BILKENT UNIVERSITY

SPRING 2021

CS353 TERM PROJECT

DESIGN REPORT

GROUP 31

Zoo Database Management System

Mehmet Yaylacı - 21802347 - Section 3

Yiğit Erkal - 21601521 - Section 2

Selin Kırmacı - 21802177 - Section 3

Selcen Kaya - 21801731 - Section 3

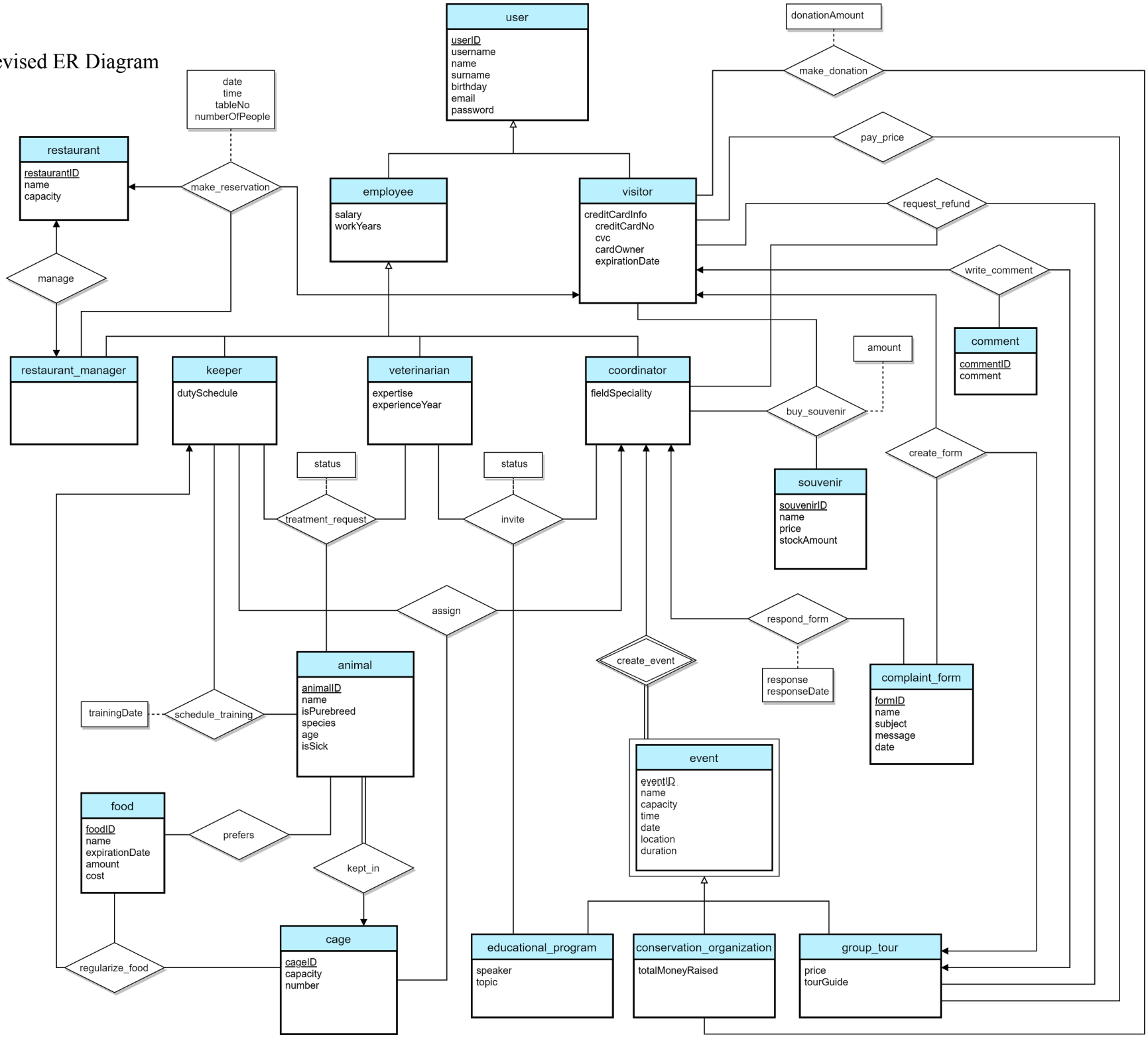
Table of Contents

1.0 Revised E/R Model

In the list below the changes to the ER diagram are given. The only change not made is adding the “price” attribute to the “pay_price” relation since “price” is an attribute of group_tour entity which is used by both “pay_price” and “request_refund” relations.

- Added the “souvenir” entity with attributes “souvenirID”, “name”, “price” and “stockAmount”.
- Added the “buy_souvenir” relation between coordinator, visitor and souvenir entities with “amount” attribute.
- Added the “restaurant_manager” entity with no attributes as a new employee type.
- Added the “restaurant” entity with attributes “restaurantID”, “name” and “capacity”.
- Added the “make_reservation” relation between restaurant_manager, restaurant and visitor entities with attributes “date”, “time”, “tableNo” and “numberOfPeople”.
- Added “username” attribute to the “user” entity.
- Added “duration” attribute to “event” entity.
- Removed the donators{} attribute from the “conservation_organization” entity.
- Removed the preferredFood{} attribute in animal entity and added the “prefers” relation between animal and food entities instead.
- Changed coordinator to 1 in “create_event” relation.
- Changed coordinator to 1 in “respond_form” relation.
- Changed complaint_form to M, group_tour to 1and visitor to 1 in “create_form” relation.
- Changed visitor to 1 in “write_comment” relation.
- Changed group_tour to 1 in “write_comment” relation.
- Changed coordinator to 1 in “assign” relation.
- Changed keeper to 1 in “regularize_food” relation.
- Removed the “respond_treatment” relation, renamed the “request_treatment” relation to “treatment_request” for clarity, and added the “status” attribute to “treatment_request” relation.
- Removed the “respond_invitation” relation, and added the “status” attribute to the “invite” relation that the veterinarian can update.
- Removed the “manage” relation.
- Removed familyMembers{} attribute from animal entity.

Figure 1: Revised ER Diagram



2.0 Relations

The relations that are converted from the revised ER diagram are given below. Each relation is presented as a subheading under which the table schema, normal form and table definitions in MySQL are listed. The normal form is given to prove that the relations are in at least 3NF and the table definitions are given to identify the attribute domains. The relations aren't as simplified as they can be (such as the kept_in relation) for clarity and intelligibility.

2.1 User

Schema

user(<u>userID</u>, username, name, surname, birthday, email, password)

primary key(s) → userID

candidate key(s) → { (userID), (email), (username) }

Normal Form

BCNF

Table Definitions

```
CREATE TABLE user (
        userID          CHAR(10) NOT NULL AUTO_INCREMENT,
        username        VARCHAR(50) NOT NULL UNIQUE,
        name            VARCHAR(30) NOT NULL,
        surname         VARCHAR(30),
        birthday        DATE,
        email           VARCHAR(50) NOT NULL UNIQUE,
        password        VARCHAR(30) NOT NULL,
        PRIMARY KEY( userID ) );
```

2.2 Visitor

Schema

visitor(visitorID, creditCardNo, cvc, cardOwner, expirationDate)

primary key(s) → visitorID

candidate key(s) → { (visitorID) }

foreign key(s) → visitorID references user (userID)

Normal Form

BCNF

Table Definitions

```
CREATE TABLE visitor (
    visitorID           CHAR(10) NOT NULL,
    creditCardNo        VARCHAR(16),
    cvc                 UNSIGNED INT(3),
    cardOwner           VARCHAR(50),
    expirationDate      UNSIGNED INT(4),
    PRIMARY KEY( visitorID ),
    FOREIGN KEY( visitorID ) REFERENCES user( userID ) );
```

2.3 Employee

Schema

employee(employeeID, salary, workYears)

primary key(s) → employeeID

candidate key(s) → { (employeeID) }

foreign key(s) → employeeID references user (userID)

Normal Form

BCNF

Table Definitions

```
CREATE TABLE employee (
        employeeID          CHAR(10) NOT NULL,
        salary              INT,
        workYears           UNSIGNED INT,
        PRIMARY KEY( employeeID ),
        FOREIGN KEY( employeeID ) REFERENCES user( userID ) );
```

2.4 Keeper

Schema

keeper(keeperID, dutySchedule)

primary key(s) → keeperID

candidate key(s) → { (keeperID) }

foreign key(s) → keeperID references employee (employeeID)

Normal Form

BCNF

Table Definitions

```
CREATE TABLE keeper (
        keeperID            CHAR(10) NOT NULL,
        dutySchedule        VARCHAR(300),
        PRIMARY KEY( keeperID ),
        FOREIGN KEY( keeperID ) REFERENCES employee( employeeID ) );
```

2.5 Veterinarian

Schema

veterinarian(vetID, expertise, experienceYear)

primary key(s) → vetID

candidate key(s) → { (vetID) }

foreign key(s) → vetID references employee (employeeID)

Normal Form

BCNF

Table Definitions

```
CREATE TABLE veterinarian (
    vetID           CHAR(10) NOT NULL,
    expertise       VARCHAR(50),
    experienceYear  INT,
    PRIMARY KEY( vetID ),
    FOREIGN KEY( vetID ) REFERENCES employee( employeeID ) );
```

2.6 Coordinator

Schema

coordinator(coordID, fieldSpeciality)

primary key(s) → coordID

candidate key(s) → { (coordID) }

foreign key(s) → coordID references employee (employeeID)

Normal Form

BCNF

Table Definitions

```
CREATE TABLE coordinator (
    coordID         CHAR(10) NOT NULL,
    fieldSpeciality VARCHAR(50),
```

```
    PRIMARY KEY( coordID ),
    FOREIGN KEY( coordID ) REFERENCES employee( employeeID ) );
```

2.7 Restaurant Manager

Schema

manager(managerID)

primary key(s) → managerID

candidate key(s) → { (managerID) }

foreign key(s) → managerID references employee (employeeID)

Normal Form

BCNF

Table Definitions

```
CREATE TABLE manager (
    managerID           CHAR(10) NOT NULL,
    PRIMARY KEY( managerID ),
    FOREIGN KEY( managerID ) REFERENCES employee( employeeID ) );
```

2.8 Restaurant

Schema

restaurant(restaurantID, name, capacity)

primary key(s) → restaurantID

candidate key(s) → { (restaurantID) }

Normal Form

BCNF

Table Definitions

```
CREATE TABLE restaurant (
        restaurantID        CHAR(10) NOT NULL AUTO_INCREMENT,
        name                VARCHAR(50),
        capacity            UNSIGNED INT,
        PRIMARY KEY( restaurantID ) );
```

2.9 Manage

Schema

manage(managerID, restaurantID)

primary key(s) → managerID, restaurantID

candidate key(s) → { (managerID, restaurantID) }

foreign key(s) → managerID references manager

→ restaurantID references restaurant

Normal Form

BCNF

Table Definitions

```
CREATE TABLE manage (
        managerID           CHAR(10) NOT NULL,
        restaurantID        CHAR(10) NOT NULL,
        PRIMARY KEY( managerID, restaurantID ),
        FOREIGN KEY( managerID ) REFERENCES manager,
        FOREIGN KEY( restaurantID ) REFERENCES restaurant );
```

2.10 Make Reservation

Schema

make_reservation(visitorID, managerID, restaurantID, date, time, tableNo,
numberOfPeople)

primary key(s) → visitorID, managerID, restaurantID

candidate key(s) → { (visitorID, managerID, restaurantID) }

foreign key(s) → visitorID references visitor
→ managerID references manager
→ restaurantID references restaurant

Normal Form

BCNF

Table Definitions

```
CREATE TABLE make_reservation (
    visitorID       CHAR(10) NOT NULL,
    managerID       CHAR(10) NOT NULL,
    restaurantID    CHAR(10) NOT NULL,
    date            DATE,
    time            TIME,
    tableNo         UNSIGNED INT,
    numberOfPeople  UNSIGNED INT,
    PRIMARY KEY( visitorID, consID ),
    FOREIGN KEY( visitorID ) REFERENCES visitor,
    FOREIGN KEY( consID ) REFERENCES conservation_organisation );
```

2.11 Souvenir

Schema

souvenir(souvenirID, name, price, stockAmount)

primary key(s) → souvenirID

candidate key(s) → { (souvenirID) }

Normal Form

BCNF

Table Definitions

```
CREATE TABLE souvenir (
        souvenirID          CHAR(10) NOT NULL AUTO_INCREMENT,
        name                VARCHAR(50),
        price               UNSIGNED INT,
        stockAmount         UNSIGNED INT,
        PRIMARY KEY( souvenirID ) );
```

2.12 Buy Souvenir

Schema

buy_souvenir(visitorID, coordID, souvenirID, amount)

primary key(s) → visitorID, coordID, souvenirID

candidate key(s) → { (visitorID, coordID, souvenirID) }

foreign key(s) → visitorID references visitor

→ coordID references coordinator

→ souvenirID references souvenir

Normal Form

BCNF

Table Definitions

```
CREATE TABLE buy_souvenir (
        visitorID           CHAR(10) NOT NULL,
        coordID             CHAR(10) NOT NULL,
        souvenirID          CHAR(10) NOT NULL,
        amount          UNSIGNED INT,
        PRIMARY KEY( visitorID, coordID, souvenirID ),
        FOREIGN KEY( visitorID ) REFERENCES visitor,
```

FOREIGN KEY(coordID) REFERENCES coordinator,
FOREIGN KEY(souvenirID) REFERENCES souvenir);

2.13 Event

Schema

event(coordID, eventID, name, capacity, time, date, location, duration)
primary key(s) → eventID
candidate key(s) → { (eventID) }
foreign key(s) → coordID references coordinator

Normal Form

BCNF

Table Definitions

```
CREATE TABLE event (
    coordID         CHAR(10) NOT NULL,
    eventID         CHAR(10) NOT NULL AUTO_INCREMENT,
    name            VARCHAR(50),
    capacity        UNSIGNED INT,
    time            TIME,
    date            DATE,
    location        VARCHAR(50),
    duration        TIME,
    PRIMARY KEY( eventID ),
    FOREIGN KEY( coordID ) REFERENCES coordinator );
```

2.14 Educational Program

Schema

educational_program(eduID, speaker, topic)

primary key(s) → eduID

candidate key(s) → { (eduID) }

foreign key(s) → eduID references event (eventID)

Normal Form

BCNF

Table Definitions

```
CREATE TABLE educational_program (
        eduID                   CHAR(10) NOT NULL,
        speaker                 VARCHAR(50),
        topic                   VARCHAR(50),
        PRIMARY KEY( eduID ),
        FOREIGN KEY( eduID ) REFERENCES event( eventID ) );
```

2.15 Invite

Schema

invite(vetID, coordID, eduID, status)

primary key(s) → vetID, coordID, eduID

candidate key(s) → { (vetID, coordID, eduID) }

foreign key(s) → vetID references veterinarian
→ coordID references coordinator
→ eduID references educational_event

Normal Form

BCNF

Table Definitions

```
CREATE TABLE invite (
        vetID           CHAR(10) NOT NULL,
        coordID         CHAR(10) NOT NULL,
        eduID           CHAR(10) NOT NULL,
        status          VARCHAR(50),
        PRIMARY KEY( vetID, coordID, eduID ),
        FOREIGN KEY( vetID ) REFERENCES veterinarian,
        FOREIGN KEY( coordID ) REFERENCES coordinator,
        FOREIGN KEY( eduID ) REFERENCES educational_event );
```

2.16 Conservation Organization

Schema

conservation_organization(consID, totalMoneyRaised)

primary key(s) → consID

candidate key(s) → { (consID) }

foreign key(s) → consID references event (eventID)

Normal Form

BCNF

Table Definitions

```
CREATE TABLE conservation_organization (
        consID          CHAR(10) NOT NULL,
        totalMoneyRaised    DECIMAL(10, 2),
        PRIMARY KEY( consID ),
        FOREIGN KEY( consID ) REFERENCES event( eventID ) );
```

2.17 Make Donation

Schema

make_donation(visitorID, consID, donationAmount)

primary key(s) → visitorID, consID

candidate key(s) → { (visitorID, consID) }

foreign key(s) → visitorID references visitor

→ consID references conservation_organisation

Normal Form

BCNF

Table Definitions

```
CREATE TABLE make_donation (
    visitorID           CHAR(10) NOT NULL,
    consID              CHAR(10) NOT NULL,
    donationAmount      DECIMAL(10, 2),
    PRIMARY KEY( visitorID, consID ),
    FOREIGN KEY( visitorID ) REFERENCES visitor,
    FOREIGN KEY( consID ) REFERENCES conservation_organisation );
```

2.18 Group Tour

Schema

group_tour(groupID, price, tourGuide)

primary key(s) → groupID

candidate key(s) → { (groupID) }

foreign key(s) → groupID references event (eventID)

Normal Form

BCNF

Table Definitions

```
CREATE TABLE group_tour (
        groupID             CHAR(10) NOT NULL,
        price               DECIMAL(10, 2),
        tourGuide           VARCHAR(50),
        PRIMARY KEY( groupID ),
        FOREIGN KEY( groupID ) REFERENCES event( eventID ) );
```

2.19 Pay Price

Schema

pay_price(visitorID, groupID)

primary key(s) → visitorID, groupID

candidate key(s) → { (visitorID, groupID) }

foreign key(s) → visitorID references visitor

→ groupID references group_tour

Normal Form

BCNF

Table Definitions

```
CREATE TABLE pay_price (
        visitorID           CHAR(10) NOT NULL,
        groupID             CHAR(10) NOT NULL,
        PRIMARY KEY( visitorID, groupID ),
        FOREIGN KEY( visitorID ) REFERENCES visitor,
        FOREIGN KEY( groupID ) REFERENCES group_tour );
```

2.20 Request Refund

Schema

request_refund(visitorID, coordID, groupID)

primary key(s) → visitorID, coordID, groupID

candidate key(s) → { (visitorID, coordID, groupID) }

foreign key(s) → visitorID references visitor

→ coordID references coordinator

→ groupID references group_tour

Normal Form

BCNF

Table Definitions

```
CREATE TABLE request_refund (
        visitorID           CHAR(10) NOT NULL,
        coordID             CHAR(10) NOT NULL,
        groupID             CHAR(10) NOT NULL,
        PRIMARY KEY( visitorID, coordID, groupID ),
        FOREIGN KEY( visitorID ) REFERENCES visitor,
        FOREIGN KEY( coordID ) REFERENCES coordinator,
        FOREIGN KEY( groupID ) REFERENCES group_tour );
```

2.21 Complaint Form

Schema

complaint_form(formID, name, subject, message, date)

primary key(s) → formID

candidate key(s) → { (formID) }

Normal Form

BCNF

Table Definitions

```
CREATE TABLE complaint_form (
        formID          CHAR(10) NOT NULL AUTO_INCREMENT,
        name            VARCHAR(50),
        subject         VARCHAR(50),
        message         VARCHAR(300),
        date            DATE,
        PRIMARY KEY( formID ) );
```

2.22 Create Form

Schema

create_form(formID, visitorID, groupID)

primary key(s) → formID, visitorID, groupID

candidate key(s) → { (formID, visitorID, groupID) }

foreign key(s) → formID references complaint_form

→ visitorID references visitor

→ groupID references group_tour

Normal Form

BCNF

Table Definitions

```
CREATE TABLE create_form (
        formID          CHAR(10) NOT NULL,
        visitorID       CHAR(10) NOT NULL,
        groupID         CHAR(10) NOT NULL,
        PRIMARY KEY( formID, visitorID, groupID ),
        FOREIGN KEY( formID ) REFERENCES complaint_form,
        FOREIGN KEY( visitorID ) REFERENCES visitor,
        FOREIGN KEY( groupID ) REFERENCES group_tour );
```

2.23 Respond Form

Schema

respond_form(formID, visitorID, groupID, response, responseDate)

primary key(s) → formID, visitorID, groupID

candidate key(s) → { (formID, visitorID, groupID) }

foreign key(s) → formID references complaint_form

→ visitorID references visitor

→ groupID references group_tour

Normal Form

BCNF

Table Definitions

```
CREATE TABLE create_form (
        formID          CHAR(10) NOT NULL,
        visitorID       CHAR(10) NOT NULL,
        groupID         CHAR(10) NOT NULL,
        response        VARCHAR(300),
        responseDate    DATE,
        PRIMARY KEY( formID, visitorID, groupID ),
        FOREIGN KEY( formID ) REFERENCES complaint_form,
        FOREIGN KEY( visitorID ) REFERENCES visitor,
        FOREIGN KEY( groupID ) REFERENCES group_tour );
```

2.24 Comment

Schema

comment(commentID, comment)

primary key(s) → commentID

candidate key(s) → { (commentID) }

Normal Form

BCNF

Table Definitions

CREATE TABLE comment (

commentID CHAR(10) NOT NULL AUTO_INCREMENT,

comment VARCHAR(300),

PRIMARY KEY(commentID));

2.25 Write Comment

Schema

write_comment(commentID, visitorID, groupID)

primary key(s) → commentID , visitorID, groupID

candidate key(s) → { (commentID , visitorID, groupID) }

foreign key(s) → commentID references comment

→ visitorID references visitor

→ groupID references group_tour

Normal Form

BCNF

Table Definitions

CREATE TABLE write_comment (

commentID CHAR(10) NOT NULL,

visitorID CHAR(10) NOT NULL,

groupID CHAR(10) NOT NULL,

response VARCHAR(300),

responseDate DATE,

PRIMARY KEY(commentID, visitorID, groupID),

FOREIGN KEY(commentID) REFERENCES comment,

FOREIGN KEY(visitorID) REFERENCES visitor,
FOREIGN KEY(groupID) REFERENCES group_tour);

2.26 Animal

Schema

animal(animalID, name, isPureBreed, species, familyMembers, age, isSick)
primary key(s) → animalID
candidate key(s) → { (animalID) }

Normal Form

BCNF

Table Definition

CREATE TABLE comment (

animalID	CHAR(10) NOT NULL AUTO_INCREMENT,
name	VARCHAR(50),
isPureBreed	BOOL,
species	VARCHAR(50),
age	UNSIGNED INT,
isSick	BOOL,

PRIMARY KEY(animalID));

2.27 Treatment Request

Schema

treatment_request(keeperID, vetID, animalID, status)
primary key(s) → keeperID, vetID, animalID
candidate key(s) → { (keeperID, vetID, animalID) }
foreign key(s) → keeperID references keeper

→ vetID references veterinarian
→ animalID references animal

Normal Form

BCNF

Table Definitions

```
CREATE TABLE treatment_request (
        keeperID            CHAR(10) NOT NULL,
        vetID               CHAR(10) NOT NULL,
        animalID            CHAR(10) NOT NULL,
        status              VARCHAR(50),
        PRIMARY KEY( keeperID, vetID, animalID ),
        FOREIGN KEY( keeperID ) REFERENCES keeper,
        FOREIGN KEY( vetID ) REFERENCES veterinarian,
        FOREIGN KEY( animalID ) REFERENCES animal );
```

2.28 Schedule Training

Schema

schedule_training(keeperID, animalID, trainingDate)
primary key(s) → keeperID, animalID
candidate key(s) → { (keeperID, animalID) }
foreign key(s) → keeperID references keeper
→ animalID references animal

Normal Form

BCNF

Table Definitions

```
CREATE TABLE schedule_training (
        keeperID            CHAR(10) NOT NULL,
        animalID            CHAR(10) NOT NULL,
        trainingDate        DATE,
        PRIMARY KEY( keeperID, animalID ),
        FOREIGN KEY( keeperID ) REFERENCES keeper,
        FOREIGN KEY( animalID ) REFERENCES animal );
```

2.29 Prefers

Schema

prefers(animalID, foodID)

primary key(s) → animalID, foodID

candidate key(s) → { (animalID, foodID) }

foreign key(s) → animalID references animal

→ foodID references food

Normal Form

BCNF

Table Definitions

```
CREATE TABLE prefers (
        animalID            CHAR(10) NOT NULL,
        foodID              CHAR(10) NOT NULL,
        PRIMARY KEY( animalID, foodID ),
        FOREIGN KEY( animalID ) REFERENCES animal,
        FOREIGN KEY( foodID ) REFERENCES food );
```

2.30 Kept in

Schema

kept_in(animalID, cageID)

primary key(s) → animalID, cageID

candidate key(s) → { (animalID, cageID) }

foreign key(s) → animalID references animal

→ cageID references cage

Normal Form

BCNF

Table Definitions

```
CREATE TABLE kept_in (
        animalID            CHAR(10) NOT NULL,
        foodID              CHAR(10) NOT NULL,
        PRIMARY KEY( animalID, cageID ),
        FOREIGN KEY( animalID ) REFERENCES animal,
        FOREIGN KEY( cageID ) REFERENCES cage );
```

2.31 Food

Schema

food(foodID, name, expirationDate, amount, cost)

primary key(s) → foodID

candidate key(s) → { (foodID) }

Normal Form

BCNF

Table Definition

```
CREATE TABLE food (
        foodID              CHAR(10) NOT NULL AUTO_INCREMENT,
        name                VARCHAR(50),
```

```
    expirationDate          DATE,
    amount                  UNSIGNED INT,
    cost                DECIMAL(10, 2),
    PRIMARY KEY( foodID ) );
```

2.32 Regularize Food

Schema

regularize_food(keeperID, foodID, cageID)

primary key(s) → keeperID, foodID, cageID

candidate key(s) → { (keeperID, foodID, cageID) }

foreign key(s) → keeperID references keeper

→ foodID references food

→ cageID references cage

Normal Form

BCNF

Table Definitions

```
CREATE TABLE regularize_food (
    keeperID            CHAR(10) NOT NULL,
    foodID              CHAR(10) NOT NULL,
    cageID              CHAR(10) NOT NULL,
    PRIMARY KEY( keeperID, foodID, cageID ),
    FOREIGN KEY( keeperID ) REFERENCES keeper,
    FOREIGN KEY( foodID ) REFERENCES food,
    FOREIGN KEY( cageID ) REFERENCES cage );
```

2.33 Cage

Schema

cage(cageID, capacity, number)

primary key(s) → cageID

candidate key(s) → { (cageID) }

Normal Form

BCNF

Table Definition

```
CREATE TABLE cage (
        cageID              CHAR(10) NOT NULL AUTO_INCREMENT,
        capacity            UNSIGNED INT,
        number              UNSIGNED INT,
        PRIMARY KEY( cageID ) );
```

2.34 Assign

Schema

assign(keeperID, coordID, cageID)

primary key(s) → keeperID, coordID, cageID

candidate key(s) → { (keeperID, coordID, cageID) }

foreign key(s) → keeperID references keeper

→ coordID references coordinator

→ cageID references cage

Normal Form

BCNF

Table Definitions

```
CREATE TABLE assign (
        keeperID            CHAR(10) NOT NULL,
        coordID             CHAR(10) NOT NULL,
```

```
cageID                  CHAR(10) NOT NULL,
PRIMARY KEY( keeperID, coordID, cageID ),
FOREIGN KEY( keeperID ) REFERENCES keeper,
FOREIGN KEY( coordID ) REFERENCES coordinator,
FOREIGN KEY( cageID ) REFERENCES cage );
```

3.0 Functional Components

The functional components are given below in two different subheadings: use cases and algorithms.

3.1 Use Cases

The use cases are presented below.

3.1.1 User

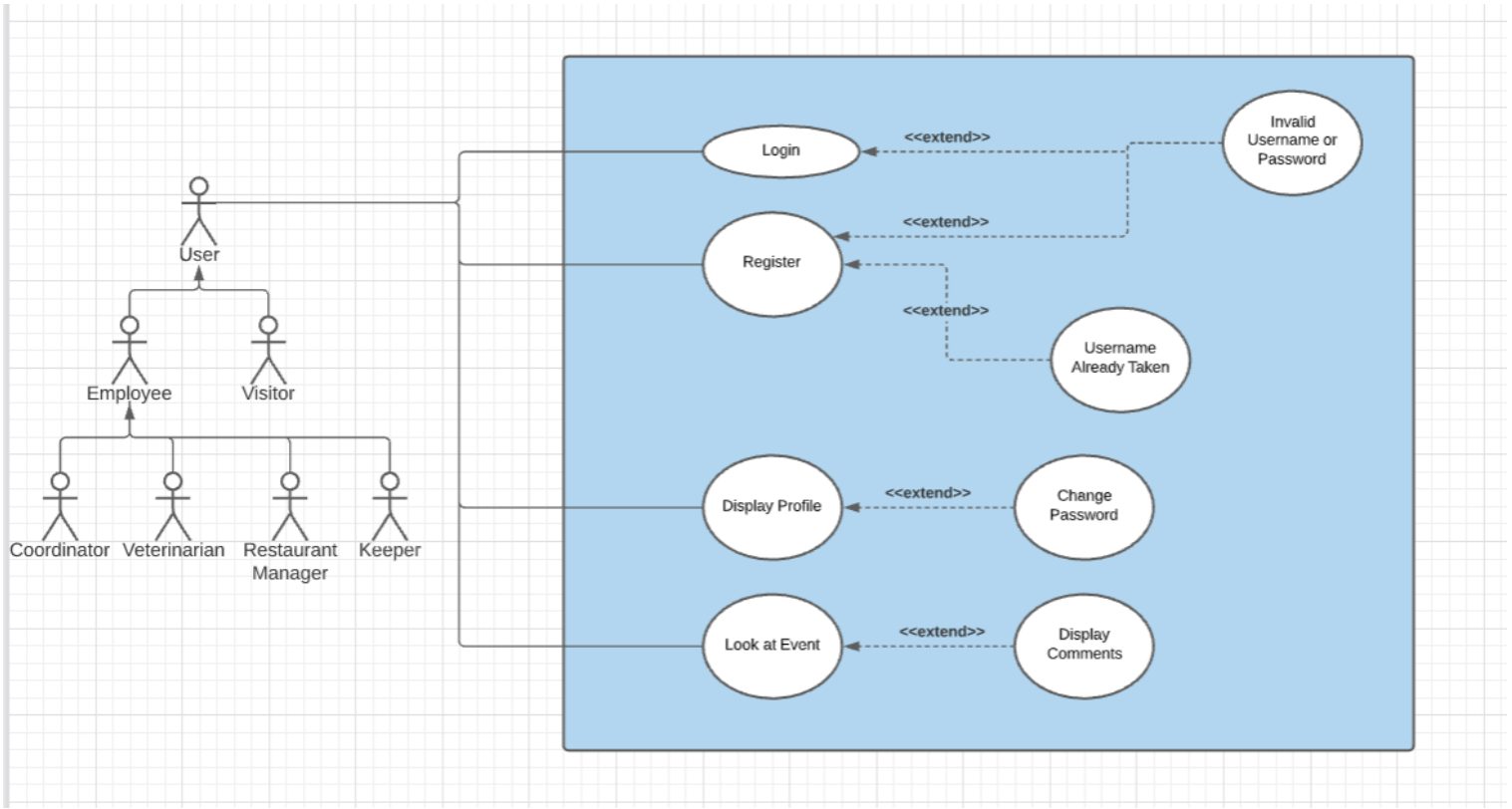


Figure 2: Use case diagram for actor “user”.

- **Login**

  A user can login to the system with their username and password.
- **Invalid Username or Password**

  This extends login and register. When a user gives the wrong password and username combination in the login case an error is given. When a user tries to make the password an non-alphanumeric value an error is given.
- **Register**

  A user can register with their username and password.
- **Username Already Taken**

  This extends register. When a user tries to take the name which is already taken.
- **Display Profile**

  A user can display profile information according to user type.
- **Change Password**

  This extends the display profile. When a user tries to change the password.
- **Look at Event**

  A user can choose to see the contents of an event.
- **Display Comments**

  This extends the “look at event” case. Displays events related to the event.

3.1.2 Visitor

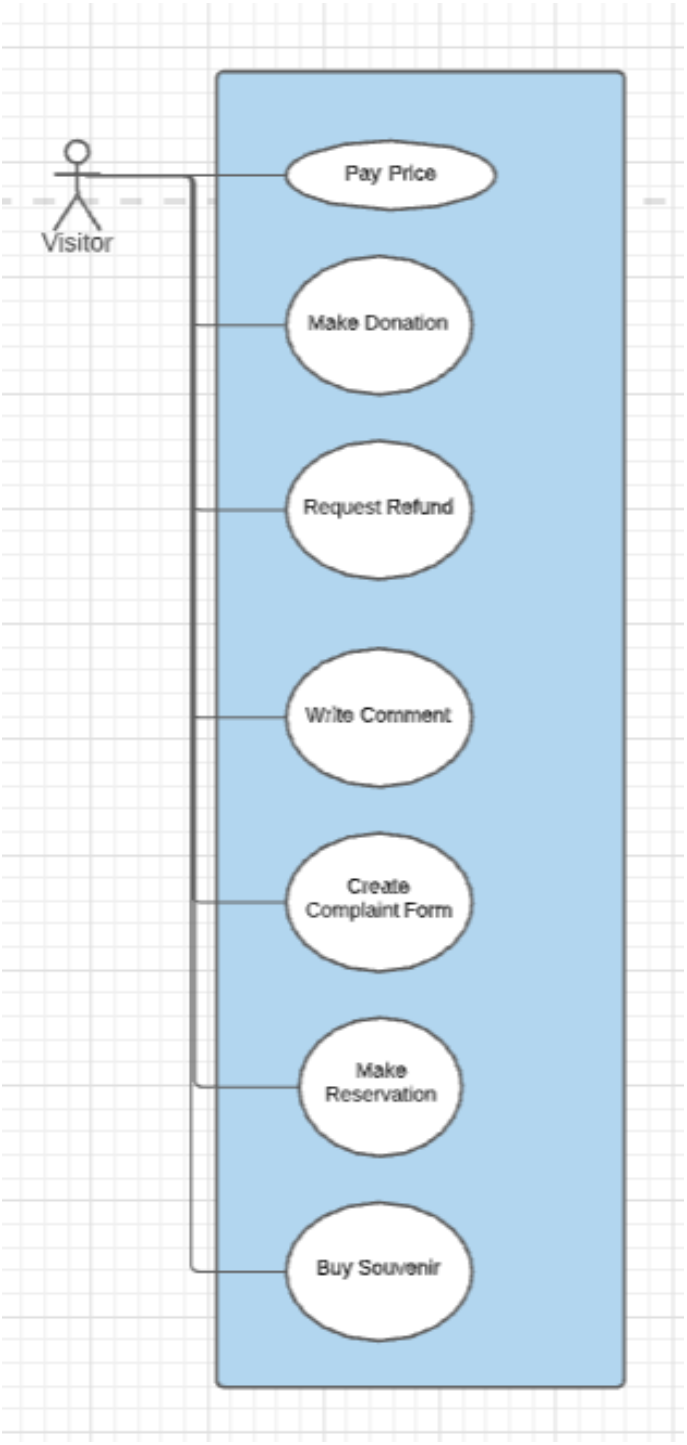


Figure 3: Use case diagram for actor “visitor”.

- **Pay Price**

  Visitors can pay the necessary ticket prices.
- **Make Donations**

Visitors can make donations to conservation organizations.

- **Request Refund**

  Visitors can request a refund for their tickets.

- **Write Comment**

  Visitors can write comments to event pages.

- **Create Complaint Form**

  Visitors can write a complaint for an event they have attended.

- **Make Reservation**

  Visitors can make reservations for the restaurant.

- **Buy Souvenir**

  Visitors can buy various souvenirs.

3.1.3 Keeper

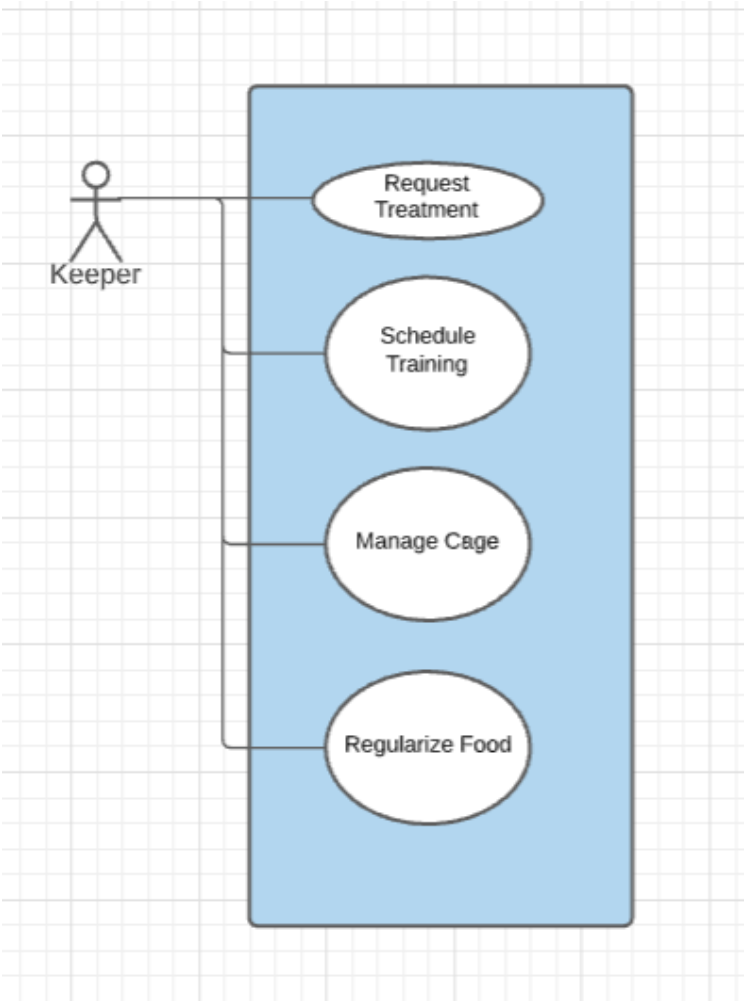


Figure 4: Use case diagram for actor “keeper”.

- **Request Treatment**

  Keepers can request treatment for the specific animal.
- **Schedule Training**

  Keepers can schedule training for the specific animal.
- **Manage Cage**

  Keepers are connected to their cages by coordinators.
- **Regularize Food**

  Keepers can regularize the given food to the animals.

3.1.4 Veterinarian

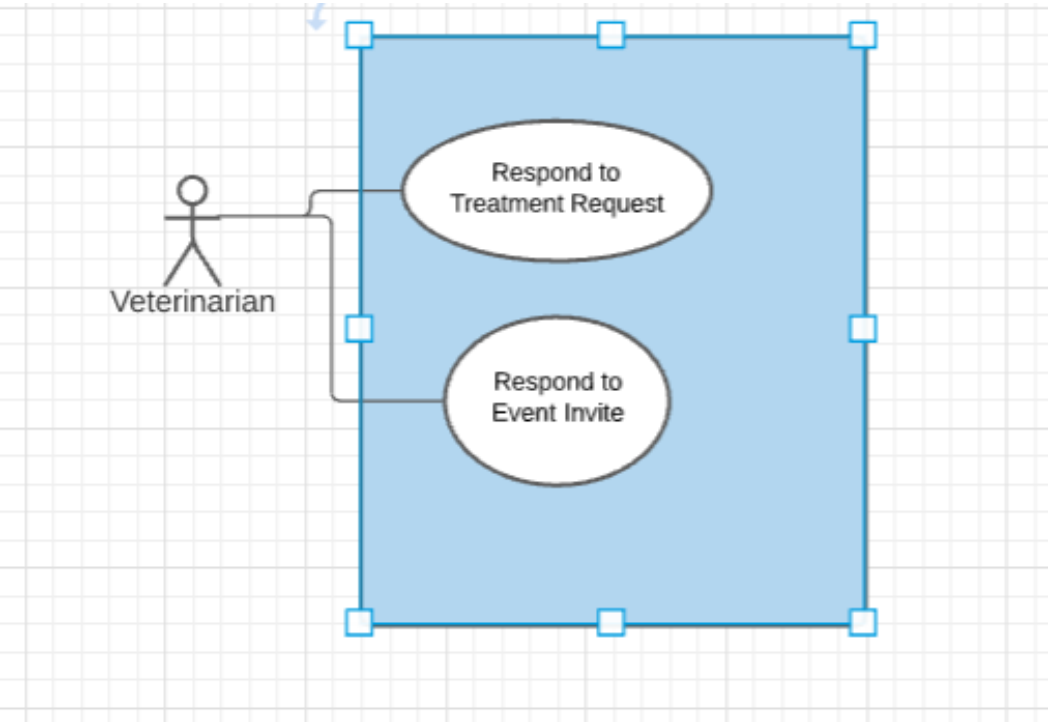


Figure 5: Use case diagram for actor “veterinarian”.

- **Respond to Event Invite**

  Veterinarians can respond to event invitations which are given by the coordinator.
- **Respond to Treatment Request**

  Veterinarians can respond to treatment requests which are given by the coordinator.

3.1.5 Coordinator

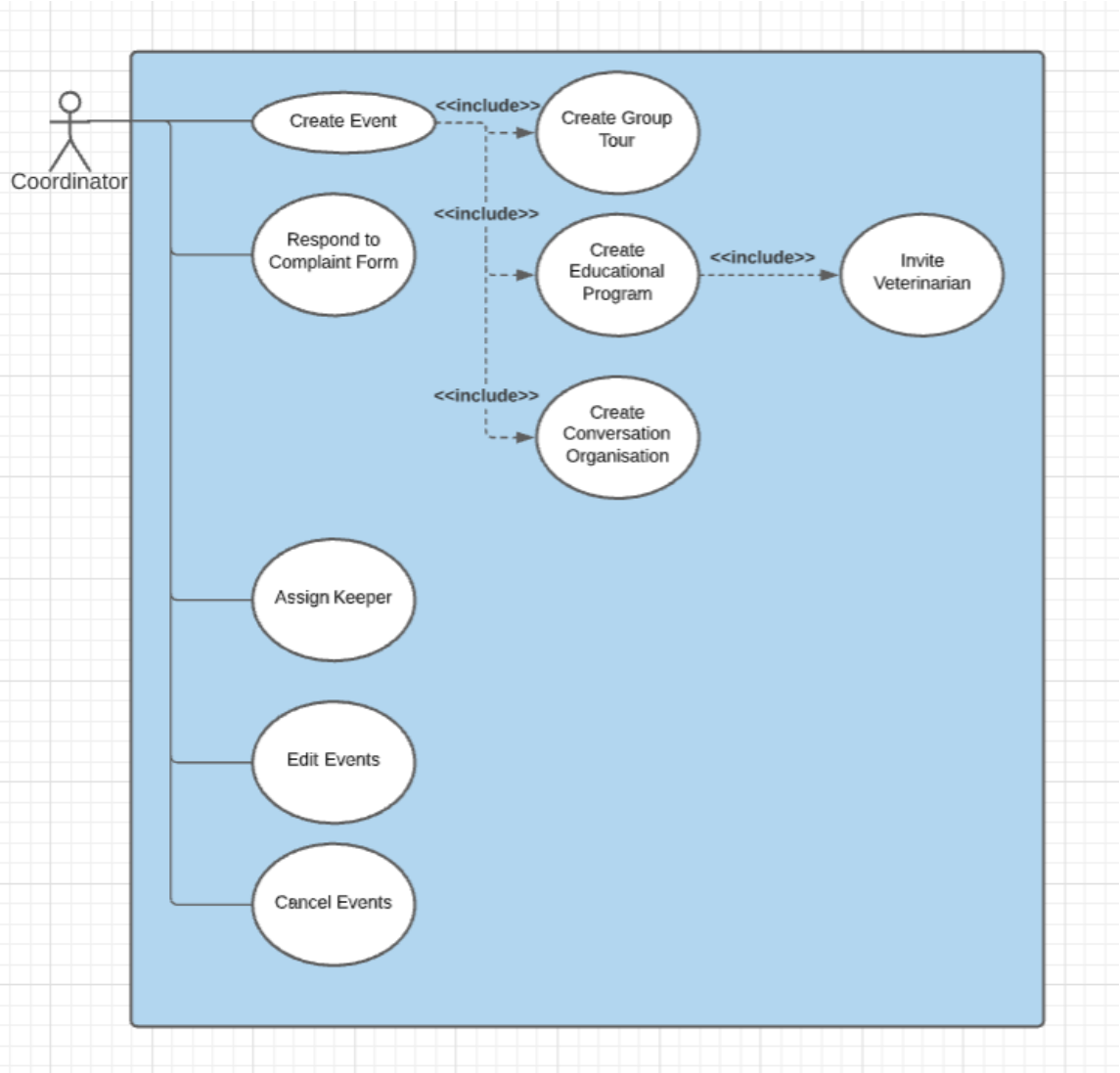


Figure 6: Use case diagram for actor “coordinator”.

- **Create Event**

  A coordinator can create new events.
- **Respond to Complaint Forum**

  A coordinator can respond to the complaint forums created by visitors.
- **Assign Keeper**

  A coordinator can assign keepers to cages.
- **Edit Events**

  A coordinator can make changes on already created events.
- **Cancel Events**

  A coordinator can cancel events.

- **Create Group Tour**
  This includes the “create event” case. A coordinator can create a group tour event.
- **Invite Veterinarian**
  This includes the “create educational program” case. A coordinator can invite a veterinarian to join the event.
- **Create Educational Program**
  This includes the “create event” case. A coordinator can create an educational program event.
- **Create Conservation Organization**
  This includes the “create event” case. A coordinator can create a conservation organization event.

3.2 Algorithms

The algorithms are given below.

3.2.1 Comment Related Algorithm

Visitors can write and edit the comments. Also, they can check the old events and related comments about the past events.

3.2.2 User-Interaction Algorithm

Four main users are visitors, keepers, coordinators and veterinarians. Users can see events. Visitors can write comments to these events or donate to them. Visitors can make reservations for the restaurant. Coordinators can invite veterinarians and assign keepers to cages. Keepers can request treatment and schedule training. Veterinarians can respond to treatment requests and event invites.

Every user has the right to register to the system. But, if they try to take an already taken username they will have an error message. Every user can look at his or her profile and they can change their passwords.

3.2.3 Cage Assigning Algorithm

A coordinator can choose a keeper from the database. Each keeper can have only one cage assigned to them. So, the keeper chosen from the database should not have any cages assigned to them. Then, the coordinator adds the connection to the ‘assign’ table.

4.0 User Interface Design and SQL Statements

Below are the user interface designs and related SQL statements for a variety of pages in two parts. The required functionalities part includes the common and topic specific functionalities, while the additional part includes all the rest.

4.1 Required Functionalities

The sign-up and login as well as the two additional features added are given down below.

4.1.1 Sign-up and Login

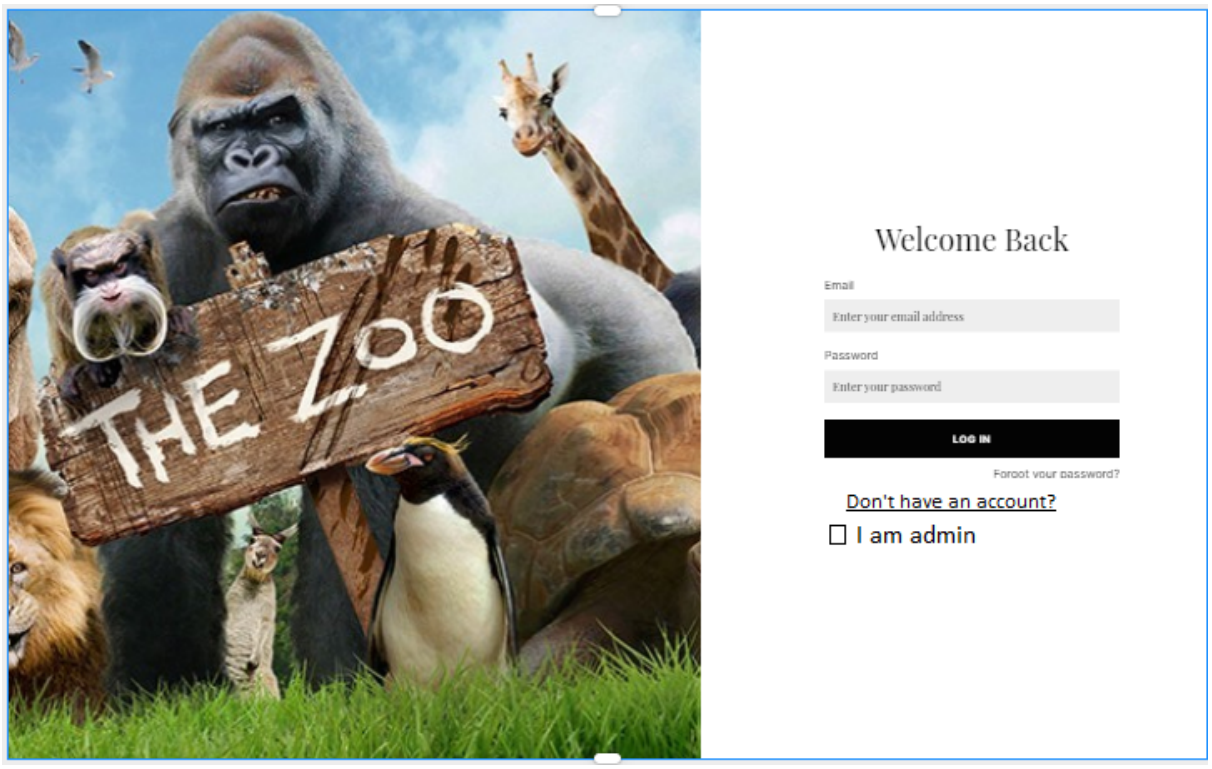


Figure 7: UI Design for the login page.

Process

Login page is the first page that every user sees when they go to the website. If the user is a visitor they will not check the “I am admin box”. They will simply enter their email and password to login to their account and will go to the visitor homepage. If they do not have an account they will click on “Don’t have an account?” link to go to the sign up page. Employees will simply check the box called “I am admin” and login to their account by entering their email and password as well.

Input

@username, @password

SQL

- **Login**

  SELECT *

  FROM user

  WHERE user.username = @username AND user.password = @password

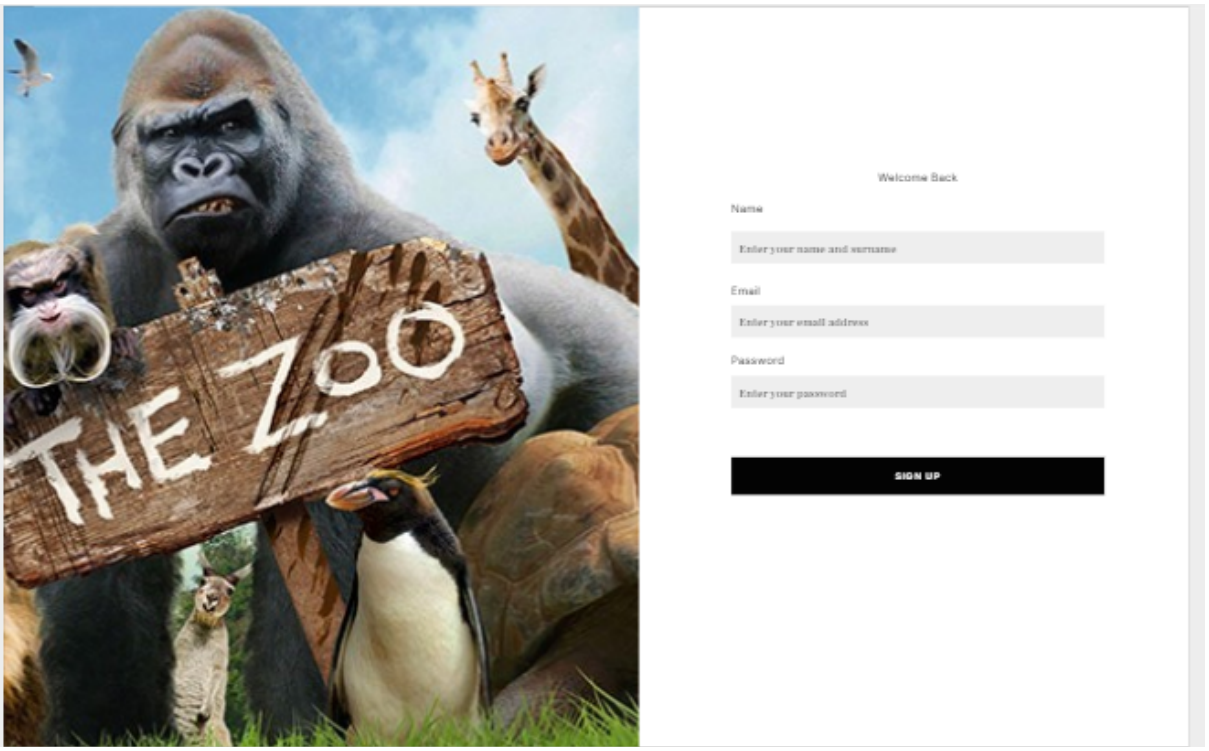


Figure 8: UI Design for the sign-up page.

Process

Register page is the page for new users to create their username, email and password combinations. If the user tries to take a name which is already taken, the system returns an

error message “This name is already taken.”. After users successfully register, they can login with the same information.

Input

@username, @email, @password

SQL

- **Sign-up for visitor**

```
INSERT INTO user (username, email, password)
        VALUES (@username, @email, @password);

INSERT INTO visitor (visitorID)
        SELECT MAX(userID)
        FROM user
```

4.1.2 Additional Feature 1 - Souvenir Shop

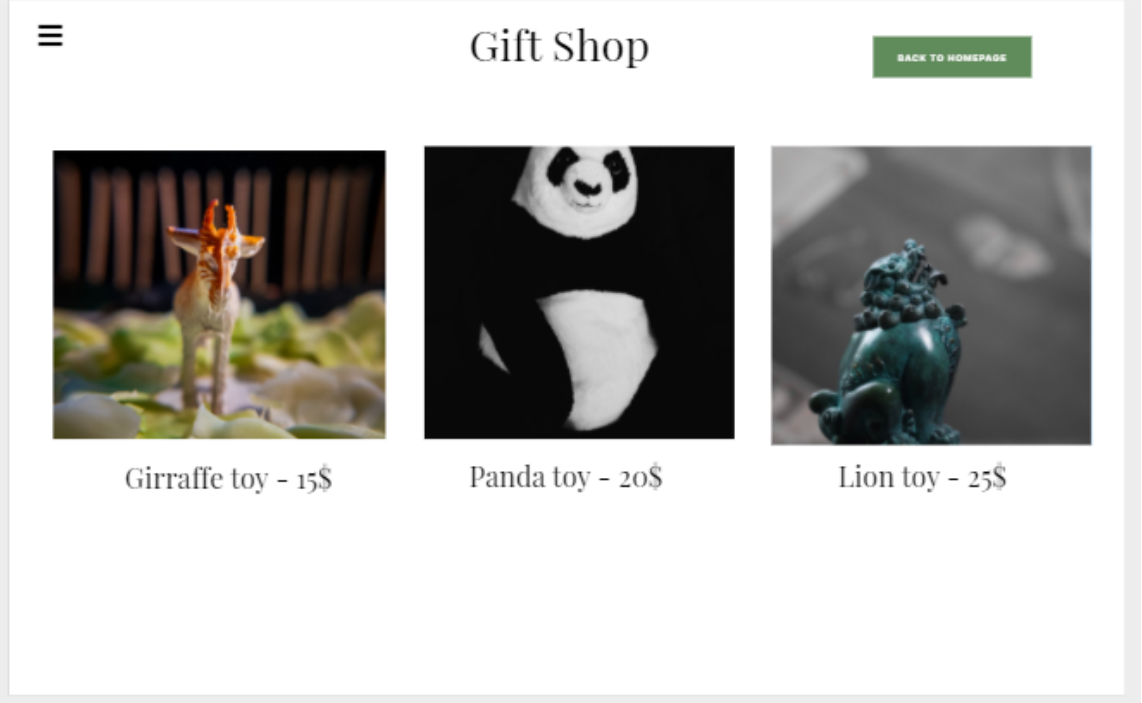


Figure 9: UI Design for the gift shop page.

Process

Gift or souvenir shop page will display many items that visitors can buy. When a visitor clicks on the item they will be directed to the payment page (Figure 26) where they

can cancel the payment or continue with the purchase. They can go back to the homepage by clicking the “Back to homepage” button.

SQL

- **Display all items in stock**

  SELECT name, price

  FROM souvenir

  WHERE stockAmount > 0

4.1.3 Additional Feature 2 - Restaurants

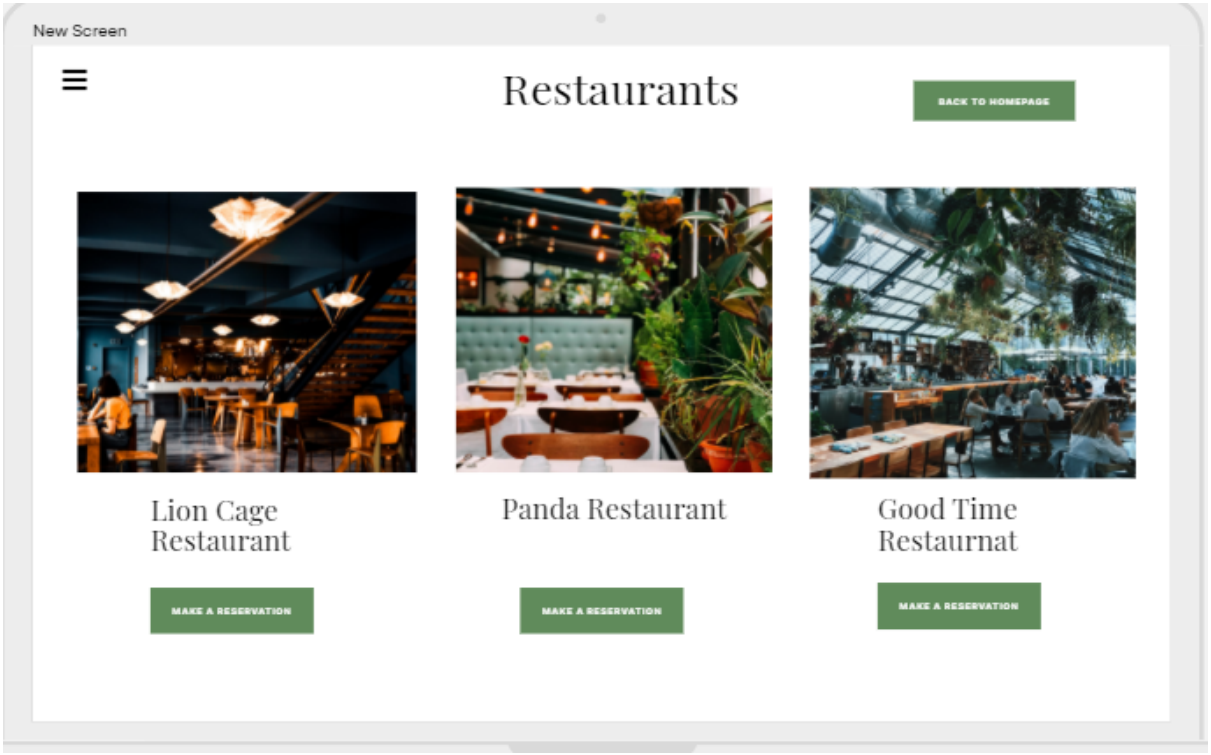


Figure 10: UI Design for the restaurants page.

Process

Visitors will be able to see all the restaurants that are in the zoo. They will be able to make a reservation from a desired restaurant. By clicking “Make a Reservation” button a modal will be shown to get the information needed to make a reservation from the visitor. Visitor can go back to the homepage by clicking the “Back to homepage” button.

SQL

- **Display all restaurants**

  SELECT name

  FROM restaurant

  WHERE capacity > 0

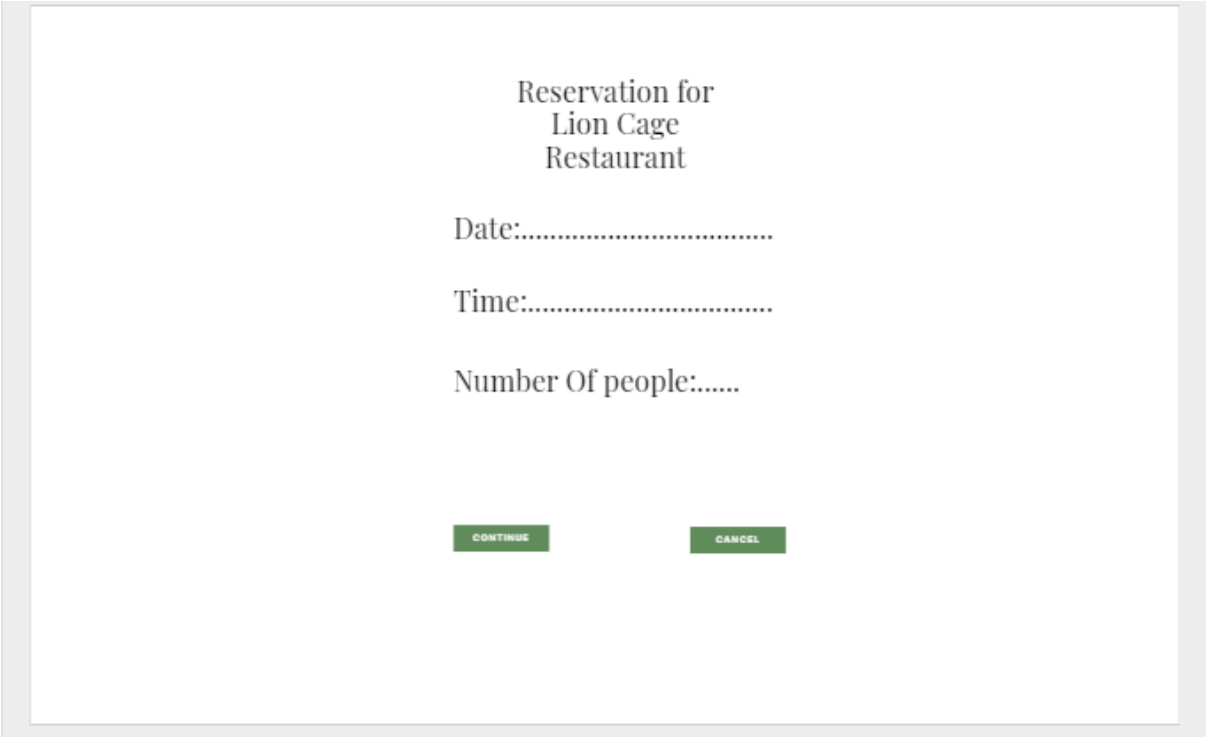


Figure 11: UI Design for the reservation page.

Process

Modal will require visitors to enter a date, time and number of people to make the reservation. Visitor can click “Continue” button to make the reservation and go back to the homepage. They can click “Cancel” to go back to the restaurants page to select a different restaurant.

Input

@visitorID, @restaurantID, @date, @time, @numberOfPeople

SQL

- **Make a reservation**

INSERT INTO make_reservation(visitorID, managerID, restaurantID, date, time,, numberOfPeople)

VALUES(@visitorID, (SELECT managerID

FROM manage

WHERE restaurantID = @restaurantID),
@restaurantID, @date, @time, @numberOfPeople);

4.1.4 Coordinator - Create Event Modal

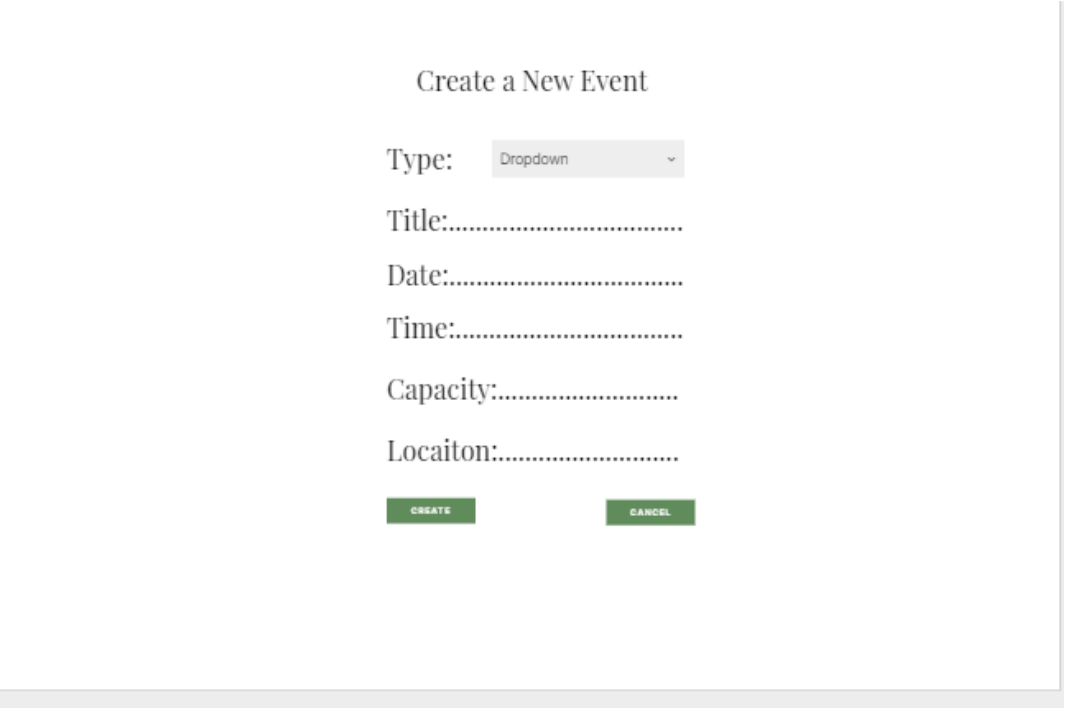


Figure 12: UI Design for the create event page for coordinators.

Process

Coordinator will select the type of the event which can be a group tour, education program or conservation organization. Depending on the type of the event additional price input will be displayed. Then the coordinator will give the needed information to create the new event. By clicking “Create” the event will be created and by clicking “Cancel” no creation will be done and the coordinator will be directed to their main page.

Input

@coordID, @type, @title, @date, @time, @capacity, @location, @duration

SQL

- **Create event**

INSERT INTO event (coordID, name, capacity, time, date, location)
VALUES(@coordID, @title, @capacity, @time, @date, @location, @duration);

Example below: if @type = educational_program

INSERT INTO educational_program (eduID)

```
SELECT MAX( eventID )
FROM event;
```

4.1.5 Coordinator - Assign Keeper to Cage Modal

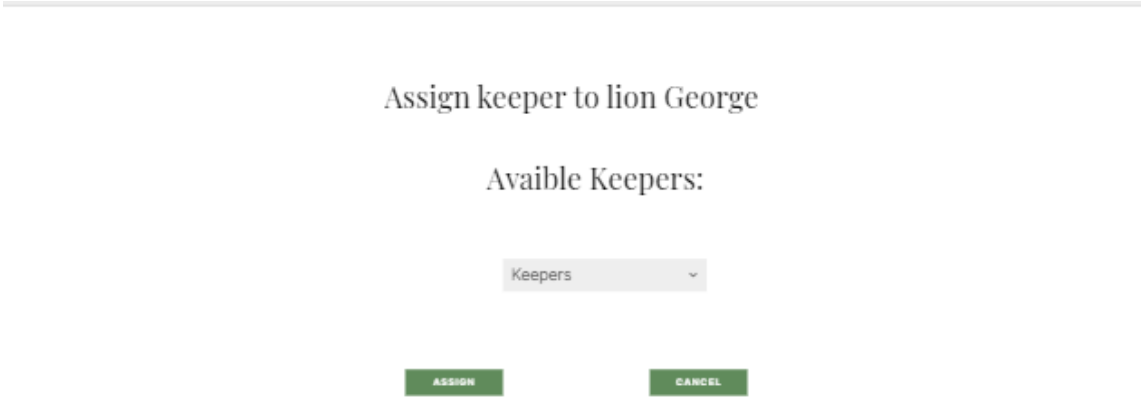


Figure 13: UI Design for the create assign keepers to cage page for coordinators.

Process

Coordinator will be directed to this page from the cage page. This page will be displayed for the cage that they have chosen. In this page coordinator will have a dropdown list that will have the available keepers and select one of them to assign them to that cage. By clicking “Assign” button keeper will be assigned.

SQL

- **Finding keepers who are not assigned to any cage**

```
SELECT *
FROM keeper K
WHERE NOT EXISTS (SELECT *
                  FROM assign A
                  WHERE A.keeperID = K.keeperID)
```

4.1.6 Keeper - List Cages Responsible For

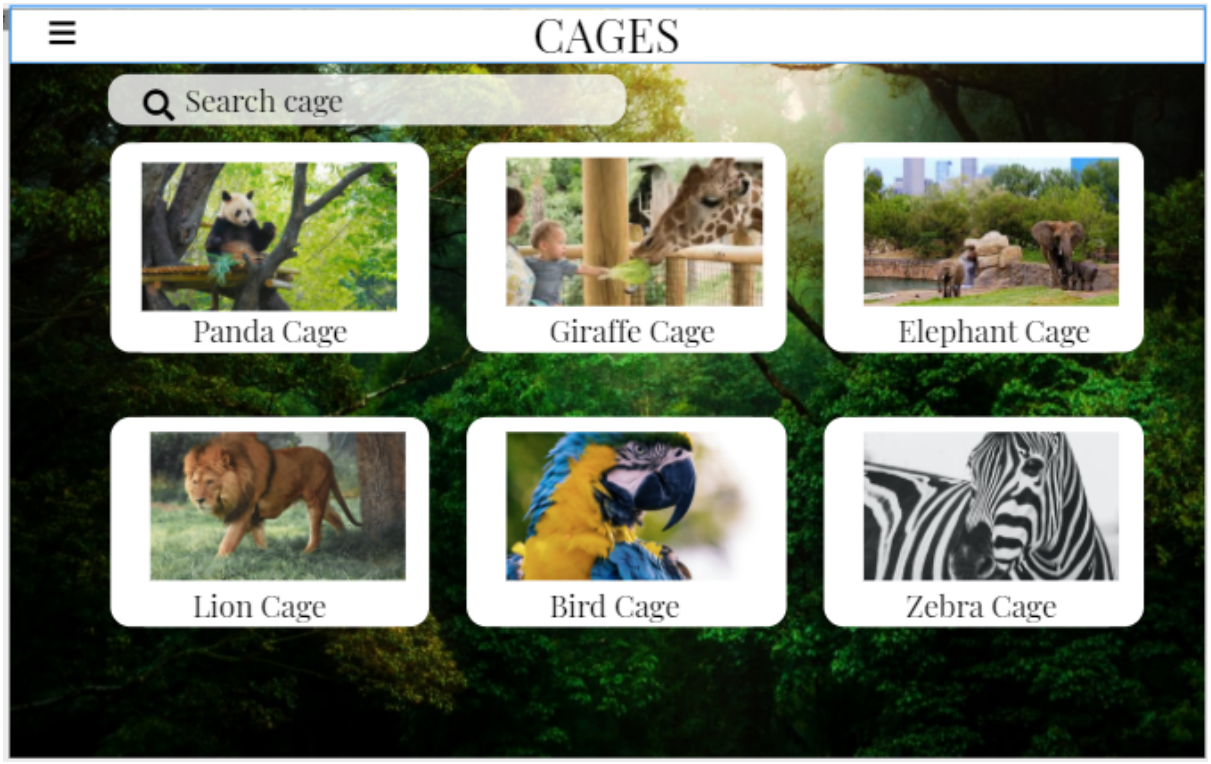


Figure 14: UI Design for the list cages responsible for page for keepers.

Process

In this page keepers will be able to see the cages that they are assigned to. By clicking one of the cards they will be directed to that cages information page (Figure 15).

SQL

- **List Cages**

  SELECT C.cageID, C.number

  FROM keeper K, cage C, assign A

  WHERE K.keeperID = A.keeperID and A.cageID = C.cageID;

4.1.7 Keeper - Regularize Food

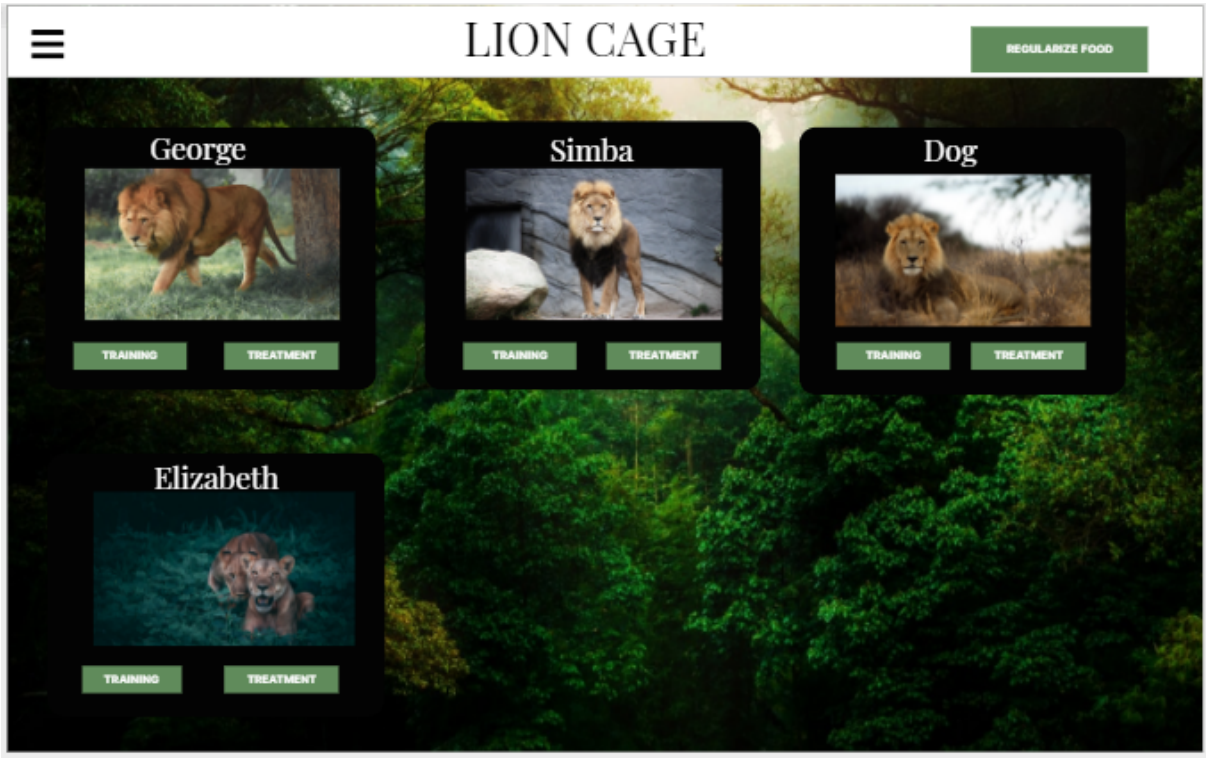


Figure 15: UI Design for the regularize food page for keepers.

Process

Keeper will be directed to this page from the cage page. All the animals that are in the cage are displayed. Keeper can click "Training" to schedule a training for that animal. Keeper can click "Treatment" to schedule a treatment for that animal. finally by clicking the "Regularize food" button to regularize food for that cage.

Input

@keeperID, @foodID, @cageID

SQL

- **Regularize new food**

  INSERT INTO regularize_food (keeperID, foodID, cageID)

  VALUES (@keeperID, @foodID, @cageID);

4.1.8 Visitor - Make Donation Modal

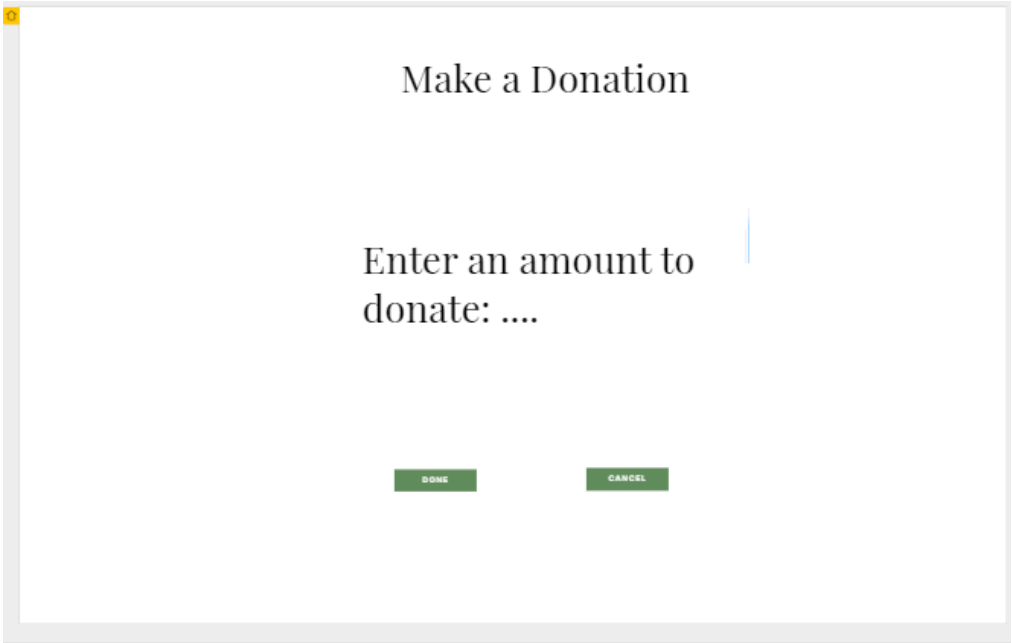


Figure 16: UI Design for the make donation page for visitors.

Process

This page will be displayed if a visitor joins a conversation organization. They will be asked to enter an amount to donate and by clicking “Done” they will be directed to the payment page. By clicking “Cancel” visitors will go back to their homepage.

Input

@visitorID, @consID, @donationAmount

SQL

- **Make Donation**

```
INSERT INTO make_donation
        VALUES (@visitorID, @consID, @donationAmount);

UPDATE conservation_organization
        SET totalMoneyRaised = totalMoneyRaised + @donationAmount
        WHERE consID = @consID
```

4.1.9 Visitor - Comment on Group Tours Modal

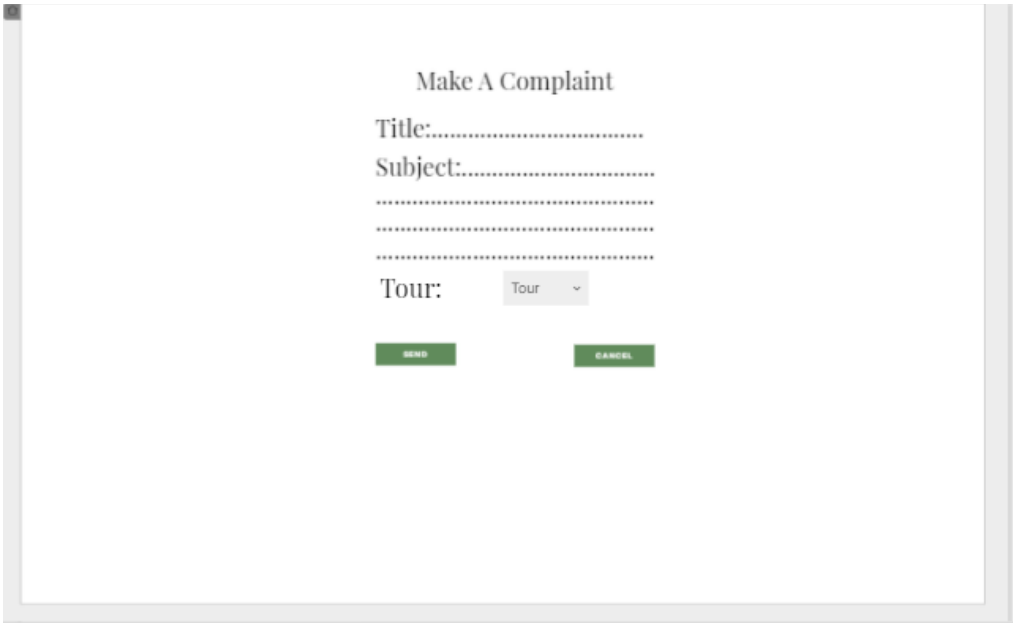


Figure 17: UI Design for the comment on group tours page for visitors.

Process

Visitors will select the tour that they want to complain about. Dropdown list will have all the events that a visitor has gone to. They will enter a title and the complaint. By clicking the “Send” button to send the complaint.

Input

@visitorID, @groupID, @subject

SQL

- **Make a new Comment**

```
INSERT INTO comment (comment)
        VALUES ( @subject );

INSERT INTO write_comment ( eduID )
        SELECT MAX( commentID )
        FROM comment;
```

4.2 Additional UI Designs

The UI designs for all other major functionalities are given below, with brief explanations.

4.2.1 User Home Page

Figure 18: UI design for user home page.

Process

This page is the first page that is displayed to the visitors after logging in. All the events are displayed categorized by group tours and conservation organizations on this page. By clicking the desired event type that is displayed at the bottom of the page all the events related to that event will be shown. By clicking the “Join” button a modal will be shown with event details. Visitors will be able to go to restaurants by clicking the “Restaurants” button and will be able to go to the gift shop page by clicking the “Gift Shop” button.

4.2.2 Coordinator - Invite Veterinarian Modal

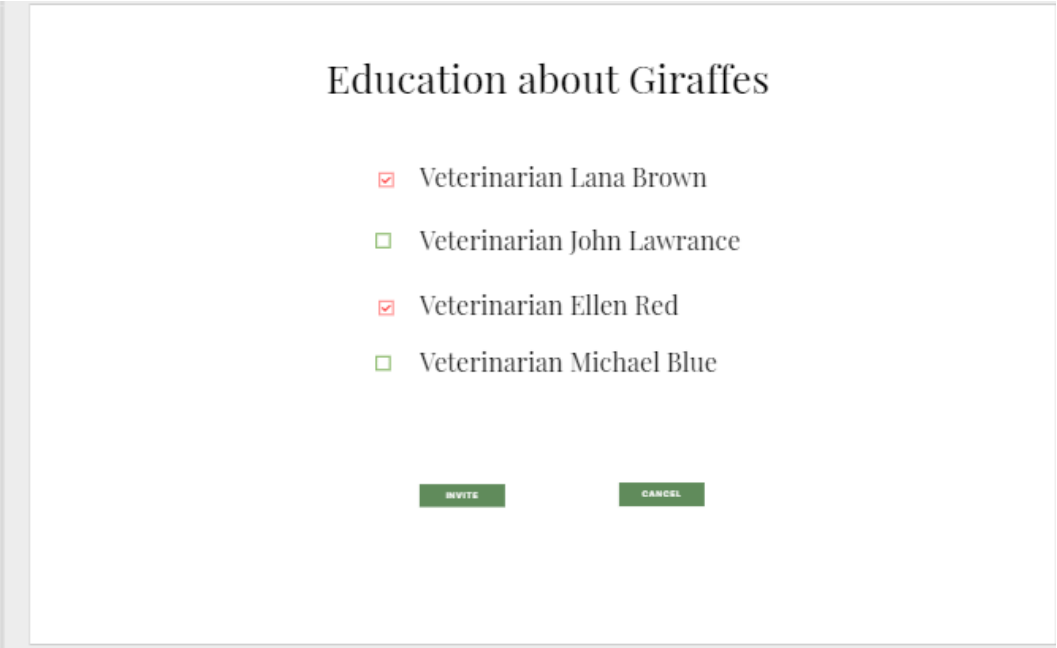


Figure 19: UI design for treatment requests for veterinarians.

Process

Coordinator will select an education program from their list (Figure 24) and by clicking the event they will be directed to this page. All the veterinarians will be displayed and the coordinator will select the ones they want to invite. By clicking the “Invite” button invitations will be sent.

4.2.3 Coordinator - Respond to Complaint Forms Modal

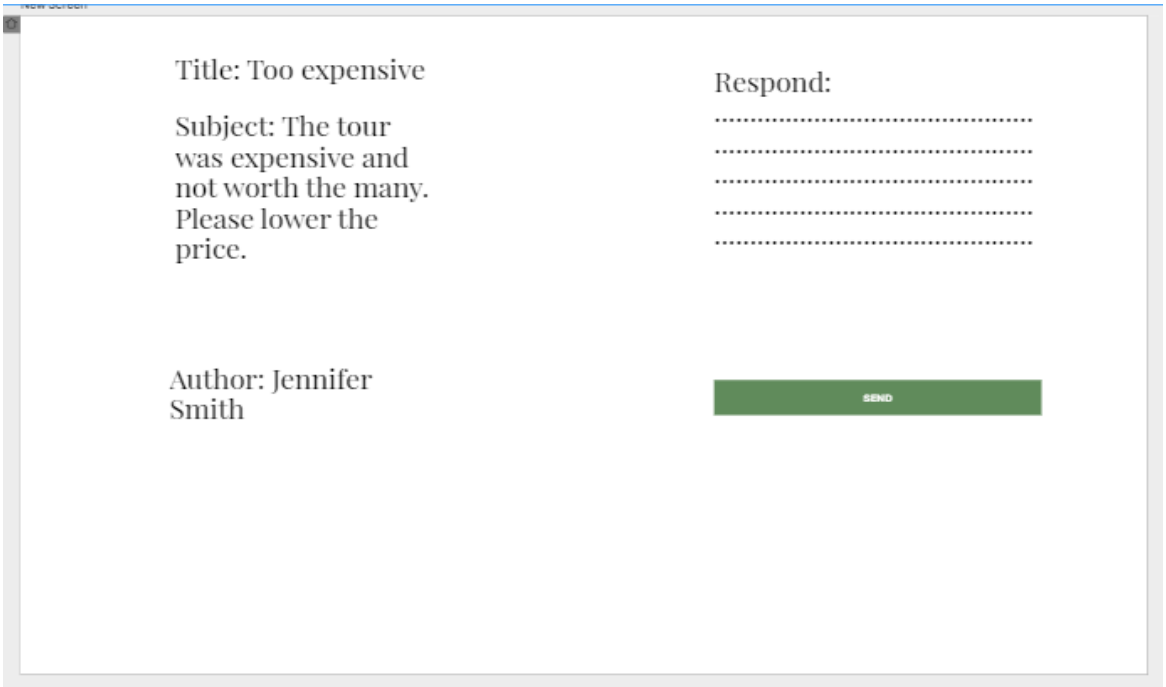


Figure 20: UI design for responding to the complaint forms.

Process

Coordinator will have a list of all the complaints and by selecting one of them they will be directed to this page. In here they will be able to see the details of the page and respond accordingly. By clicking “Send” respond will be sent to their email address.

4.2.4 Coordinator - Animal Page

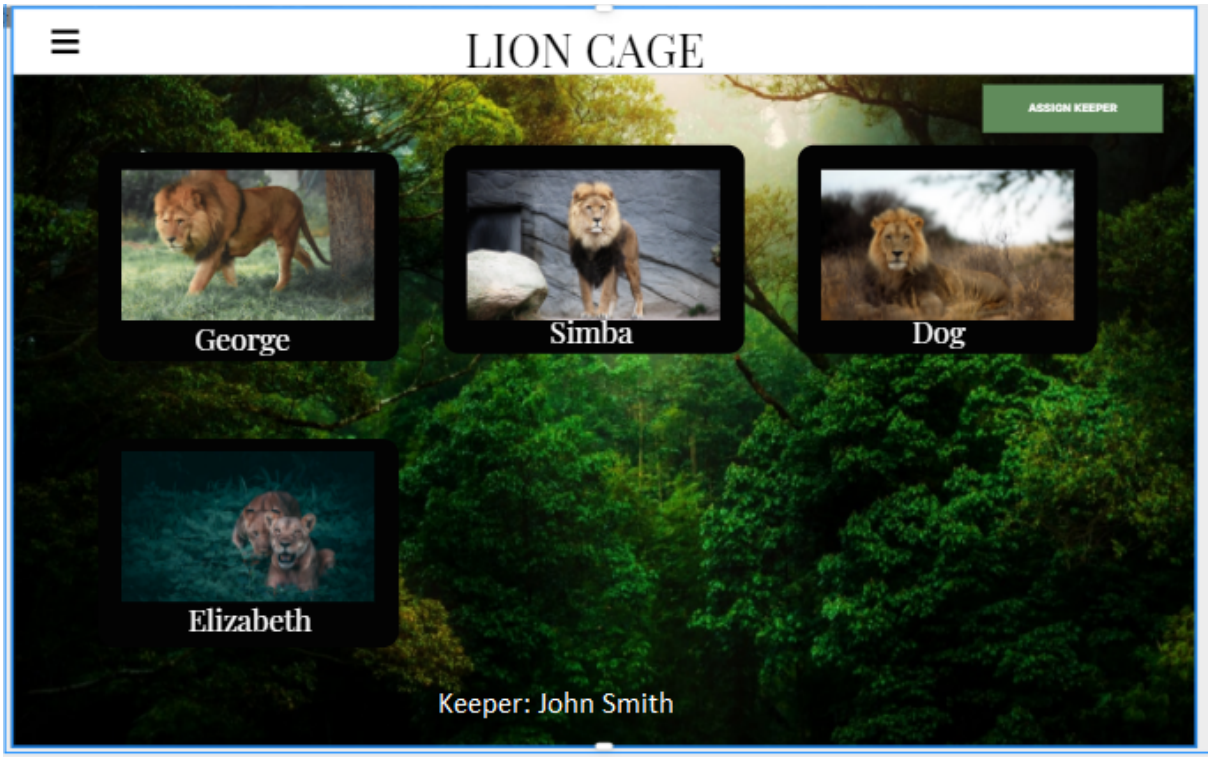


Figure 21: UI Design for the animal page for coordinators.

Process

This is the page for coordinators where they will be able to see the animals of the cage that they have chosen from the cage page (Figure 23.). Keeper name who is responsible for that cage will be shown on the bottom of the page. By clicking the animal they will be directed to the animal information page (Figure 22). By clicking the “Assign Keeper” button they modal for assigning will be shown (Figure 13).

4.2.5 Coordinator - Animal Information Page

Figure 22: UI Design for the animal information page for coordinators.

Process

Here the animal's information is displayed along with the scheduled training and treatments if they have any.

4.2.6 Coordinator - Cage Page

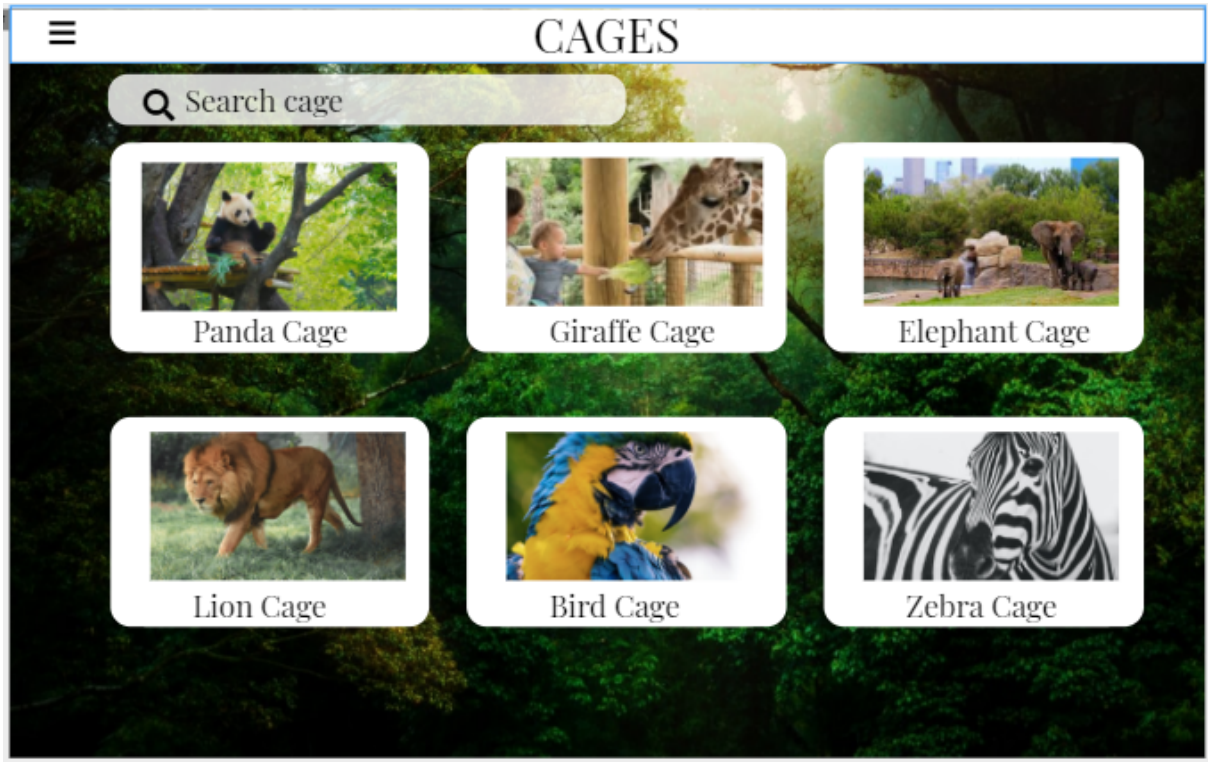


Figure 23: UI Design for the cage page for coordinators.

Process

In here coordinators will be able to see all the cages of the zoo. By clicking one of the cards they will be directed to that cage's information page (Figure 22).

SQL

- **List all the cages that are assigned to keepers by this coordinator with necessary information related to each cage. - Input: @coordID**

  SELECT C.number, C.capacity

  FROM cage C, assign A

  WHERE A.coordID = @coordID AND A.cageID = C.cageID

4.2.7 Coordinator - Display Events Page

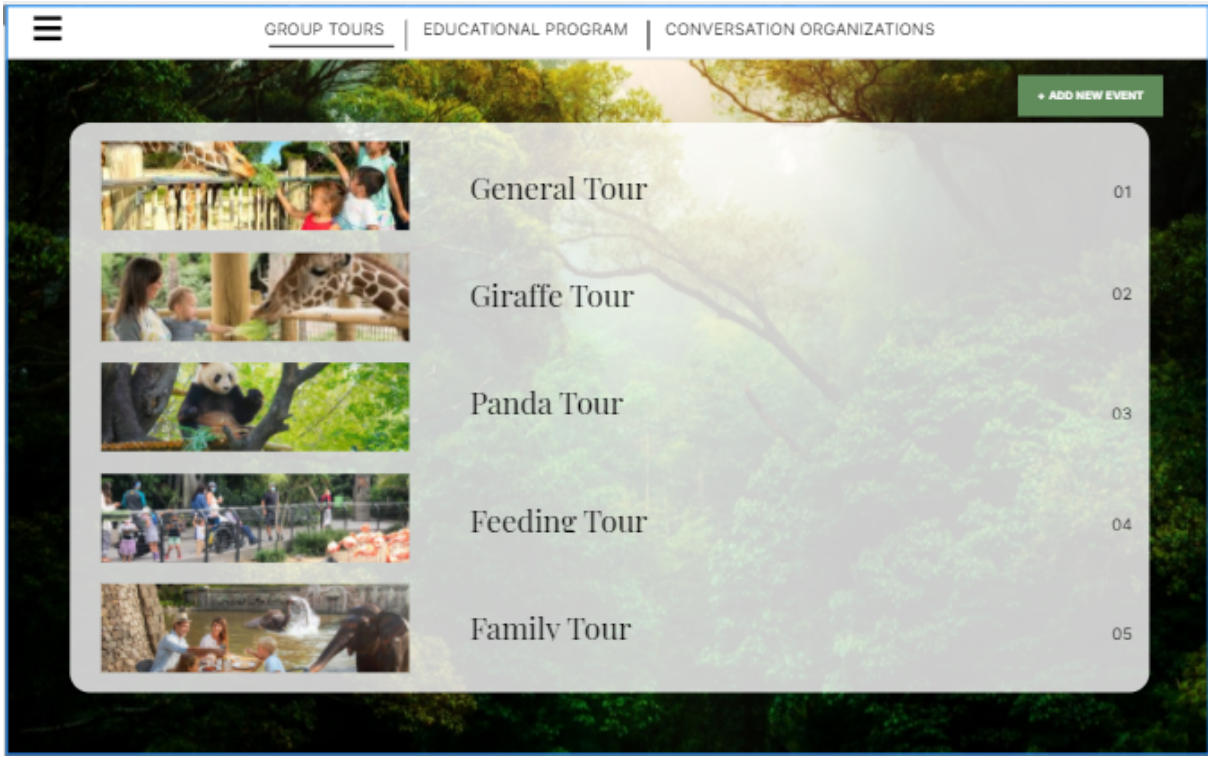


Figure 24: UI Design for the display events page for coordinators.

Process

This page displays all the events in a categorized manner for all the types of events. Coordinator will be able to see all the events and add new events by clicking the “Add New Event” button where a modal is shown(Figure 12).

4.2.8 Veterinarian- Display Events Page

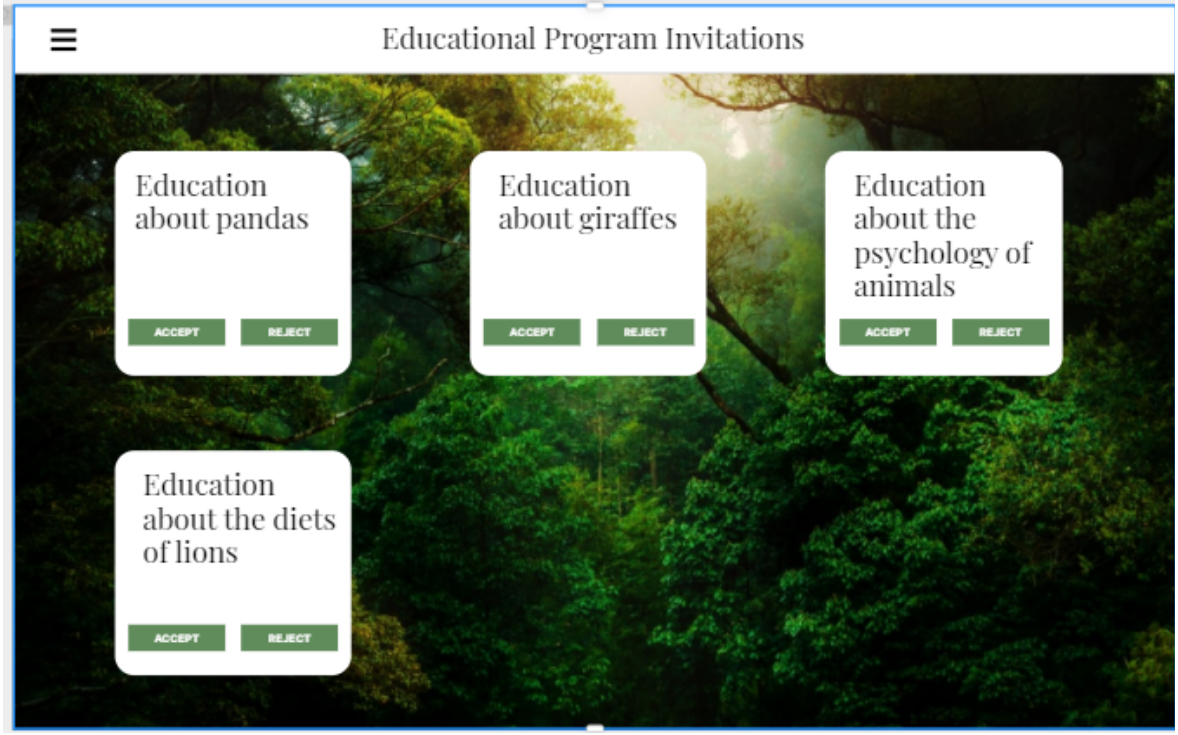


Figure 25: UI Design for the display events page for veterinarians.

Process

This page displays all the invitations that are sent to that veterinarian. Veterinarians will be able to accept the invitation by clicking “Accept” and reject it by clicking “Reject”.

4.2.9 Visitor - Payment Page

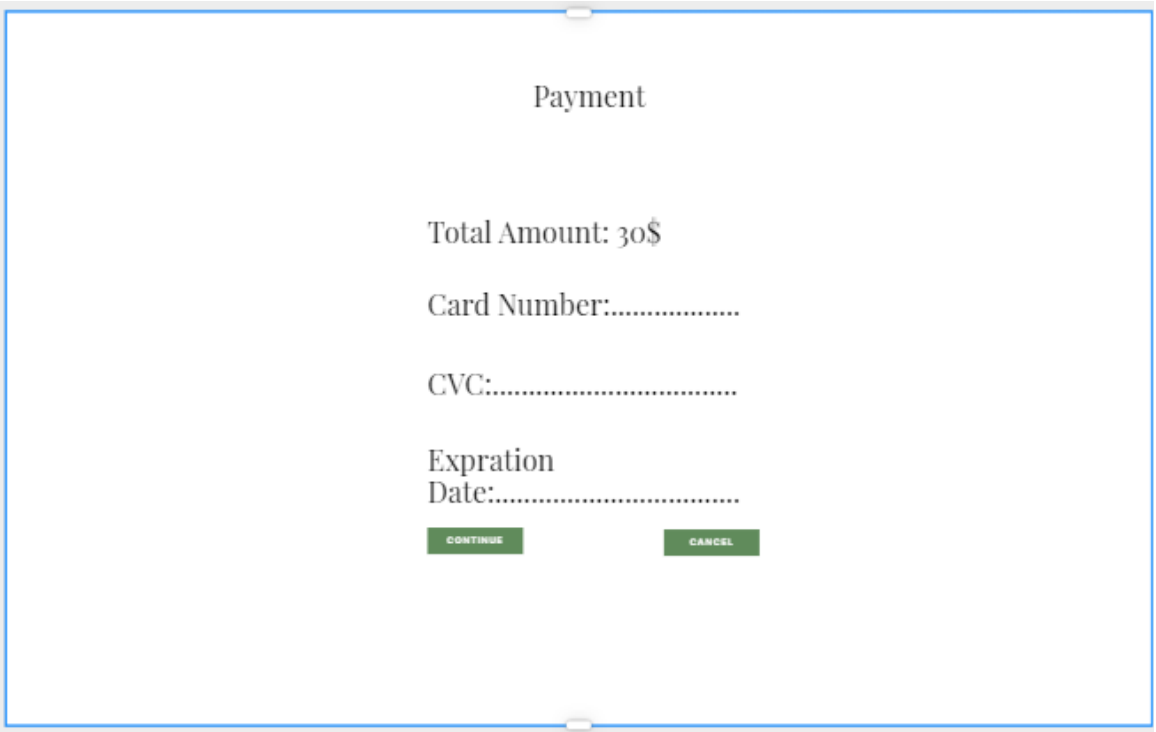


Figure 26: UI Design for the payment page for visitors.

Process

Visitors will be directed to this page when they want to purchase something from the gift shop or join a group tour that they have selected or when they want to make a donation. Amount that they will have to pay will be displayed at the top. All the card information will be written to make the purchase like card number, CVC and expiration date of the card. By clicking the “Continue” button they will complete the purchase and by clicking “Cancel” they will cancel the purchase and go back to their homepage.

4.2.10 Veterinarian- Display Treatment Requests Page

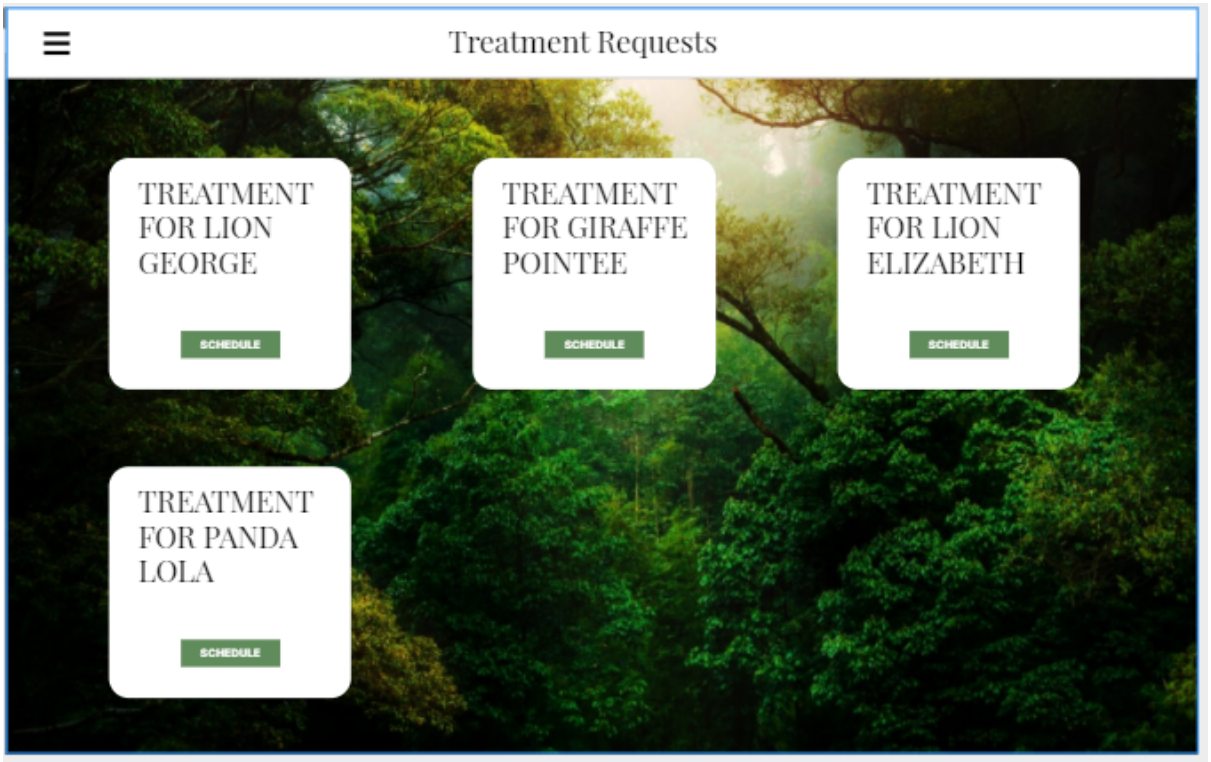


Figure 27: UI Design for the display treatment requests page for veterinarians.

Process

Veterinarians will be able to see all the requests for treatment and will be able to schedule that treatment by clicking the “Schedule” button which will direct them to the scheduling a date for a treatment page (Figure 30).

4.2.11 Visitor - Profile Page

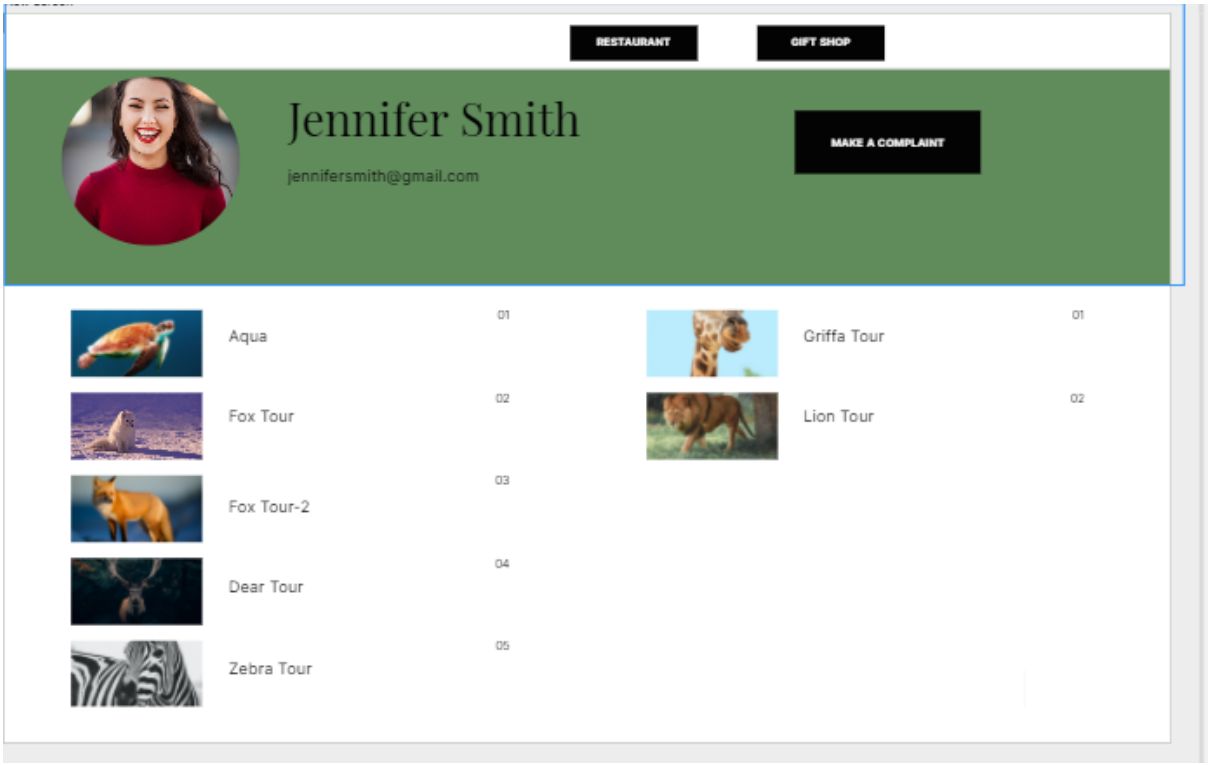


Figure 28: UI Design for the profile page for visitors.

Process

This page will display the visitors information like name, surname, email address and all the events that they have registered to. By clicking the "Make a Complaint" button complaint modal will be shown (Figure 17) where they can make their complaints. "Restaurant" and "Gift Shop" buttons are displayed if they want to go to those pages.

4.2.12 Visitor - Event Info Modal

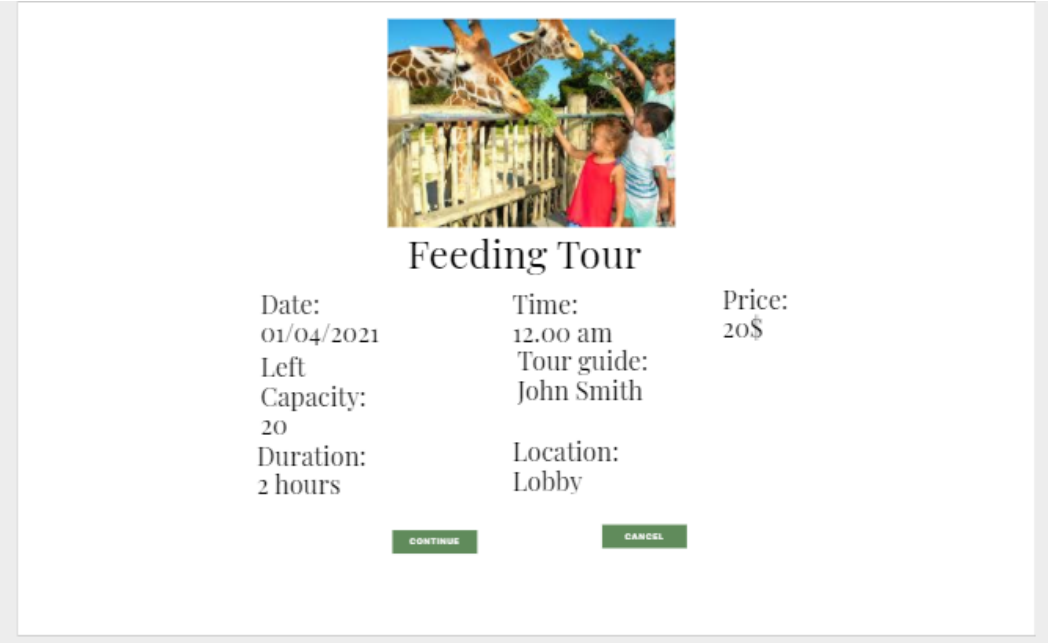


Figure 29: UI Design for the event detail modal for visitors.

Process

When a visitor clicks on an event card displayed in Figure … this modal will be shown where it has all the information about the event. By clicking the "Continue" button visitors will be directed to the "Payment" page (Figure 26) where they will continue with the purchasing of the event. By clicking the "Cancel" button modal will be closed and the visitor will be on the homepage again (Figure 18).

4.2.13 Veterinarian- Scheduling Date Modal

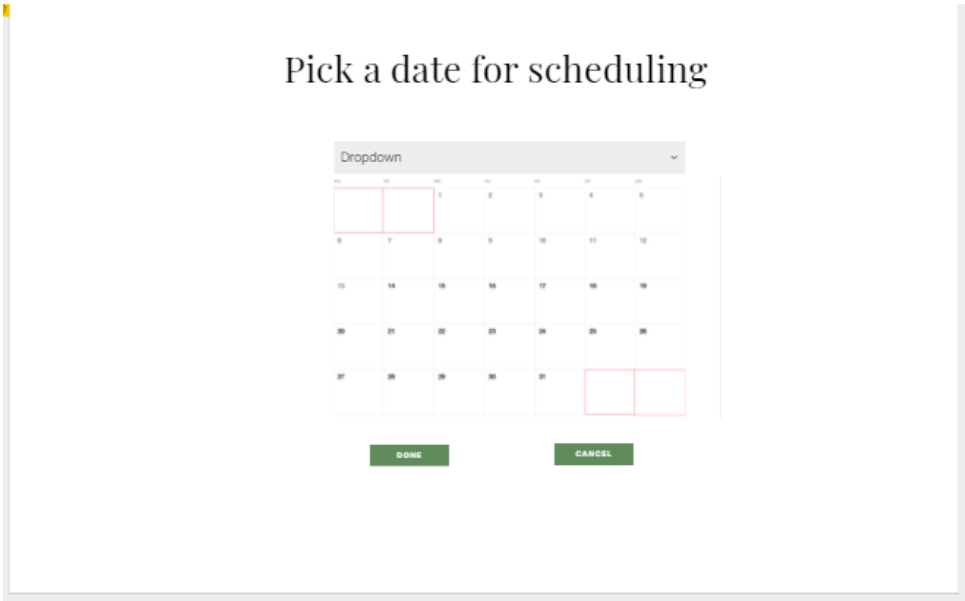


Figure 30: UI Design for the date selection modal for veterinarians.

Process

When a veterinarian decides to accept a treatment request and clicks on the "Schedule" button in Figure 27 this modal will be shown where they can pick a date to schedule. By clicking the "Done" button scheduling will be done and by clicking the "cancel button" modal will be closed without any scheduling done.

4.2.14 Keeper - Training Schedule

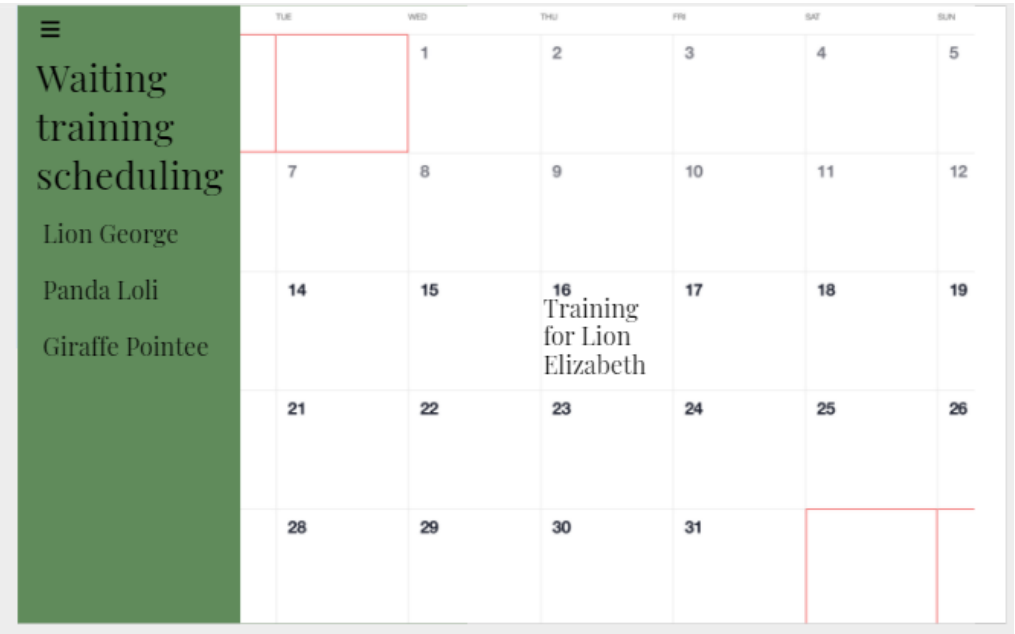


Figure 31: UI Design for the training schedule for keepers

Process

In this page the keeper will be able to see the animals who are waiting for a scheduling of their training along with a calendar which contains all the previous appointments. Keeper will be able to put the animals to the available spots to make the scheduling.

4.2.15 Veterinarian- Treatment Calendar Page

Figure 32: UI Design for the treatment schedule for veterinarians.

Process

This page displays all the scheduled treatments for the animals in a calendar.

5.0 Advanced Database Components

The advanced database components are given below in headings views, reports, triggers and constraints.

5.1 Views

Available Free Group Tours

```
CREATE view [AvailableFreeTours] as
    SELECT *
    FROM group_tour
    WHERE price = 0.00 ;
```

Active Treatment Requests

```
CREATE view [ActiveTreatmentRequests] as
    SELECT *
    FROM treatment_request
    WHERE status = “open”;
```

Currently Working Keepers

```
CREATE view[CurrentlyWorkingKeepers] as
    SELECT *
    FROM keeper K, assign A
    WHERE K.keeperID = A.keeperID
```

5.2 Reports

Individuals That Have Made Reservation For Tomorrow

```
SELECT userID
```

```
FROM user, make_reservation
WHERE visitorID = userID and
        date >  @current_date;
```

Number Of Events That Are More Than 50 Capacity And Named As Whale Performance

```
SELECT count(eventID)
FROM event
WHERE capacity > 50 and
        name = “Whale Performance”;
```

Number Of Educational Program With 30 Capacity

```
SELECT count(eduID)
FROM event NATURAL JOIN educational_program
WHERE eduID = eventID AND
        capacity = 30;
```

Number Of Treatment Requests That Are Made To Veterinarian Who Expertise Lion

```
SELECT count(vetID)
FROM treatment_request T, veterinarian V
WHERE T.vetID = V.vetID AND
        V.expertise = “Lion”;
```

Tomorrow Events Which Are Done By Coordinator Ahmet

```
SELECT E.eventID
FROM event E
WHERE E.capacity > 30 and
        date = @current_date + 1
        E.coordID = (SELECT coordID
                        FROM coordinator, employee
                        WHERE coordID = employeeID and employeeID IN
                        (SELECT userID
                          FROM user, employee
```

```
WHERE userID = employeeID AND
    name = “Ahmet” ) );
```

5.3 Triggers

- When the event is cancelled, it is removed from all pages that contain that event.
- When the user profile is deleted, all of its complaint forms about events are also deleted.
- When an animal in the treatment period, that cage will be unavailable.
- When the restaurant capacity reaches full and the user tries to make a reservation from that place.

5.4 Constraints

- When the date of the event is passed, it does not show that event anymore at the events page.
- Complaint forms have char limitations for storage efficiency.
- When assigning a keeper to the cage, the program shows only the available keepers.
- When adjusting the schedule for treatment requests, the program shows only available dates.
- Only the visitors attending events can create a complaint form.
- All events and cages are searchable.
- When the treatment request is accepted, that cage will be closed for a given time.

5.5 Stored Procedures

Stored procedures will be used to speed up the development duration.

- Showing joined events in the profile page.
- Showing complaint forms according to the tour type.
- Showing old training of that animal.

- Showing an old regularized food list of that animal.
- Showing the past treatments of that animal.
- Showing available veterinarians and keepers for a certain job.

6.0 Implementation Plan

In our project to implement a database system, MySQL will be used. Furthermore, to provide a neat user interface and fundamental system functionalities, JavaScript and Java will be used for the front-end implementation. For the back-end implementation JavaScript will be used.

7.0 Webpages

This section provides the webpage and the (currently private) Github page for the term project of our group.

Webpage: https://mehmetyaylacci.github.io/index.html

Github Page: https://github.com/mehmetyaylacci/CS353-31-ZooDBMS