Review for ENSE Master Race Hell Support Software Tool
Created by Team DCC
Last updated on April 11th, 2018

0. Preamble

We believe the code is well constructed and easy to understand. The overall approach to the solution was in a similar vein as our own and many other groups. Vue provides a rather nice to read front-end syntax and Javascript is a solid foundation for developing web services.

1. Cleanliness

Following the code is quite simple, the modularity of Vue really makes it easy to understand what a function is doing, as opposed to an abundance of queries inline in a php page.

1.1 Website

The Website is visually appealing, the color scheme seems to be adapted from EHealth color patterns. User experience is quite nice, visually smooth experience. The login page has a lack of feedback, which is something that could be improved upon. An example would be invalid logins, which give no prompt to the user.

1.2 GitHub

The github is clean, A lot of overhead folders seem to be required for Vue. One recommendation would be to keep the database login information off of the github. We made a similar mistake as well, but it's a good idea to keep it in mind.

2. Code Smells

Code smells were difficult to find, since we are new to Vue, we can't easily identify any overhead generated with the content. We did identify some long method spots and bloating, such as shown below where the code checkEmail, and checkPassword was directly inserted into the validate function. The refactor solution was already present, but nonetheless, the smell exists within the example function.

```
validate: function () {
  var email = document.getElementById('email').value
  var emErrorMsg = ''
  if (!(/^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/.test(email))) {
    emErrorMsg += 'You have entered an invalid email address <br> '
  }
  if (emErrorMsg.length > 0) {
    document.getElementById('em__msg').innerHTML = emErrorMsg
  } else {
    document.getElementById('em__msg').innerHTML = ''
  }
  // password
  var password = document.getElementById('password').value
  var pwErrorMsg = ''
  if (password.length < 8) {
    pwErrorMsg += 'You have to enter at least 8 characters <br> '
  }
  if (pwErrorMsg.length > 0) {
    document.getElementById('pw__msg').innerHTML = pwErrorMsg
  } else {
    document.getElementById('pw__msg').innerHTML = ''
  }
  if (emErrorMsg.length > 0 || pwErrorMsg > 0) {
    return false
  } else {
    return true
  }
},
```

```
    },
    checkEmail: function () {
      var email = document.getElementById('email').value
      var errorMsg = ''
      if (!(/^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/.test(email))) {
        errorMsg += 'You have entered an invalid email address'
      }
      if (errorMsg.length > 0) {
        document.getElementById('em__msg').innerHTML = errorMsg
      } else {
        document.getElementById('em__msg').innerHTML = ''
      }
    },
    checkPassword: function () {
      var password = document.getElementById('password').value
      var errorMsg = ''
      if (password.length < 8) {
        errorMsg += 'You have to enter at least 8 characters'
      }
      if (errorMsg.length > 0) {
        document.getElementById('pw__msg').innerHTML = errorMsg
      } else {
        document.getElementById('pw__msg').innerHTML = ''
      }
    }
  }
}
</script>
```

3. Refactoring

For the most part, their code is great, there isn't very much to refactor. Touching on the previous code snippet, the regex for the email could be stored in a static variable, as well as transitioning to full reliance on modular functions would be a strong start.

```
const {ApproverList} = require('../models')

module.exports = {
  async getAllApprovers (req, res) {
    try {
      const ApproverList = await ApproverList.findAll({
      })
      res.send(ApproverList)
    } catch (err){
      res.status(500).send({
          error: 'An error has occured while retrieving the Approver list'
      })
    }
  },
```

The code is clean and simple designed and variable names are easy to understand. Which makes the project very flexible for developers to make changes and maintain the software. We also discovered some abstraction designs during the code view such as ApproverList and controller. Most of the code are short and simple, that means fewer bugs.

```
if(error){
  switch(error.details[0].context.key){
    case 'first__name' :
      res.status(400).send({
        error: `The first name provided failed to match the following rules:
        <br>
        1. It must contain only the following characters: lower case, upper case
        <br>
        2. it must be at least 3 characters in length and not greater than 30 characters
        `
      })
      break
    case 'last__name':
    res.status(400).send({
      error: `The last name provided failed to match the following rules:
      <br>
      1. It must contain only the following characters: lower case, upper case
      <br>
      2. it must be at least 3 characters in length and not greater than 30 characters
      `
    })
      break
    case 'email':
      res.status(400).send({
          error: 'You must provide a valid email address'
      })
      break
```

Most of the temp variables are already replaced by queries. This has made the code more readable and easily extended in the future. Another good thing is that the functions have local variables utilizing passed parameters, by using this method they minimize the confusion and reduced unwanted consequences, such as spaghetti code and external side-effects within functions.

```
module.exports = (sequelize, DataTypes) => {
  const User = sequelize.define("User", {
    id:{
      type: DataTypes.INTEGER,
```

In conclusion, team ENSE master race really puts effort in this quality work. Some things to take into consideration when refactoring, or producing a mvp 1.2 would be to enhance abstraction, modulation, and keep change in mind by utilizing static variables. The team applied the technical knowledge learned from class to the project they made. However, not all of the functions applied those methods, there's still some minor tweaks to work on in future releases.