ENDEAVOUR STUDIOS

Cat Logistic Appointment and Work Scheduler (CLAWS)

S2c - Technical Design Document

SENG 321 - Group 4
March 15, 2012

Author	Contribution
Jared Griffis	Executive Summary, Overview, Management Plan, Editing
Jordan Smythe	Users
Jeremy Moseley	Appointments, Editing
Michael Pattie	Employee Shift Scheduling, General Scheduling
Wes Alcock	Appointments
Braden Simpson	Cat-a-logue, Test Plan
Theo Prentice	Helped with Test Plan

Executive Summary

The structure of CLAWS can be broken down into three major components, or applications: scheduling, catalogue, and users. Each relies on a main “manager” class that provides most of the functionality of each app.

The scheduling app is the largest of the three. It contains classes that represent all the event types that will be used, namely employee shifts, grooming appointments, and kenneling appointments. The scheduling app is run by the ScheduleManager, which keeps track of the overall schedule. The Scheduler manager can be queried to check if a new appointment can be fit into the schedule, and it can also add or remove events from the schedule. Combining the three types of events into one schedule is necessary because of interactions between them; the number of grooming appointments allowed on a single day depends on the number of staff present on that day, for example. The scheduling app provides all the views associated with customer or employee appointment management and employee scheduling, as well as the Catcam and logs associated with specific appointments.

The users app controls user accounts and access. Though Django handles much of the work already with its User class, the a UserManager is necessary to handle the specifics of CLAWS. In particular, customer accounts will not be active until activated by an employee. User accounts will have their permissions set depending on their type to control access to certain pages.

The catalogue app is only responsible for the Cat-a-logue. The Item class represents a single item in the catalogue. The CatalogueManager can add or remove items from the catalogue.

Testing of CLAWS components will consist of unit, built, functional, integration, and acceptance tests. Unit tests will be developed as part of the coding process, and run before commits and on integration. Build tests will be run nightly on multiple environments to ensure that the project is still functioning correctly. Functional tests will be conducted by members responsible for a module, and will cover common use cases. Integration tests will be performed on dates listed in the test plan to verify that modules are interacting properly. Performance tests will also be conducted along with integration tests to measure system conformance. Finally, an acceptance test will be performed for Purrfur Cat Grooming and Kennelling. Bugs will be reported and tracked using a custom Google Form.

Table of Contents

1.0 Overview

The functionality of CLAWS can be decomposed into three parts. The first, scheduling, manages cat appointments and the employee schedule. The second, the catalogue, handles adding and editing of items in the catalogue. The third, users, is responsible for logging users in and determining their permissions. Each of these modules fits into the Django framework as individual applications, or “apps.”

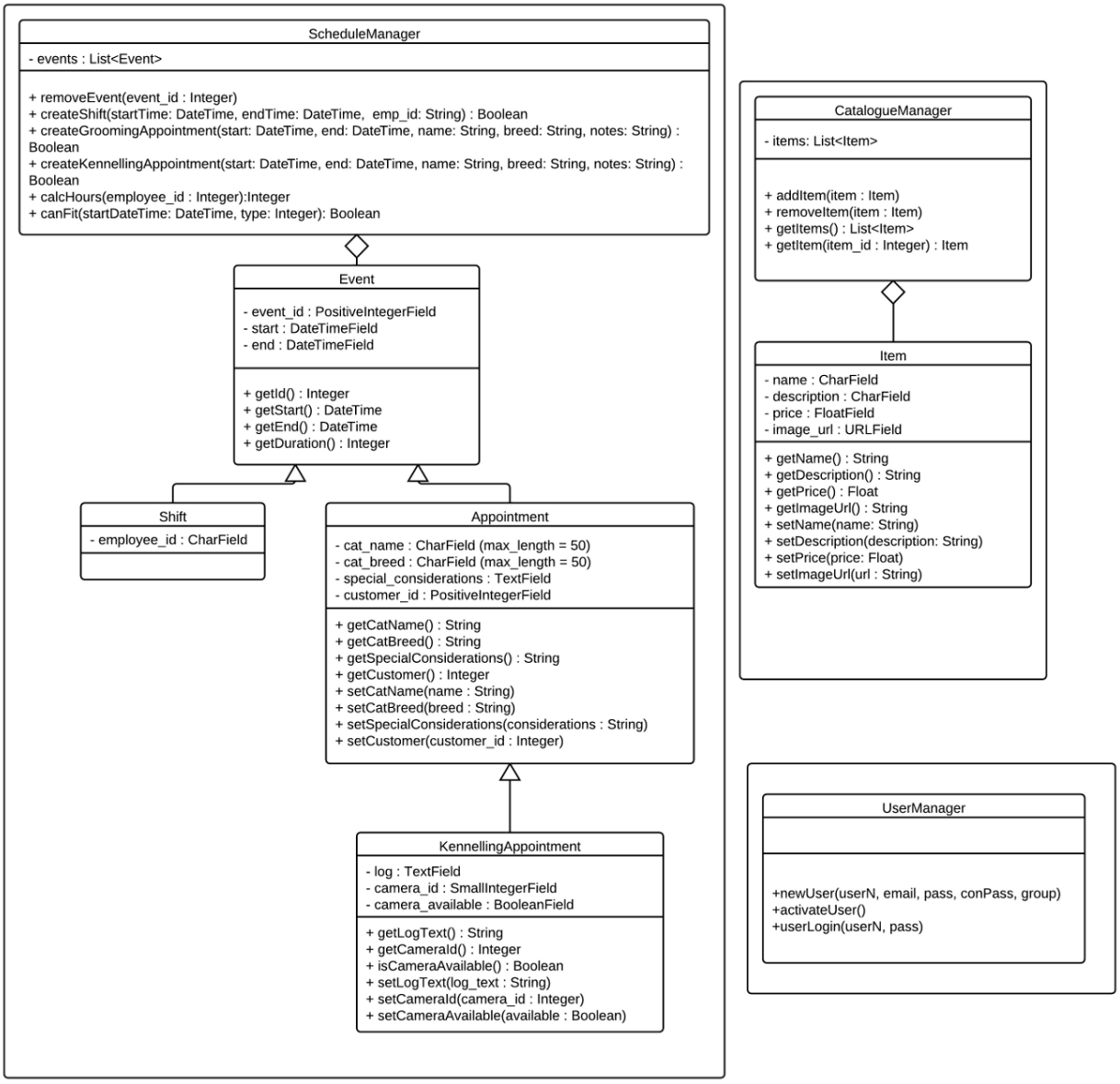


Figure 1.0 - CLAWS Class Diagram

Most of the heavy lifting in the scheduling module is handled by the ScheduleManager. The ScheduleManager is called on to determine if a new appointment can be fit into the schedule or to add a new appointment. Appointments and employee shifts are all specializations of the base Event class, which has a start datetime and an end datetime. Shifts add an employee id, while cat appointments add the cat’s name and breed, along with special considerations for the appointment

and a reference to the customer who booked the appointment. KennelingAppointments further specialize Appointments by adding a field for the log, the number of an associated catcam, and a flag indicating whether the cat is currently out of the kennel.

In the catalogue app, the CatalogueManager adds or removes individual items from the catalogue. Each item is represented by an Item object, which contains the item's name, description, and price, along with a path to the item's image.

The users app relies on the UserManager, which verifies users and provides them with a session when they log in. Users are handled in part by the Django framework.

2.0 Users

There are 3 types of users in the CLAWS system, Manager, Employee, and Customer. Only a single Manager and Employee account exist within the system while multiple Customer accounts are possible. Although there will be only one manager and one employee account upon release of the software, the user management will be written such that allowing additional manager and employee accounts upon customer request could be done easily and quickly.

2.1 Account Responsibilities

2.1.1 Manager

The manager account has the most power and responsibilities in the system. This account can edit the information of all other account types, confirm new customer accounts, create the employee work schedule, create customer appointments, and edit the Cat-a-logue.

2.1.2 Employee

The employee account has reduced responsibility from the manager account. This account can edit customer account information, confirm new customer accounts, create customer appointments, and edit the Cat-a-logue.

2.1.3 Customers

The customer account has very minimal responsibility. This account can only edit its own information and create appointments for its own account.

2.2 Creating Users

2.2.1 Manager & Employee

Only a single Manager and Employee account are in the system and are created by Endeavour Studios before the product is released. The staff at Purrfur will then have to be manually given the passwords to the account at the discretion of the manager. Once Purrfur receives the software the manager may edit the fields of the accounts.

2.2.2 Customers

Customers have the ability to create there own accounts through the web-page. To create an account the customer must fill out the following fields: Desired Username, email, password, and Confirm password. Once a customer has filled in all these fields and there is no conflict with existing users a request will be sent to the manage users page for manager or employee confirmation.

2.3 Managing Users

2.2.1 Manager

Managers are able to manage all users including the employee account. The manager account is able to change all the information about any account as well as the ability to confirm customer accounts upon there creation. Using the admin interface of Django the manager will be able to manage all users from one page. A batch task for confirming multiple customer accounts will be provided in the system in addition to being able to confirm each account individually.

2.2.2 Employee

Employees have the ability to confirm customer accounts. The employee account can change customer account. The employee account will also have some access to the admin interface such that they can also use the tasks to confirm users but will have some fields such as editing the employee account as read only.

Customer Confirmation: Confirming user accounts will be done using the inherent Django user flag set_active; When a manager or an employee confirms an account the set_active flag will become true. This flag will then be checked when a customer tries to log into the system to see if there account has been confirmed.

2.4 User Permissions

2.3.1 Manager

The manager account will have all permission and be able to see everything. Django allows this user to be easily created by instantiating it as a superuser which will automatically give it access to the entire system. In future implementations of this system the manager account will become a user group within the system such that Django will automatically give the proper permissions to each new manager.

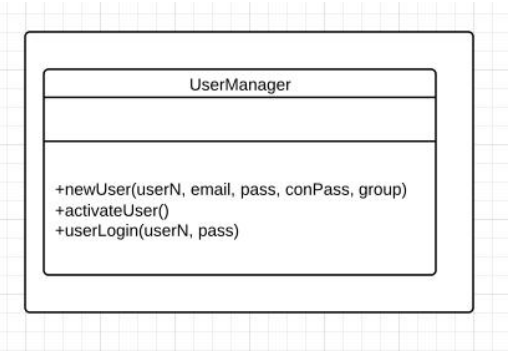
2.3.2 Employees

The employee account will have to be given specific permissions such that i cannot access the editable scheduler, the employee account information, but still have access to editing customer accounts and appointments. In future implementations of this system the employee account will become a user group within the system such that Django will automatically give the proper permissions to each new employee.

2.3.3 Customers

The customer account must have very minimal access to the system, as they can only create appointments and edit there account. Because there will be many users and it is not good design to have each of the users permission created separately, we will use Django's user groups to instantiate a customer group and all of the permissions for each customer will come from being within that group. This will allow the system to be more expansible because changes to permissions or additional permissions can be added more easily then if given to each user on creation.

2.5 UserManager



2.5.1 newUser()

This method creates a new user in the system; The function firsts checks that there is no conflict with the chosen username. If the username is already in use then the user will be prompted to reenter their username and password. If the username is available the function will then check that the password and the confirmed password are identical otherwise the user will be notified and asked to reenter the password. The function will then use Django's built in create_user() method to register the user. The final parameter group will be used to set the permissions for the new user.

At deployment group will automatically be assigned to customer because there is only one manager and employee account in the system. This parameter has been added so that the functionality for creating new manager and employee accounts can be more easily added if requested.

2.5.2 activateUser()

This method will be used to confirm the validity of an account in the system. After a account has been confirmed by a manager or employee this function will be called to set the user accounts set_active flag to true, allowing the user access to the system.

2.5.3 userLogin()

This method is used to authenticate and login a user to the system. Once a user has entered there username and password the system will check to see if it exists within the system, if it does the user will be checked to see if set_active flag on the user has been set to true. If the flag is false the user will recieve a message telling them that there account is not active yet, otherwise it will call Django's login() method to make a session for the user.

2.6 User Test Cases

Test Case Name: Create User
Test Case Purpose: To test the ability to create a new user in the system
Test Case Function:

- Allow users to create new accounts

Test Case Input: Username, Email, Password, Confirm Password

Test Case Expected Output: System should display a message confirming that the new account has been sent for employee approval.

Test Case Procedure:

Test Case Input	**Test Case Output**
User enters a Username that already exists in the system	System displays a message telling the user that the username is already taken
User enters a malformed email address	System tells user that an improper email address has been entered
Users password and Confirm password fields are different	System displays message that the passwords do not match

Test Case Name: Confirm New User Account
Test Case Purpose: To verify that once a user account has been confirmed they have access to the system
Test Case Function:

- Allow employees to confirm users

Test Case Input: Employee selects confirm button by account name or selects multiple accounts and selects the batch approve task.

Test Case Expected Output: User account is removed from pending accounts page and the user then has access to the system.

Test Case Procedure:

Test Case Input	**Test Case Output**
Employee confirms a single User account	User account is removed from pending page
Employee selects and confirms multiple User Accounts	All selected user accounts are removed from pending user page

Test Case Name: User Login
Test Case Purpose: To test that verified users can access the system and unverified users cannot
Test Case Function:

- A verified User account can access the system

- An unverified User account cannot access the system

Test Case Input: Username, Password

Test Case Expected Output: logging in with a verified account should grant access to the system, logging in with an unverified the user should be shown a message

Test Case Procedure:

Test Case Input	**Test Case Output**
Login with a verified user account	Access is granted to the system and a user session starts
Login with an unverified user account	A message telling the user that there account has not been verified by an employee yet is displayed

3.0 Scheduling

The schedule manager deals with subclasses of event. It can determine whether there is room for another event at a given time and can add or remove events as requested. The event super class has the following attributes:

Attribute	**Type**	**Description**
start	DateTimeField	Start date and time of event
end	DateTimeField	End date and time of event
event_id	IntField	Unique event identifier

The event class has two subclasses: Appointment and employeeShift. Each of these inherits the startDate and endDate attributes.

3.1 Operations

3.1.1 bool canFit(startDate, appointmentType)

This method calculates, whether an appointment of appointmentType can fit in a given slot in the schedule. Based on the type, the method checks a set amount of time (1 hour for a grooming appointment and 1 day for a kennelling) starting from the start time. Any appointment found to overlap with this section is said to be in conflict with it and added to the total. Because the time being used is the smallest unit of each appointment type, there are no partial overlaps. The total must be less than the limit for the appointment type.

inputs:

startDate: must be a valid date and time.

appointmentType: must be one of APPOINTMENT (for grooming appointments), KENNELLING_APPOINENT (for kenneling appointments), or SHIFT (for employee shifts).

outputs:

TRUE if the endDate is after the startDate and there is room for the appointment.

FALSE otherwise.

Pseudocode:

```
determine time slice from type and start time
count number of appointments of the same type that overlap the time slice
if (the number of overlapping appointments is less appointment limit for that type)
        return true
else
        return false
```

3.1.2 int CalcHours(startDate, endDate, id)

This method calculates the hours worked by an employee, or the hours a certain customer has had their cat groomed. Every hour between the start date and the end date that corresponds with the id.

Inputs:

startDate: must me a valid date and time.

endDate: must me a valid date and time after the start time.

employee_id: must be a valid employee id.

Outputs:

The number of hours worked by the employee. -1 if the employee_id is invalid.

3.1.3 Boolean createEmployeeShift(startTime, endTime, employee_id)

This method creates a new employee shift taking all of its attributes in as paramaters.

inputs:

startDate: must be a valid date and time.

endDate: must be a valid date and time.

employee_id: a unique identifier of the

outputs:

TRUE if the endDate is after the startDate and the shift is successfully added.

FALSE otherwise.

3.1.4 Boolean createAppointment(start, end, type, cat_name, cat_breed, considerations)

This method creates a new grooming or kennelling appointment taking all of its attributes in as parameters.

inputs:

startDate: must be a valid date and time.
endDate: must be a valid date and time.
cat_name: The cat's name.
cat_breed: The cat's breed.
considerations: any special issues relating to the cat. The string in considerations must be able to hold a sizable message.

outputs:

TRUE if the endDate is after the startDate, the and the appointment is successfully added.
FALSE otherwise.

3.1.5 Schedule Manager Operation Unit Tests

Input	Return Code
Canfit with fewer appointments than employees - 1	return true
canFit with fewer than the total kennels filled	return true
canFit with employees=appointments	return false
canFit with all kennels filled	return false
calcHours with invalid emp_id	return -1
calcHours with startTime > endTime	return -1
calcHours with no hours worked	0
calsHours with 8 hours	8

3.2 Employee Schedule

3.2.1 Employee Shift Object

Attribute	Type	Description
employee_id	CharField	unique identifier for an employee

3.2.2 Shift Creation

The manager view of the employee schedule contains a form with a start date, end date, and employee_id. When the manager clicks the create button, the information in the form is validated and entered into a new shift object.

3.2.3 Shift Management

When a manager clicks on a shift, the shift information is retrieved and placed in the shift creation form. The “create” button is replaced with an “update” button, which will be disabled until the manager has changed a value. When the manager clicks the update button, the shift information in the database is updated and the shift chart is redrawn.

3.2.4 Shift Viewing

The gantt chart view of the schedule will always display the current schedule. This will need to have a specifiable date range with horizontal and vertical scroll bars to maintain readability. To view the details of a shift, the manager can click on it in the gantt chart. Until they change something and click the update button, they are simply viewing the information.

3.2.5 Shift Deletion

In addition to the “update” button, there will also be a delete button. When this is clicked, the shift must be deleted and the gantt chart must be redrawn to match.

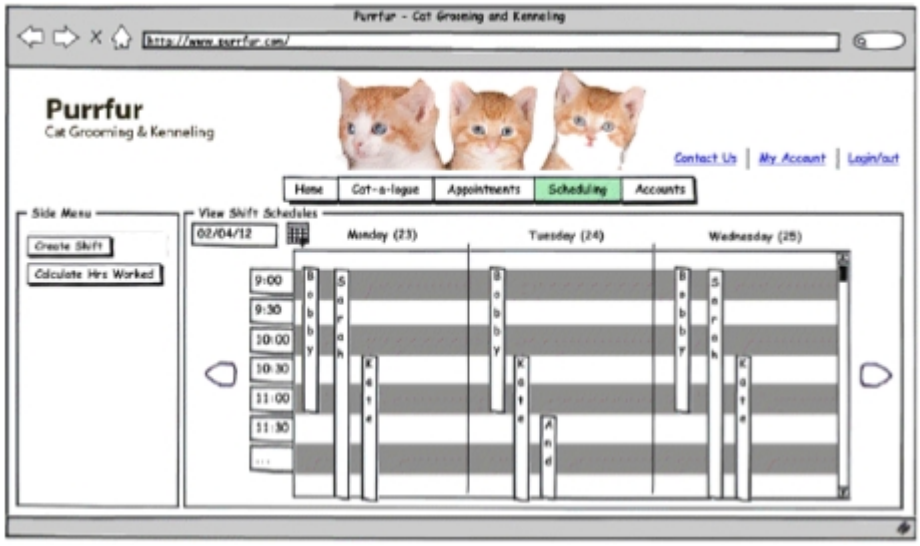


Figure 4.3.1 - Manager View of Shift Schedule

Test case Procedure:

Test Case Input	***Test Case Output***
click create with valid startDate, endDate, and employee_id	shift is created and appears in the gantt chart. if clicked in the chart, it will populate the shift fields with correct data.

createShift with valid startDate, and employee_id, but endDate earlier than startDate	shift is not created. An error message is displayed.
select shift click delete	shift disappears from chart and is not found in database.
select update with the shift end time earlier than it was before the update	shift shrinks in the gantt chart (end time moves up in the chart) and when the bar is reselected, the new end time is displayed in the chart.

3.3 Appointments

3.3.1 Appointment Object

The Appointment object is a class containing all the extra information pertaining to an appointment of either grooming or kennelling type. It is the parent class of the KennellingAppointment object and it is a child of the Event object.

Attribute Name	**Type**	**Description**
cat_name	CharField, max_length = 50	Cat Name
cat_breed	CharField, max_length = 50	Cat Breed
special_considerations	TextField	Special Considerations

3.3.2 KennelingAppointment Object

This object contains the extra information pertaining to a kenneling appointment including logs, and cat cam number (if applicable). It is a child of the Appointment object.

Attribute Name	**Type**	**Description**
log	TextField	Kenneling Log Record
camera_id	SmallIntegerField	Cat Cam Number (-1 if NA)
camera_available	BooleanField	If the cat cam is available (or temporarily disabled)

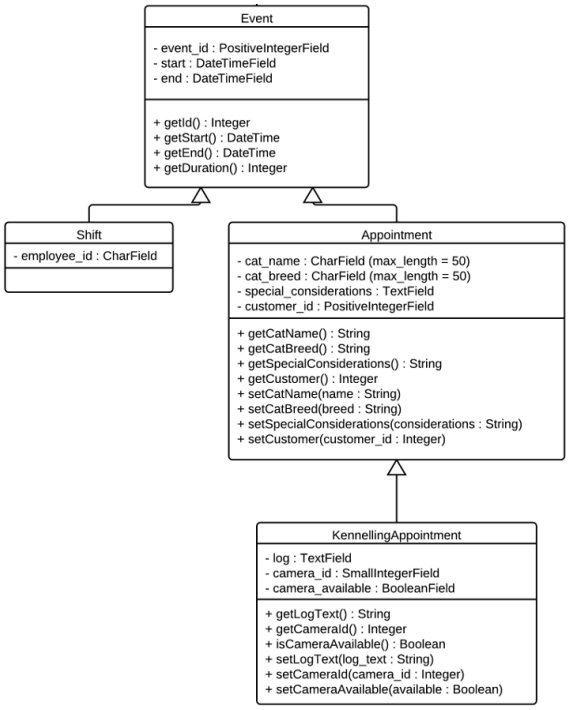


Figure 4.3.2 - Appointment Class Diagram

3.3.3 Creating Appointment

Appointment info is provided by a logged in user (either Customer, Employee, or Manager) through the Create Appointment page. This page is accessed through the “Create Appointment” link in the top navigation bar. This page provides a form allowing the user to specify the appointment type (kennelling or grooming), cat name, and special considerations. Based on the appointment type a pop-up calendar of available time slots will be provided and populated. See Section 3.3.3.1 for more information. The submit button for this form displays “Create Appointment”.

This form will validate the information once it has been submitted, ensuring that the required data is provided and the inputs will fit within the attributes. Once the input has been validated an appointment object will be created using the CreateAppointment function of the ScheduleManager object (see Section 3.1.4). If CreateAppointment returns an error code an error message is displayed to the user in a pop-up, informing the user of the error and the reason based on the error code, and the form (with input) is left in-tact so the user can try again.

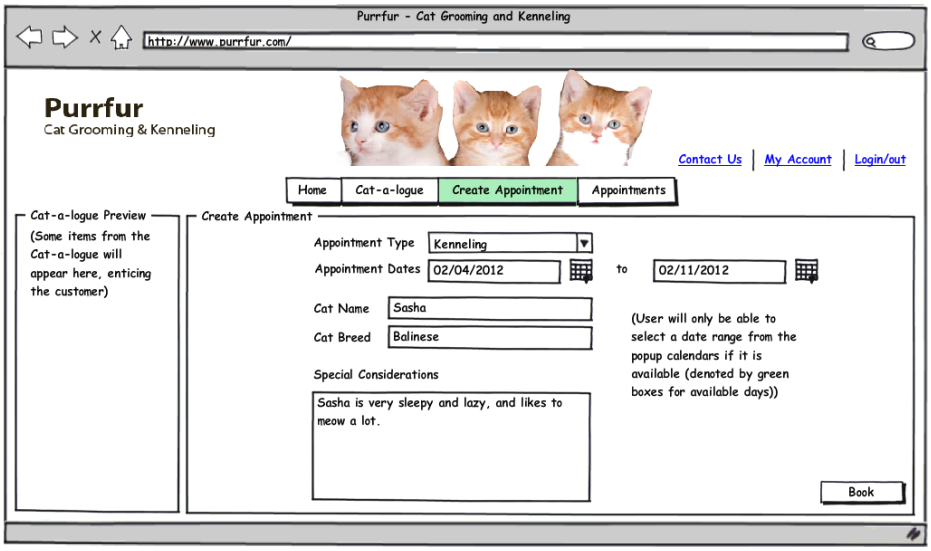


Figure 4.3.3 :: Create Appointment, Kenneling

3.3.3.1 Date/Time Selection Calendar

The Date/Time Selection Calendar is a dynamic time selector, allowing the user to only select available appointment times. Depending on appointment type this calendar will function by hourly slots for grooming appointments, or by day for kenneling appointments. The calendar will have green color in available time ranges, and red where unavailable. The availability information for each slot is obtained from the canFit function of the ScheduleManager object (see Section 3.1.1). A data flow diagram (DFD) is provided for the CreateAppointment process, which makes use of the canFit function (see Figure 3.3.3.1).

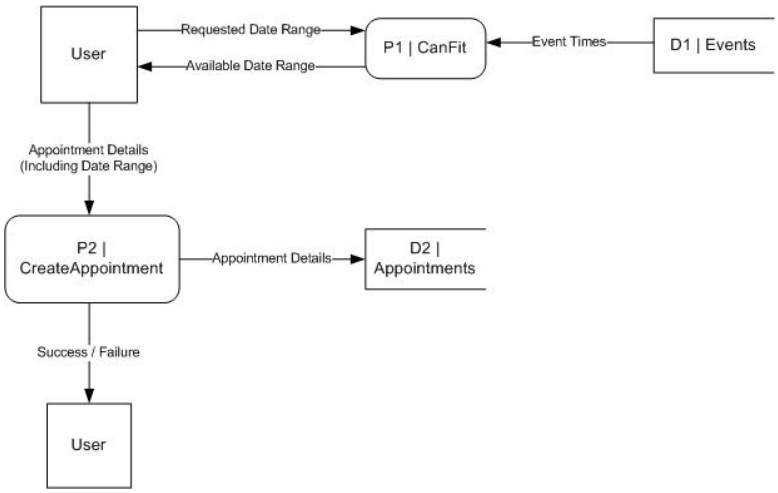


Figure 4.3.3.1 :: Data Flow Diagram - CreateAppointment Process

In the case of a grooming appointment being selected, the calendar will display the current day, and the consecutive two days, as well as provide arrows on the right and left allowing the user to scroll the calendar to look at future days. Each day will be divided into time slots equal to the length of a grooming appointment, and each time-slot will be colored red or green indicating if an appointment is available at this time or not. The availability will be determined by successive calls to the canFit function for each time-slot. Each time the user clicks, the calendar will shift by one day, and again

appropriately color the calendar using successive calls to canFit. When the user selects a time-slot the pop-up will close and populate the date/time field of the form. Opening the calendar and clicking a different slot will re-populate the date/time field, changing the users choice.

In the case of a kennelling appointment, the calendar will have a different view, displaying five days, and a similar arrows allowing the user to shift the days displayed by the calendar. Each day will be colored red and green, again by using successive calls to canFit. When the users selects the day to start the kenneling appointment the form will auto-populate the start date, and auto-populate the end date with the following date (the next day). The user can then show the calendar for the ending date of the kennelling appointment. This calendar will have an identical display to above, with the exception that only days after the start date will be available (green), and the availability will end at the first unavailable (red) day. If the user shows the calendar for the end date, without having a start date populated, only days after the current day will be available.

3.3.4 Managing Appointments

Using the navigation bar, the user accesses the appointments calendar by clicking the Appointments link. Depending if the user is a customer or an employee/manager a different view is displayed.

3.3.4.1 Customer Access

The appointments page displays a list of all the customers appointments (past and future) in a paginated list, sorted in descending order by start date. This list displays the appointment start and end time/date in a standard, readable format, the appointment type, and the name of the cat. The user has the option to sort the list by the other displayed attributes as well. Beside each appointment are buttons: “Modify”, “Delete”, ”Catcam”, and “Log”. The “Catcam” and “Log” buttons are only displayed if the appointment is of the kennelling type. The “Modify” and “Delete” buttons are only displayed if the appointment is more than 48 hours in the future.

The “Modify” button takes the user to a page with a form the same as is seen on the Create Appointment page, with the difference that the fields are already populated with the appointment info, and the submit button displays “Modify Appointment”.

The “Delete” button displays a prompt, confirming the users desire to delete the appointment, and informing them that once deleted the appointment cannot be recovered. This dialog has two buttons, “Confirm” and “Cancel”. The “Confirm” button deletes the appointment (and refreshes the appointment list to show the change), and the “Cancel” button returns the user back to the list of appointments.

The “Catcam” button takes the user to a page showing the video stream of the cat, or a message informing the user that the cam is temporarily unavailable if the cam_avail attribute is False. There is also a “Back” button that returns the user to the appointments list.
The “Log” button takes the user to a page displaying the contents of the log, with a scroll bar if necessary. There is also a “Back” button that takes the user back to the appointments list.

3.3.4.2 Employee/Manager Access

The appointments page displays a scrollable calendar, showing all appointments (past and future) in the calendar. By default this calendar displays the current day and the two upcoming days, but the

user can change the displayed days by clicking left and right arrows located at the sides of the calendar. See Figure 3.3.4.2 for an example.

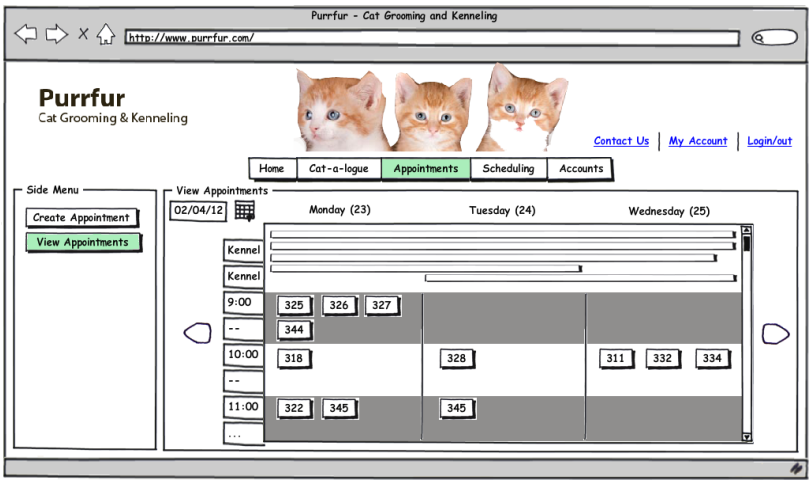


Figure 3.3.4.2 - View Appointments, Manager

3.3.5 Appointment Test Cases

3.3.5.1 Create Appointment Test Case

Test case Name: Create Appointment
Test case Purpose: To ensure that appointments are created with valid start and end times
Test case Function:

- To verify that only correct/valid start and end times are selected
- To ensure required fields contain input
- Allow a user to create an appointment

Test case Input: An appointment date range, and various text fields (cat name, etc.)

Test case expected Output: The user should only be allowed to select from a set of valid start and end times, as returned by the CanFit function. If the appointment booking is successful, a confirmation message should be returned to the user.

Test case Procedure:

Test Case Input	***Test Case Output***
A valid (available) start and end time, along with a valid input of cat name and cat breed	The user is redirected to a confirmation page stating the appointment creation was successful
A valid (available) start and end time, along with **no** input for cat name and cat breed	Client-side form validation returns a message asking the user to supply a cat name and cat breed
Invalid (unavailable) start and end time is attempted to be selected.	The date and time cannot be selected, and the form cannot be submitted.

3.3.5.2 Manage Appointment Test Case

Test case Name: Manage Appointment
Test case Purpose: To ensure that an appointment can be modified successfully
Test case Function:

- To verify that only correct/valid start and end times are selected
- To ensure required fields contain input
- To ensure the user can only modify their appointments
- Allow a user to modify an existing appointment

Test case Input: An existing appointment, along with the new date range and various text fields (cat name, etc.)

Test case expected Output: The user should only be allowed to modify appointments pertaining to them. The user should only be allowed to selected from a set of valid start and end times, as returned by the CanFit function. If the appointment booking is successful, a confirmation message should be returned to the user.

Test case Procedure:

Test Case Input	***Test Case Output***
An appointment not owned by the user is attempted to be modified	The list of appointments in the users 'manage appointments' page are only those owned by them. Therefore, the appointment cannot be selected to modify.
A valid appointment is selected to modify and the date range is changed to a valid (available) start and end time. Cat name and cat breed remain unchanged.	The user is redirected to a confirmation page stating the appointment modification was successful, and the date/time range for this appointment has been changed.
A valid appointment is selected to modify and the date range is changed to a valid (available) start and end time. Cat name and cat breed are removed.	Client-side form validation returns a message asking the user to supply a cat name and cat breed.
A valid appointment is selected to modify and the user attempts to change the date range to an invalid (unavailable) start and end time	The new date and time cannot be selected, and the form cannot be submitted.

4.0 Cat-a-logue

4.1 Overview

The Cat-a-logue is a data store of items, chosen by the manager and employee users to display to the customers in a multiple ways. The interactions allowed are based on the currently logged in user, and will support adding, editing, removing, viewing list of items, and viewing individual items. There

are three views for the Cat-a-logue; one product listing on the login page, for anonymous users, one product listing in the main “Cat-a-logue” tab in the dashboard, and one “Product” page described in section 4.3.1 / 4.3.2.

4.2 Class Diagram

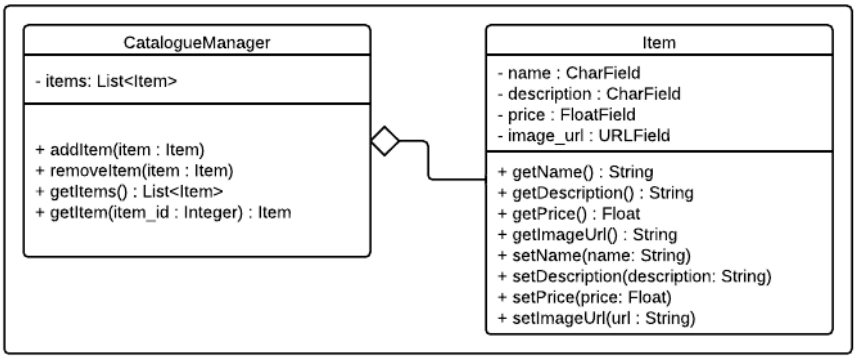


Figure 4.2.1 - Class Diagram for the Cat-a-logue components.

4.3 Views

4.3.1 Product Listing (Customer)

The “Product Listing” view, is modeled by a table of images for the different products, as well as the name and underneath. This table receives the data from the server by calling the CatalogueManager getItems() method and displays it. The allowed functions here are simply clicking on the products to enter the “Product View” described in 4.3.3.

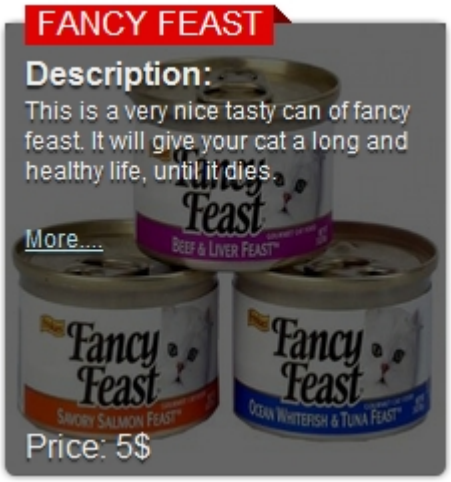


Figure 4.3.1.1 :: Example of an Item

4.3.2 Product Listing (Manager)

This retrieves the data the same as the customer version of the product listing, except it allows for more functions and has some slight display modifications. Each product will have an extra options

for removing it, editing it, and and adding new products. The new products must provide all the parameters for a new instance of an Item, seen in 4.2. The information will be retrieved via a dialog, and will display confirmation to the user when the product is created by retrieving the updated list.

4.3.3 Single Product View

This is a expanded view on a single product, it will have a larger version of the picture in the Product Listing's thumbnail, and have the full description displayed. The product to be shown will be sent in a parameter in the URL. There will no other functions here for a customer user, but for the manager or employee users, it will provide edit and delete buttons, with the same functionality as in the Product Listing.

4.4 UI

4.4.1 Login Page

The login page will offer a product listing version of the Cat-a-logue, showing a simple list of the items in a table format. This products listing will be smaller and on the bottom of the page. It offers the users something to look at even if they don't have an account with the site. From this page the user can access the Single Product View seen in 4.3.3.

4.4.2 Main Cat-a-logue Page

The main Cat-a-logue Page will have a larger product listing than the login page. The user can still click on an item to bring them to the Single Product View for that Item, but the main difference is that this page has the editable functions for employees and managers on it. This means that there will be buttons on the sidebar to add a Product, and the product items will have remove and edit buttons on them.

4.5 Test Plan

4.5.1 Create New Product Test

Test case Name: Create New Product Test
Test case Purpose: To ensure that a new product may be created by the manager.
Test case Function:

- To verify that the manager and employee accounts are able to access their add function for products.
- To verify that the add product dialog has the correct fields
- To verify that when a product is created successfully, it displays a success message
- To verify a successfully created product shows up in the Cat-a-logue views.

Test case Input: A new product (Name:CharField, Description:CharField, Price:Float, ImageURL:CharField).
Test case expected Output: The employee/manager should be able to create a product which displays a success message, then displays the new product in all Cat-a-logue views.

Test case Procedure:

Test Case Input	***Test Case Output***
Manager or Employee log in. Select the " "Cat-a-logue" tab.	The current list of Cat-a-logue entries are shown in the Product Listing view, as described in 4.3.2. Administrative functions are displayed (add, edit, remove).
User clicks add.	The "Create new product" dialog pops up and has fields for the following: Name, Price, Description, ImageURL with a file browser.
Users fills all the fields and clicks OK.	The system responds with "Created Product Successfully" and reloads the Product Listing View with the new product in it.

4.5.2 View Single Product Test

Test case Name: View Single Product Test
Test case Purpose: To ensure that Items may be viewed in the Single Product View
Test case Function:

- To verify that products show up in the Product Listing views.
- To verify the link isn't broken between product listings and single product pages.
- To verify the data retrieved for a single product is correct.

Test case Input: The field input data from Test case 4.5.1, and a logged in user accessing the "Cat-a-logue" view, and the created product from 4.5.1.

Test case expected Output: The user should be able to view the created product, and verify that the data used to create it is identical to the data retrieved in the Single Product View.

Test case Procedure:

Test Case Input	***Test Case Output***
Customer log in. Select the " "Cat-a-logue" tab.	The current list of Cat-a-logue entries are shown in the Product Listing view, as described in 4.3.2. Administrative functions are displayed (add, edit, remove). The specific entry created in 4.5.1 is present in the list of items, with correct price and name, and the thumbnail is correct.
User clicks on the Cat-a-logue entry created in 4.5.1	The system responds by showing the "Single Product View" described in 4.3.3, with all the information filled in correctly for the fields: Name, Price, Description, and the Image displayed is a enlarged version of the data given in 4.5.1.

5.0 Test Plan

5.1 Objectives

The CLAWS test plan provides a comprehensive method of ensuring top quality software at the end of the development cycle. It explains in detail the stages of the testing process and which methods will be used on which modules. It provides a solution for integration of modules and methods to

prevent errors to propagate to the late stages of the project. The test plan will explain in detail the methods of monitoring and reporting defects, as well as the procedure for resolving them.

5.2 Test Schedule and Responsibilities

5.2.1 Schedule

Testing will be carried out throughout development, with testers performing small scale testing on completed and in-progress modules alongside of development. Once development is completed testing will begin doing in-depth, large scale tests to fully test the implementation and flush out any bugs that may arise.

Test Case	**Developer**	**QA Tester(s)**	**Deadline**
Unit Tests	All	All	Before Commiting
Build Tests (Continuous Integration)	**Developer**	**QA Testers(s)**	**Deadline**
Build Tests (Including all unit tests)	All	N/A	Nightly
Functional Tests	**Developer**	**QA Tester(s)**	**Deadline**
User Tests (2.6)	Jordan	Theo, Braden	After every Integration
Employee Schedule (3.0)	Michael	Theo, Jordan	After every Integration
Appointments Tests (3.3.5)	Jeremy	Theo, Michael, Wes	After every Integration
Cat-a-logue Tests (4.5)	Braden	Theo, Jeremy	After every Integration
Acceptance Test (5.3.6)	All	Client	March 29th
Additional Tests (7.3)	All	Theo, Jared	March 22nd, March 26th

5.2.2 Responsibilities

Small scale testing of individual modules will be carried out by the developers as the modules are developed and the large scale full implementation will be performed by the designated testers.

5.3 Test Procedures and Criteria

5.3.1 Testing Environment

The tester must perform all tests on all our supported browsers (Firefox 5+, Safari 5, Chrome), as well, the tester must deploy each new version of the server on all supported operating systems (Windows, Linux (Debian), OSX). When deploying the new version of the project, the tester must

receive the new build from our build system, which will be created nightly, ensuring that new defects are discovered quickly.

5.3.2 Unit Tests

Unit testing will be performed after writing any server side code, and before any commits are accepted into the team's project repository, all the tests must pass. If any new methods are implemented, or any new logic added, there must be corresponding unit tests written as a separate commit. Any commits causing unit tests to fail will not be accepted upstream.

5.3.3 Integration Testing

The modules will be integrated in order of most critical to least critical, this provides more time testing the critical modules, and allowing the later, less critical features to be tested less. With this integration method, the more critical features will be central, and other features will need to conform to them, which is the ideal model. This model is also Integration testing will be performed iteratively over our predefined list of test cases, as new unit-tested components are pushed upstream. The test cases will be executed on these components by our dedicated tester. As new features are added, new test cases written by the tester must be added to the list of tests to be performed during integration testing. If a defect is encountered during integration testing, the tester will make a new defect report with our provided template (7.1). When created it will be placed in the defect backlog and the developer(s) assigned to fixing defects will start working on it; once a fix is implemented, the defect will be marked as pending and is assigned back to the tester for re-testing, and the process is repeated until the defect is resolved as closed(fixed) by the tester.

5.3.4 Functional Testing

Functional testing will be a part of the Integration test procedure. As the tester performs the testing portion of the integration procedure, it will be using functional testing methods. Each functional test case written will be executed the following way:

- Identify functions that the software is expected to perform
- Create data based on the function's specifications
- Determine output based on the function's specifications
- Execute the test case
- Compare actual and expected outputs

The test cases are written by the tester who does not have access to the code base, making these tests truly functional black-box tests. Some of the functional tests to be performed on each component are in their respective sections in this document.

5.3.5 Performance Evaluation

The performance evaluation will be done at the end of each integration test, and will consist of quantitative tests on throughput of different types of actions in CLAWS. These tests will be executed automatically, and will measure the speed of our system in storing doing all of the actual processing and data access.

5.3.6 Acceptance Test

The Acceptance test will be in our demo to the client on the second of April, 2012. It will include a demonstration of our product and have the acceptance tests run.

6.0 Management Plan

The following table outlines coding responsibilities.

Module / Aspect	Members Responsible
Templates (HTML, CSS)	Braden, Jared
Client-side Coding (JavaScript, AJAX)	Jared
Scheduling	Michael
Scheduling (Appointments)	Jeremy, Wes
Users	Jordan
Catcam / Logs	Jeremy, Jordan
Cat-a-logue	Braden

The following table outlines expected module completion dates, the dates the modules should be fully tested by.

Module / Aspect	Completion Date	Testing Completion Date
Templates (HTML, CSS)	Mar. 20	Mar. 22
Client-side Coding (JavaScript, AJAX)	Mar. 20	Mar. 22
Scheduling	Mar. 27	Mar. 29
Scheduling (Appointments)	Mar. 27	Mar. 29
Users	Mar. 24	Mar. 26
Catcam / Logs	Mar. 24	Mar. 26
Cat-a-logue	Mar. 24	Mar. 26

7.0 Appendix

7.1 Defect Creation Form

https://docs.google.com/spreadsheet/viewform?formkey=dDRrWmJVRVhmRGN4M2xlSWxfTlNOV2c6MQ

7.2 Additional Functional Tests

https://docs.google.com/spreadsheet/ccc?key=0AgM7Ob93dWeSdEFaSDVCZXlSalhnQnlOTG9nM0hnMVE

7.3 URL Patterns

https://docs.google.com/document/d/1BkkY4ZjRNyWoVRYJGajEBE3tHRctlg2A98oaMMb043U/edit