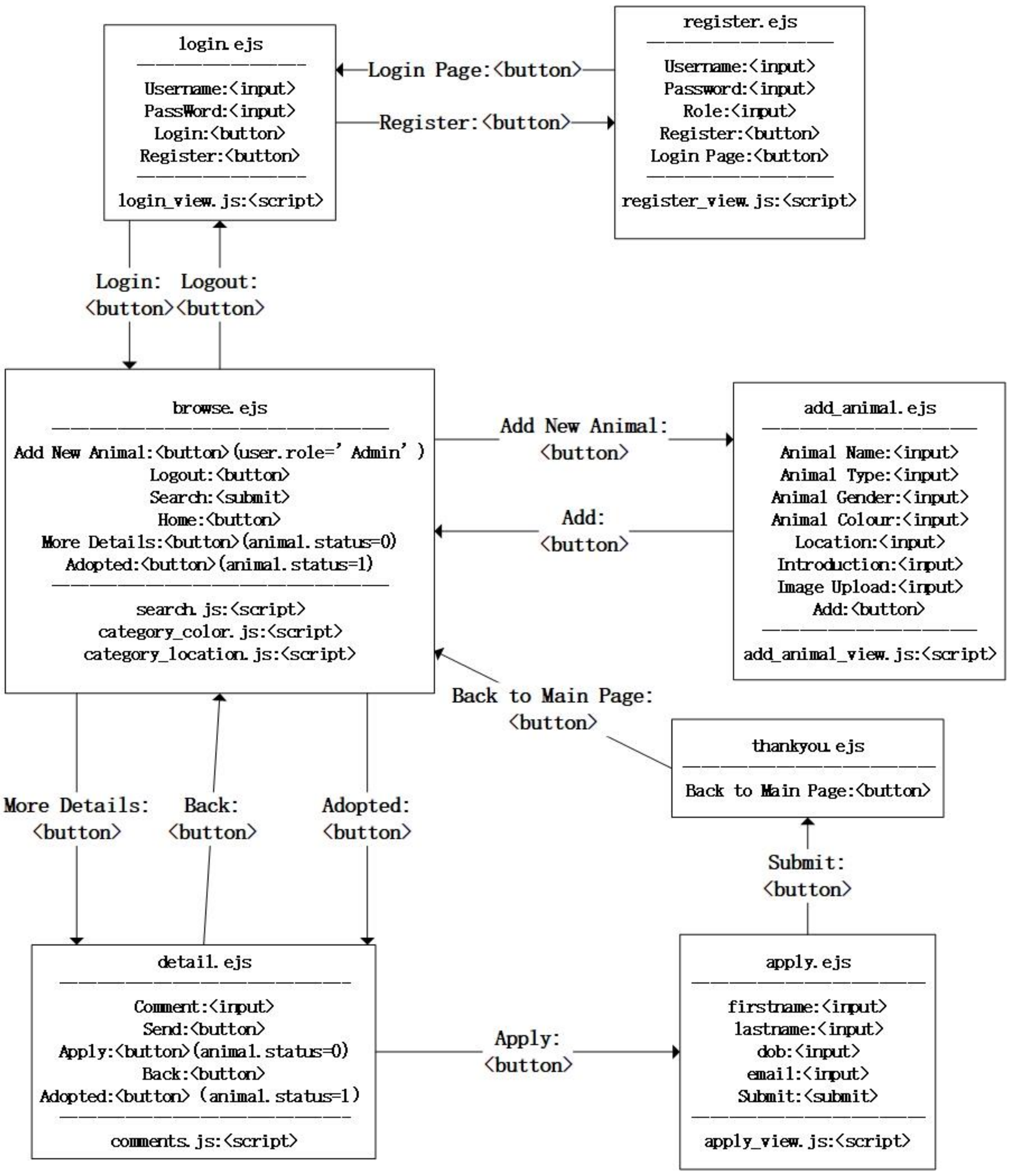
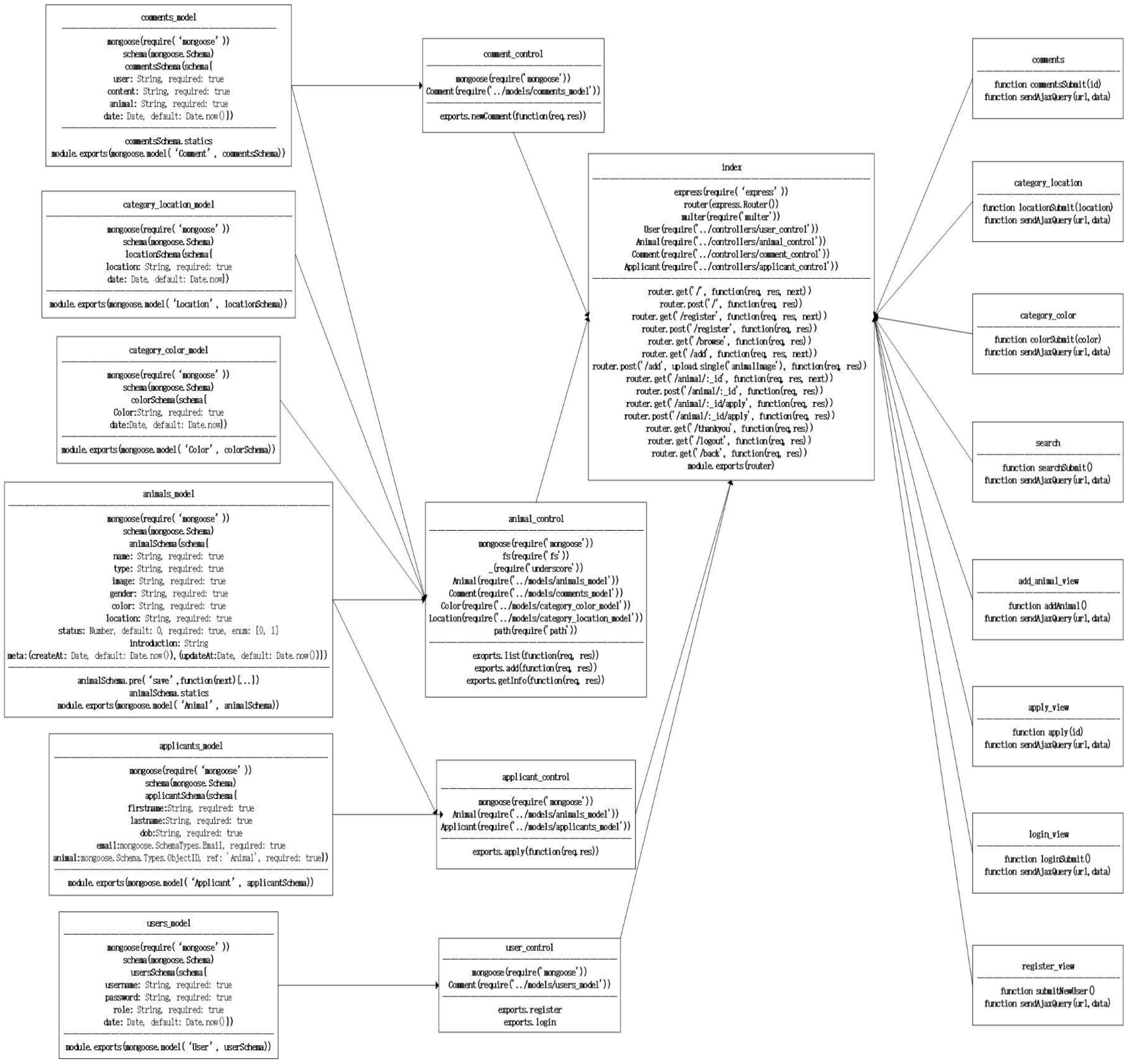
Animal Adoption Website Report

Group Members:
Cheng-Hsiang, Wang: 190199845
Jing He: 190244190
Fuxian Gu: 190254052

Introduction





```
comments_model
-----------------------------------------
mongoose(require( 'mongoose' ))
schema(mongoose.Schema)
commentsSchema(schema{
user: String, required: true
content: String, required: true
animal: String, required: true
date: Date, default: Date.now()})
-----------------------------------------
commentsSchema.statics
module.exports(mongoose.model( 'Comment' , commentsSchema))
```

```
category_location_model
-----------------------------------------
mongoose(require( 'mongoose' ))
schema(mongoose.Schema)
locationSchema(schema{
location: String, required: true
date: Date, default: Date.now})
-----------------------------------------
module.exports(mongoose.model( 'Location' , locationSchema))
```

```
category_color_model
-----------------------------------------
mongoose(require( 'mongoose' ))
schema(mongoose.Schema)
colorSchema(schema{
Color:String, required: true
date:Date, default: Date.now})
-----------------------------------------
module.exports(mongoose.model( 'Color' , colorSchema))
```

```
animals_model
-----------------------------------------
mongoose(require( 'mongoose' ))
schema(mongoose.Schema)
animalSchema(schema{
name: String, required: true
type: String, required: true
image: String, required: true
gender: String, required: true
color: String, required: true
location: String, required: true
status: Number, default: 0, required: true, enum: [0, 1]
introduction: String
meta:(createAt: Date, default: Date.now()), (updateAt:Date, default: Date.now()})
-----------------------------------------
animalSchema.pre( 'save' ,function(next){...})
animalSchema.statics
module.exports(mongoose.model( 'Animal' , animalSchema))
```

```
applicants_model
-----------------------------------------
mongoose(require( 'mongoose' ))
schema(mongoose.Schema)
applicantSchema(schema{
firstname:String, required: true
lastname:String, required: true
dob:String, required: true
email:mongoose.SchemaTypes.Email, required: true
animal:mongoose.Schema.Types.ObjectID, ref: 'Animal', required: true})
-----------------------------------------
module.exports(mongoose.model( 'Applicant' , applicantSchema))
```

```
users_model
-----------------------------------------
mongoose(require( 'mongoose' ))
schema(mongoose.Schema)
usersSchema(schema{
username: String, required: true
password: String, required: true
role: String, required: true
date: Date, default: Date.now()})
-----------------------------------------
module.exports(mongoose.model( 'User' , userSchema))
```

```
comment_control
-----------------------------------------
mongoose(require('mongoose'))
Comment(require('../models/comments_model'))
-----------------------------------------
exports.newComment(function(req.res))
```

```
animal_control
-----------------------------------------
mongoose(require('mongoose'))
fs(require('fs'))
_(require('underscore'))
Animal(require('../models/animals_model'))
Comment(require('../models/comments_model'))
Color(require('../models/category_color_model'))
Location(require('../models/category_location_model'))
path(require('path'))
-----------------------------------------
exoprts.list(function(req, res))
exports.add(function(req, res))
exports.getInfo(function(req, res))
```

```
applicant_control
-----------------------------------------
mongoose(require('mongoose'))
Animal(require('../models/animals_model'))
Applicant(require('../models/applicants_model'))
-----------------------------------------
exports.apply(function(req.res))
```

```
user_control
-----------------------------------------
mongoose(require('mongoose'))
Comment(require('../models/users_model'))
-----------------------------------------
exports.register
exports.login
```

```
index
-----------------------------------------
express(require( 'express' ))
router(express.Router())
multer(require('multer'))
User(require('../controllers/user_control'))
Animal(require('../controllers/animal_control'))
Comment(require('../controllers/comment_control'))
Applicant(require('../controllers/applicant_control'))
-----------------------------------------
router.get('/', function(req, res, next))
router.post('/', function(req, res))
router.get('/register', function(req, res, next))
router.post('/register', function(req, res))
router.get('/browse', function(req, res))
router.get('/add', function(req, res, next))
router.post('/add', upload.single('animalImage'), function(req, res))
router.get('/animal/:_id', function(req, res, next))
router.post('/animal/:_id', function(req, res))
router.get('/animal/:_id/apply', function(req, res))
router.post('/animal/:_id/apply', function(req, res))
router.get('/thankyou', function(req,res))
router.get('/logout', function(req, res))
router.get('/back', function(req, res))
module.exports(router)
```

```
comments
-----------------------------------------
function commentsSubmit(id)
function sendAjaxQuery(url,data)
```

```
category_location
-----------------------------------------
function locationSubmit(location)
function sendAjaxQuery(url,data)
```

```
category_color
-----------------------------------------
function colorSubmit(color)
function sendAjaxQuery(url,data)
```

```
search
-----------------------------------------
function searchSubmit()
function sendAjaxQuery(url,data)
```

```
add_animal_view
-----------------------------------------
function addAnimal()
function sendAjaxQuery(url,data)
```

```
apply_view
-----------------------------------------
function apply(id)
function sendAjaxQuery(url,data)
```

```
login_view
-----------------------------------------
function loginSubmit()
function sendAjaxQuery(url,data)
```

```
register_view
-----------------------------------------
function submitNewUser()
function sendAjaxQuery(url,data)
```

Task

- ### Registering/Sign in:
  Solution:

  When the user gets access to the system, first he/she would see the login page (**login.ejs**), provided by the server. If having no user account, the register button could be clicked to direct the user to the registration page (**register.ejs**). If the user has an account, he/she could enter the username and password, and click the login button. After the login button is clicked, **login_view.js** file would collect the input data from the user and send an ajax query to the server with the route **'/'** using the **'POST'** method. In **index.js**, the **router.post** method with the parameter **'/'** would be executed, which links to the **login function** in the controller file **'user_control'**. The validation of the login data would be carried out here by fetching the user data in the database and compare, after which the server would come up with a response of the user data if the validation is passed or any issue occurring during the process back to **login_view.js**. If successful, the user would be directed to the browsing page, otherwise, the error message would be printed and the route would not change.

  If the user has no account at first, the registration form should be completed and submitted. **register_view.js** would handle the user input and send the ajax query to the server with the route **'/register'** using the **'POST'** method. Getting the information from the query, **index.js** will decide which way the query should go, and the function **'register'** in the controller, **'user_control'** would be executed. What the function does is to check if the input data could be registered as a new user account, and return the message whether the registration succeeds. After the ajax response received, the message returned would be shown on the user interface. The route would not redirect by itself, so the user could decide to click the login page button and **get back to the login page** or **keep filling the registration form**. The benefit is that even when the registration succeeded, the user could keep registering for another user account, or get to the login page with just one further click.

  Requirements

  In the Registering and Sign-in section, we meet all the requirements in the assignment.

  (1) In the registration interface, two different account types can be registered (**Admin & General**). This registration function meets the requirement of registering in two different user roles in the Assignment. What's more, the information of registration is stored in the database table (**users**).

  (2) In the login interface, the registered users can enter their username and password to log into the browsing page and in the **browsing page**, they can click the **'Logout'** button when they want to logout.

  (3) If the users' accounts are admin accounts, after they login, they can see the **'Add New Animal'** button in the browsing page and click this button to add a new animal if they need. However, if the users' accounts are general accounts, they cannot see the 'Add New Animal' button which means they don't have permission to add animals.

  Issues and Limitations:

  The verification parts are simple and could be done more perfectly. When designing, we had been struggling about how to send the user data to the browsing page without using the redirect function on the server side, and finally we decided to use cookies. It may not be a perfect solution, but at least it solves our problem and successfully does its job.

Another thing could be made better for users is the role part in the registering page. Radio buttons could be used instead of letting users to type which increases the possibility of error.

- **Browsing/Searching for animals and displaying results:**
  Solution:

  After the user login successfully, the route would be changed to **'/browse'**, and the home page (**browse.ejs**) would be rendered. Here only for Admin users there would be an **'Add New Animal'** button, on which when the user clicks, he/she would be redirected to the add animal page. The home page also shows all the animals in the database in the beginning. If users want to search for a specific animal, they could type the name of the animal in the search bar and click search. The keyword would be sent to the **search.js** file and packed into a json type data to send to the server via ajax query. This is by using the **'GET'** method and to the url **'/browse'**. **Index.js** would call the **list function** in the **'animal_control'** file, and the function would search in the database to find animals whose name matched with the keyword using regular expression. Finally **stores the found animals and the keyword in cookies** and responds with a json formatted response to **search.js** file, which will redirect users to the new browsing page with searched results shown on the page and a string telling the users how many results has been found.

  There is also a filter function which could narrow down the shown animals regarding their colours or locations. When users click on any of the colour tags or the location tags, the **'category_color'** or the **'category_location'**js file would send the ajax query with json type data through the **'/browse'** route and the **list function** in **'animal_control.js'** file would find the matched results and send back a response. Like the search function, the results would be stored in cookies and the new browsing page would be rendered.

  At any time if users want to see all of the animals, they could click on the home button or the title and the cookie with the searched keyword and animals would be cleared, after which the browsing page with all the animals would be rendered.

  There is an image, a name and a button on each of the animals shown on the page. If the animal is not adopted yet, the word on the button would be **'details'** and the background colour would be **green**, otherwise the word would be **'adopted'** and the background colour would be **blue**. When users click on the button, the route would be redirected to the corresponding animal's animal page with details of the animal.

  Requirements:
  In Browsing/Searching for animals and displaying results section, we meet all the requirements in the assignment.

  (1) After the administrator adds new animal objects through the **admin** account, all users can use the search function in the browsing page. When users want to search for a specific animal, they need to enter the animal's name in the search bar and click the **'Search'** button, the result will be shown on the browsing page as a list. What's more, the tips for search results (**"<% total %>" result related to "<%= keyword %>"**) will be displayed below the search bar.

  (2) All users can see two tags (**Color tag & Location tag**) in the browsing page. When there is no data in the animal database (**animals**), these two tags have no content behind them. When the administrator adds the first new animal, the animal's colour (**animal.color**) and the animal's location (**animal.location**) will separately display behind the **Color tag** and **Location tag**. After this operation, when the administrator adds new animals which have the same colour or/and the same location as the first animal, the new animals will be assigned to the same tag(s). When the user clicks the tag name (**category_color.color /**

category_location.location), all of the same types (**animal.color / animal.location**) of the animals will be displayed on the browsing page.

(3) In the browsing page, whether the user is browsing all animals, searching specific animals or viewing the same tag of the animals, all animals are displayed in a list.

Issues and Limitations:

I am not sure whether using cookies to store the keyword and the found animals is a good way, but it gets our work done successfully. Maybe when the number of animals stored in the database rises to a threshold, we would need to find another way like sessions to deal with it. The search bar is only allowed to enter the name of the animals to search, rather than enter a series of terms to search.

- ### Applying to adopt an animal：

Solution:

In **'applicants_control.js'**, we first create two classes (**Applicant & Animal**) which are used to store the applicant's data and search animal's status data. In the **'.apply'** method of 'applicants_control.js', we create applicant's data (**applicantData**) and the id of the animal to be adopted (**id**), we also use a new applicant object (**newApplicant**) to represent a new applicant from **Applicant** class. All the **applicantData** attributes are stored in **newApplicant**. It is worth noting that the value of the **'animal'** in the applicant object represents the animal's id (**id**). Then we use the **'newApplicant.save'** method to save **newApplicant**. After finishing the save operation, we use the **'Animal.updateOne'** method and the animal id value (**id**) from **newAplicant** to update the adopted animal's status to '1', which means this animal is adopted. Finally, we use **'newApplicant._doc.toURL='/thankyou''** to remind **apply_view.js** to redirect to **thankyou.ejs** and return **newApplicant** data as JSON type.

In **'apply_view.js'**, we use the **'apply(id)'** method to collect the users' input (**firstname, lastname, dob, email**) from the form of the **apply.ejs**. What's more, the **'sendAjaxQuery(url, data)'** method is to change the data type of the form into JSON. In the **'apply(id)'** method, the name of url is **'/animal/'+id+'/apply'**.

In **'index.js'**, we use **router.get('/animal/'+id+'/apply, function(req, res))** to get the animal id and applicant's data from **apply.ejs**, then use **router.post** method to send animal id and applicant's data into **'.apply'** method in **'applicant_control.js'** in order to update the animal status and store applicant's data into applicants database. The reason why we choose this kind of design is that it can easily meet all the applying requirements in the assignment and what's more, it is easy to design.

Requirements:

In Applying to adopt an animal section, we meet all the requirements in the assignment.

(1) When the users find the animal they are interested in, they can click the **'More details'** button to see the animal's details in the animal page. If the users want to adopt the animal, they can click the **'Apply'** button then they will jump into the **apply page**. In the apply page, adopters need to enter their details and click the **'Submit'** button and now the animal is adopted (**animal.status=1**).

(2) After the users click the **'Submit'** button in the apply page, the applicant's database (**applicants**) will store their details.

(3) After the users click the **'Submit'** button on the apply page, the interface will jump into **thankyou page**. When users click the '**Back to Main Page**' button, the interface will return to the **browsing page**.

Issues and Limitations:

We didn't create another page for the adopted animal. When a user wants to see his/her adoption information, the website cannot offer the adoption information, the user needs to go back to the browsing page and search by himself/herself, which is unfriendly for the user. What's more, all the animals in the animal database will be shown in the browsing page, there is no interface module designed to display adopted and unadopted animals separately, which is also not user friendly. We have tried to hide the adopted animal on the browsing page but we find some bugs in the search function. We finally use **'More details'** and **'Adopted'** to distinguish between the adopted animal and unadopted animal.

- ## The animal page:

  Solution:
  When users click on the button for each animal in the browsing page, they would be directed to the animal page. To show the information of the animal, the **animal_control.js** file and **index.js** file is needed in the backend operation.

  Detail.ejs is the interface where the animal's information is listed, and **index.js**, which is the heart of the project, decides where the queries go and which function to use to handle them. If the method is '**GET**' (when users click the button linked to the animal page), the getInfo function in the '**animal_contro**l' file would be executed. Here we use **findById** function to find the corresponding animal's data from the database, and then use the fetch function declared in the **comments_model.js** file to get all the comments of the animal. After that the animal page with animal details and comments are rendered.

  If any user wants to adopt an animal, the '**apply now**' button is there linking to the applying page if the animal has not been adopted yet. Users could also click the back button to get back to where they click the button to this animal page, or they could click the title to get back to the home page with all the animals shown.

  Requirements
  Users can click the **'More details**' button or '**Adopted**' button to enter the animal page. In the animal page, users can see animal photos (**animal.image**) and animal information including animal name (**animal.name**), animal type (**animal.type**), animal gender (**animal.gender**), animal colour (**animal.color**), animal address (**animal.location**) and the introduction of the animal (**animal.introduction**).

  Issues and Limitations:

  Users can see the details of each animal (**name, type, gender, image, color, location, introduction**) and make comments in the animal page. What's more, in the animal page users can click the **'Apply'** button if the animal is unadopted. However, users can't search the similar animals in the animal page, we haven't created the tag or search function that has the ability to navigate to similar animals. We want to use the fuzzy query method of mongodb to achieve the function of similar animals' navigation, but we are still under learning, we did not add this function in our project.

  One of the most obvious limitations is that we can only show one animal photo in the animal page because when we design add new animal function, we have not set up to allow multiple image upload, which may not be user friendly enough.

- ## Creating a new page for an available animal:

  Solution:
  The value **'role'** from the cookie is judged, and if the user is an administrator (i.e. role='**Admin**'), a light blue button **'Add New Animal'** is generated on the top right corner of the browse.ejs page. By clicking this button, the data is transferred into the router. In the router, an **'upload'** object of the **'multer'** class is created. The **'upload'** object saves the image

of the animal in the path given and modifies the image name.

In the **‘.add’** method of **‘animal_control.js’**, a new **‘animal’** object is generated and inserted into the database's **‘Animal’** collection, which holds information about the animal that the user entered in the form, such as animal name, animal type, color, location, image and so on. For the status of the animal, the default value is set to 0, which means the animal has not been adopted. The **‘.add’** method also determines whether the animal's color and location exist in the **‘category_color’** and **‘category_location’** sets and if not, the new tag is saved. After the **‘animal’** object is saved in the database, the server sends the client an **‘animals’** object containing all animals' information in JSON format. Finally by using **'animal._doc.toURL = '/back''**, the page returns back to home page (browsing page with all the animals listed).

Requirements
In Creating a new page for an available animal section, we meet all the requirements in the assignment.

(1) If the user logs in the browsing page through the admin account and wants to add a new animal, they need to click the ‘**Add New Animal**’ button to jump into the add page. In the add page, users can see a form where they could enter the information about the animal they are adding.

(2) In the add page, user can enter the animal’s information including animal name (**animal.name**), animal type (**animal.type**), animal gender (**animal.gender**), animal colour (**animal.color**), animal address (**animal.location**) and the introduction of the animal (**animal.introduction**). Users can also insert animal photos (**animal.image**). After finishing entering the information of the animal and inserting the animal photos, the user needs to click the ‘**Add**’ button then the interface will jump into the **browsing page** and this new animal is available for every user.

(3) After adding operation, two tags (**Color tag** & **Location tag**) in the browsing page will show this animal colour and this animal address as a new tag name.

Issues and Limitations:
The verification of the textboxes was not done perfectly, merely making it impossible to be empty. The sex of the animal should be achieved using a radio box. When filling the form with the color and location, the existing data in the collection can be listed as radio boxes so that the user does not have to fill it in again.

- ### Commenting on an animal:

Solution:
The form is submitted in **'detail.ejs'**, the comment data is passed into **'comment_model.js'** and converted to json format. Data includes the user who sent the comment, comment content, the animal being commented, and comment date. The comment data is routed into the **'newComment'** method of **'comment_control.js'**. In this method, the new comment is added to the database. After sending the client with the comment content in JSON format, the comment is listed in the corresponding animal page.

Requirements
In Commenting on an animal section, we meet all the requirements in the assignment.

(1) Users can make their own comments on the animal page. At the bottom of the animal page, users can enter their words into the ‘**Comment-content**’ bar, after finishing entering, users need to click the ‘**Send**’ button to post their comment. After clicking the ‘**Send**’ button, the comment with username (**‘comment.user.username’: ‘comment.content’**) will be shown under the introduction of the animal.

(2) After the users click the ‘**Send**’ button, their comments will be stored in the comments database (**comments**). What’s more, The username of the user (**comment.user**) who makes this comment and the id of the commented animal (**comment.animal_id**) can be found in the comments database.

(3) All users can see all comments of each animal in each animal page.

Issues and Limitations:
Did not use other packages for creating a better comment panel, which allows users to modify their text style and use emoji when commenting. Only use an unordered list in the comments area to list the username and the corresponding comments, and the username will appear everytime the comments are sent even by the same user. Furthermore, the comments section does not show more details, such as the date of the comment.

In addition, when the user's cookie expires, if the user stays on the comments page, the user can continue to comment, except that the user name is changed to 'undefined', but these comments will not be saved in the database, so the next time you go to the same page, these comments will be lost. A better way is to prompt the user to log in again when the user's cookie expires.

- ## The Server Architecture:

Solution:
Several **'.js'** files are created to process the data from the view pages. First, after clicking the submit button of the view pages, the corresponding form data is passed through the **'onclick function'**. The form's data is serialized into an array, and after a call to **'sendAjaxQuery()'**, the array is converted to JSON data format and routed to the appropriate controller. In addition, passing data from database back to the client requires conversion to JSON format using **'JSON.stringify()'**.

Requirements:
The server implementation uses Express/NodeJS, with **'.ejs'** as the template, and uses JSON as the format for data exchange.

Issues and Limitations:
According to Sethiya (2018), the limitation of Express/NodeJS is single-threaded. Single-threaded can save resources and the use of computer memory (RAM). What’s more, because NodeJS operations are all asynchronous operations, multiple processes can occur in the same thread. If there is an error in one of the processes, the entire thread will crash and eventually, all the processes will fail to run because of the thread crash. Therefore, Express/NodeJS is suitable for projects where the code is not too logical.

- ## Database:

Solution:
In this project, we used the MongoDB database, and we created several models, including **'animals_model'**, **'applicants_model'**, **'category_color_model'**, **'category_location_model'**, **'comments_model'** and **'users_model'**. And we use a collection of databases by requiring a corresponding model.

Requirements:
Data storage needed for holding data related to different users and animals.

Issues and Limitations:
To achieve the function of displaying animals by different categories, we created

separate collections for the color and location of the animals. However, we did not establish a link between these categories collections and the animals' Id, which resulted in a reduction in efficiency. Besides, we wanted the adopted animals not to be shown at first, and this could be done by judging the adoption status in the **'browse.ejs'** page. However, the category of the adopted animals was still shown, and clicking on that category would not have the animal shown. Finally, we had to show animals that had been adopted but marked 'Adopted' to distinguish them.

- **Responsive web design:**

  Solution:

  In this project, Bootstrap's grid system is used to make web pages responsive. The login and registration pages are relatively simple. First, the elements are placed in the container, and then the container's **'margin-top'** and grid layout are adjusted to centre the display. In the **'browse.ejs'** page, the grid system plays an important role in the responsive layout. Code **'col-xs-12 col-sm-6 col-md-4 col-lg-2'** is used to adjust the number of animals displayed on different device pages.

  Requirements:

  On the **'login.ejs'**, **'register.ejs'**, **'add_animal.ejs'** and **'apply.ejs'** page, the form can stay in the middle of the page, regardless of the size of the page.
  On the **'browse.ejs'** page, when the display device is gradually reduced to a minimum, the number of animal modules displayed will change from 6 per row to 3 columns, 2 columns, and finally 1 column.

  Issues and Limitations:

  On the browse page, when the page shrinks to a minimum, the button on the right runs below the search bar, but in expectation, the button should float above the search bar. In general, the aspect ratio of animal pictures are different, and I have not yet made it possible for the picture to be centred vertically in each frame.

Conclusion

(1) In the **'Upload Image'** function, we can run the teacher's code in the lab project, but when we write the same code in our project or copy the code to our project, we cannot save the picture anyway. The error number is **'POST 500'**, but no exact error message is given. As a result, we wasted much time in this debug process, and we also used package capture software but did not find out the cause of the error. Later, we built a new node project and rewrote the code to upload pictures, and it was a success. We realized that the root directory of our project was different from that of the teacher's project, so we found **'modules.xml'** in the **'.idea'** directory and changed our root directory to make our image path consistent.

(2) Official documentation plays a significant role when using unfamiliar technologies such as **Bootstrap**. Some teaching websites, such as **'w3schools'** are also beneficial for us to get started as quickly as possible. Websites like **'stackoverflow''** can help us solve problems when we encounter bugs we have not seen before. There are a lot of people in this online community who have the same problems as we do, and there are many geeks who answer these questions.

Extra Information

For extra information, we used cookies to store the user's data, keyword and searched animals' data. However, because we set the cookies' survival time to an hour, the data stored would disappear after the period and we have not yet fulfilled the function to remind users to login again. So there might be some exceptions occurring after login for an hour.

In this project we fulfill the function to render different browsing page content for different roles. Only the admin type user could add new animals to the server. There could be more admin-only functions to extend like deleting animals or updating animals' information in the future.

Bibliography

Sethiya, S., (2018). Advantages & Disadvantages of Using @Nodejs [online]. *medium*[online]. 28 July 2018. [Viewed 22 May 2020]. Available from:https://medium.com/@shailendrasethiya/advantages-disadvantages-of-using-nodejs-a2ae640fc141

Lanfranchi, V. (2020) Lab Week10-MongoDB-file upload [PDF presentation]. COM6517, Web Technologies. 18 May. [Viewed 25 May] Available from: https://learn-eu-central-1-prod-fleet01-xythos.s3.eu-central-1.amazonaws.com/5c8f80ee07c44/7584730?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27COM3517%25206517%2520Lab%2520Week%252010%2520-%2520MongoDB%2520upload%2520file.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200529T120000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLYI3L4QWN%2F20200529%2Feu-central-1%2Fs3%2Faws4_request&X-Amz-Signature=bdabfa7e94d29f1e79d0afc1854b870dfc8d59a08abe504a100256e43d7ee6fc

Lanfranchi, V. (2020) Lab Week9-MongoDB-list-and-delete [PDF presentation]. COM6517, Web Technologies. 11 May. [Viewed 19 May] Available from: https://learn-eu-central-1-prod-fleet01-xythos.s3.eu-central-1.amazonaws.com/5c8f80ee07c44/7265830?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27COM3517%25206517%2520Lab%2520Week%25209%2520-%2520MongoDB%2520%2520list%2520and%2520delete.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200529T120000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLYI3L4QWN%2F20200529%2Feu-central-1%2Fs3%2Faws4_request&X-Amz-Signature=3a60beec09ad7a8ee2b31f8e30592f4bb025a09b1efa38c90d7881932c532922
Lanfranchi, V. (2020) Lab Week8-MongoDB [PDF presentation]. COM6517, Web Technologies. 4 May. [Viewed 4 May] Available from: https://learn-eu-central-1-prod-fleet01-xythos.s3.eu-central-1.amazonaws.com/5c8f80ee07c44/7153680?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27COM3517%25206517%2520Lab%2520Week%25208%2520-%2520MongoDB%25281%2529.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200529T120000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLYI3L4QWN%2F20200529%2Feu-central-1%2Fs3%2Faws4_request&X-Amz-Signature=afcf8f277aa41a65f878dd24b5fb947feccbc078db9fa118e47fd8a858c63cc4

Lanfranchi, V. (2020) Lecture Week8-MongoDB [PDF presentation]. COM6517, Web Technologies. 4 May. [Viewed 4 May] Available from: https://learn-eu-central-1-prod-fleet01-xythos.s3.eu-central-1.amazonaws.com/5c8f80ee07c44/7153679?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27COM3517-6517%2520Week%25208%2520-%2520MONGODB.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200529T120000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLYI3L4QWN%2F20200529%2Feu-central-1%2Fs3%2Faws4_request&X-Amz-Signature=1a9382271fca694fad28048652b2891d834b39b9bc1eacc0681b95dac172b058

Lanfranchi, V. (2020) Lab Week7-JSON and Nodejs [PDF presentation]. COM6517, Web Technologies. 27 April. [Viewed 27 April] Available from: https://learn-eu-central-1-prod-fleet01-xythos.s3.eu-central-1.amazonaws.com/5c8f80ee07c44/7086396?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27COM3517%25206517%2520Lab%2520Week%25207%2520-%2520JSON%2520and%2520NodeJS.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200529T120000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLYI3L4QWN%2F20200529%2Feu-central-1%2Fs3%2Faws4_request&X-Amz-Signature=886e883b28365e2d6f982fca704f67da356dcdee951720131377f8c3d60652b9

Lanfranchi, V. (2020) Lecture Week7-JSON and Nodejs [PDF presentation]. COM6517, Web Technologies. 27 April. [Viewed 27 April] Available from: https://learn-eu-central-1-prod-fleet01-xythos.s3.eu-central-1.amazonaws.com/5c8f80ee07c44/7086392?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27COM3517%25206517%2520Week%25207%2520-%2520JSON%2520and%2520NodeJS.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200529T120000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAZH6WM4PLYI3L4QWN%2F20200529%2Feu-central-1%2Fs3%2Faws4_request&X-Amz-Signature=9a71cb9e5140b3e7255800ba30eae0b93eb53fa82f8d90ebd42d8d8152ac2fbd