SW Engineering CSC 648/848 Fall 2019

CATDOG

Team Number 5

Tahar Touati - ttouati@mail.sfsu.edu

Amir Anjomshoaa - Aanjomshoaa@mail.sfsu.edu

Ivan Briseno - ibriseno@mail.sfsu.edu

Melissa Estrada - mestrada7@mail.sfsu.edu

Xiaopeng Rong - xrong@mail.sfsu.edu

"Milestone 4"

12/19/2019

History Table	
Version	Submission date
Milestone 1 V 1.0	10/02/2019
Milestone 1 V 2.0	10/30/2019
Milestone 2 V 1.0	10/30/2019
Milestone 2 V 2.0	11/18/2019
Milestone 3 V 1.0	12/03/2019
Milestone 4 V 1.0	12/5/2019
Milestone 4 V 2.0	12/19/2019

Table of Contents

1- Product Summary 3

2- Usability Test Plan 5

3- QA Test Plan 7

4- Code Review 9

5- Self-Check: Adherence to Original Non-Functional Specs 15

Product Summary

Product Name: CatDog

Product Url: http://34.67.160.125/

List of Priority 1 functional requirements:

1. Guests shall be able to sign up and create an account on the website:
   1.1. Create account as worker.
   1.2. Create account as Client.
2. Guests shall be able to send message to support team admin.
3. Guests shall be able to access a demo tour of the website.
4. Guests shall be able to skip the demo.
5. User shall be able to sign into the website.
6. User shall be able to get support from admin.
7. Clients shall be able to create Pet profiles.
8. Clients shall be able to create posts.
9. Client should include the date period for the post
10. Clients shall be able to create Pet profile in sign up.

<u>All the registered users:</u>

11. User shall be able to log out from the website.
12. User shall be able to get support from admin.
13. User shall be able to change his/her password.
14. Clients shall be able to see who booked the post.
15. Clients shall be able to use search bar:
    Filter workers by booking number.
    Filter workers by username.
16. Workers shall be able to accept to book a client's post.
17. Workers shall be able to use search bar to search for available jobs in location
18. Worker shall be able to see the available post to book.
19. Administrators shall be able to sign in to the website.
20. Administrators shall be able to log out from the website.
20. Administrators shall be able to delete/ban user.

Product Summary:

CatDog is a website that provides pet sitting by accredited sitters. Our website is targeted to two types of users, the first type are pet owners who frequently travel and need a service that provides pet care during their travel duration, and the second type are accredited sitters and pet-friendly people wanting to earn extra money while taking care of a pet. This is the main focus of our market for catdog.

Users with a client account will be able to create pet profiles for their pets, create a post that seeks a pet sitting service from workers by notifying them and includes the date period and location for the service, select a sitter from those who booked his post and finally confirm the booking.

Users with a sitter account will be able to search clients by location, book a client's post from the available posts or from the post feed, get notified when a client accepts them.

This is the URL of our website: http://34.67.160.125/

Usability Test Plan

Test Objectives:

The feature being tested is for the client side of our website. The feature is going to let the client to be able to create a post. The feature will be tested with making sure of the effectiveness and accuracy due to being one of the main functionalities of our website. The test is to make sure the Client side of our website is able to create a post for their pet. This process contains updating the database so the Worker side of our website is able to see the post created by the client moments after.

Test Description:

- **System Setup:**
  The user will be provided a laptop with windows 10 OS running on it, and also with a working internet connection and a chrome installed on it with the version 78.0.39 (latest version for windows).
  We also going to guide the user through the sign-up process as a client so the user is able to test the functionality on their own profile.
- **Starting Point:**
  After making sure user has a working account we could start the test by the login page which would be /login which would redirect the user after login into the page /clientprof?user=[].
- **Intended Users:**
  For this test we are looking for people who never used our website, and also are cat or dog pet owners who are adapted to the lifestyle that would cause them to leave the city often a lot and can't take the responsibility of taking their pet with them on the trips. This is group of people is our main focus of marketing team.
- **Url to be tested and measured:**
  The first Url user would see is the /clientprof?user=[] which is query string user=[username] and this would allow the system to make sure with the cookie and authenticate the user to be on that page. And every api call is communicating with the back-end through the request body. And use satisfaction would be measured at last by the Likert scale questionnaire.

Usability Task Description:

- User first needs to sign up with the help of us as a client in our website.
- After the user creates the account we would redirect them to login page and from their they should be able to navigate forward
- After the user logs into the website it would be redirected to the profile page and from their the tab for creating a post is visible to them and easy to navigate.
- And lastly the form is easy to use to fill up and choose for what pet they are choosing and what range of date.

Questionnaire:

- The post creating feature was easy to find and navigate to (Circle one):

Strongly Disagree Disagree Neutral Agree Strongly Agree

- The form for creating the post was simple and easy to fill out (Circle one):

Strongly Disagree Disagree Neutral Agree Strongly Agree

- Choosing a date and time for the post was straight forward (Circle one):

Strongly Disagree Disagree Neutral Agree Strongly Agree

- I am satisfied with the post creation feature (Circle one):

Strongly Disagree Disagree Neutral Agree Strongly Agree

QA Test Plan

Test Objectives:
The objective of the test is to explore the possibilities offered by the post feature, try all the error cases and make sure that the posts gets published in the workers posts page when posted, and also in the active posts when they are booked by a worker.

Hardware and Software setup:
Hardware Setup: a laptop connected to the internet connection and running windows 10.
Software Setup: google chrome installed and open the website's signup page which URL is http://34.67.160.125/Signup

Features to be tested:
Client creating a post and publishing the post for the workers to request to book.

QA Test Table:
For Chrome and Firefox:

#	Test	Test Description	Input	Expected output	PASS/ FAIL
1	The Calendar Date Range	Client X creates a post with a starting date of yesterday.	Start date: 12/4/2019 End date: 12/09/2019	Display an error with a message "please check the dates of your post"	FAIL
2	Post visibility in the client page as a the post gets created inside the tab pending post.	Client X creates a valid post with correct pet name and date range and time for pickUp	petName: [userInput] Date Start: [userInput] Date End: [userInput] Location: [userInput	The post appears inside the clients page in the pending post tab.	PASS

3	Post being request to be booked by worker	Worker Y requests to book the post that was created by the client X	Clicking on the booking button accept on the worker profile page	Client receives an email that his post was requested to booked and the posts shows up in the request post page on the client profile	FAIL

Code Review

Other team members reviewd my code for my routes which contains my api routes and post routes:

I received comments in my code with his name next to it:

```
var express = require('express');



const router = express()

const session = require('express-session')
const search = require('./data/db').search

const client = require('./data/clientDb')



const addSub = require('./data/db').addSub

const dbConnect = require('./data/db').connect
const dbDisconnect = require('./data/db').disconnect

const bcrypt = require('bcryptjs')

const uuid = require('uuid/v4')

//Jose Castanon -- more descriptions as to what each route does would be nice.

dbConnect()

const auth = (req,res,next) => {
   console.log(req.session)
   if(req.session.loggedIn){
       next()
   }
   //res.status(403).send("Access not allowed")
}



router.get('/', function(req, res, next) {

   req.session.loggedIn = true
   res.render('index', { title: uniqueId  });
 });
router.get('/test', auth ,(req,res) => {
   console.log(req.session.loggedIn)
   res.send('Hi')
})
```

```
router.post('/db/search', (req,res) => {
   console.log(req.body.searchValue)
   search(req.body.searchValue).then((result)=>{
       res.json(result)
   }).catch((e) => {
       console.log(e)
   })
})

router.post('/db/addSub', (req,res) => {
   console.log(req.body)
   addSub(req.body.firstName,req.body.lastName,req.body.email).then((result) => {
       res.send(result)
   }).catch((e) => {
       res.send("Already added")
   })

})

router.post('/db/addClient', async (req,res) => {




   console.log(req.body)

   try{
   //Jose Castanon -- request body can be passed to add client, that way you can have
less variables and cleaner code.
   const user = req.body.userName;
   const firstName = req.body.firstName
   const lastName = req.body.lastName
   const street = req.body.street
   const city = req.body.city
   const zipCode = req.body.zipCode
   const email = req.body.email
   const petQuantity = 0
   const petId = 0

   const password = req.body.password

   const hashPassword = await bcrypt.hash(password, 3)

   console.log(hashPassword)

   client.addClient(user,firstName, lastName,street,
city,zipCode,email,hashPassword).then((result)=>{

       req.session.loggedIn = true;
       console.log(req.session)

       res.status(201).send("Created")
   }).catch((e) => {
```

```
        res.send(e)
    })

    }catch(error){

         console.log(error)

    }

})

router.post('/db/addPet', auth ,async(req,res) => {
    try{
        const result = await
client.addPet(req.body.user,req.body.name,req.body.description)
        res.status(201).send('All good')
    }catch(error){
        res.status(200).send('User Taken')
//Jose Castanon -- errors could be more descriptive. Error says user taken on an add pet
route. Does this mean that the add pet route also checks if the user has been registered
already?
    }

})

router.post('/login', async (req,res) => {
    try{
    const user = req.body.user
    const password = req.body.password



    // const isMatch = await bcrypt.compare(password)
    client.getPassword(user).then(async (result) => {
        //console.log
        const hashPassword = result;
        const isMatch = await bcrypt.compare(password,hashPassword)

        if(isMatch){
            console.log("Matched")
        }else{
            res.json({error: "Incorrect Password"})
        }
    }).catch((e) => {
            res.status(406).send('Username Already Taken') //Jose Castanon  -- username
check should be done when registering a user
    })

    }catch(error){
        console.log(error)
    }



 module.exports = router;
```

Internal Team Member Melissa Code review:

The review is done by Amir on Melissa's code for the clientProfile page

The code is avaliable on the github branch in
"/application/client/component/creatingClient/clientprof.js"

Too big to post here the whole code.

Comments for her code from Amir:

/*

- There are some packages being imported into the project which are not being used for best practice I think its better to take them off.
- Some comments available inside the code makes the just for the back-end team easier to navigate through your code for fixing the states of our website.
- Overall great code and usage of css and thank you for making our website look great Melissa

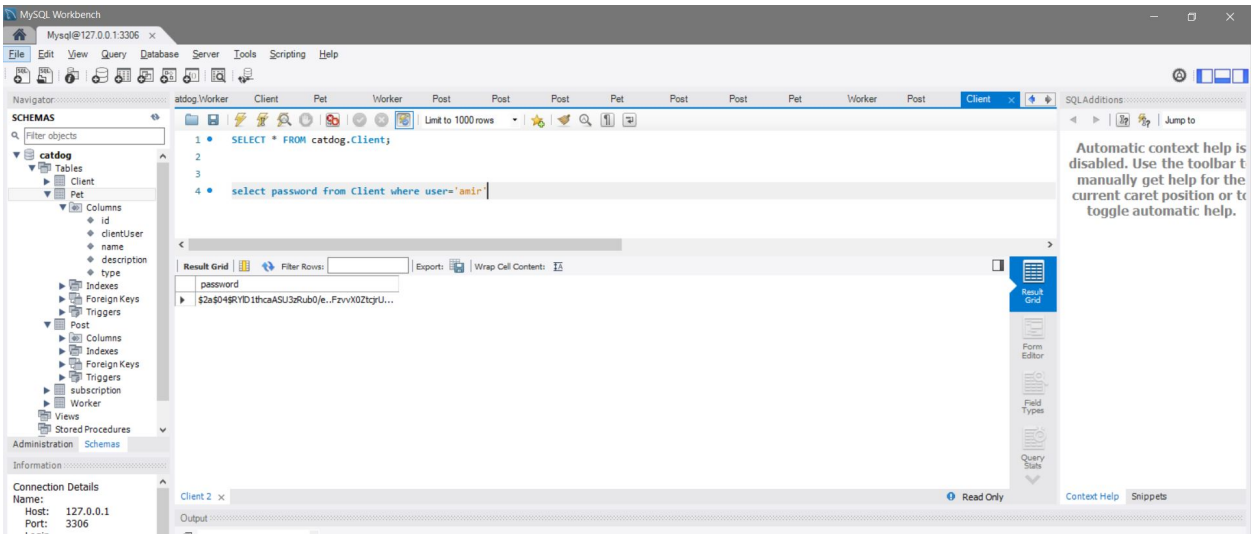
***/**

Self-Check on Best Practice for Security

The assets being proctected in our websites are:

- passwords are encrypted and also salted when placed inside the database.
- Search bar inputs are being secured for any SQL injection.
- Client and profile pages are secured and also visibile to the user with correct username and password.
- Data is being communicated to the back-end with the help of proxy for more security

Proof for salting and encrypting password:




```
const password = req.body.password

const hashPassword = await bcrypt.hash(password, 3)
console.log(hashPassword)
```

Figure 1 - Password encryption

```
client.addClient(user,firstName, lastName,street, city,zipCode,email,hashPassword).
```

Figure 2 – Adding the Hashed password

```
router.post('/login', async (req,res) => {
    console.log(req.body)

    const user = req.body.user
    const password = req.body.password
    const type = req.body.type
    authjs.getPassword(user,type).then(async(result) =>{
        const hashPassword = result;
        const isMatch = await bcrypt.compare(password,hashPassword)
        if(isMatch) {
            req.session.loggedIn = true;
            req.session.user = user;
            req.session.type = type;
            res.status(201).send("Ok")
        }
        else {
            req.session.destroy()
            res.send('User/Password Incorrect')
        }
```

Figure 3 – Checking the Hashed password

Data Validation:

For the forms validation we are implementing errors checkers for such as correct email was input or the user does not the each other primary user name key. Also for the search we're validating the input depending on the side of the service. If the client is searching for a worker, they are only allowed to search by booking # or username of the worker. Bookings are only numbers so if the user input is only number gets validated for the booking id and if it contains letter it would be username. The search for client is designed to not search for any special character in their input string. This is also being offered to the worker side of our website for safety of our database and the user information

- **Password is being validated and hashed**
- **Search bar input is being validated for any mysql injection and detecting ;**

Self-Check: Adherence to Original Non-Functional Specs

Copy all original non-functional specs as in high level application document published at the very beginning of the class. Then for each say either:

DONE if it is done;

ON TRACK if it is in the process of being done and you are sure it will be completed on time;

or ISSUE meaning you have some problems and then

Non- functional Requirements	Status
Functionality	
1. The site should be developed and deployed using the stack tools and servers that was approved by the Class CTO.	Done
2. Each WWW page needs to have a functional navbar and search bar with the logo included at the top of the page.	Done
3. Guest user should be able to see the functionality of the website in a demo before signing up.	Done
4. Application shall be very easy to use and intuitive	Done
5. Application shall be hosted and deployed on Gcloud client server as specified in M0.	Done
6. Application shall be optimized for standard desktop/laptop browsers. All users should be able to contact support.	Done
7. Client should be able to contact Admin in case of emergency.	Done
Compatibility:	
8. Site should have a logo to be displayed next to the title in every browser.	Done

9. The site shall be compatible with the latest version of Chrome browser (77.0.3865.90)	Done
10. The site shall be compatible with the latest version of Firefox browser (68.0.1)	Done
11. The site shall be compatible with the latest version of Safari browser (5.1.7)	Done
Performance	
12. Loading time for site shall be less than 10 seconds for any page.	Done
13. Search bar result should be shown in less than 10 seconds.	Done
Security	
14. Registered user should be able to login with the credentials it made during sign up.	Done
15. Client should input his/her name, address and driver's licence # to be collected as data.	Done
16. Client's pet needs to be California licensed and the ID # needs to be inputted into the data. Data.	Done
17. Guest user should not be able to see any live data before signing up.	Done
18. Registered user's password should be encrypted and saved in the database.	Done
19. ` Worker should input his name, address, and driver's licence #, and pet sitting certification to be collected as le to change their password if needed.	Done
Coding Standards	
20. Admin should be able to ban workers and clients if needed	Done
21. Developer should develop code to be easy to read.	Done
22. Methods/functions should have one and only one purpose.	Done
23. Developer should be commenting on the specific methods on his/her code.	Done
24. Developer should be consistent in the naming convention of variables.	Done
25. Back-End developer should develop noncomplex api.	Done
26. Developer should write test cases for their functions.	Done
Layout	Done
27. Website should be professional looking	Done
28. Website should be user-friendly	Done
Data Integrity & Capacity & Reliability	
29. User's data shall be inputted into MYSQL database.	Done
30. Data in the data base should be back up every week.	Done
31. The server storage shouldn't exceed 80%	Done
32. Inform the users if the website is going to be down for maintenance.	Done
33. If the server is down, it should be restarted and back on running in less than 1 hour in a month.	Done
34. Pay functionality shall not be implemented.	Done