed with other people's ideas.

The Future of Education

Education in America has been trying to implement new technologies into their education systems for decades now. I remember my middle school and high school years when schools were trying to provide laptops and Ipads for the students. However, Even to this day many of the educational applications are slow to catch up to the new technologies. At EON Reality, I was working on one of the educational VR application and was notified by the project managers that there are not enough educational meterials for the schools to use their Ipads properly.

Also, one of the studies done in South-Western University addresses some of the problems about modern e-learning systems.

"[T]he crucial question is not 'What technological tools are to be used during the development process of e-learning courses?' The core problem is 'How to design and plan an e-learning course that ensures the achievement of the educational objectives?'" (Tuparova IV.12-1)

"A modern e-learning environment not only has to offer the most recent technologies but it has to possess high level of usability also. This environment is to be adaptive to different learning models, e.g. constructivist learning, collaborative learning, experimental learning, problem based learning. In our opinion it is obligatory" (Tuparova LV.12-2)

The Current Trend

The book Trend Korea 2019 discuses about the current trends in Korea and predicts how it will affect businesses in Korea. The book is based on research done from Seoul Universities business department and published every year. This book addresses the trend of the upcoming millennial families and how more and more people are focusing on self-improvements (Kim 1064).

Conclusion

Even though there are many e-learning tools, it can not keep up with the paste of the technology industry, and it lacks in many aspects of learning. Also, with the demand for e-learning due to self-improvement trends calls for a flexible e-learning solution. BiBimBox, with its project management functionality and course management system, can provide a flexible environment to instructors and students. Also, being able to do hands-on learning through different types of projects and being able to create custom courses as an instructor will make BiBimBox an ideal platform for people to implement different learning methods.

References

Kim, Nan-do, et al. *트렌드 코리아 2019*. EPUB, Mirae Ui Ch'ang, 2018.

Tuparova, Daniela. “Didactical Issues of e-Learning-Problems and Future Trends.” *Academia.edu*, www.academia.edu/16847546/Didactical_issues_of_e-learning-Problems_and_future_trends.