e-Valuate

Developer Guide

Version 1.0.0

Revision History

Date	Version	Description	Author
04/May/20	0.0.1	initial draft, introduction, and web application development	Action Ghimire
05/May/20	0.0.2	Initial draft of deployment section. This includes all subsections except 4.3 and 4.4 and all passwords are redacted.	Jordyn Dent
05/May/20	0.0.3	Reorganize and reword the introduction, and frontend sections. Write the backend section.	Kenny Houston
06/May/20	0.0.4	Finished sections 4.3 and 4.4	Jordyn Dent
06/May/20	1.0.0	Formatting	Kenny Houston

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to provide an overview of the development requirements of the e-Valuate web application. For a developer wishing to make changes to the application, but who is unfamiliar with its structure, this document is meant to guide them.

1.2 Project Overview

Each year, the CCSC (Consortium for Computing Sciences in Colleges) South Central Region hosts a conference in which contestants can submit research posters to be judged. Any given year can have dozens of posters, but since the adoption of this conference, organization and judging efforts have been completed by hand (and often on paper) by the chair and his colleagues.

When contestants sign up for this event, they must contact the chair directly. This requires time consuming and unnecessary work on the part of the chair. Then, during the event, judges must grade each poster using a paper rubric and a pen. Then, all judges' scores are compiled together into an excel spreadsheet. This is a very time-consuming and error prone process which requires a lot of moving pieces to be successful.

e-Valuate is an online event management and judging platform which aims to streamline this process. Our solution removes the headache of doing everything manually and allows everyone involved with the conference to utilize the benefits of the service. e-Valuate also allows for similar contests to reap the benefits of our digital service, by leaving the structure of events and rubrics to be customizable.

e-Valuate is a web application built using Spring Boot for its backend and React for its front end, all running on AWS.

1.3 Technology Stack and Document Overview

e-Valuate is a web application built using Spring Boot for its backend and React for its front end, all running on AWS. Each technology receives an indepth look within its own section. Here, we give a very breif explanation as to the purpose of each technology.

1.3.1 Frontend - React

The React framework is a single page application, consisting of JavaScript and HTML. The graphical interface uses a decoupled architecture to interact with the backend.

1.3.2 Backend - Spring Boot

Spring Boot is a Java based framework using a layered architecture. Spring Boot interfaces incredibly well with MySQL using JPA Repositories. Therefore, MySQL is our database, but as developers we need not interface with it directly.

1.3.3 Server - AWS

Amazon Web Services is an incredibly popular platform that took care of all of our needs, even on the free tier. On the server, which is described in depth in section 4, exists a web server for our application and a MySQL server for our data.

2. Frontend - React

2.1 Yarn Package Manager

Yarn is very similar to npm and is our package manager of choice. The frontend directory is the home of the yarn project. This directory contains some package files and lock files, in addition to directories containing source code and external resources.

2.2 Material UI Library

Material UI is a React component library full of useful and visually appealing components.

2.3 Source Code

All frontend source code can be found in the "frontend/src" directory.

2.3.1 Individual Files

Here, we have 3 high level files: index.js, Evaluate.js, and Utils.js. index.js simply starts the react app; nothing more. Evaluate.js routes our single page application to different components, creating the illusion of a multi-page application. Utils.js holds utility functions to be used in multiple pages.

2.3.2 Pages

The bulk of the source code is contained here. Many of the pages are divided up into directories based on their intended user. Pages intended for multiple users are not contained within subdirectories. These include the landing page, the event details page (which can take different forms when viewed by different types of users), and the dashboard, which is used by both chairs and judges.

Each page extends the React Component class. We take advantage of the Component class by overriding certain features and functions. The typical page in the application has a constructor method, a componentDidMount method (where necessary), and a render method. While the componentDidMount method is optional, the constructor and render are basically required. The constructor initializes the state and binds functions. The render method sets up the dynamic html content. The componentDidMount method typically fetches the initial set of dynamic content.

The state is an incredibly important feature. It is a javascript object that literally contains the state of the page, including all dynamic information. It can be updated by calling the setState function and passing a javascript object. The passed object should contain all the key-value pairs which need to be updated in the state. It need not include the entire state.

Each page also includes a plethora of other functions in order to organize the various tasks required. Above the page declaration are global functions that return visual components. These are helpful for organizing the repetitive visuals, like table rows or cards.

2.3.3 Components

The components directory contains dynamic visual components to be used within multiple pages. The components do not extend the Component class, so they do not have a state. They simply take a set of props, which determine their visual output. Examples include standardized buttons and textfields. There are notable components which we will cover next.

2.3.3.1 Layout

The Layout component is used in and surrounds the main content of every page. It is both the header at the top of the page and the footer at the bottom. It controls the user drop down menu in the top right corner.

2.3.3.2 Screens

Screens exports a few components, but the most useful is the StateScreen component. When used properly, it acts as a loading screen, submitting screen, and an error screen. It takes a single prop called "state", which is an integer. Depending on the value of this integer, it can show different error or loading screens.

This functionality is typically implemented by keeping a "loading" attribute in the state of a page. Most pages start with a value of 0 (loading), then move to a value of 1 (loaded) once the componentDidMount function wraps up. When passed a value of -1, the screen is "submitting." All values greater than 1 are treated as error codes and trigger error screens. An if-statement at the beginning of the render function should act as a filter to the main content of the page. If you are looking for examples, any page with dynamic content should do.

2.3.4 Static

The static directory contains two subdirectories: one for images and one for styles. Both are fairly self explanatory.

2.4 References

Yarn: https://yarnpkg.com/
React.js: https://reactjs.org/
Material UI: https://material-ui.com/

3. Backend - Spring Boot

3.1 IDE

We mostly used Eclipse, though intelliJ should do just fine. Both have plugins that can be helpful.

3.2 Maven

Maven is a common dependency manager utilized by spring boot. Its configurations lie in pom.xml. There are plenty of online tools if you need to dive into this file but hopefully you shouldn't. The pom.xml also contains build information that builds the frontend and incorporates it into the "fat" jar which is placed in the target directory. This can be fairly time consuming and may want to be removed during development.

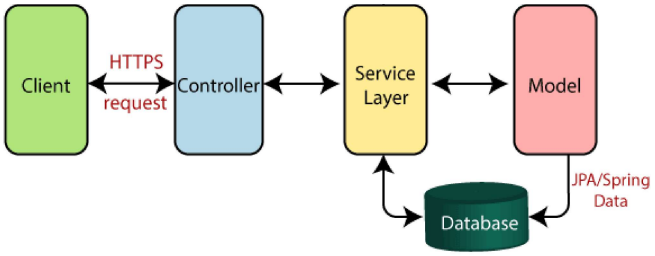
3.3 Source Code

The src directory contains all of the backend source code. There are three main packages within it: src/main/java, src/main/resources, and src/main/java. The bulk of the source code is contained within sub packages of the src/main/java package. Meanwhile, src/main/resources really only contains the application.properties file, and src/test/java contains unit tests for their src/main/java counterparts. The details of src/main/java are expanded upon in the Architecture section.

3.4 Architecture

Spring boot's architecture is highly layered. The horizontal layers of our application are standard among spring boot applications. The vertical slices of our application are less typical. We have opted for each slice to pertain to a page within the frontend, rather than have each slice pertain to a model in the backend. For example, there is a CreateEventController and CreateEventService. Rather than an EventController and EventService.

Each of these layers (Controller, Service, Model) has a package within the backend. Other packages include the repositories, shiro, and bootstrap packages.



3.4.1 Controllers

A controller's only job is to interface with the client and make calls to the service layer. No (or very little) business logic is done within the controller layer.

3.4.2 Services

All business logic is performed here. Relationships between models are formed and saved here. The MailService and HashService are services which are not intended for a single controller. The MailService interfaces with evaluate.tcu@gmail.com to send emails to users. The HashService allows other services to encrypt different information, like user's passwords, in the same way that Shiro does.

3.4.3 Models

Models are the data representation of everything. For the most part, they are simple java classes with getters and setters. However, built into the setters are means of making every relationship reciprocal. This makes work within the service layer a bit more straightforward. The models also contain plenty of annotations to control how the entities translate into the MySQL database.

3.4.4 Repositories

Speaking of the database, all queries are done indirectly through JPA Repositories. The JpaRepository interface allows us to simply specify the model class and the data type of its ID, and then have nearly all the access to the database we need. If more specific queries are required, only the method names need be declared (in a specific format), and the interface takes care of the rest.

3.4.5 Bootstrap

Upon handoff, the important parts of the DevBootStrap class within the bootstrap package have been commented out. The DevBootStrap class made it easy to insert testing/sample data into the database during development. Fake emails and simple passwords in plain text are obviously not ideal upon deployment, so they are nonexistent. But for development, the commented out insertions still exist.

3.5 Shiro

Shiro is a security framework from apache. Near the end of the project, we began to see some of the drawbacks of this framework. Instead, we likely should have chosen Spring Security as it likely pairs better with the Spring Boot Framework. With Shiro, features like "Remember Me" are buggy and insecure, so they have to be left out. But overall, Shiro does the job, and interfaces well with our database if the configuration is set up correctly.

All shiro configuration can be found within the Shiro package. We use Shiro to differentiate between anonymous users and authenticated users. Shiro also filters through requests, preventing certain requests from ever reaching the controller layer if they require authentication. However, more complicated business logic surrounding read/write permissions to specific entities has to be done manually in the Service layer. A useful function for verifying authorization is the SecurityUtils.getSubject().getPrincipal() function. It returns an instance of the Account model for the current user. From there, all entity relations can be confirmed or denied.

3.6 Application Properties

The application.properties file is a simple configuration file which exists in the src/main/resources package. It almost exclusively contains configurations concerning the database and how information is loaded from it. Most importantly, the url, username, and password for the MySQL server must be present for the application to build and run properly.

3.7 References

Eclipse IDE: https://www.eclipse.org/
Eclipse Spring Tools 4 Plugin: https://marketplace.eclipse.org/content/spring-tools-4-aka-spring-tool-suite-4
Maven: http://maven.apache.org/
IntelliJ IDEA: https://www.jetbrains.com/idea/
Spring Boot: https://spring.io/projects/spring-boot
Shiro: https://shiro.apache.org/
MySQL: https://www.mysql.com/

4. Server - AWS

4.1 AWS EC2 Instance

Created an Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type server of size t2.micro.

Configured the security group to open ports 22 (SSH), 80 (HTTP), 443 (HTTPS), 3306 (MySQL), 8080 (spring).

Accessible via ssh using the key file, e-Valuate.pem

4.2 MySQL on the Server

On the server, mySQL has one user, evaluate-user, and one database, evaluate. The evaluate-user is the user that the application leverages to read and write data, and this user can only create, select, insert, update, delete, alter, and reference data for the evaluate database.

Root MySQL access is not allowed from remote connections, so if you have access to the server, you can use the MySQL root password to make administrative changes to the database.

4.3 Apache as Proxy

With how Spring is set up, the application's jar file has a built-in Tomcat web server that broadcasts the app on port 8080. While the port used by the server can be customized, it cannot naively be set to the HTTP (80) or HTTPS (443) ports, so a proxy must be used to show the application on ports 80 or 443. The following virtual host is customized to take the application endpoint being broadcasted at port 8080 and move it to port 80. Then, once the certificate in section 4.4 is in place, the certificate is utilized to broadcast the app on port 443 and encrypt all traffic moving to and from the server.

```
<VirtualHost *:80>
    DocumentRoot "/www/evaluate"
```

```
    ServerName e-valuate.net
    ServerAlias www.e-valuate.net
    ProxyPreserveHost on
    RequestHeader set X-Forwarded-Proto https
    RequestHeader set X-Forwarded-Port 443
    ProxyPass / http://127.0.0.1:8080/
    ProxyPassReverse / http://127.0.0.1:8080/
    # Other directives here
    RewriteEngine on
    RewriteCond %{SERVER_NAME} =e-valuate.net [OR]
    RewriteCond %{SERVER_NAME} =www.e-valuate.net
    RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
</VirtualHost>
```

4.4 LetsEncrypt Certificate for HTTPS

Once the virtual host in section 4.3 was set-up, certbot was utilized to provide a certificate for the domains (www.e-valuate.net and e-valuate.net) and encrypt traffic to and from the server. Certbot is a tool that creates an official SSL/TLS certificate with LetsEncrypt, which is a free, automated, and open certificate authority (CA).

A cron job was set up to check the status of the certificate every day. The certificate lasts for 90 days and will be auto-renewed when it expires.

4.5 Updating e-Valuate

Generate a personal access token on GitHub:
github.com -> settings -> developer settings -> personal access tokens

For the scope, which defines the token's access, select "repo"

Being a private repository, in order to get the code from the e-Valuate GitHub repository onto your machine, you must have access to the repository and generate a personal access token for your GitHub account. You only need to do this once for e-Valuate. If you prefer to use GitHub desktop, this is not required, but you still must have access.

To initialize the eValuate directory:
git clone https://github_username:access_token@github.com/tcuseniordesigncourse/eValuate.git

If directory exists, to update the eValuate directory:
cd eValuate
git pull https://github_username:access_token@github.com/tcuseniordesigncourse/eValuate.git

Make sure you are in the eValuate directory for each of the following commands

Open the application.properties file:
vim src/main/resources/application.properties

For the following lines, append the evaluate-user's data after the equal signs then save and quit:
spring.datasource.username=evaluate-user
spring.datasource.password=_____

Add execute privileges to mvnw script:
chmod u+x mvnw

Build the app (you must have Yarn installed):
./mvnw clean install

If successful, send the jar to the server with SFTP (or use FileZilla):
sftp -o "IdentityFile=path_to_key_file" ec2-user@ec2-18-220-166-169.us-east-2.compute.amazonaws.com
put target/evaluate-1.0.0.jar

Connect to the e-Valuate server

On the server, kill the current application process if it's running:
./stop

Start the new version of e-Valuate:
./start