MODULE SELECTOR

Project 6: Module Selection Website

Client:

James Toner, Centre for Technology Enhanced Learning,
King's College London.

Team: Make Code Not War

Dawan Abdulla

Tahoor Ahmed

Petru (Armand) Bancila

Kaé Dupuy

Radhika Gorecha

Irid Kotoni

Adrian Kusiak

Hani Tawil

Maria Veneva

Introduction:

The objectives for the three groups of users encompassing core and additional features were to: Allow a standard user (student) to:

- Search for modules with keywords
- Refine the results through specific tag categories such as 'Careers' and 'Skills'
- Create an account and view a tutorial first time user logs into the website
- Be able to track modules and get recommendations based on tracked modules
- View more information about a module
- Build a course based on selected course by adding and removing modules optional modules
- View recommended modules

Additionally:

- Access the website from mobile and tablets
- Reset password via email if the user has forgotten their password
- Edit account details
- Deactivate account and provide feedback

Allow a moderator to:

- Do everything a user can do
- Add, edit and remove modules, degrees and tags
- Add data for tags, degrees and modules in bulk (by uploading a CSV file)
- View, download and remove feedback from users

Allow an administrator to:

- Do everything a moderator can do
- Change the access group of specific users to make them moderators

Outcome:

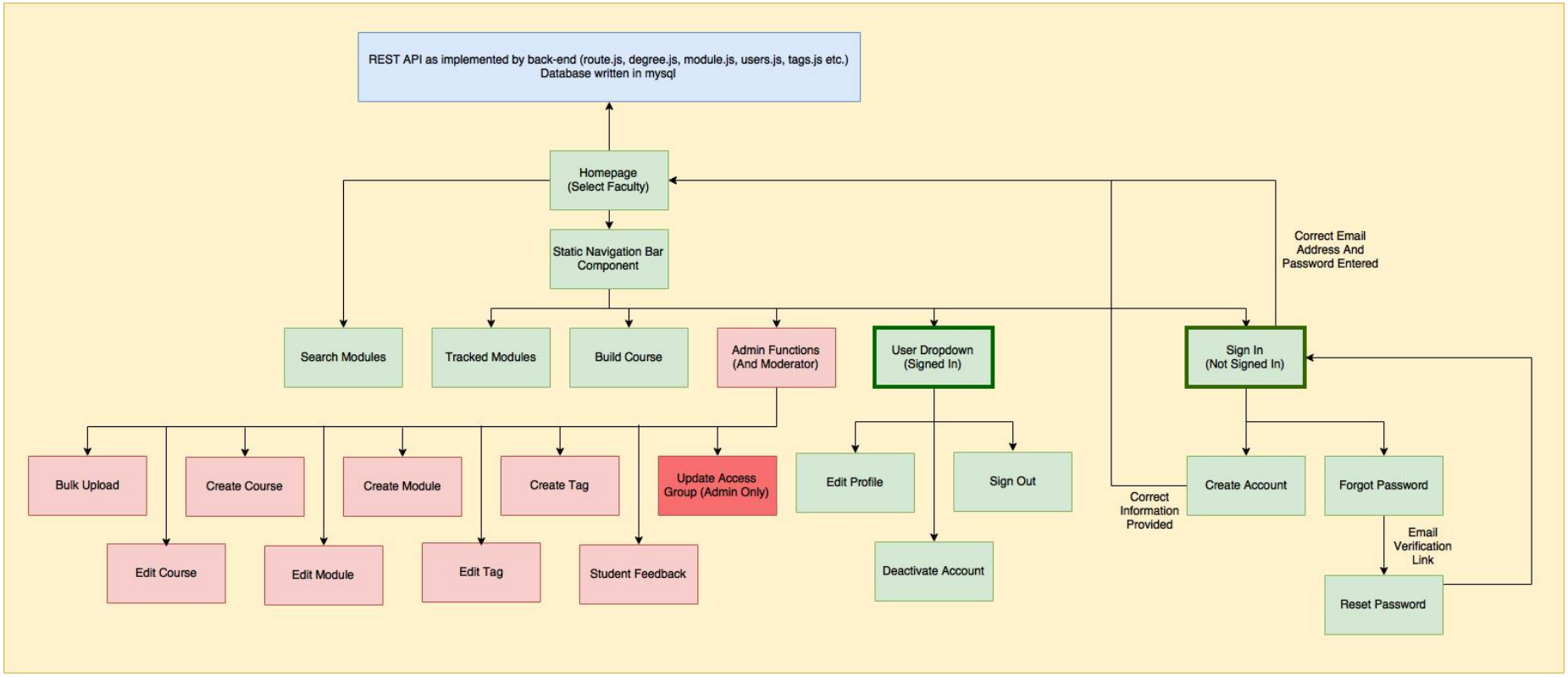
We have built an application called 'Module Selector' that acts as an information tool for students and potential students allowing them to search and track modules taught at King's College London with the aim to aid them in crafting a degree course that best fits their target criteria (e.g. career, skills learned). To fulfil our client's need of providing a simple representation of module choice in degree courses and being able to represent these choices with additional attributes to help a user, we decided to split this functionality into two clear cut features.

The first is a filterable search where all modules can be found based on module properties or Admin assigned tag words similar in fashion to many ecommerce catalogues filters, and then tracked as form of 'pinning' functionality. This will allow the administrator/client to attribute any additional properties to modules that they deem helpful, without limit, making it quite extensible. The second feature consists of a separate page allowing the user to both view which degrees fit best with their tracked selections and personalise the structure of those degrees by fitting optional modules around compulsory ones in a clear, colour coded fashion. Module dependencies and recommendations are interactively shown on user choice.

The application is a variant of the MEAN (MongoDB,Express, Angular, Node) stack web application utilising MySQL as a replacement of MongoDB. The variant is less common however allows more complex queries to be formed. This complemented our domain where the formation of the data to the returned (i.e. retrieving existing user constructed degrees with internal module-to-module dependencies etc.) was more important than the ease of IO of the data store that NoSQL would be prominent in given it works fluently with JSON objects. The stack allowed us to build an integrated web app where the server (Express module running on Node.js) provides both access to the model and serves the client-facing application statically,

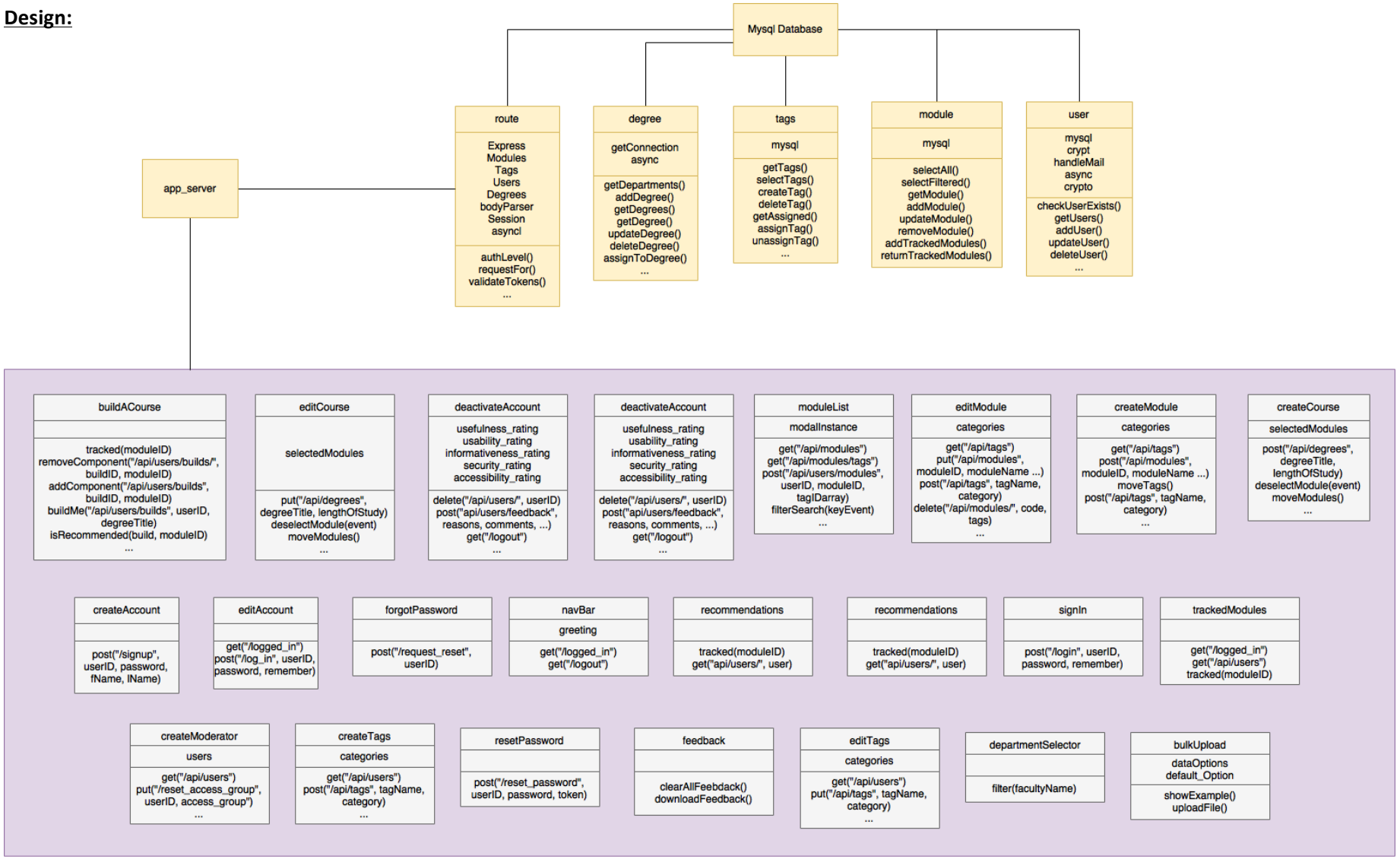
giving Angular full control over data representation. This means the server is quite separated from the client view and both can be maintained and extended independently.

The server uses the Passport.js module alongside express-MySQL-session to build on top of automatically created express session objects and allows us to attach client sessions to users where a corresponding session id is stored in a user agent's cookie. This serves as both an authentication method, allowing signup and login, and as a persistence tool allowing us to attribute the applications user-specific state to both a signed in user and a guest visitor. To increase a user's ease of use, a quick video tutorial for first-time users will appear after they have created an account. We have added an additional feature represented by a cog in the website which allows users to change the colour scheme of the website, with the benefit of visually impaired users in mind, who may not be able to distinguish between the different components on the website in a singular scheme.



The diagram above illustrates the structure of the module selection website. The main navigation is a static navigation bar component that can always be accessed by the user from any page within the application. The navigation bar updates based on the access group of the user who has logged in. All users are able to view all of the components coloured green. Moderators and administrators can access all of the components coloured pink. Only Administrators can visit the component coloured in red. The two components that have a dark green border cannot exist simultaneously so if a user is signed in they can use the dropdown to access more components alternatively if they are not signed in, clicking the sign in button on the navigation bar will take them to a sign in page where they can either enter their credentials or access links for create account and forgotten password.

Design:


Mysql Database
app_server
route
Express
Modules
Tags
Users
Degrees
bodyParser
Session
asyncl
authLevel()
requestFor()
validateTokens()
...
degree
getConnection
async
getDepartments()
addDegree()
getDegrees()
getDegree()
updateDegree()
deleteDegree()
assignToDegree()
...
tags
mysql
getTags()
selectTags()
createTag()
deleteTag()
getAssigned()
assignTag()
unassignTag()
...
module
mysql
selectAll()
selectFiltered()
getModule()
addModule()
updateModule()
removeModule()
addTrackedModules()
returnTrackedModules()
user
mysql
crypt
handleMail
async
crypto
checkUserExists()
getUsers()
addUser()
updateUser()
deleteUser()
...
buildACourse
tracked(moduleID)
removeComponent("/api/users/builds/", buildID, moduleID)
addComponent("/api/users/builds", buildID, moduleID)
buildMe("/api/users/builds", userID, degreeTitle)
isRecommended(build, moduleID)
...
editCourse
selectedModules
put("/api/degrees", degreeTitle, lengthOfStudy)
deselectModule(event)
moveModules()
...
deactivateAccount
usefulness_rating
usability_rating
informativeness_rating
security_rating
accessibility_rating
delete("/api/users/", userID)
post("api/users/feedback", reasons, comments, ...)
get("/logout")
...
deactivateAccount
usefulness_rating
usability_rating
informativeness_rating
security_rating
accessibility_rating
delete("/api/users/", userID)
post("api/users/feedback", reasons, comments, ...)
get("/logout")
...
moduleList
modalInstance
get("/api/modules")
get("/api/modules/tags")
post("/api/users/modules", userID, moduleID, tagIDarray)
filterSearch(keyEvent)
...
editModule
categories
get("/api/tags")
put("/api/modules", moduleID, moduleName ...)
post("/api/tags", tagName, category)
delete("/api/modules/", code, tags)
...
createModule
categories
get("/api/tags")
post("/api/modules", moduleID, moduleName ...)
moveTags()
post("/api/tags", tagName, category)
...
createCourse
selectedModules
post("/api/degrees", degreeTitle, lengthOfStudy)
deselectModule(event)
moveModules()
...
createAccount
post("/signup", userID, password, fName, lName)
editAccount
get("/logged_in")
post("/log_in", userID, password, remember)
forgotPassword
post("/request_reset", userID)
navBar
greeting
get("/logged_in")
get("/logout")
recommendations
tracked(moduleID)
get("api/users/", user)
recommendations
tracked(moduleID)
get("api/users/", user)
signIn
post("/login", userID, password, remember)
trackedModules
get("/logged_in")
get("/api/users")
tracked(moduleID)
createModerator
users
get("/api/users")
put("/reset_access_group", userID, access_group")
...
createTags
categories
get("/api/users")
post("/api/tags", tagName, category)
...
resetPassword
post("/reset_password", userID, password, token)
feedback
clearAllFeebdack()
downloadFeedback()
editTags
categories
get("/api/users")
put("/api/tags", tagName, category)
...
departmentSelector
filter(facultyName)
bulkUpload
dataOptions
default_Option
showExample()
uploadFile()

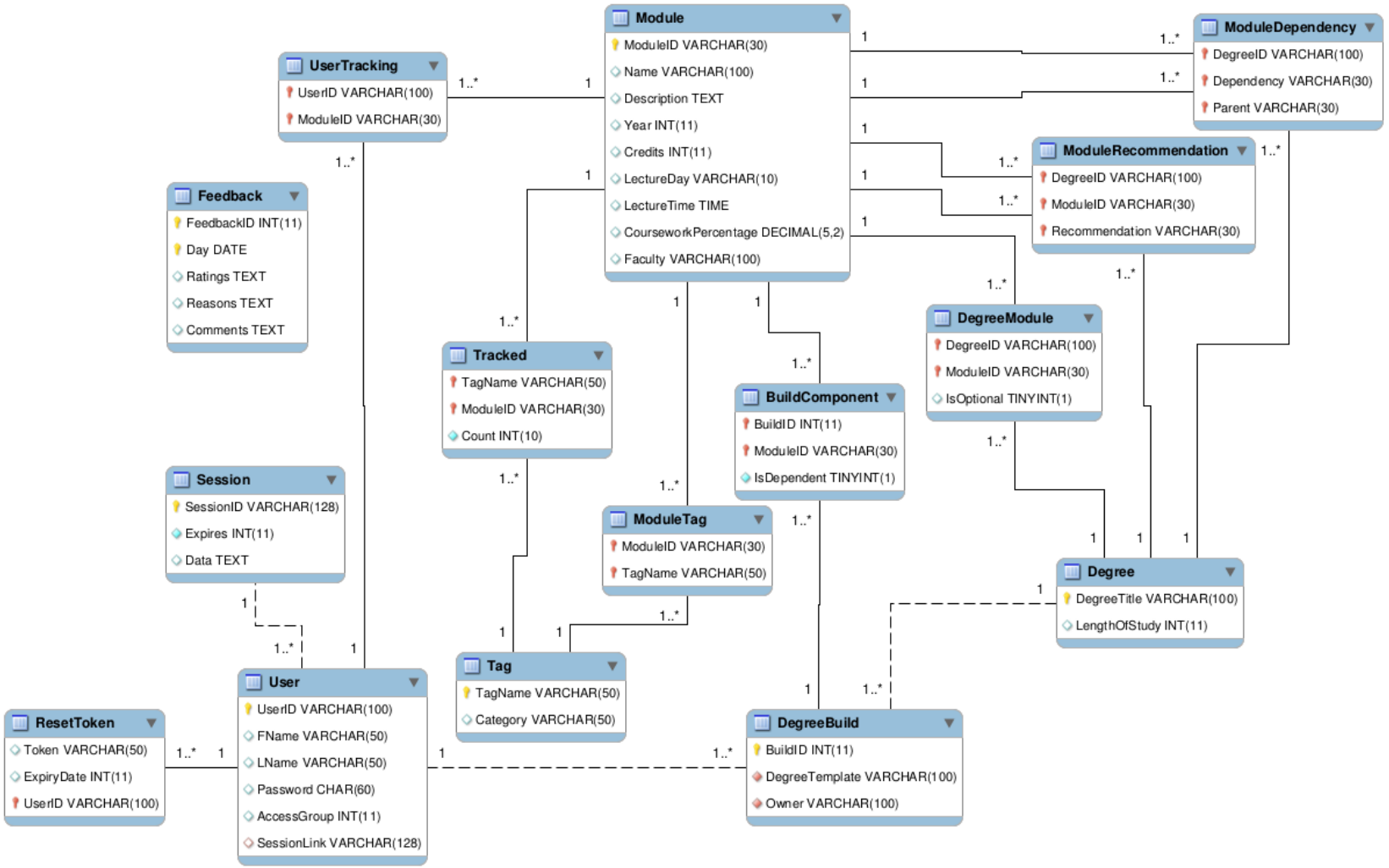
The UML class diagram on the previous page shows both the modules of the backend (server-side code written using node.js and express) and frontend (client-side code written using Angularjs modules with components that have their own html templates and controllers). The server side components are depicted in yellow and the client side JavaScript modules are in a large purple box. The purpose of the functions in these modules is to store, retrieve, update and delete information about various instances of entities that the application requires. The front-end implementation uses the architectural design pattern of Model View Controller. Each angular.js module contains a component, which has a visual part (the template) and a functional part (the controller) and always includes directives.

Server:
The structure of the server application took from a standard express application layout (MVC variant not feature separated). This mimics the Model-View-Controller delegation where the express server serves the front-side statically from a given public folder, (in our case called app) and is configured with HTTP routes from the route.js script acting as the controller interface listening for the views requests. The routes themselves delegate the data manipulation to functions defined in the model scripts (e.g. user.js, modules.js, degrees.js), which purposefully define functions to manipulate their respective resource in the MySQL database.

API:
The defined routes themselves are structured in a Hybrid RESTful design. Data manipulation functionality which can be directly translated to a resource are part of the '/api/' routes where all resources are provided with the standard CRUD methods (POST for create, GET for retrieve, PUT for update, and DELETE). Additional auxiliary routes are defined in non-REST format for ease of use to any front-end implementation, such as /login and /signup which manipulate both user data and client sessions. All routes utilise express-validator middleware to ensure proper bad request detection. Validation combined with the exception safety of the majority of the the middleware involved means the server is highly resistant to input error, and will continue to function correctly for other users even if an internal error occurs. It will log all errors to STDERR, fatal or not, for further review.

Client-Side
In our case of using a MEAN technology stack, since our front-end Angularjs application is not itself composed of just basic static html and css this implementation uses it's own pattern of Model View Controller to structure its functionality. Each angular.js module contains a component, which has a visual part (the html template) and a functional part (the javascript controller) intertwined with angular directives. We have a created a one-page website with a static navigation bar where we have chosen not to use standard anchor links that redirect to separate html documents. Instead, we have used deep linking between components. The components are added to the html file and managed by the file app.config.js, which dynamically changes the contents of the ng-view element whenever the user switches to a different section of the website. For example, in "http://www.moduleselection.com/#!/sign-in", the part "sign-in" changes the main view to the signIn(shown in the diagram) component.

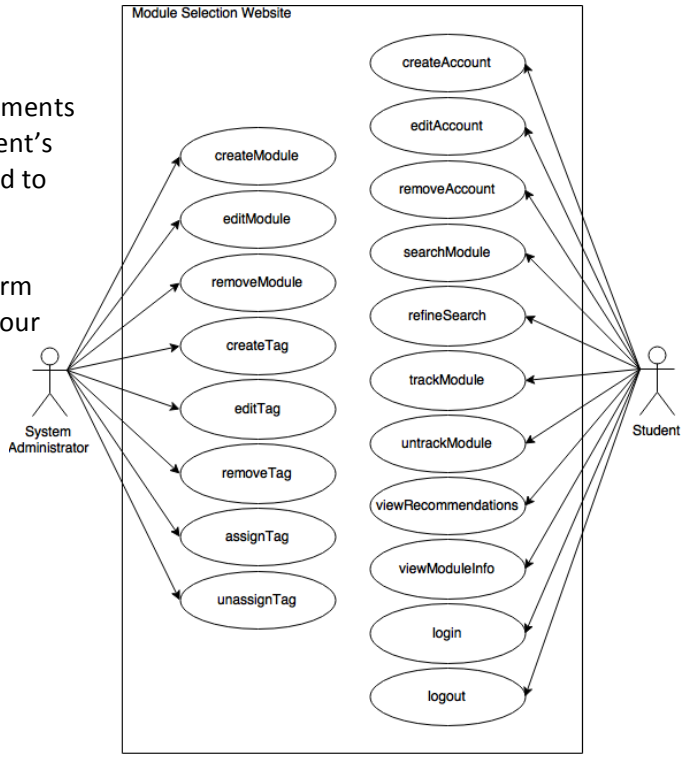
Database Schema:



Software Quality:

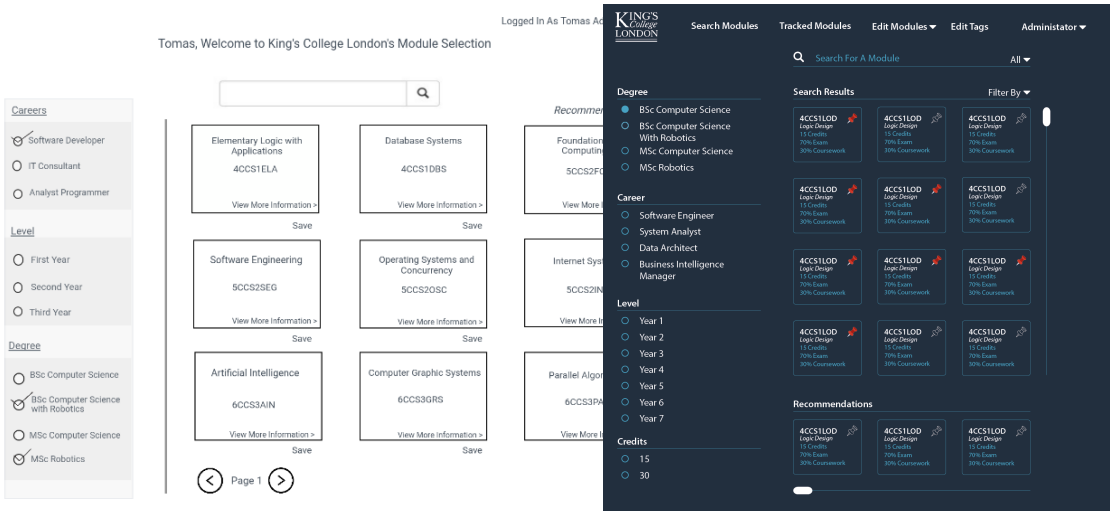
Requirements Elicitation:

At the start of the project we performed requirements elicitation by thoroughly reading through the client's requirements and prioritising which tasks needed to be implemented before other tasks and we also created an initial use-case diagram to depict the operations the administrator and user can perform using the system. The screenshot on the right is our use-case diagram that depicts the initial user requirements:



Design:

After the requirements analysis phase, some team members began designing the user interface of the website using the use-case diagram. Kaé created an initial design of the website which Tahoor took inspiration from and extended and began developing the designs for all of the webpages in the website. The web pages were designed to be very user-friendly and they were all consistent to ensure no confusion. We felt this was an important aspect to decide on early so we start coding the base of the application with a specific expectation of how it would be used. The screenshot below on the left is Kaé's design and the one on the right is Tahoor's design:



Implementation:

Our choice of the Kanban Agile approach also played a key role in ensuring that our application maintained its usability as it ensured our code base stayed highly organised in anticipation of requirement changes. We agreed to use an agile approach as it allows for this and we were prepared for the client making adjustments to his initial requirements given that they were, at first, quite open. Throughout the project the team engaged in various activities to complete the application to good standards, such as pair programming, and towards the end of the project the team held regular large coding sessions. This allowed team members to collaborate with one another and fix bugs together as a team.

Using an agile approach allowed us to select a handful of the client's requirements, start working on them until completion before taking on the next load of requirements. Furthermore, it allowed us to check the application would perform adequately in production by maintaining a prototype that we deployed on a heroku hosting dyno. As a result our application is highly configurable with environment variables, such as for specifying listening port to bind to, database connection and user information, and whether to load bulk data using a faster LOAD DATA INFILE statement for bulk uploads which is usually disabled by third party MySQL hosts, instead of INSERT. The live prototype also provided the benefit of the client having functional

access to latest integrated features at all times, allowing them to think of improvements or changes they would want or need.

Another reason why we decided to use Kanban to maintain code quality as helps improve and utilise the communication between team members to ensure everyone is keeping up with their assigned work as well as maintaining no redundancy in our task assignments. For example, if a team member's selected Trello card was in the 'Currently in implementation' phase for quite some time, other team members would visually recognise they might need some help and at the same time, user stories marked as under implementation will signify that they are being worked, ensuring no two people will implement the same features twice. Such restrictions increased our ability to maintain and merge code in our repository with practically no code being lost or damaged by repeated code of the same nature.

Testing:
Once a team member completed their implementation of functionality, they tested the functionality thoroughly to ensure it works well, using this principle allowed us to test our application whilst we were developing. This allowed us to easily mitigate bugs when they appeared. For example, in development of the server interface, all routes were tested using POSTMAN on an as-made basis. In addition, to ensure continuous high quality code even after refactoring, the backend team performed rigorous automated testing to ensure all API routes and functions work as intended. The team used the Mocha framework that allowed us to integrate the more thorough chai assertion library and sinon mocking library as a replacement for the default.

Chai-http plugin allowed us to perform integration tests on the routes in addition to the unit tests of both configuration files and model files. Tests were written for each function in each script to verify whether the correct error message appears and to check if data was correctly manipulated or returned. Istanbul provides a coverage report of the tests carried out and the backend team were able to achieve 100% branch coverage of all scripts in the server application folder as well as the database integrity confirmation script. The front-end team has performed unit testing using karma and e2e testing which is used to test the interaction between the user and the website. The screenshot below shows the coverage report generated for the server scripts:

100% Statements `1884/1884` **100%** Branches `683/683` **100%** Functions `270/270` **100%** Lines `1502/1502`

File	Statements		Branches		Functions		Lines	
MCNW-Module-Selector/	100%	75/75	100%	12/12	100%	4/4	100%	70/70
MCNW-Module-Selector/server/	100%	630/630	100%	279/279	100%	147/147	100%	579/579
MCNW-Module-Selector/server/config/	100%	153/153	100%	53/53	100%	21/21	100%	135/135
MCNW-Module-Selector/server/models/	100%	1026/1026	100%	339/339	100%	98/98	100%	718/718
routes.js	100%	630/630	100%	279/279	100%	147/147	100%	579/579
app_server.js	100%	25/25	100%	6/6	100%	0/0	100%	24/24
database_setup.js	100%	50/50	100%	6/6	100%	4/4	100%	46/46
connect_db.js	100%	51/51	100%	24/24	100%	8/8	100%	42/42
mail_config.js	100%	10/10	100%	8/8	100%	2/2	100%	10/10
passport.js	100%	37/37	100%	15/15	100%	6/6	100%	35/35
validation.js	100%	55/55	100%	6/6	100%	5/5	100%	48/48
degrees.js	100%	424/424	100%	112/112	100%	33/33	100%	270/270
modules.js	100%	274/274	100%	112/112	100%	24/24	100%	217/217
tags.js	100%	117/117	100%	46/46	100%	24/24	100%	93/93
user.js	100%	211/211	100%	69/69	100%	17/17	100%	138/138

Team Organisation:

At the start of the project we decided that we would split our team into three sub teams with one team attributed to the backend, frontend and integration respectively. In a traditional structure we would have assigned teams based on functionality, however, due to our limited experience with web development we felt such a split would be unnecessarily detrimental at that stage. Therefore the teams were split to focus learning on specific portions of our technology stack. As the entire stack is JavaScript centric, this ensured that during the end of the project, integration could be done smoothly with better understanding.

Subteam 1 consisted of Adrian, Kaé and Radhika. Subteam 1 was the backend team who were focused on integrating the backend components and the database. Sub team 2 consisted of Hani, Irid and Dawan. Subteam 2 was the first frontend team who were focused on creating and integrating certain web pages with a focus on API integration. Subteam 3 consisted of Tahoor, Maria and Petru. Subteam 3 was the second frontend team who were also focused on creating and integrating web pages with a focus on design. Below is a list of the team members and their roles in the project:

- Dawan Abdulla: front-end web developer
- Tahoor Ahmed: front-end web developer and web designer
- Petru (Armand) Bancila: front-end web developer
- Kaé Dupuy: back-end web developer, web designer and client liaison
- Radhika Gorecha: back-end web developer
- Irid Kotoni: front-end web developer
- Adrian Kusiak: back-end web developer and client liaison
- Hani Tawil: project leader and front-end developer
- Maria Veneva: front-end web developer

We used Trello as a tool for project management to ensure everyone in our team is up to date with his or her work. We did this by creating cards in Trello that represent use-cases, if a team member has implemented this functionality as prescribed on the Trello card, it will move from the 'Product backlog' state to the 'Currently in implementation' state. This allowed the team to view requirements more clearly and they were able to distinguish between tasks that had been completed and tasks that were still in product backlog. Team members would regularly select cards from the backlog and work on them until completion. The board also allowed easy modification to add specificity to requirements and improve their descriptions or priority the further along the project we got.

We also used Google Drive to manage team files in a team folder, this allowed files to be kept together and were accessible by every team member. Some files we included were the SQL schema, page designs and a team timetable file that was created at the start of the project and was used to ensure that every team member could attend meetings and coding sessions. We used Slack as our primary means of communication and we used it to create subgroups for the three subteams in our team, this allowed team members to easily communicate with one another. We also used Slack to share code and updated to-do lists of functionalities that still required implementation. If a team member was unable to attend a meeting, they could join the meeting using Slack and they would be informed of the latest team updates.

We used Google Calendar to arrange team meetings, client meetings and coding sessions as well as set reminders to team members that they are required to attend team meetings, client meetings and coding sessions. Also, in the Google Calendar, for the team meetings description, we added agenda's so the team knew what would be discussed in the following meeting. The team engaged in various activities to complete the application such as pair programming and towards the end of the project the team held regularly large coding sessions. This allowed team members to collaborate with one another and fix bugs together as a team. The team also had two group meetings a week, where one meeting was used to check in on the progress of the application and the other meeting was used to assign more tasks to subteams as well as discuss the client's feedback on the application if a meeting had taken place that week with the

client. All team members were present for these team meetings and coding sessions, if a team member was not present or was late to a meeting this was reported on Team Feedback. The website can be accessed through the URL module-selection.herokuapp.com. Admin login details are user ID: admin, password: multipass.

Other Information:

The screenshots below show how our team used Google Docs, Trello and Google Calendar to manage the teams resources and Slack to communicate with one another.