Bilkent University

Department Of Computer Engineering

CS 353

Database Management Systems Project

Project Design Report

Zoo Database System

Ant Duru 21704108

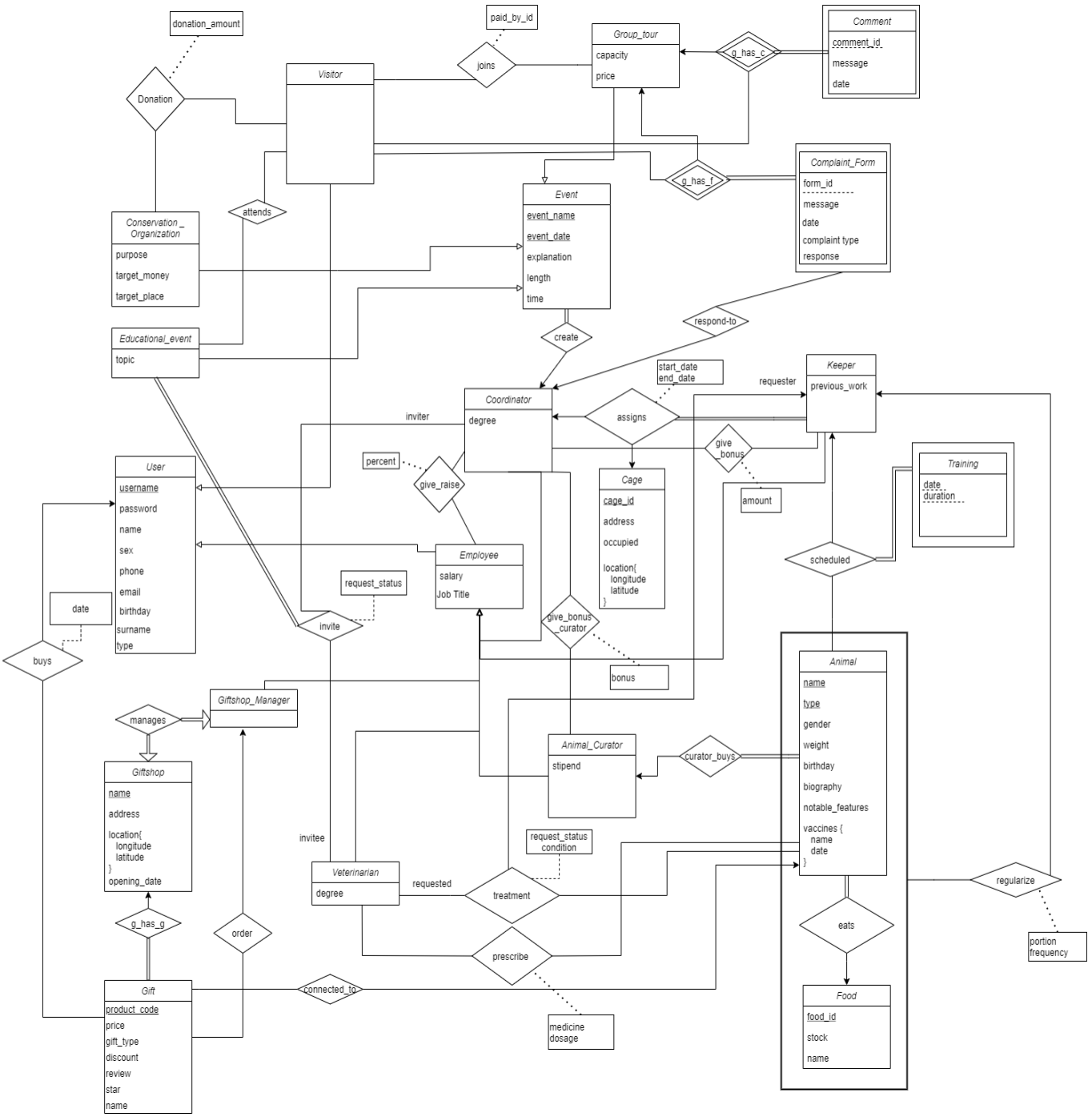
Mehmet Ata Yurtsever 21702135

Çerağ Oğuztüzün 21704147

Atay Kaylar 21703284

02.04.2021

1.Revised ER Diagram



Regarding our assistants feedback we have revised our ER diagram with the following changes:

- We added an additional functionality of a zoo gift shop where visitors can buy Gifts. Gift is also an entity.
- We added an additional user type which is the Gift Shop Manager. This user inherits the Employee entity.
- We added an Animal Curator entity as a new employee type, who can buy animals from the zoo.
- We added the User entity which is parent of Visitors and Employees.
- We added the username, password information to users for the login functionality.
- We added Cage as an entity rather than a relation.
- We enabled each visitor to be able to comment on a group tour by implementing a “has-a” relationship between Visitor and Comment entities.
- We specified each individual event type with “is-a” relationship to the Event entity.
- We removed the aggregate relation for training and introduced a relationship called scheduled for Training entity.
- We removed the “can-take” aggregate relation and introduced the “regularize” aggregate relation to the Keeper entity.
- We removed the “group” relation between Visitor and Group Tour.
- We improved the attributes in each entity.
- We added an accept/reject status attribute for invitation to the Veterinarian entity.
- We fixed any cardinality and participation errors.
- Complaint From no longer has Comment as parent and it is a weak entity.

2.Relational Schema

1.

a. Types

i. vaccine

```
CREATE TYPE vaccine AS (
                name char(120),
                vac_date date
        )
```

ii. location

```
CREATE TYPE location AS (
                longitude real,
                latitude real
        )
```

b. User

Relational Model

User(username, password, name, surname, sex, phone, email, birthday, created_at, type)

Functional Dependencies

username → password, name, surname, sex, phone, email, created_at, type
email → password, name, surname, sex, phone, username, created_at, type

Candidate Keys

{(username), (email)}

Normal Form

3NF

Table Definition

```
CREATE TABLE User
    (username varchar(20) PRIMARY KEY,
    password varchar(40), NOT NULL
    name char(40),
    surname char(40),
    sex char(1),
    phone bigint,
    email varchar(100) UNIQUE,
    birthday DATE
    created_at timestamp,
    type varchar(10))
```

c. Visitor

Relational Model

Visitor(username, created_at)

Functional Dependencies

username → created_at

Candidate Keys

{(username)}

Normal Form

BCNF

Table Definition

CREATE TABLE Visitor
(username varchar(20) PRIMARY KEY,
created_at timestamp,
)

d. Employee

Relational Model

Employee(<u>username</u>, created_at, salary, job_title)

Functional Dependencies

username → created_at,paid, salary, job_title

Candidate Keys

{(username)}

Normal Form

BCNF

Table Definition

CREATE TABLE Employee
(username varchar(20) PRIMARY KEY,
created_at timestamp,
salary money,
job_title varchar(20))

e. Giftshop_Manager

Relational Model

Giftshop_Manager(<u>username</u>, created_at, giftshop_name)

Functional Dependencies

username → created_at, giftshop_name

Candidate Keys

{(username)}

Normal Form

BCNF

Table Definition

CREATE TABLE Giftshop_Manager
(username varchar(20) PRIMARY KEY,
created_at timestamp,
giftshop_name char(20) NOT NULL,
FOREIGN KEY (giftshopName) REFERENCES Giftshop(name))

f. Veterinarian

Relational Model

Veterinarian(<u>username</u>, created_at, degree)

Functional Dependencies

username → created_at, degree

Candidate Keys

{(username)}

Normal Form

BCNF

Table Definition

CREATE TABLE Veterinarian
(username varchar(20) PRIMARY KEY,
created_at timestamp,
degree text)

g. Animal_Curator

Relational Model

Animal_Curator(<u>username</u>, created_at, stipend)

Functional Dependencies

username → created_at, stipend

Candidate Keys

{(username)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE Animal_Curator
    (username varchar(20) PRIMARY KEY,
    created_at timestamp,
    stipend decimal)
```

h. Keeper

Relational Model

Keeper(<u>username</u>, created_at, previous_work, assigned_by)

Functional Dependencies

username → created_at, previous_work, assigned_by

Candidate Keys

{(username)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE Keeper
    (username varchar(20) PRIMARY KEY,
    created_at timestamp,
    previous_work text[],
    assigned_by varchar(20) NOT NULL,
    FOREIGN KEY(assigned_by) REFERENCES Coordinator(username))
```

i. Coordinator

Relational Model

Coordinator(<u>username</u>, created_at, degree)

Functional Dependencies

username → created_at, degree

Candidate Keys

{(username)}

Normal Form

BCNF

Table Definition

CREATE TABLE Coordinator
(username varchar(20) PRIMARY KEY,
created_at timestamp,
degree text)

j. Event

Relational Model

Event(<u>event_name</u>, <u>event_date</u>, created_at, time, explanation, length, coord_un)

Functional Dependencies

event_name, event_date → created_at, time, explanation, length, coord_un

Candidate Keys

{(event_name, event_date)}

Normal Form

BCNF

Table Definition

CREATE TABLE Event
(event_name varchar(40),
event_date date,
created_at timestamp,
explanation text,
length decimal,
coord_un varchar(20)
PRIMARY KEY (event_name, date),
FOREIGN KEY (coord_un) REFERENCES Coordinator(username))

k. Conservation Organization

Relational Model

Conservation_Organization(<u>event_name</u>, <u>event_date</u>, created_at, purpose, target_money, target_place)

Functional Dependencies

event_name, event_date → created_at, time, created_at, purpose, target_money, target_place

Candidate Keys

{(event_name, event_date)}

Normal Form

BCNF

Table Definition

CREATE TABLE Conservation_Organization
(event_name varchar(40),
event_date date,
created_at timestamp,
purpose text,
target_money decimal,
target_place char(40),
PRIMARY KEY (event_name, date))

l. Educational Event

Relational Model

Educational_Event(<u>event_name</u>, <u>event_date</u>, created_at, topic)

Functional Dependencies

event_name, event_date → created_at, topic

Candidate Keys

{(event_name, event_date)}

Normal Form

BCNF

Table Definition

CREATE TABLE Event
(event_name varchar(40),
event_date date,
created_at timestamp,
topic text,
PRIMARY KEY (event_name, date))

m.Group Tour

Relational Model

Educational_Event(<u>event_name</u>, <u>event_date</u>, capacity, price)

Functional Dependencies

event_name, event_date → created_at, capacity, price

Candidate Keys

{(event_name, event_date)}

Normal Form

BCNF

Table Definition

CREATE TABLE Group_Tour
(event_name varchar(40),
event_date date,
created_at timestamp,
capacity int,
price decimal,
PRIMARY KEY (event_name, date))

n. Animal

Relational Model

Animal(<u>name</u>, <u>type</u>, created_at, gender, weight, birthday, biography, notable_features, vaccines, bought_by, food_id, cage_id)

Functional Dependencies

name, type → created_at, gender, weight, birthday, biography, notable_features, vaccines, bought_by, food_id, cage_id

Candidate Keys

{(name, type)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE Animal
    ( name char(40),
    type char(120),
    created_at timestamp,
    gender char(1),
    weight float,
    birthday date,
    biography text,
    notable_features text[],
    vaccines vaccine[],
    bought_by varchar(20),
    food_id uuid NOT NULL,
    cage_id uuid NOT NULL,
    PRIMARY KEY (name, type),
    FOREIGN KEY (bought_by) REFERENCES Animal_Curator(username),
    FOREIGN KEY (food_id) REFERENCES Food)
```

o. Food

Relational Model

Food(food_id, stock, name)

Functional Dependencies

food_id → stock, name

Candidate Keys

{(food_id)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE Food
    ( food_id uuid PRIMARY KEY,
    stock decimal,
```

name char(120))

p. Giftshop

Relational Model

Giftshop(name, address, shop_location, opening_date)

Functional Dependencies

name → shop_location, opening_date

Candidate Keys

{(name)}

Normal Form

BCNF

Table Definition

CREATE TABLE Giftshop
(name varchar(40) PRIMARY KEY,
address text,
shop_location location,
opening_date date)

q. Gift

Relational Model

Gift(product_code, price, animal_name, animal_type, shop, discount, review, star, name)

Functional Dependencies

product_code → price, animal_name, animal_type, shop, discount, review, star,name

Candidate Keys

{(prduct_code)}

Normal Form

BCNF

Table Definition

CREATE TABLE Gift

```
( product_code uuid PRIMARY KEY,
price money,
name varchar(40),
animal_name char(40),
animal_type char(120),
shop varchar(40),
discount float(2,1),
review text,
star float(1,1),
FOREIGN KEY (animal_name, animal_type) REFERENCES Animal(name,
type),
FOREIGN KEY (shop) REFERENCES Giftshop(name) )
```

r. Cage

Relational Model

Cage(cage_id, address, occupied, cage_loc)

Functional Dependencies

cage_id → address, occupied, cage_loc

Candidate Keys

{(cage_id)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE Cage
( cage_id uuid PRIMARY KEY,
address text,
occupied bool,
cage_loc location)
```

s. invite

Relational Model

invite(event_name, event_date, inviter, invitee, request_status)

Functional Dependencies

event_name, event_date, inviter, invitee → request_status

Candidate Keys

{(event_name, event_date, inviter, invitee)}

Normal Form

BCNF

Table Definition

CREATE TABLE Invitations
(event_name varchar(40),
event_date date,
inviter varchar(20),
invitee varchar(20),
request_status char(1),
PRIMARY KEY (event_name, event_date, inviter, invitee),
FOREIGN KEY (event_name, event_date) REFERENCES Educational_Event(name, date),
FOREIGN KEY (inviter) REFERENCES Coordinator(username),
FOREIGN KEY (invitee) REFERENCES Veterinarian(username))

t. Buys

Relational Model

Buys(product_code, username, buy_date)

Functional Dependencies

username, product_code → buy_date, type

Candidate Keys

{(username, product_code)}

Normal Form

BCNF

Table Definition

CREATE TABLE Buys
(username varchar(20),
product_code uuid,
buy_date date,
FOREIGN KEY (username) REFERENCES User(username),
FOREIGN KEY (product_code) REFERENCES Gift(product_code),
PRIMARY KEY (product_code))

u. Treatment

Relational Model

Treatment(requested, requester, animal_name, animal_type, request_status, condition)

Functional Dependencies

requested, requester, animal_name, animal_type→ request_status, condition

Candidate Keys

{(requested, requester, animal_name, animal_type)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE Treatment
    ( requested varchar(20),
    requester varchar(20),
    animal_name char(40),
    animal_type char(120),
    request_status char(1),
    condition text,
    FOREIGN KEY (requested) REFERENCES Veterinarian(username),
    FOREIGN KEY (requester) REFERENCES Keeper(username),
    FOREIGN KEY (animal_name, animal_type) REFERENCES
Anima(name,type),
    PRIMARY KEY(requested, requester, animal_name, animal_type))
```

v. Donations

Relational Model

Donations(event_name, date, username, amount)

Functional Dependencies

event_name, date, username→ amount

Candidate Keys

{(event_name, date, username)}

Normal Form

BCNF

Table Definition

CREATE TABLE Treatment
(revent_name varchar(40),
date date,
username varchar(20),
amount money,
FOREIGN KEY (event_name, date) REFERENCES
Conservation_Organization,
FOREIGN KEY (username) REFERENCES Visitor,
PRIMARY KEY(event_name, date, username))

w. Join_tour

Relational Model

Join_tour(<u>event_name</u>, <u>date</u>, <u>username</u>, paid_by_id)

Functional Dependencies

event_name, date, username→ paid_by_id

Candidate Keys

{(event_name, date, username)}

Normal Form

BCNF

Table Definition

CREATE TABLE Join_tour
(event_name varchar(40),
date date,
username varchar(20),
paid_by_id varchar(20) NOT NULL,
FOREIGN KEY (event_name, date) REFERENCES
Conservation_Organization,
FOREIGN KEY (username) REFERENCES Visitor,
PRIMARY KEY(event_name, date, username))

x. Training

Relational Model

Training(keeper_name, animal_name, animal_type, duration, training_date)

Functional Dependencies

keeper_name →animal_name, animal_type

Candidate Keys

{(duration, training_date)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE Treatment
        ( keeper_name  varchar(20),
        training_date date,
        animal_name char(40) NOT NULL,
        animal_type varchar(120) NOT NULL,
        duration int,
        FOREIGN KEY (keeper_name) REFERENCES Keeper(username),
        FOREIGN KEY (animal_name, animal_type) REFERENCES Animal(name,
type),
        PRIMARY KEY(duration, training_date))
```

x.give_raise

Relational Model

give_raise(giver_un, receiver_un, percent)

Functional Dependencies

giver_un, receiver_un →percent

Candidate Keys

{(giver_un, receiver_un)}

Normal Form

BCNF

Table Definition

CREATE TABLE give_raise
(giver_un varchar(20),
receiver_un varchar(20),
percent float(2,1),
FOREIGN KEY (giver_un) REFERENCES Coordinator(username),
FOREIGN KEY (receiver_un) REFERENCES Employee(username))
PRIMARY KEY(giver_un, receiver_un))

y. # respond-to

Relational Model

respond-to(<u>form_id,</u> username)

Functional Dependencies

form_id→username

Candidate Keys

{(form_id),(username)}

Normal Form

3NF

Table Definition

CREATE TABLE respond-to
(form_id uuid PRIMARY KEY,
username varchar(20) NOT NULL,
FOREIGN KEY (form_id) REFERENCES Complaint_Form,
FOREIGN KEY (username) REFERENCES Employee,
)

z. # give_bonus

Relational Model

give_bonus(<u>bid,</u>c_username, k_username, amount)

Functional Dependencies

bid -> c_username, k_username amount

Candidate Keys

{(bid)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE give_bonus
        ( c_username varchar(20),
         k_username  varchar(20) NOT NULL,
        amount money,
        bid uuid PRIMARY KEY,
        FOREIGN KEY (c_username) REFERENCES Coordinator(username),
        FOREIGN KEY ( k_username ) REFERENCES Keeper(username))
```

aa. regularize

Relational Model

regularize(name, type, food_id , portion, frequency, username)

Functional Dependencies

name, type -> portion, frequency, username, food_id

Candidate Keys

{(name, type)}

Normal Form

3NF

Table Definition

```
CREATE TABLE regularize
        ( name char(40) ,
          food_id uuid,
        portion int,
        frequency int,
        username  varchar(20),
        type varchar(120),
        FOREIGN KEY (name, type ) REFERENCES Animal,
        FOREIGN KEY (food_id ) REFERENCES Food,
        FOREIGN KEY (username ) REFERENCES Keeper,
        PRIMARY KEY(name, type, food_id))
```

bb. give_bonus_curator

Relational Model

give_bonus_curator(bid, coord_un, curator_un, bonus)

Functional Dependencies

bid → coord_un, curator_un bonus

Candidate Keys

{(coord_un, curator_un)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE give_bonus_curator
    ( coord_un varchar(20),
    curator_un varchar(20) NOT NULL,
    amount money,
    bid uuid PRIMARY KEY,
    FOREIGN KEY (coord_un) REFERENCES Coordinator(username),
    FOREIGN KEY (curator_un) REFERENCES Animal_curator(username))
```

cc. prescribe

Relational Model

prescribe(username, name, type, medicine, dosage)

Functional Dependencies

username, name, type→ medicine, dosage

Candidate Keys

{(username, name, type)}

Normal Form

BCNF

Table Definition

CREATE TABLE prescribe

(username varchar(20),
name char(40),
type char(120),
medicine text,
dosage int,
FOREIGN KEY (username) REFERENCES Veterinarian,
FOREIGN KEY (name, type) REFERENCES Animal,
PRIMARY KEY(username, name, type))

dd. g_has_c

Relational Model

g_has_c(comment_id, message, event_date, username, event_name)

Functional Dependencies

comment_id,-> message, date, username, event_name

Candidate Keys

{(comment_id)}

Normal Form

BCNF

Table Definition

CREATE TABLE g_has_c
(comment_id uuid PRIMARY KEY,
message text,
event_date date,
username varchar(20),
event_name varchar(40),
FOREIGN KEY (event_name, event_date) REFERENCES Group_tour,
FOREIGN KEY (username) REFERENCES Visitor)

ee. g_has_f

Relational Model

g_has_f(form_id, message, event_date, complaint_type, response, username, event_name)

Functional Dependencies

form_id -> message, date, complaint_type, response, username, event_name

Candidate Keys

{(form_id)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE g_has_f
    ( form_id uuid PRIMARY KEY,
    message text,
    event_date date,
    complaint_type varchar(20),
    response text,
    username varchar(20),
    event_name varchar(40),
    FOREIGN KEY (event_name, event_date) REFERENCES Group_tour,
    FOREIGN KEY (username) REFERENCES Visitor)
```

ff. assigns

Relational Model

assigns(<u>k_username, cage_id,</u> <u>c_username</u>, start_date, end_date)

Functional Dependencies

k_username, cage_id, c_username →start_date, end_date

Candidate Keys

{(k_username, cage_id, c_username)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE assigns
    ( k_username varchar(20),
    cage_id int,
    c_username  varchar(20),
    start_date date,
    end_date date,
    FOREIGN KEY (cage_id) REFERENCES Cage(cage_id),
    FOREIGN KEY (k_username) REFERENCES Keeper(username),
    FOREIGN KEY (c_username) REFERENCES Coordinator(username),
```

PRIMARY KEY(k_username, cage_id))

gg. scheduled

Relational Model

scheduled(schedule_date, username, duration, name, type)

Functional Dependencies

schedule_date, username, name, type-> duration

Candidate Keys

{(schedule_date, username, name, type)}

Normal Form

BCNF

Table Definition

CREATE TABLE scheduled
(schedule_date date,
username varchar(20),
name char(40),
type varchar(120),
FOREIGN KEY (name, type) REFERENCES Animal,
FOREIGN KEY (username) REFERENCES Keeper,
PRIMARY KEY(schedule_date, username, name, type))

hh. connected-to

Relational Model

connected-to(product_code, name, type)

Functional Dependencies

product_code, name, type

Candidate Keys

{(product_code, name, type)}

Normal Form

BCNF

Table Definition

CREATE TABLE connected-to
(product_code uuid,
name char(40),
type varchar(120),
FOREIGN KEY (name, type) REFERENCES Animal,
FOREIGN KEY (product_code) REFERENCES Gift,
PRIMARY KEY(product_code, name, type))

ii. attends

Relational Model

attends(<u>username, event_name, date</u>)

Functional Dependencies

username, event_name, date

Candidate Keys

{(username, event_name, date)}

Normal Form

BCNF

Table Definition

CREATE TABLE attends
(event_name varchar(40),
username varchar(20),
event_date date
FOREIGN KEY (event_name, event_date) REFERENCES Educational_event
FOREIGN KEY (Visitor.username) REFERENCES Visitor(username)

jj. Complaint_Form

Relational Model

Complaint_Form(event_name, <u>form_id</u>, message, event_date, complaint_type, response)

Functional Dependencies

form_id →event_name, message, event_date, complaint_type, response

Candidate Keys

{(form_id)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE Complaint_Form
    ( event_name varchar(40),
    form_id uuid PRIMARY KEY,
    message text,
    event_date date,
    complaint_type  varchar(40),
    response  text,
    FOREIGN KEY (event_name, event_name) REFERENCES Group_tour
    )
```

kk. Comment

Relational Model

Comment(event_name, comment_id, message, event_date)

Functional Dependencies

comment_id -> message, event_date, event_name

Candidate Keys

{(comment_id)}

Normal Form

BCNF

Table Definition

```
CREATE TABLE Comment
    (event_name varchar(40) ,
    comment_id uuid PRIMARY KEY,
    message varchar(40),
    event_date date,
```

FOREIGN KEY (event_name, event_date) REFERENCES Group_tour)

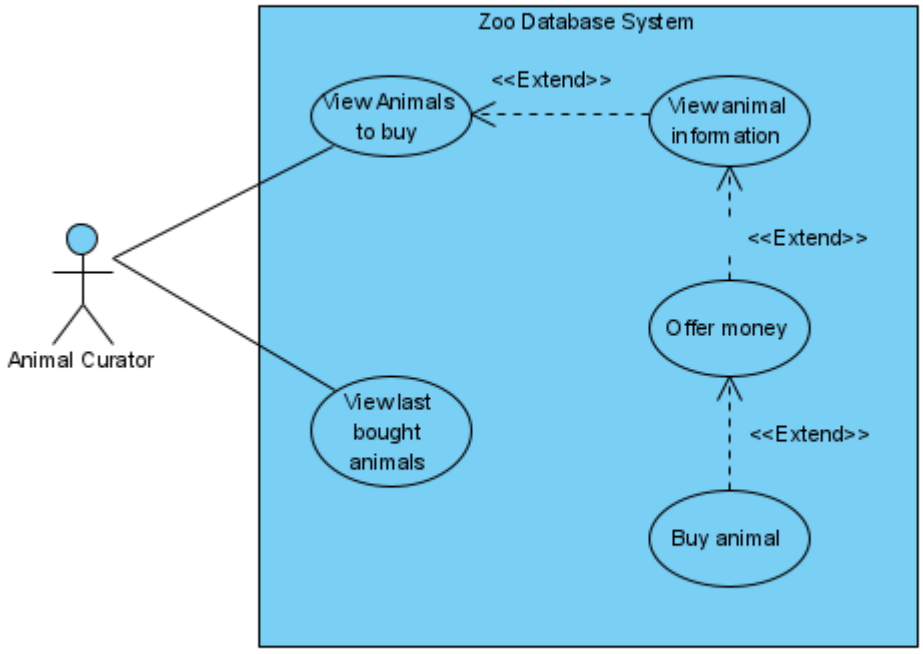
3. Functional Dependencies and Normalization Of Tables

The Relational Schemas part of the report includes the normal forms as well as the dependencies for each schema. We used Boyce-Codd Normal Form (BCNF) for every relation. Hence, we did not need any normalization and decomposition.

4. Functional Components

4.1. USE CASES / SCENARIOS

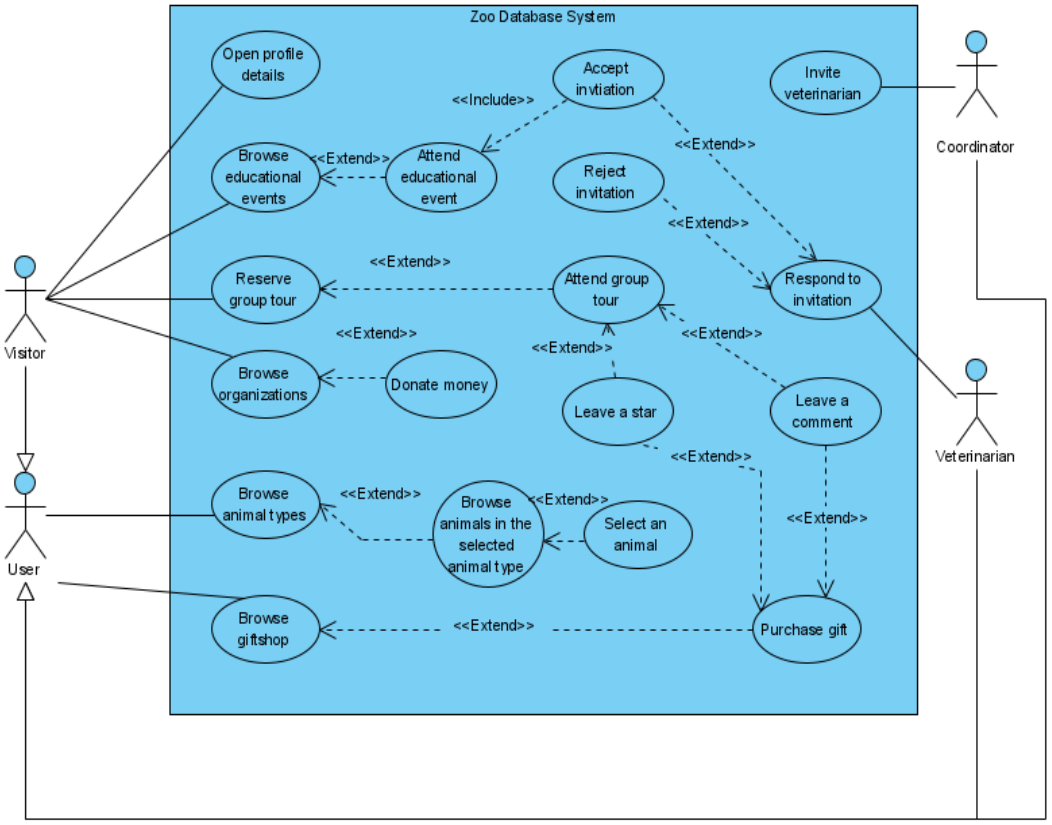
4.1.1 Animal Curator Use Case Diagram



- **View Animals to buy:** Animal curator can browse animals for buying.
- **View animal information:** Information of animals can be viewed by the animal curator.
- **Offer money:** Animal curator can make an offer to buy an animal.

- **Buy animal:** If the offer is accepted, animal curator can buy the animal.
- **View last bought animals:** Animal curator can see the lastly bought animals.

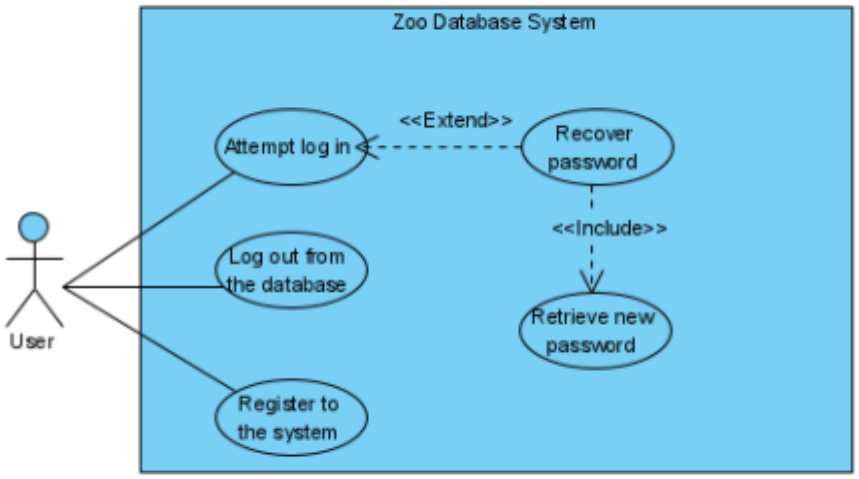
4.1.2 Generic User Use Case Diagram



- **Open profile details:** Visitor can see their profile details.
- **Browse educational events:** Visitor can see the educational events that will occur.
- **Attend educational event:** Visitor can attend an educational event
- **Accept invitation:** Veterinarian can accept an invitation for an educational event.
- **Reject invitation:** Veterinarian can reject an invitation for a n educational event.
- **Invite Veterinarian:** Coordinator can invite a veterinarian to an educational event
- **Respond to invitation:** Veterinarian responds to invitation as mentioned in previous cases
- **Reserve Group Tour:** Visitor can make reservation for a group tour.
- **Attend Group Tour:** Visitor can attend a group tour.
- **Leave a comment:** Visitors can leave comments for the group tour they attended.
- **Browse organizations:** Visitor can browse organizations that accept donations.
- **Donate money:** Visitor can donate money to a conversation organization

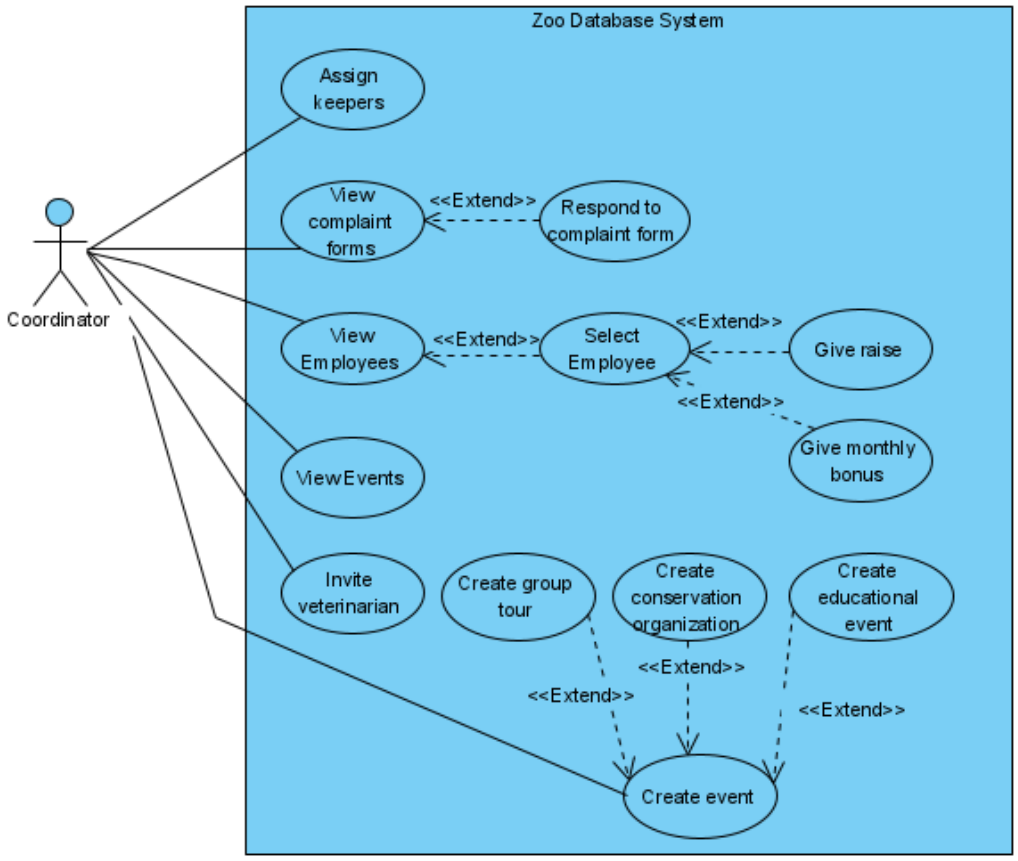
- **Leave a star:** Visitor can leave a star for a group tour
- **Purchase gift:** Visitor can purchase a gift from the gift shop
- **Browse animal types:** Users can browse animals in the zoo.
- **Browse giftshop:** Visitor can see the gifts the gift shop
- **Browse animals in the selected animal type:** Visitors can search for animals of a specific type.
- **Select an animal:** Visitor can see information of an animal

4.1.3 User Use Case Diagram



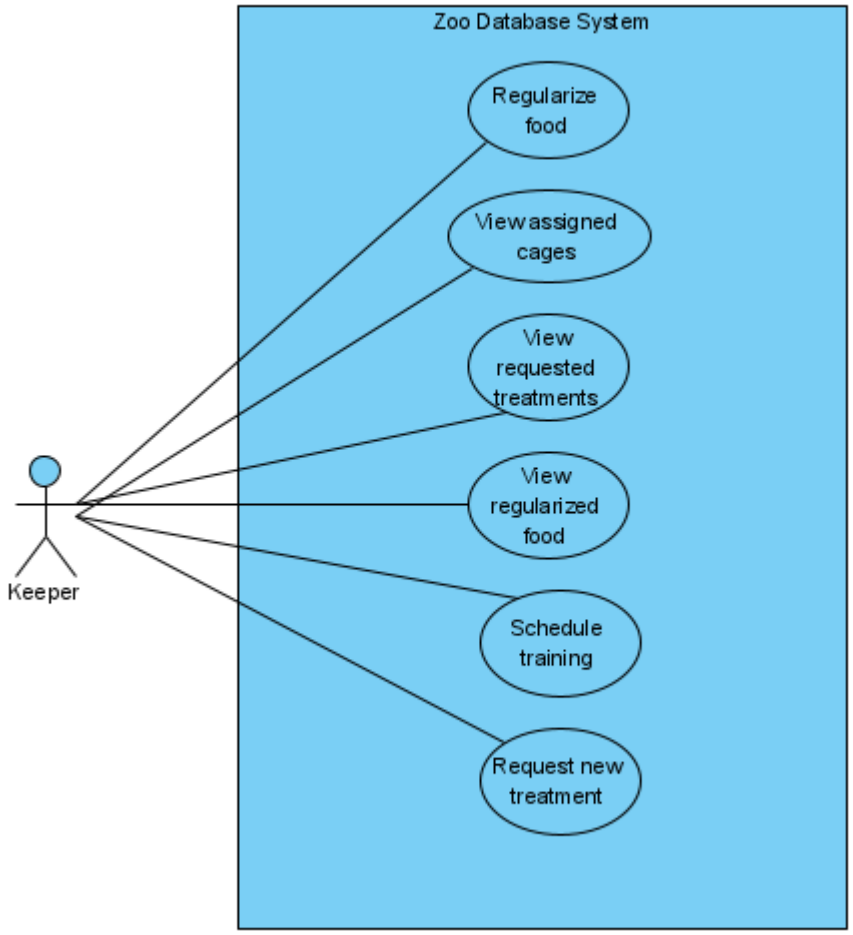
Attempt login: User can attempt to login to the system
Recover Password: An email is sent to the email of the user who forgot their password for renewal.
Retrieve new password: User can retrieve the new password
Log out from the database: User can logout form the database.
Register to the system: User can register (sign-up) to the system

4.1.4 Coordinator Use Case Diagram



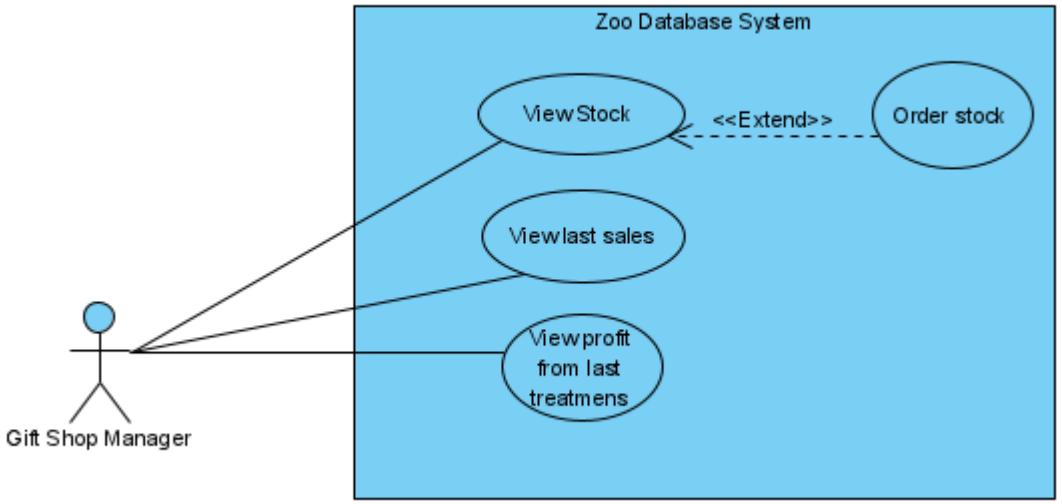
Assign keepers: Coordinator can assign keepers to cage.
View complaint forms: Coordinators can see all complaint forms.
Respond to complaint form: Coordinators can respond to complaint forms.
View employees: Coordinator can see all the employees
Select employee: Coordinator can select an employee and see their profile page
Give raise: Coordinator can give an employee raise
Give monthly bonus: Coordinator can give a monthly bonus to a keeper or an animal curator
View Events: Coordinator can see all events.
Invite veterinarian: Coordinators can invite veterinarians to educational events.
Create group tour: Coordinators can create group tours.
Create conservation organization: Coordinators can create a conservation organization.
Create educational event: Coordinator can create an educational event
Create event: Coordinator can create an event

4.1.5 Keeper Use Case Diagram



- **Regularize food:** Keepers can regularize food for animals for their assigned cages.
- **View Assign cages:** Keeper can see the cage it got assigned to.
- **View requested treatments:** Keepers can view the requested treatments to the veterinarians.
- **View regularized food:** Keepers can view the regularized food for the animals..
- **Schedule training:** Keeper can schedule training for an animal
- **Request new treatment:** Keeper can request a new treatment for an animal

4.1.6 GiftShop Manager Use Case Diagram



- **View Stock:** GiftShop Manager can see gift stocks.
- **Order Stock:** Giftshop Manager can order more gift stocks.
- **View last sales:** Last sales can be viewed with their info.
- **View profit from last month:** Gift shop manager can see the last month's profit.

5. User Interface Design and Corresponding SQL Statements

5.1. Visitor UI

Welcome Screen

Logout Login Register

Welcome to the Zoo Webpage

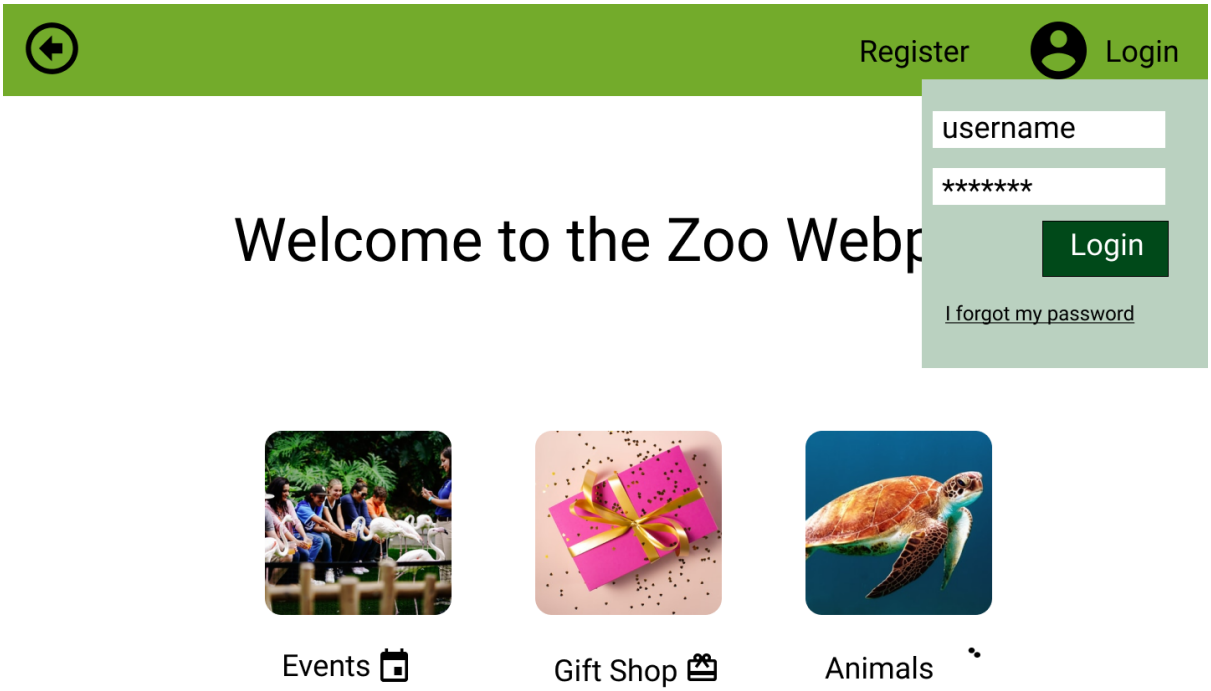
Events

Gift Shop

Animals

Inputs: -
Process: Welcome to the Zoo Webpage welcomes visitors and employees- in other words, all users-. On this main page, a user can log in or register by clicking the appropriate buttons and for registration, the user is navigated to a new page and for login, a pop-up screen comes. The user, also, can click on three buttons to see events, gift shop, animals, respectively. The user can always click on the Logout link to logout.
SQL Statements: -

Login Popup

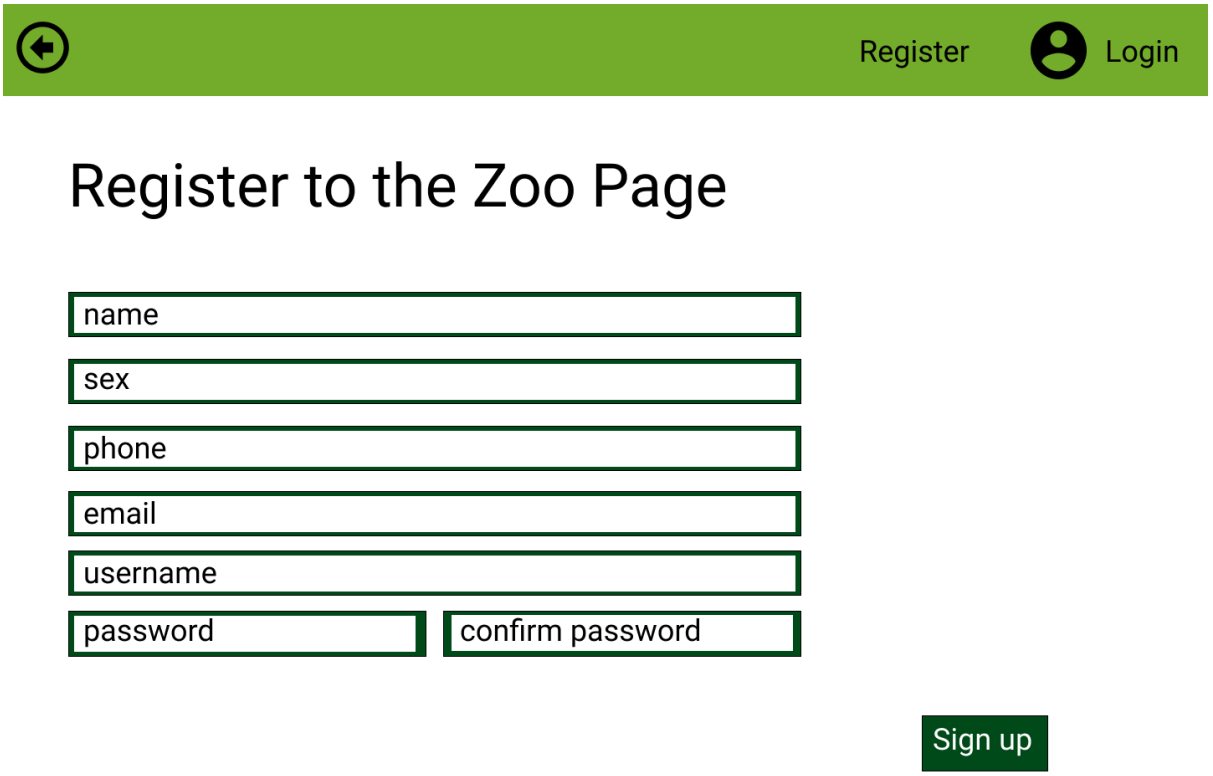


Inputs: @username, @password
Process: When the user clicks on Login, the user must enter their username and password, and click login to sign in to their profiles. If the user forgot their password, the new password is sent to their email address. The system checks whether the username, password pair that the user enters is the same as any pair in the database.
SQL Statements:

Login
SELECT username, password
FROM User
WHERE username = @username AND password = @password

Change Password
UPDATE User
SET password = @newpassword
WHERE username = @username

Register Screen



Inputs: @name, @sex, @phone, @email, @username, @password, @confirmPassword
Process: The user can register to the Zoo webpage by creating a new user by filling in the respective fields and click on the Sign-up button.
SQL Statements:
Checking If User Exists
SELECT username
FROM user
WHERE @username = username;

Registering New User
INSERT INTO User
VALUES(@username, @password, @name, @surname, @sex, @phone, @email, @birthday, now(), “Visitor”)
WHERE @password = @confirmPassword
INSERT INTO VISITOR
VALUES(@username, now())
WHERE @password = @confirmPassword

Visitor Profile Screen

Inputs: -
Process: The visitor profile page displays the information about the user. The events the visitor has chosen to join are also displayed with their name, date and time information.
SQL Statements:

Displaying Visitor Info

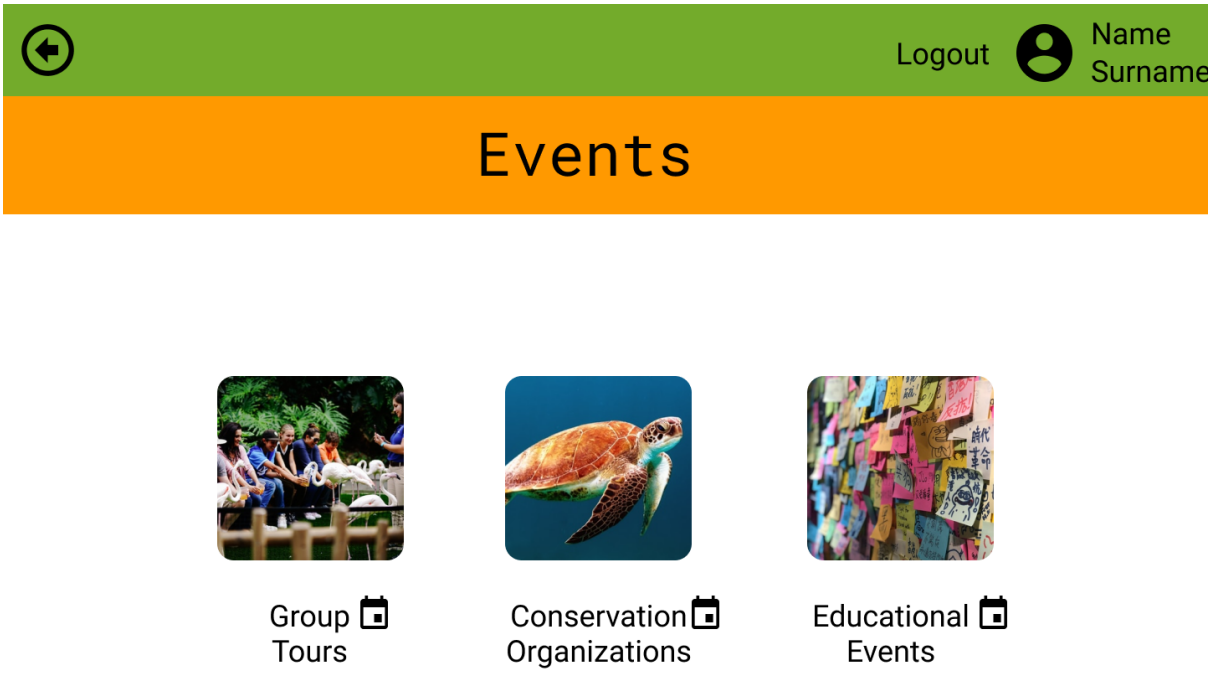
```
SELECT username, password, name, sex, phone, email, birthday, surname
FROM Visitor
WHERE username = @username
```

Displaying Joined Events of Visitor

```
SELECT event_name, date, time
FROM Visitor V, attends A
WHERE A.username = V.username
UNION
SELECT event_name, date, time
FROM Visitor V, joins J
WHERE J.username = V.username
UNION
```

```
SELECT event_name, date, time
FROM Visitor V, donation D
WHERE D.username = V.username
```

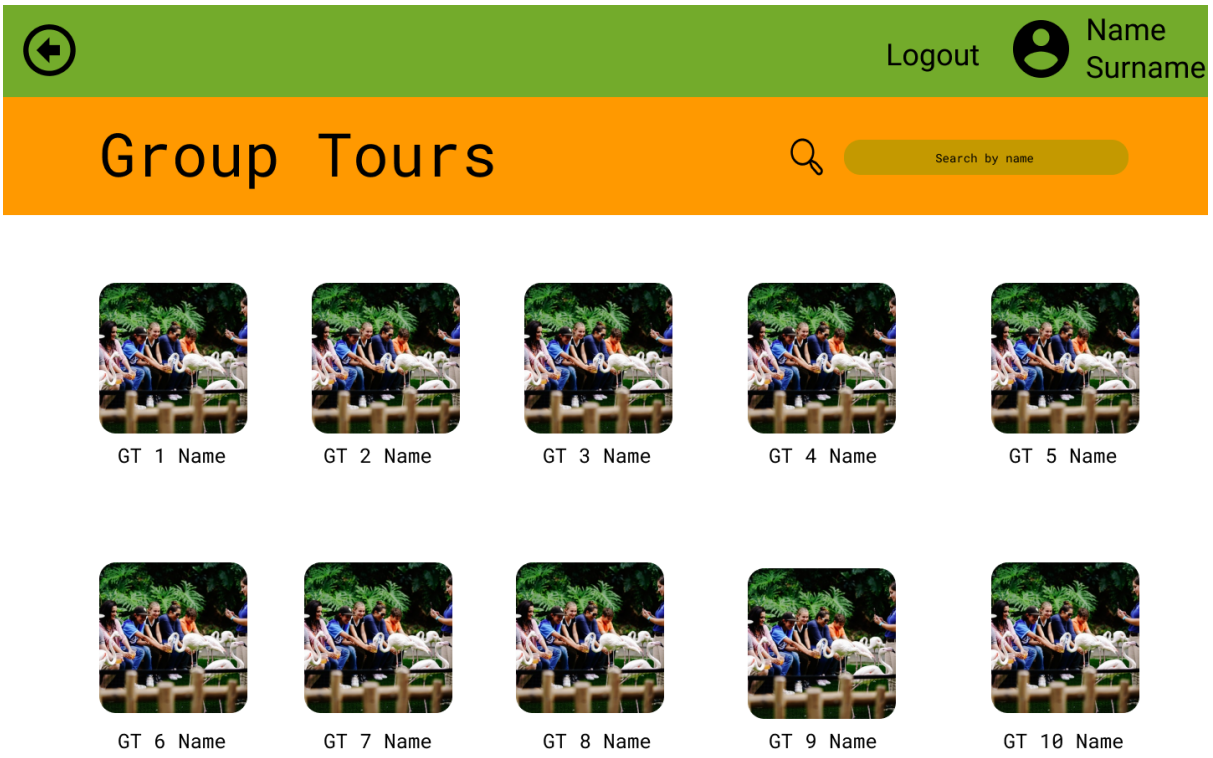
Events Main Screen



Inputs: -
Process: Three types of events are Group Events, Conservation Organizations and Educational Events. The user can browse events of a particular type by clicking on the images and they will be navigated to the respective browse screen.

SQL Statements: -

Browse Group Tours Screen

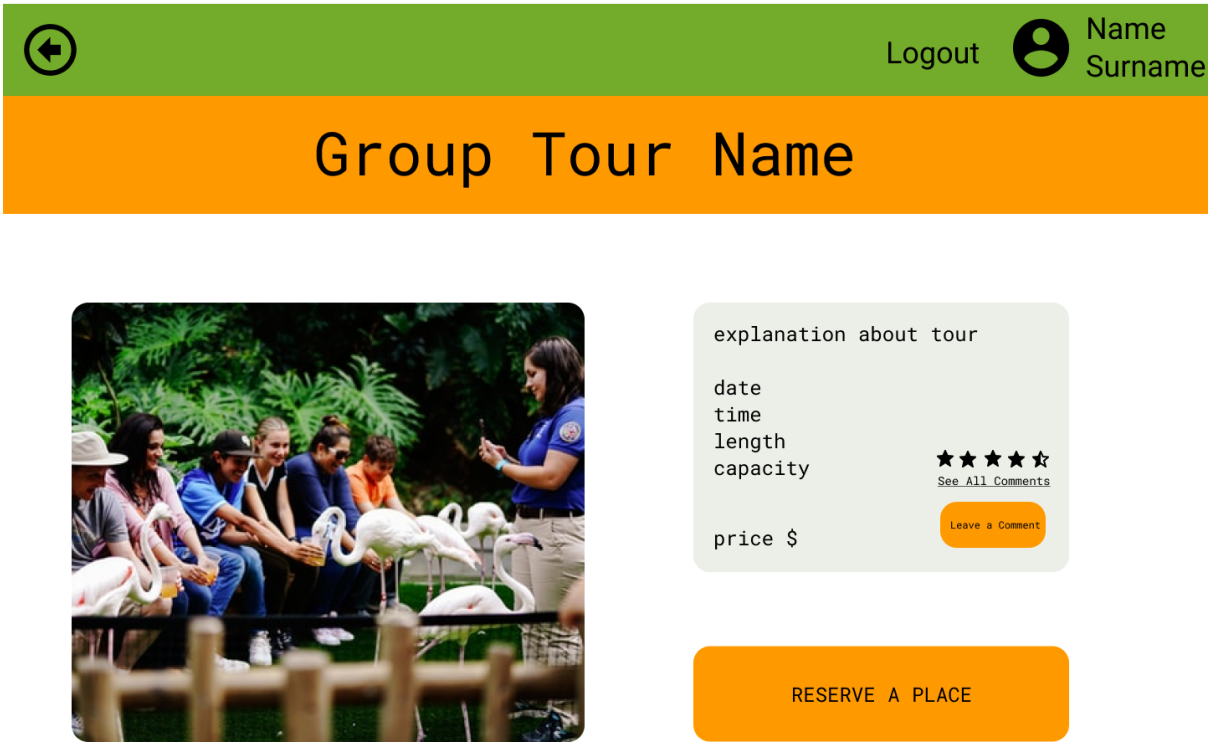


Inputs: @searchName
Process: On this page, group tours that will happen in the future and still has available slots are being displayed. The user can click on the group tour image that they want to attend to see further information about that tour. There is, also, a search bar to search a group tour by its name.
SQL Statements:

Display Group Tours
SELECT event_name
FROM Group_tour

Search by Group Name
SELECT event_name
FROM Group_tour
WHERE event_name LIKE “%@searchName%”

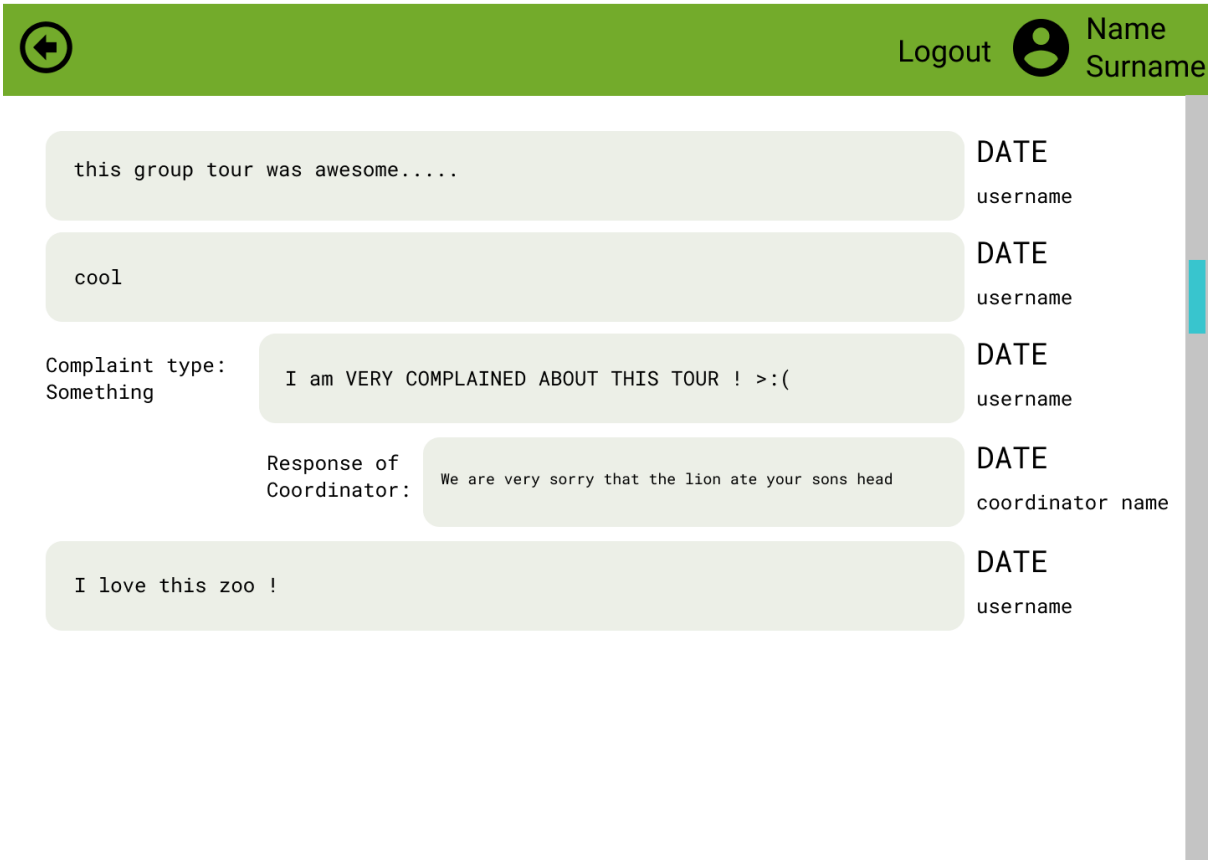
Group Tour Details Screen



Inputs: -
Process: On this page, the user can see detailed information about the group tour that they have clicked. The explanation about the tour, the date, the starting time, the duration, the capacity and the price of the tour is displayed on the page. The user can reserve a place by clicking on the respective button.
SQL Statements:

Displaying Group Tour Details
SELECT explanation, date, time, length, capacity, price
FROM Group_tour NATURAL JOIN Event
WHERE event_name = @event_name

See All Comments Screen



Inputs: -
Process: Here, previous comments on a specific group tour are displayed. The user can also see the responses that have been made by the coordinators.
SQL Statements:

Displaying Comments
SELECT Comment.message, Comment.date, Visitor.username
FROM GroupTour NATURAL JOIN Comment
WHERE GroupTour.event_name = @event_name

Displaying Complaint Forms and Their Responses
SELECT Complaint_Form.message, Complaint_Form.date, Complaint_Form.complaint_type, Complaint_Form.response, Visitor.username, Coordinator.username
FROM Complaint_Form NATURAL JOIN g_has_f NATURAL JOIN Event NATURAL JOIN respond-to NATURAL JOIN Coordinator
WHERE GroupTour.event_name = @event_name

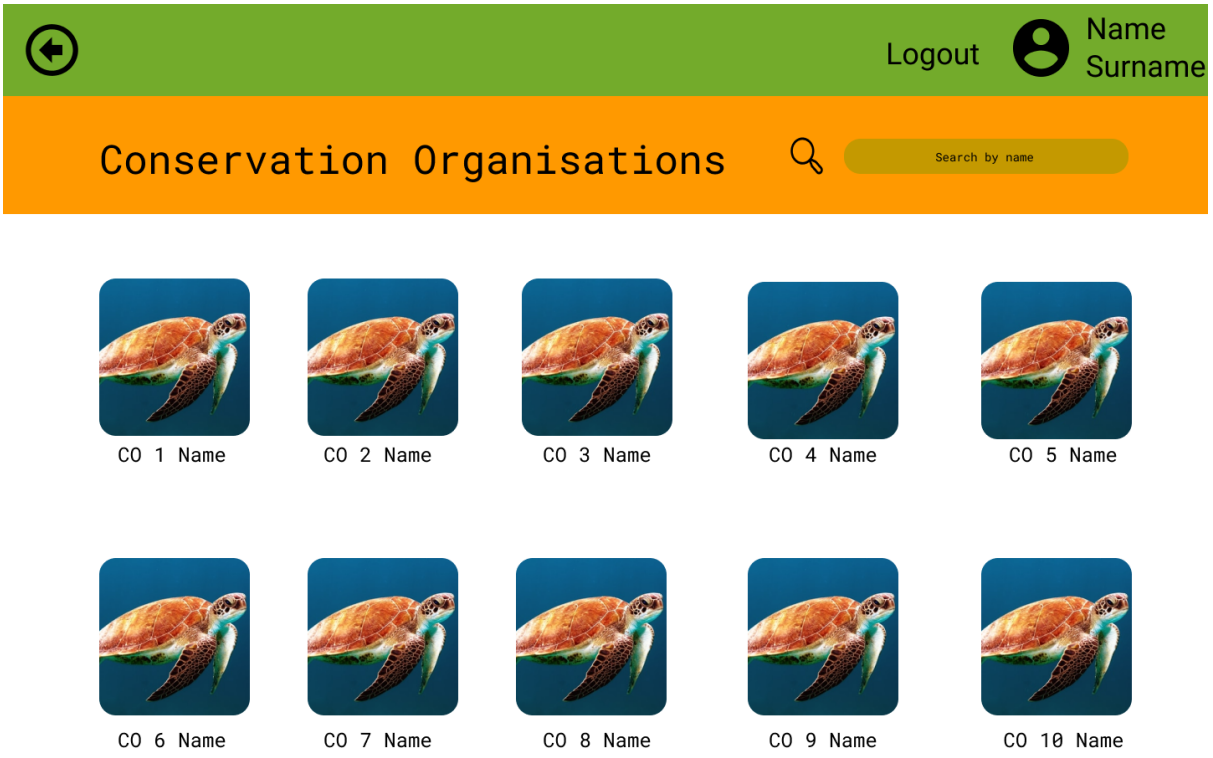
Leave Comment Screen

Inputs: @message, @complaintType, @event_name
Process: On this page, the user can make a comment about a group tour that they attended. Also, If they want to make it a complaint, they can choose the complaint type.
SQL Statements:

Write Comment
INSERT INTO Comment
VALUES(@event_name, Random.uuid(), @message, @date)

Write Complaint Form
INSERT INTO Complaint_Form
VALUES(@event_name, Random.uuid(), @message, @date, @complaintType, NULL)

Browse Conservation Organizations Screen



Inputs: @searchName
Process: On this page, the user can see the conservation organizations that the user can donate money to. The user can see detailed information about a conservation organization by clicking the image of it. This page, also, includes a search bar. The user can search for a conservation organization by its name.
SQL Statements:

Display Conservation_Organization
SELECT event_name
FROM Conservation_Organization

Search by Conservation_Organization Name
SELECT event_name
FROM Conservation_Organization
WHERE event_name LIKE “%@searchName%”

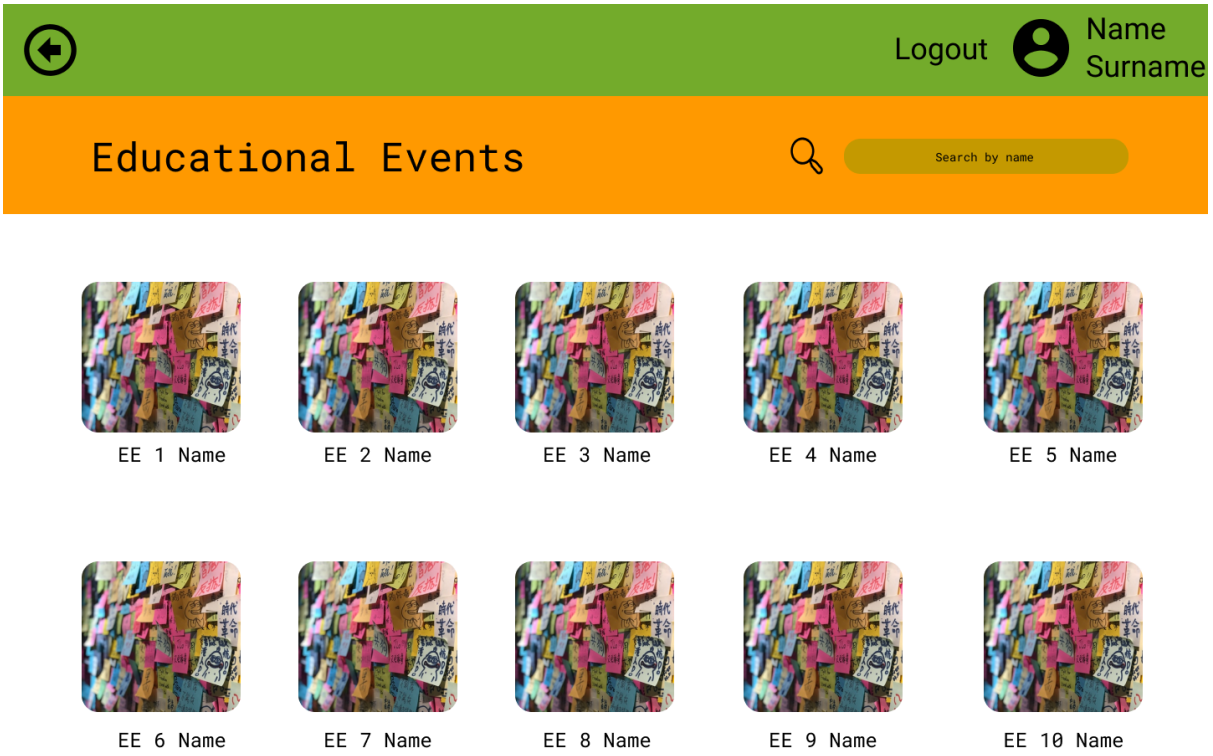
Conservation Organization Details Screen

Inputs: @amountToDonate
Process: On this page, the user can see detailed information about the conservation organization that they have clicked. This detailed information consists of an explanation about the organization, the purpose of the organization, the date, the starting time, the duration, the target money and the place of the organization will occur. The user, can also, donate an amount of money by entering the amount and clicking the left-bottom button.
SQL Statements:

Displaying Conservation Organization Details
SELECT explanation, purpose, date, time, length, target_money, target_place
FROM Conservation_Organization NATURAL JOIN Event
WHERE event_name = @event_name

Donation
INSERT INTO Donation(username, event_name, date, donation_amount)
VALUES (@username, @event_name, @date, @amountToDonate)

Browse Educational Events Screen



Inputs: @searchName
Process: On this page, the user can see the educational events that the user can attend. The user can see detailed information about an educational event by clicking the image of it. This page, also, includes a search bar. The user can search for an educational event by its name.
SQL Statements:

Display Educational_Events
SELECT event_name
FROM Educational_Events

Search by Educational_Events Name
SELECT event_name
FROM Educational_Events
WHERE event_name LIKE “%@searchName%”

Education Event Detail Screen

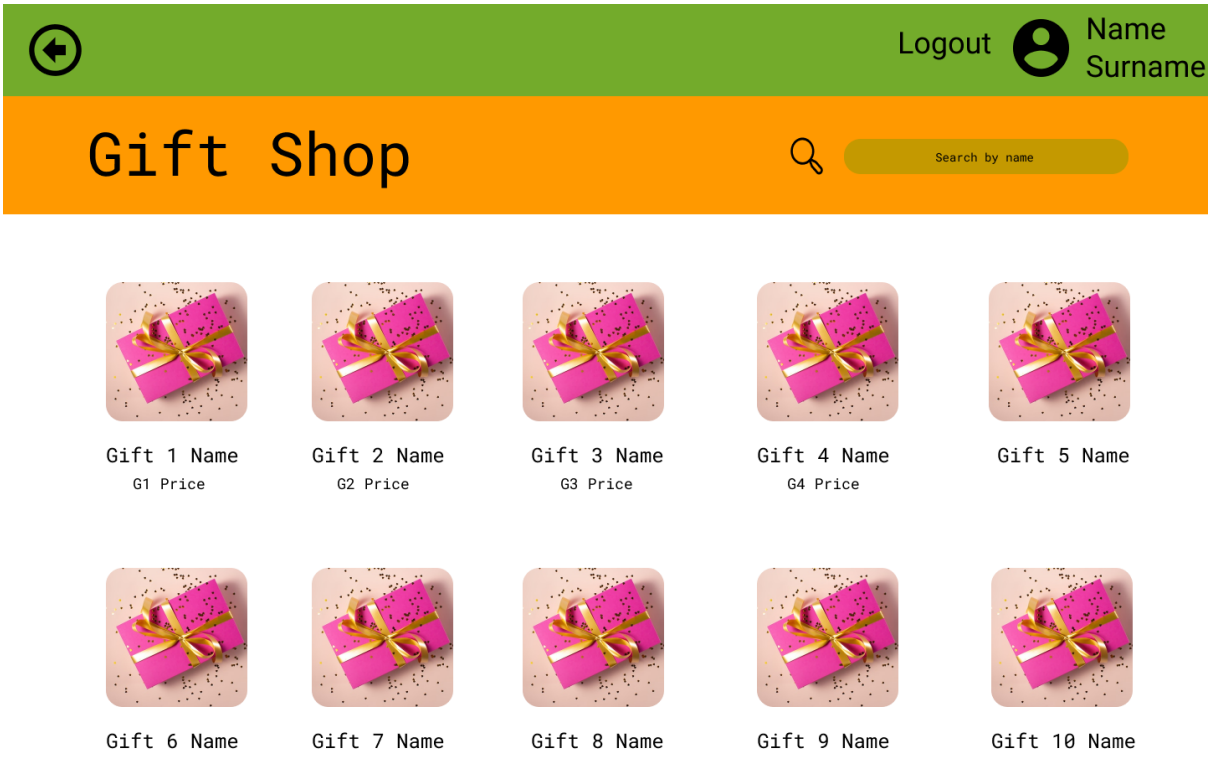
Inputs:
Process: On this page, the user can see detailed information about the educational event that they have clicked. This detailed information consists of an explanation about the event, the explanation of the event, the date, the starting time, the duration, the topic of the event and the veterinarians that will join this event. The user can join the event by clicking the left-bottom button.
SQL Statements:

Displaying Educational Event Details
SELECT topic, date, time, length, explanation
FROM Education_event
WHERE event_name = @event_name

Displaying Veterinarian Name
SELECT V.name, V.surname

FROM Veterinarian V, Educational_event E, invite I
WHERE E.event_name = @event_name AND I.username = V.username AND
I.request_status = ‘A’

Gift Shop Screen

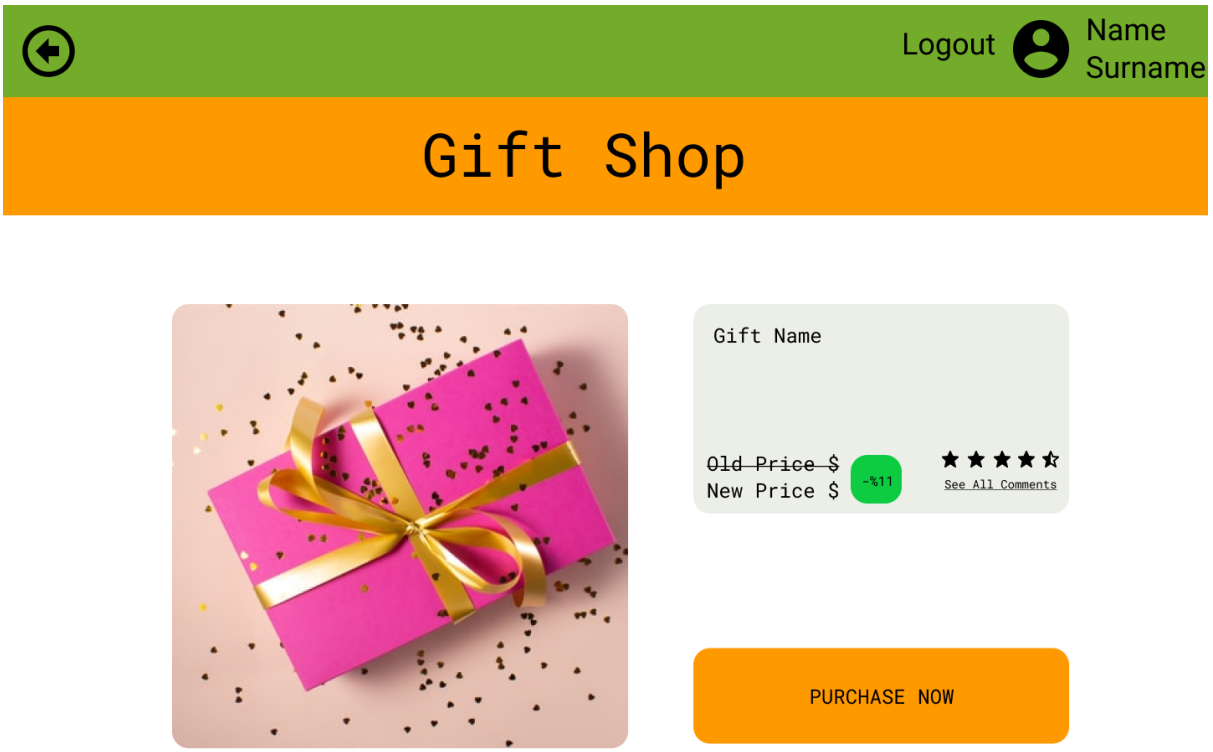


Inputs: @product_code
Process: This is the gift shop of our zoo. There are several gifts that the user can buy. The user can see detailed information about a gift by clicking on the image of it. There is also a search bar to search for a gift by its name.
SQL Statements:

Display Gifts
SELECT name,prdouct_code
FROM Gift

Search by Gifts Name
SELECT product_code,name
FROM Gift
WHERE product_codeLIKE “%@name%”

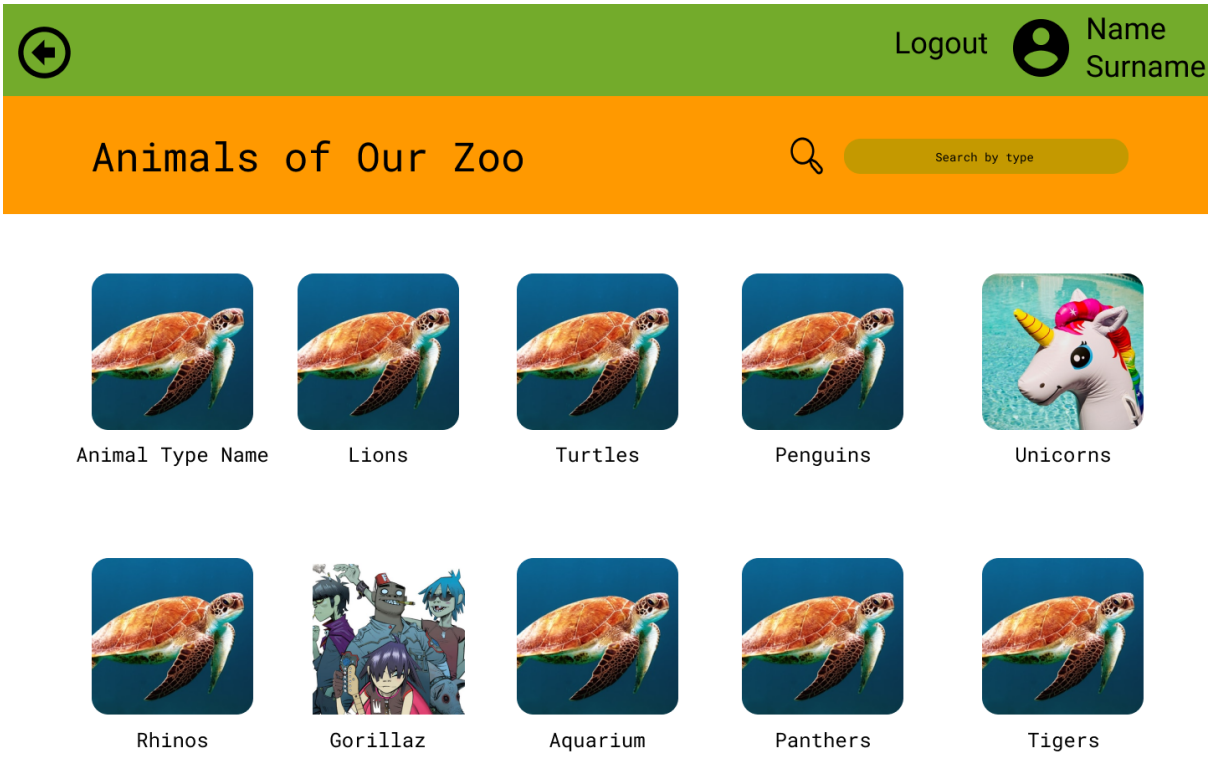
Purchase Gift Screen



Inputs:
Process: On this page, detailed information about the gift that the user has clicked is displayed. The name of the gift, the old and new price, the discount, the rating and the comments on the gift are also displayed. The user can buy a gift by clicking on the purchase now button.
SQL Statements:

Displaying Gift Details
SELECT product_code, price, gift_type, discount, review, star
FROM Gift
WHERE product_code= @product_code

Browse Animals Screen



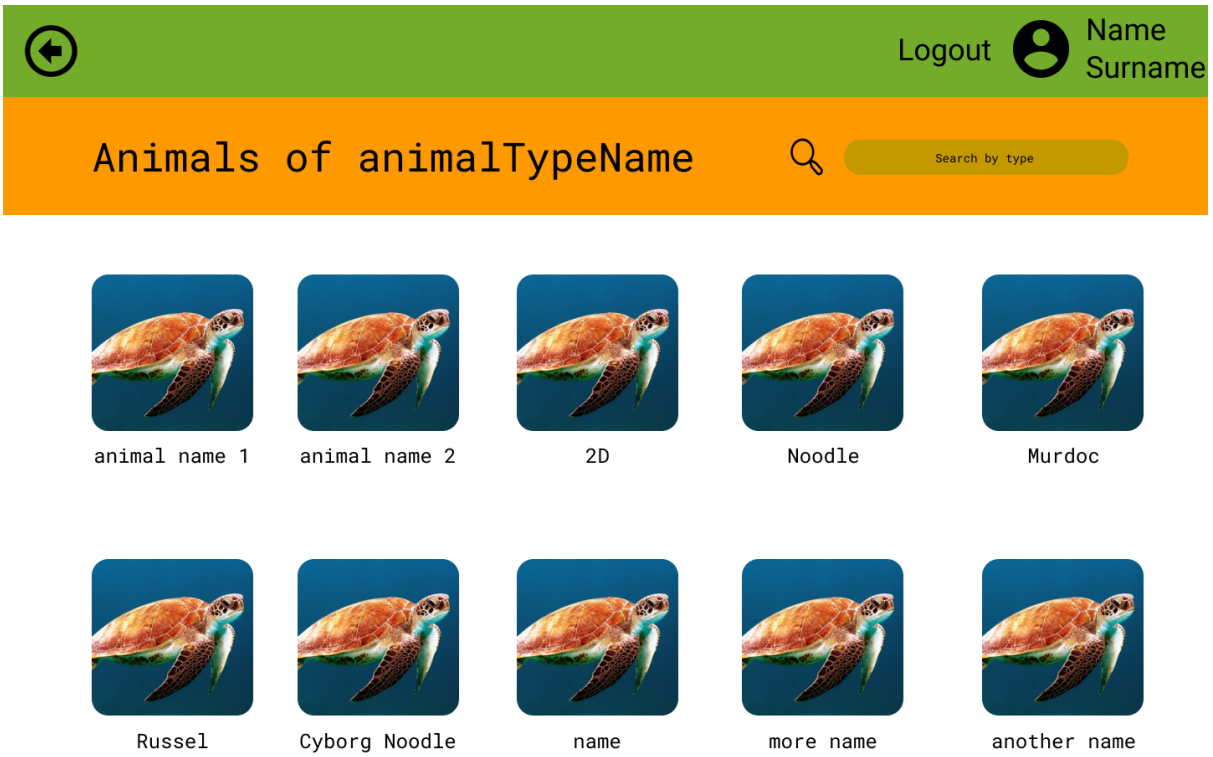
Inputs: @type
Process: This is the page that comes when the user clicked on animals button on the main page. The user can see groups of animals according to their types. When the user clicks on a type, they will see each animal in that type.
SQL Statements:

Display Animals by Type
SELECT type
FROM Animal

Search by Animals Type
SELECT type

FROM Animal
WHERE type LIKE “%@type%”

Browse Animals of a Type Screen

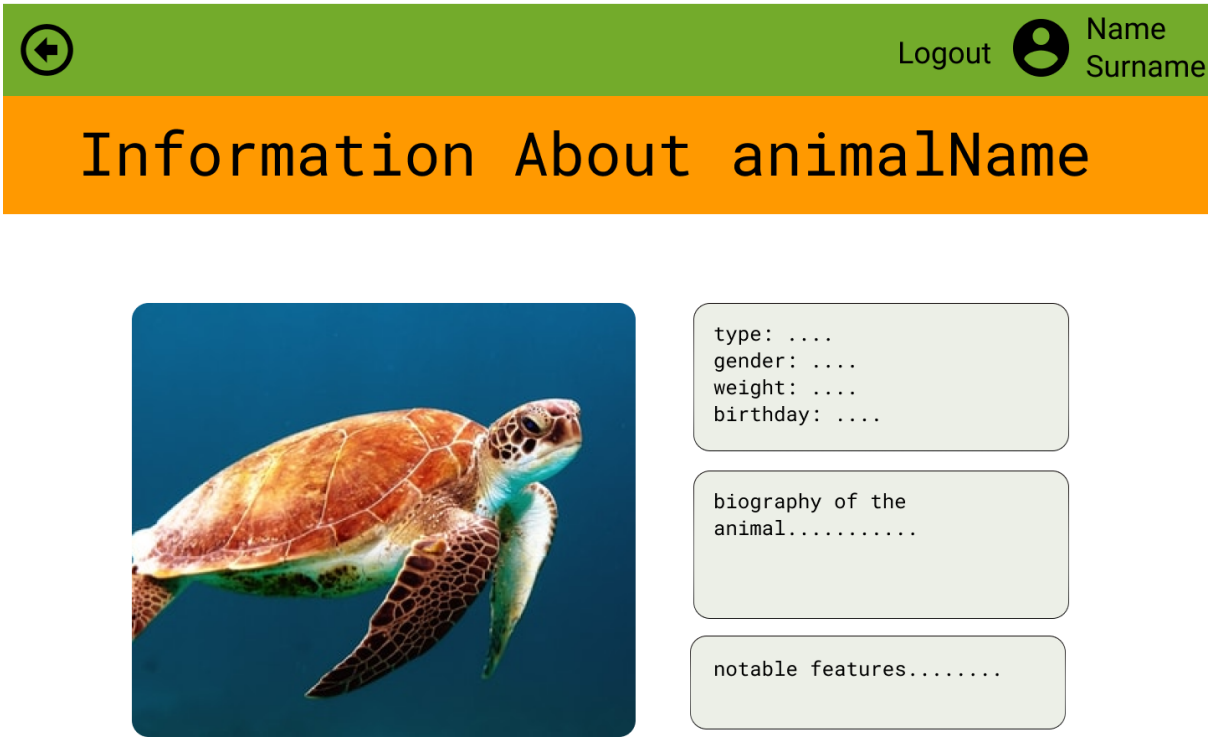


Inputs: @name
Process:
SQL Statements:

Display Animals by Name
SELECT name
FROM Animal

Search by Animals Name
SELECT name
FROM Animal
WHERE name LIKE “%@name%”

Animal Detail Screen



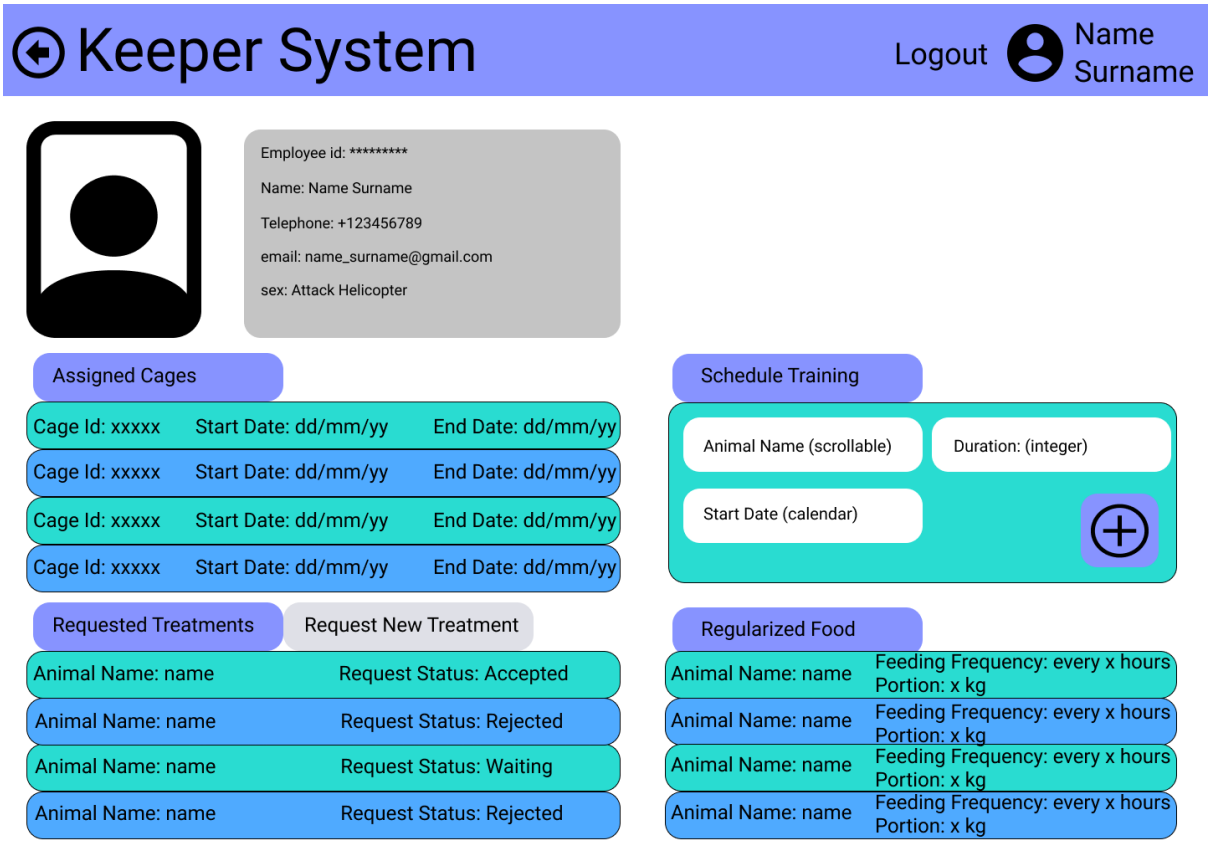
Inputs:
Process: On this page, specific information about the selected animal is displayed. The type of the animal, the gender, the weight and height of the animal, the birthday, the biography and notable features about the animal is displayed.
SQL Statements:

Displaying Animal Details
SELECT name, type, gender, weight, birthday, biography, notable_features
FROM Animal
WHERE name= @name

Employee UI

5.2.Keeper UI

Keeper Profile Screen with Requested Treatments Tab Open



Inputs: @animalName, @duration, @startDate, @username
Process: This is how the keeper profile seems when a keeper enters the system. On this page, the picture of the keeper and the general information, such as employee id, name, telephone, email and sex, of the keeper. Below, the keeper can see the cages that they have been assigned to, as well as the starting and ending time of their cage duty. There is also a part to schedule training for a specific animal, given a start date and duration (number of days). Under these two, there is a part to see the regularized food for animals, containing the

detailed information about their regularized food. At bottom-left, the keeper can see the treatments that they requested from a veterinarian for an animal, and their request's status. The keeper can change the tab to request a new treatment.

SQL Statements:

View Keeper Info

```
SELECT *
FROM Keeper
WHERE username = @username
```

View Assigned Cages

```
SELECT cage_id, start_date, end_date
FROM Keeper NATURAL JOIN assigns
WHERE username = @username
```

View Requested Treatments

```
SELECT name, request_status
FROM Keeper NATURAL JOIN treatment
```

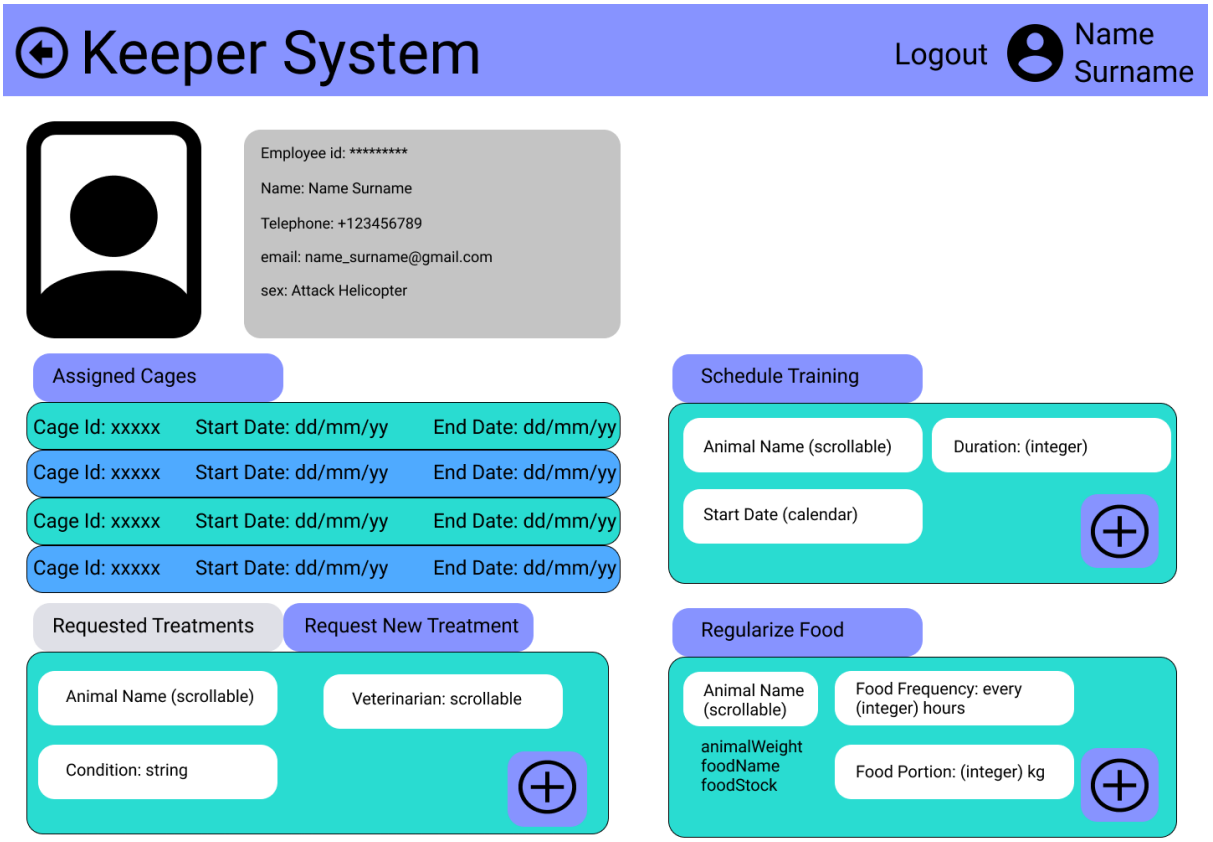
View Regularized Food

```
SELECT name, frequency, portion
FROM Keeper NATURAL JOIN regularize
```

Schedule Training

```
INSERT INTO Training
VALUES(@username, @animalName, @type, @duration, @startDate)
```

Keeper Profile Screen with Request New Treatments Tab



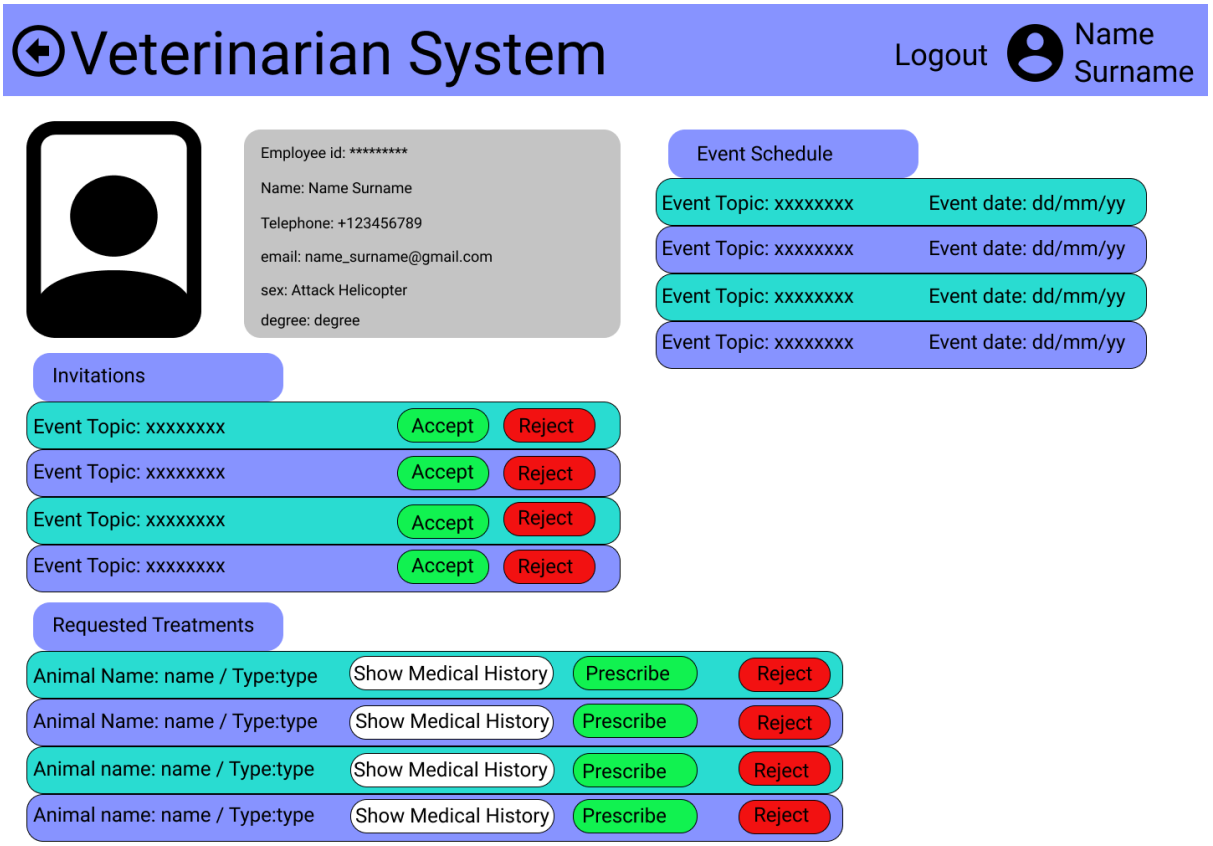
Inputs: @animalNameForTreatment, @vet, @condition, @animalNameForFood, @frequency, @portion, @food_id, @animalTypeForFood, @animalTypeforTreatment
Process: This is the same page with request new treatment tab and regularize food tab enabled. The keeper can request a treatment for an animal by entering the name of the animal, their condition and the requested veterinarian for the treatment. The keeper can regularize an animal's food by entering the animal name, the frequency of their food and the portion.
SQL Statements:

Request New Treatment
INSERT INTO treatment(requested, requester, name, type, request_status, condition)
VALUES(@vet, @username, @animalNameForTreatment, @AnimalTypeforTreatment, “W”, @condition)

Regularize Food
INSERT INTO regularize
VALUES(@animalNameForFood, @animaltypeForFood, @food_id, @portion, @frequency, @vet)
WHERE (SELECT SUM(portion) FROM regularize WHERE @food_id = food_id) + @portion < stock

5.3.Veterinarian UI

Veterinarian Profile Screen



Inputs:

Process: This is the profile page of a veterinarian when they enter the system. Again, at the top there is general information about the veterinarian. A veterinarian can see the scheduled events that they will attend. They can also respond to invitations from educational events- they can either accept or reject it- and they can see the requested treatments for animals. They can either reject the treatment, see the animal's medical history, or they can prescribe a medicine for an animal. A pop-up screen comes when the prescribe button is clicked.

SQL Statements:

View Veterinarian Info

```
SELECT *
FROM Veterinarian
WHERE username = @username
```

View Invitations

```
SELECT topic
FROM invite
WHERE invitee = @username AND request_status <> 'W'
```

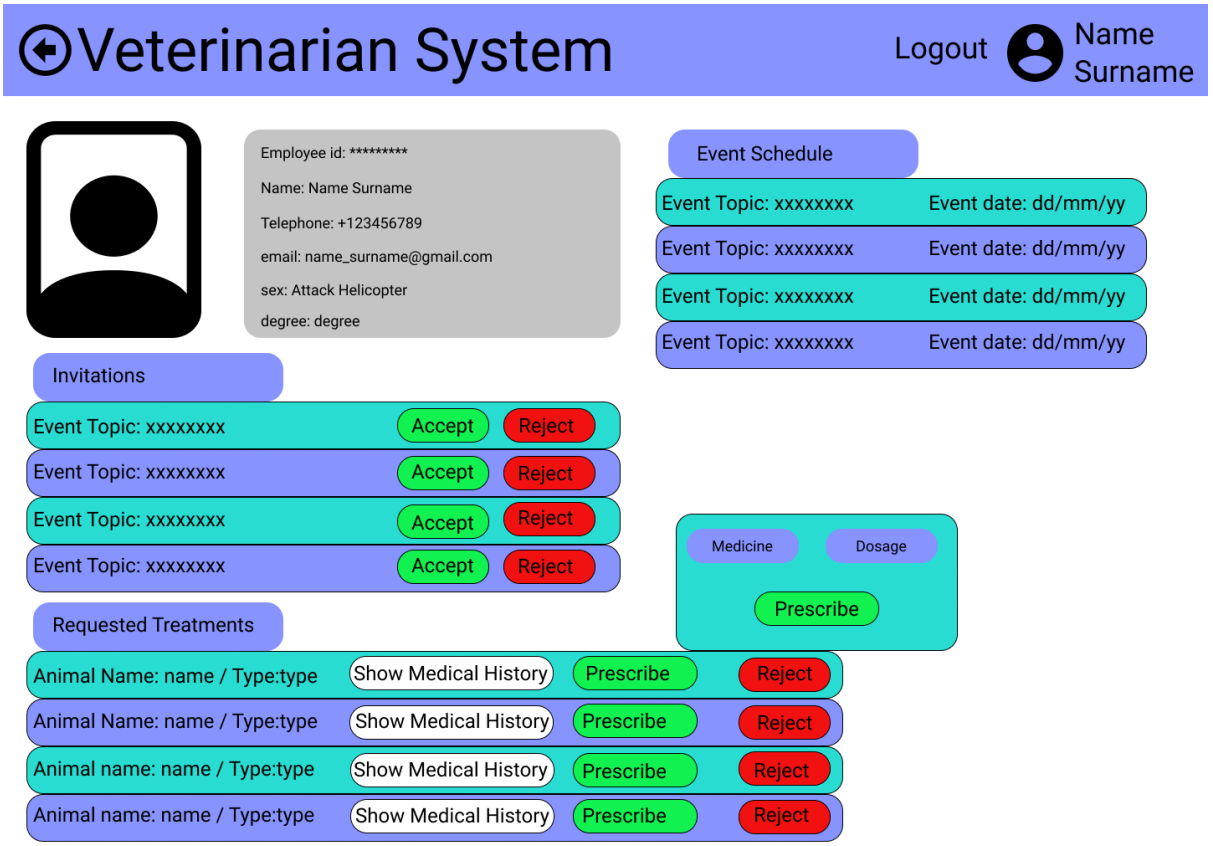
View Requested Treatments

```
SELECT name, condition
FROM treatment
WHERE username = @username
```

View Event Schedule

SELECT topic, date
FROM invite NATURAL JOIN Educational_event
ON invitee = username
WHERE request_status = ‘A’

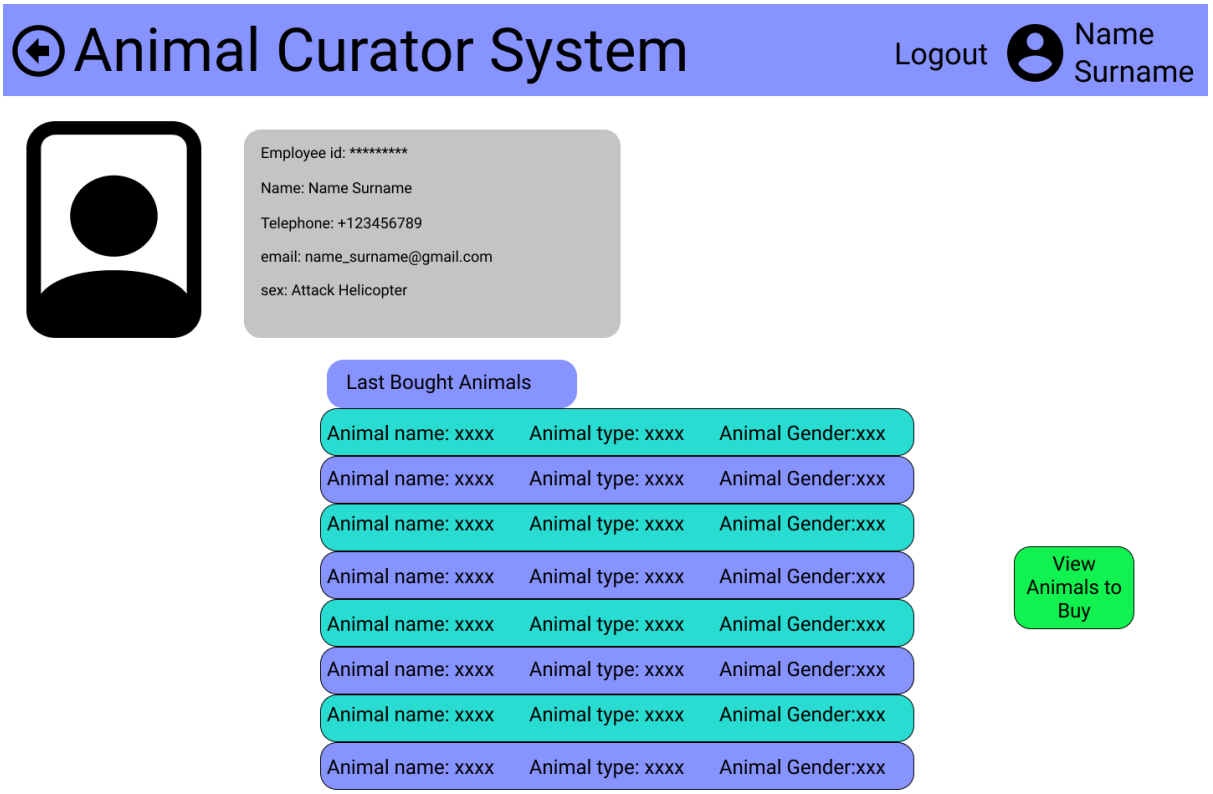
Prescribe Pop-up



Inputs:
Process: This is the same page when the prescribe button is clicked. The veterinarian can select the specific medicine and the dosage for a treatment and finish the prescription.
SQL Statements: same as the SQL for Veterinarian Profile Screen

5.4.Animal Curator UI

Animal Curator Profile Screen

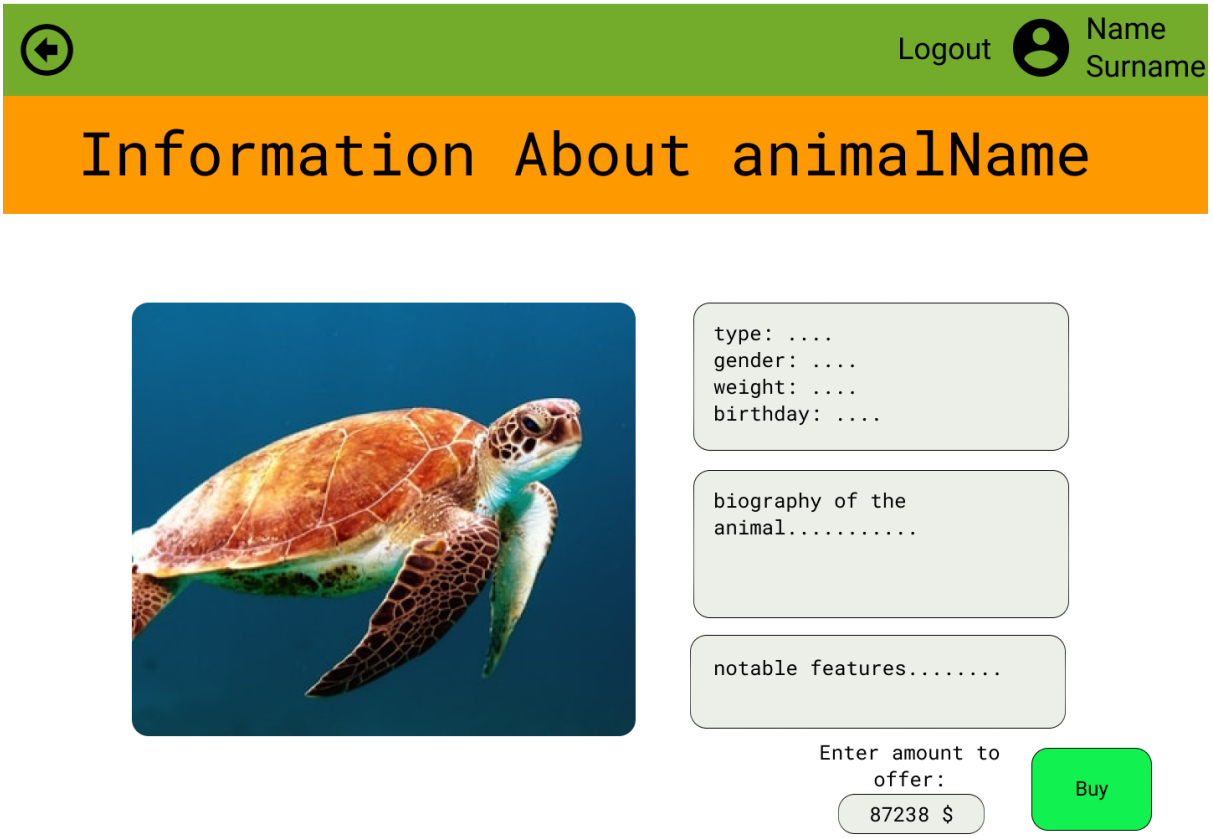


Inputs:
Process: This is the profile page of the animal curator when they enter the system. At the top, again, there is general information about the animal curator. The animal curator can see the lastly bought animals and when they click on the “view animals to buy” button, they are navigated to a new page.
SQL Statements:

View Animal Curator Info
SELECT *
FROM Animal_Curator
WHERE username = @username

View Bought Animals
SELECT name, type, gender
FROM Animal
WHERE bought_by = @username

Buy Animal Screen



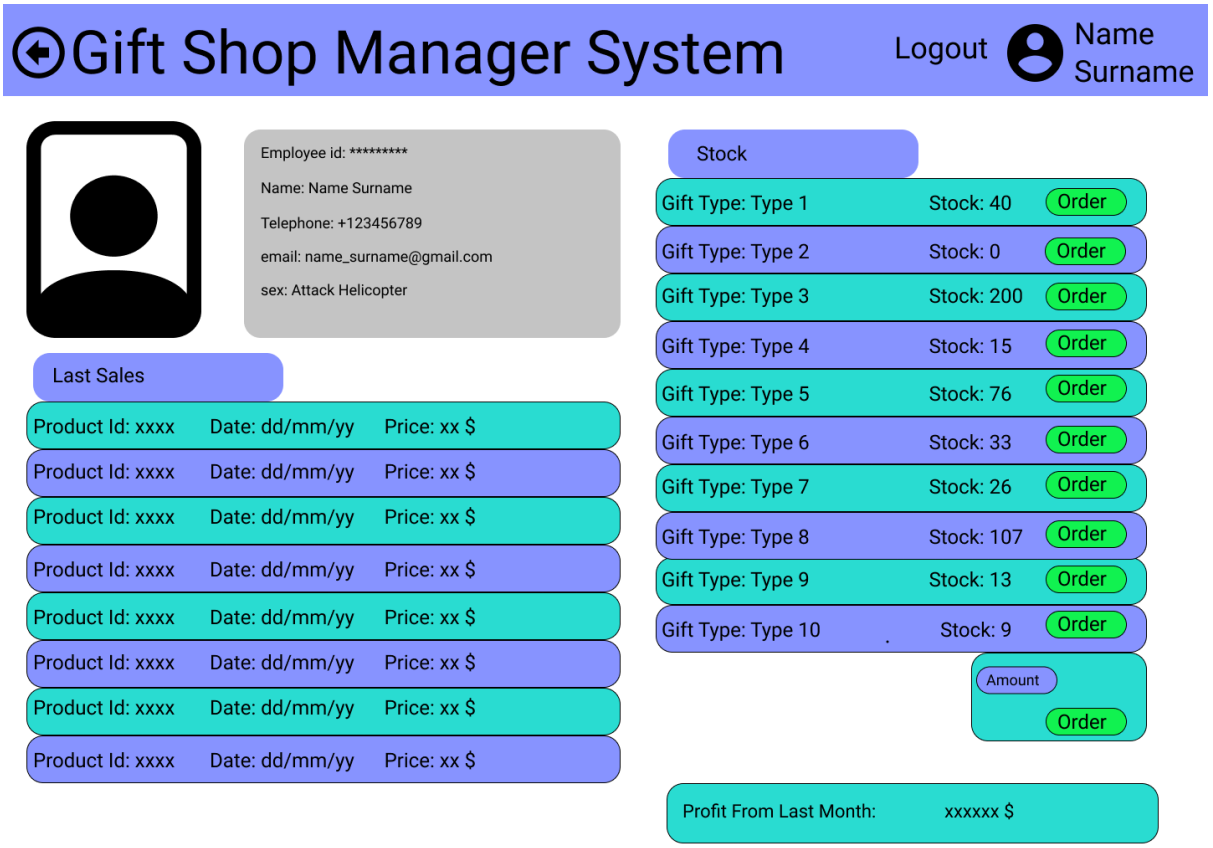
Inputs: @offerToBuyAnimal
Process: This is the page when an animal curator searches for an animal to buy. The animal curator can state an offer for an animal.
SQL Statements:

Displaying Animal Details
SELECT name, type, gender, weight, birthday, biography, notable_features
FROM Animal
WHERE name= @name

Buy Animal
INSERT INTO buys
VALUES(name, @username, @offerToBuyAnimal)

5.5.Gift Shop Manager UI

Gift Shop Manager Profile Screen



Inputs:

Process: This is the profile page of a gift shop manager when they enter the system. At the top, again, there is general information about the gift shop manager. The gift shop manager can see the last sales, the stock of the products and the profit the gift shop made last month.

SQL Statements:

View Gift Manager Info

```
SELECT *
FROM Giftshop_Manager
WHERE username = @username
```

View Sales

```
SELECT product_code, date, price
FROM Gift NATURAL JOIN Giftshop_Manager
ON shop = giftshop_name
WHERE username = @username
```

View Stock

```
SELECT name, stock as COUNT(product_code)
FROM Gift
GROUPED BY name
WHERE stock > 0
```

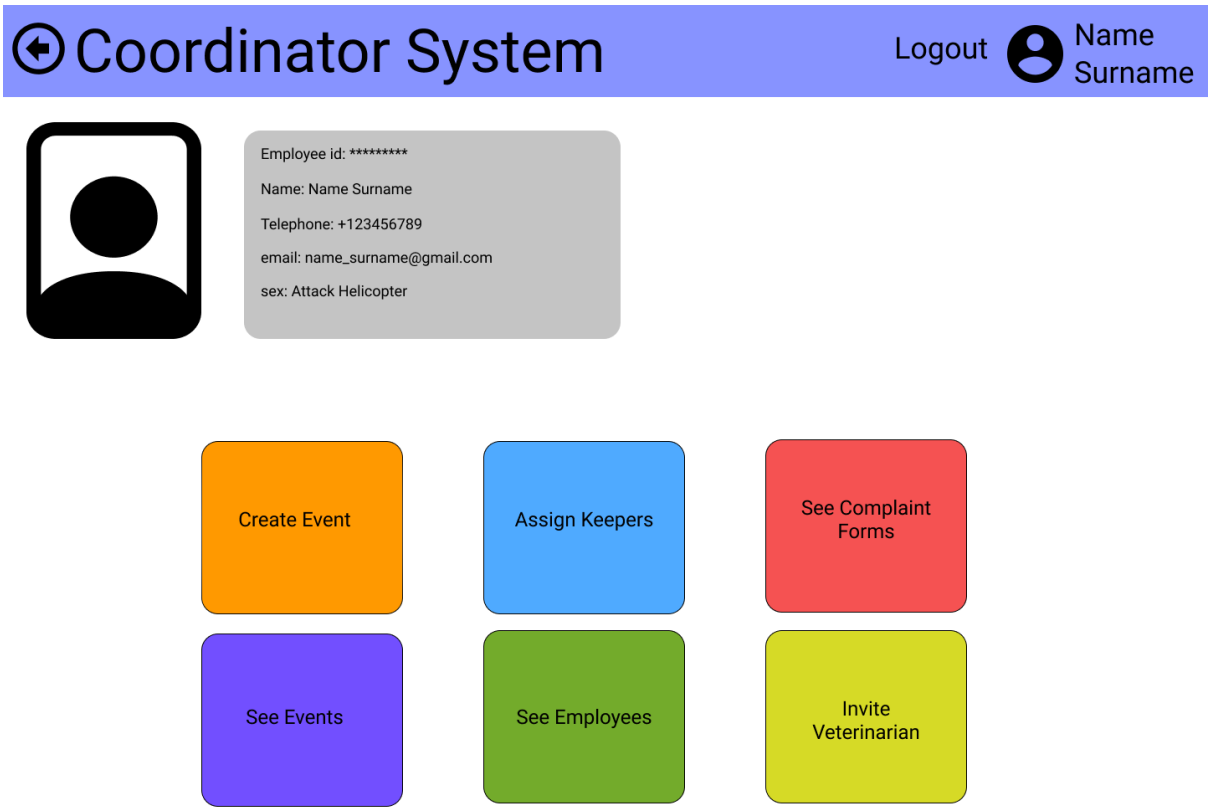
View Profit From Last Month

```
SELECT p as SUM(price)
```

FROM Buys NATURAL JOIN Gift NATURAL JOIN Giftshop
ON Gift.shop = Giftshop.name HAVING DATEDIFF(mm, NOW(), buy_date) < 1

5.6.Coordinator UI

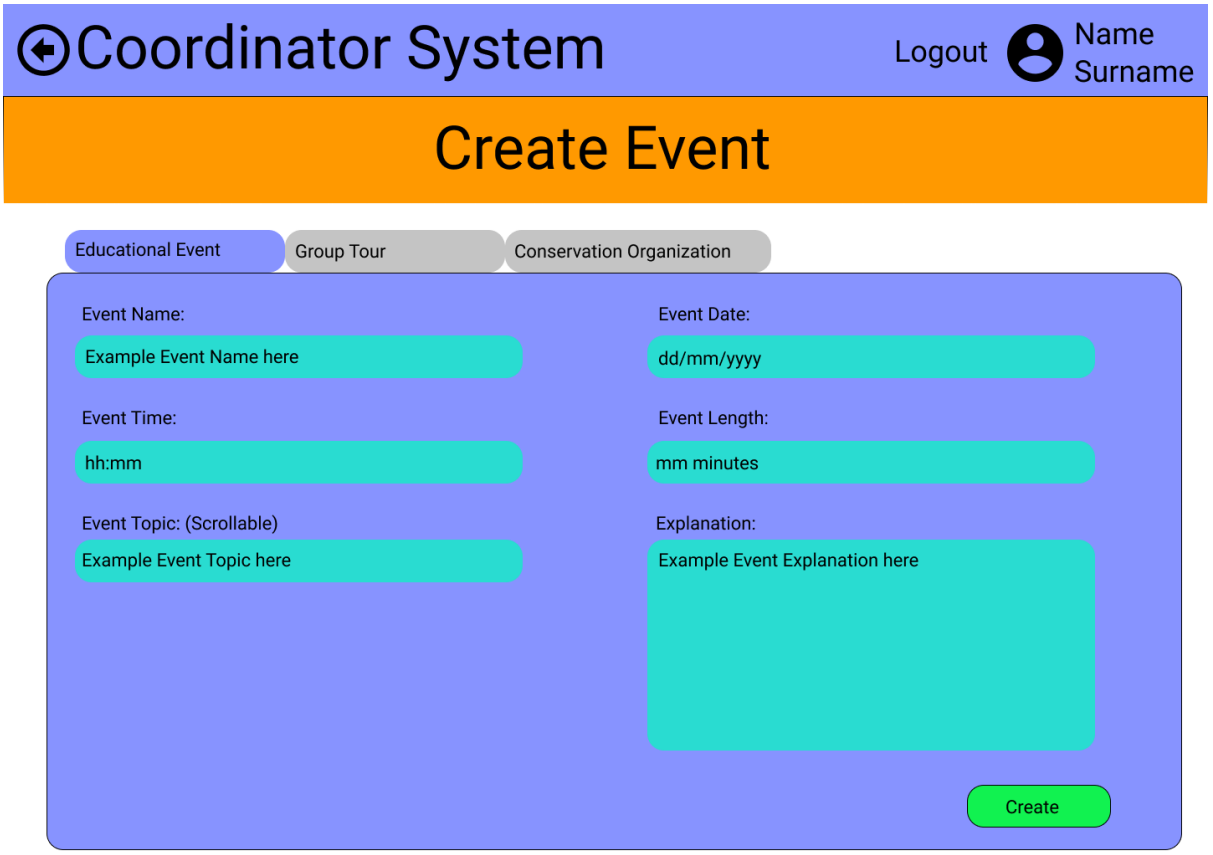
Coordinator Profile Screen



Inputs:
Process: This is the profile page of a coordinator when they enter the system. At the top, again, there is general information about the coordinator. The coordinator can see the pages to create an event, see the events, assign keepers to cages, see all employees, see complaint forms and invite veterinarians by clicking the respective button.
SQL Statements:

View Coordinator Info
SELECT *
FROM Coordinator
WHERE username = @username

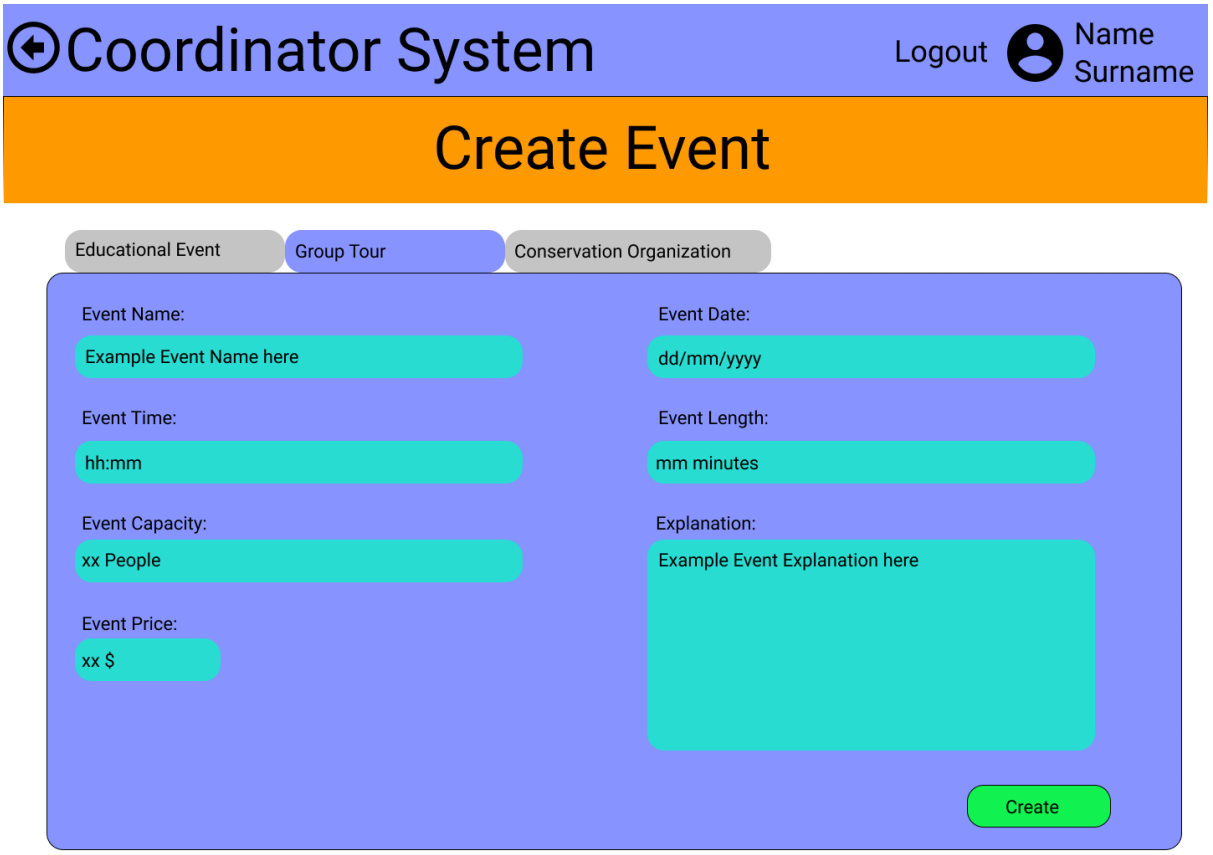
Create Educational Event Screen



Inputs: @eventName, @time, @topic, @date, @length, @explanation
Process: This is the page when the create event button is clicked by the coordinator. The coordinator can select the type of the event from the tabs. This is the display when the educational event is selected. The coordinator should enter an event name, event date, starting time, event length, event topic and the explanation of the event. Then, clicking the create button, the event is created.
SQL Statements:

Create Educational Events Info
INSERT INTO Educational_events
VALUES(@eventName, @date, @topic)
INSERT INTO Event
VALUES(@eventName, @date, @explanation, @length, @time)

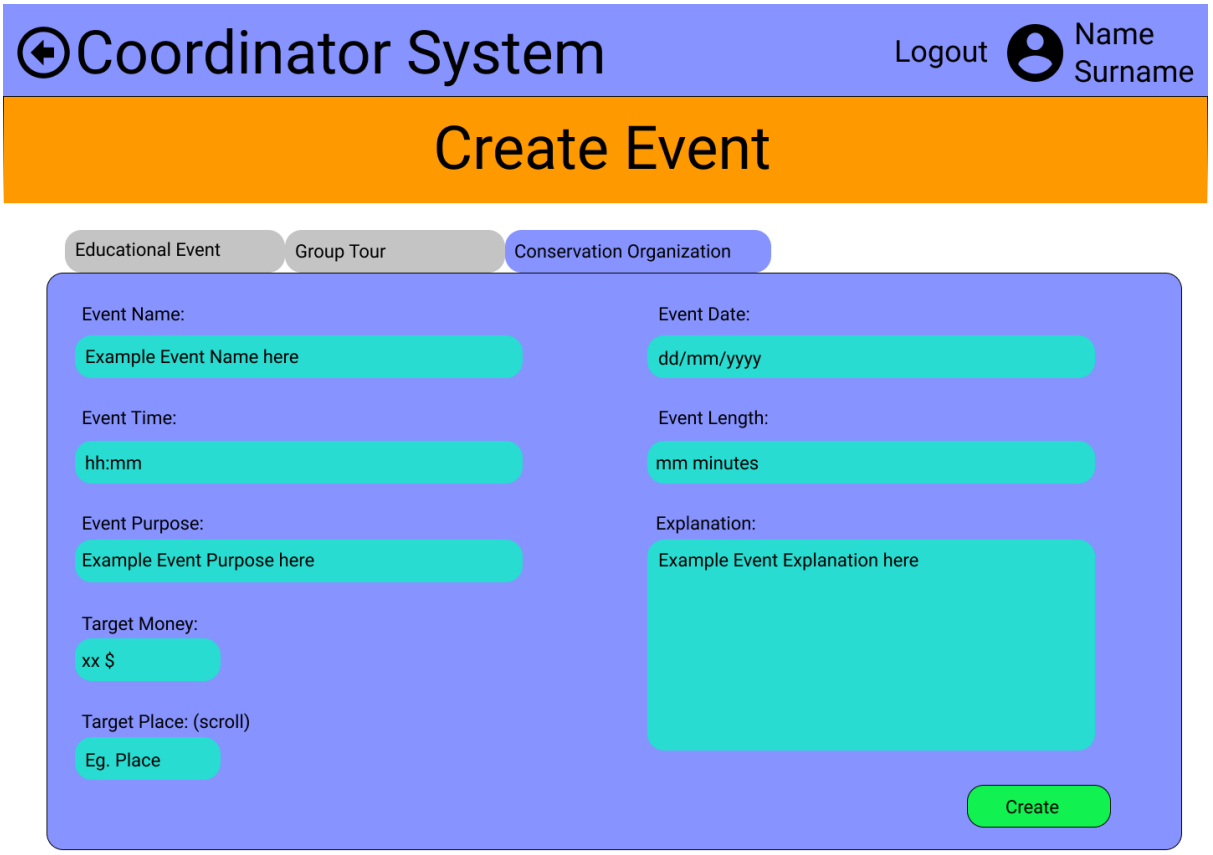
Create Group Tour Screen



Inputs: @name, @time, @capacity, @price, @date, @length, @explanation
Process: This is when the group tour tab is selected. The coordinator should enter the event name, event date, starting time, event length, event capacity, the price and the explanation of the event. Then, clicking on the create button, the event is created.
SQL Statements:

Create Group Tour Events Info
INSERT INTO Group_tour
VALUES(@eventName, @date, @explanation, @length, @time, @capacity, @price)
INSERT INTO Event
VALUES(@eventName, @date, @explanation, @length, @time)

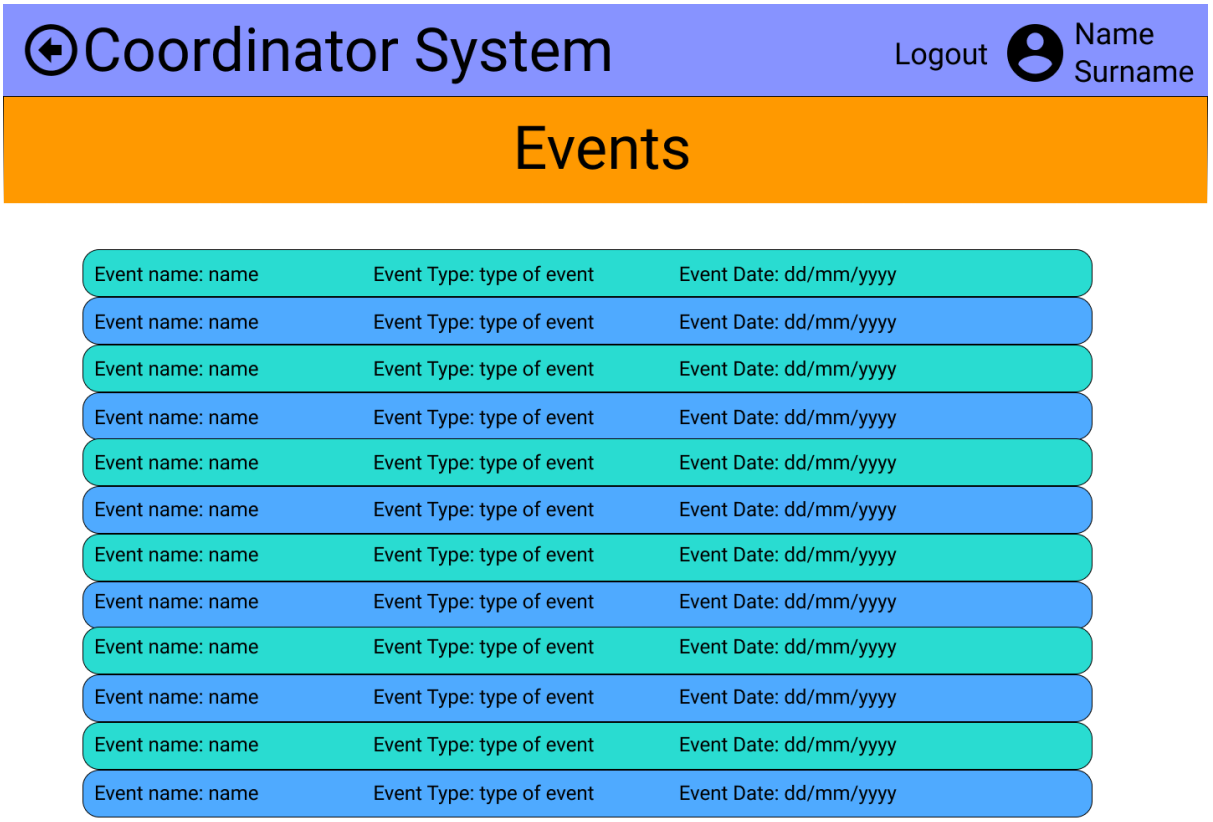
Create Conservational Organization Screen



Inputs: @name, @time, @purpose, @targetMoney, @targetPlace, @date, @length, @explanation
Process: This is how the page seems when the conservation organization tab is selected. The coordinator should enter the event name, event date, starting time, event length, event purpose, the target money that the organization wants to reach, the place that the event will occur and the explanation of the event. Then, by clicking the create button, the event is created.
SQL Statements:

Create Conservation Organization Events Info
INSERT INTO Conservation_Organization
VALUES(@eventName, @date, @explanation, @length, @time, @purpose, @targetMoney, @targetPlace)
INSERT INTO Event
VALUES(@eventName, @date, @explanation, @length, @time)

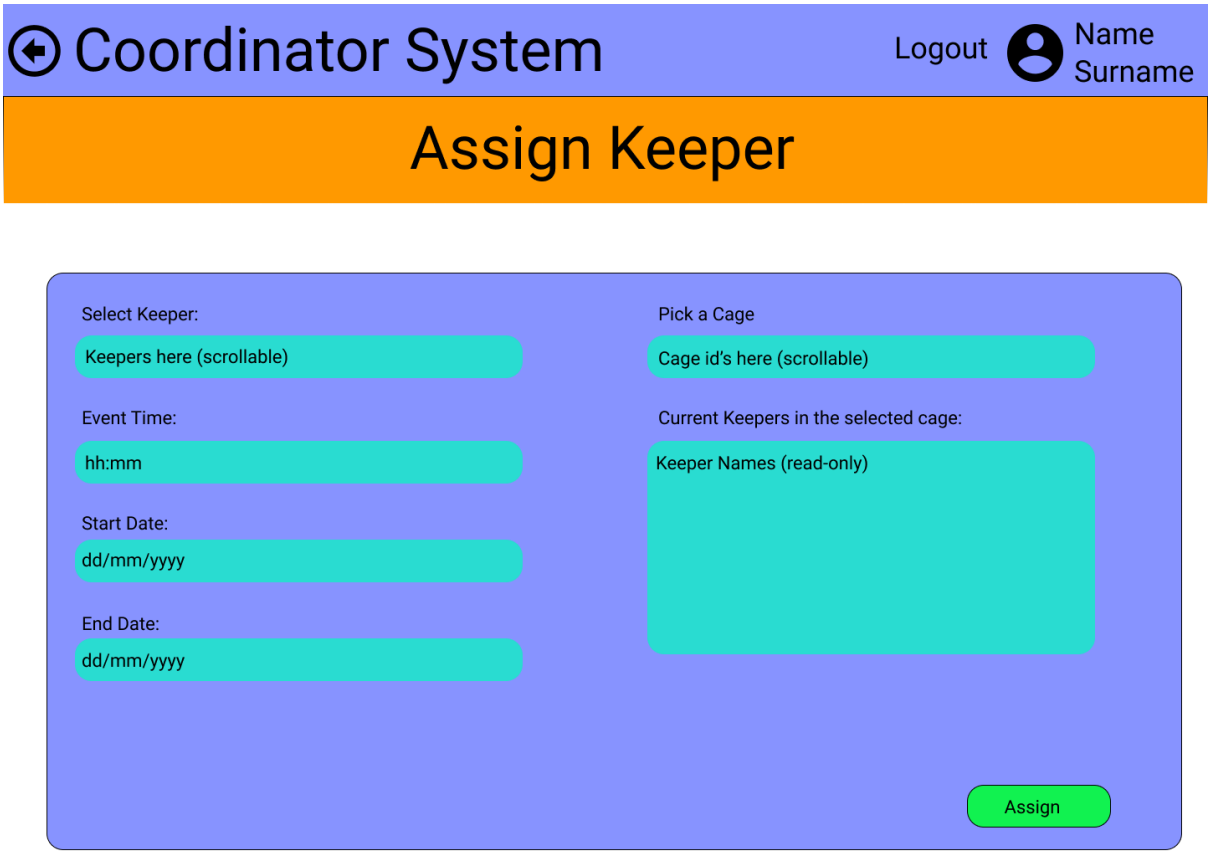
Browse Events Screen



Inputs:
Process: This the page when the events button is clicked by the coordinator. The coordinator, simply, sees all the events that will happen.
SQL Statements:

View Events
SELECT event_name, event_type, event_date
FROM Events

Assign Keeper Screen

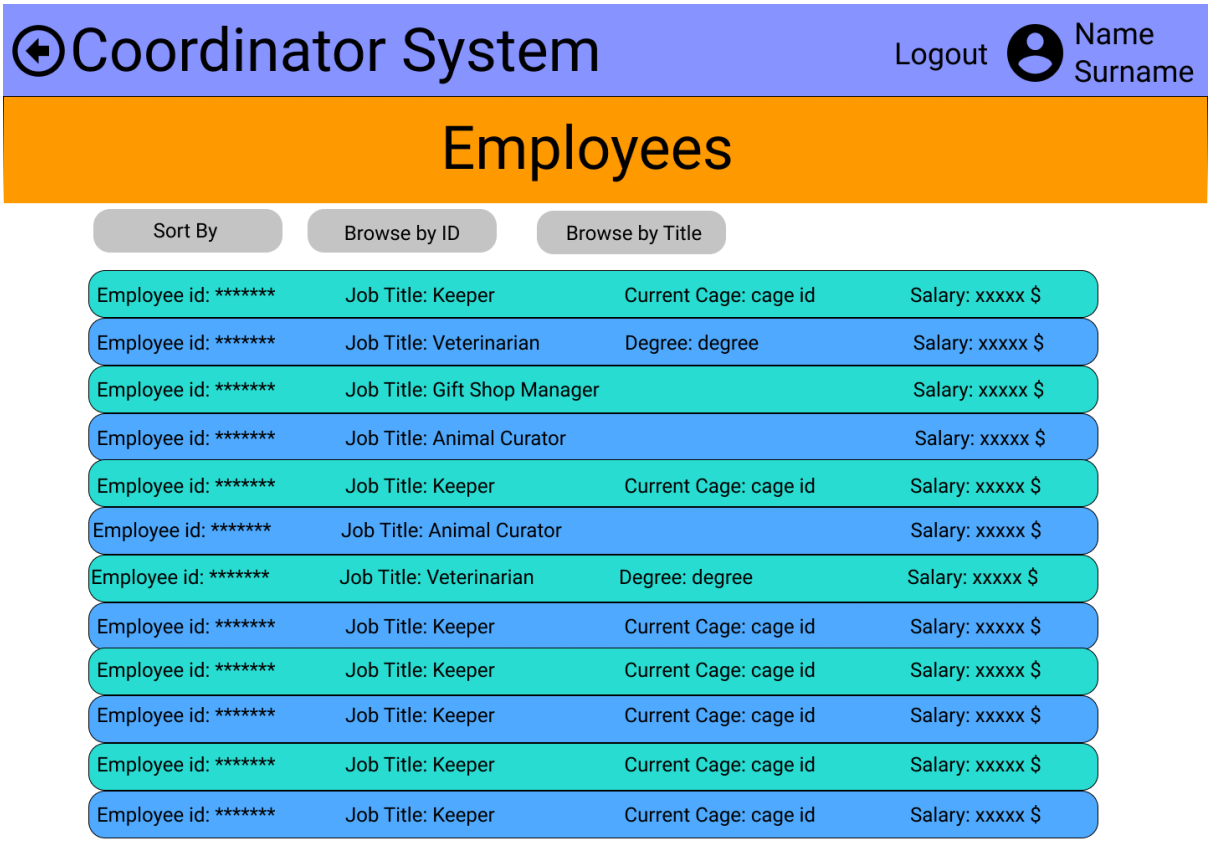


Inputs: @keeperUsername, @time, @startDate, @endDate, @cage_id, @coordUsername
Process: This is the page when assign keeper button is clicked by the coordinator. On this page, a coordinator can assign keepers to cages. The coordinator should select the keeper, the cage that the leeper will be assigned to, the time, the starting and ending date. Current keepers in the selected cage will be displayed as read-only. Then, clicking the assign button, the keeper will be assigned to the cage.
SQL Statements:

View Currently Assigned Keepers
SELECT name, surname
FROM assigns NATURAL JOIN Keeper
WHERE @cage_id = cage_id

Assign Keepers
INSERT INTO assigns
VALUES(@coordUsername , @cage_id@keeperUsername, @time, @startDate, @endDate, @cage_id)

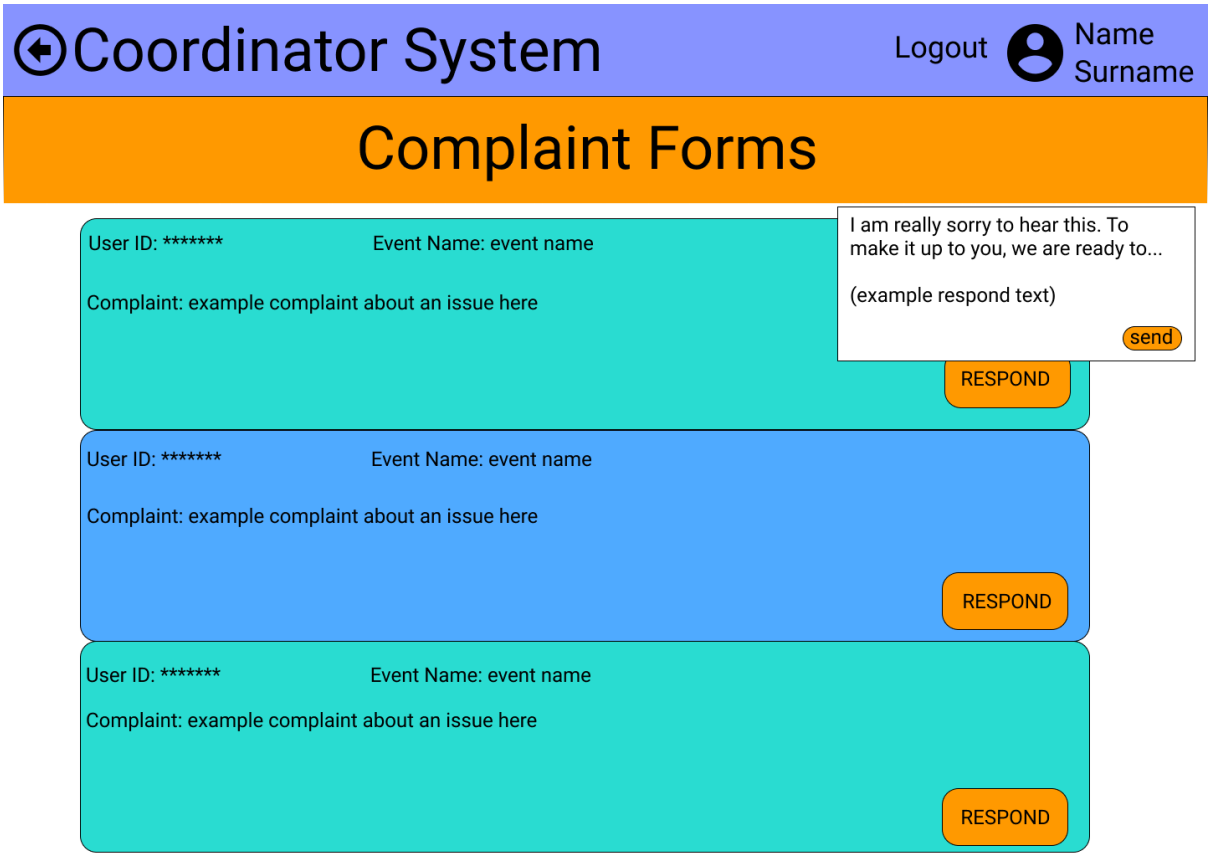
View Employees Screen



Inputs:
Process: This is the page when see employees button is clicked by the coordinator. The coordinator, simply, sees all the employees in the zoo. They can sort the employees by name, salary or job title. The coordinator can also browse an employee by their id or their job title.
SQL Statements:

View Employees
SELECT username, job_title, salary
FROM User
WHERE type <> “Visitor”

View Complaints Screen

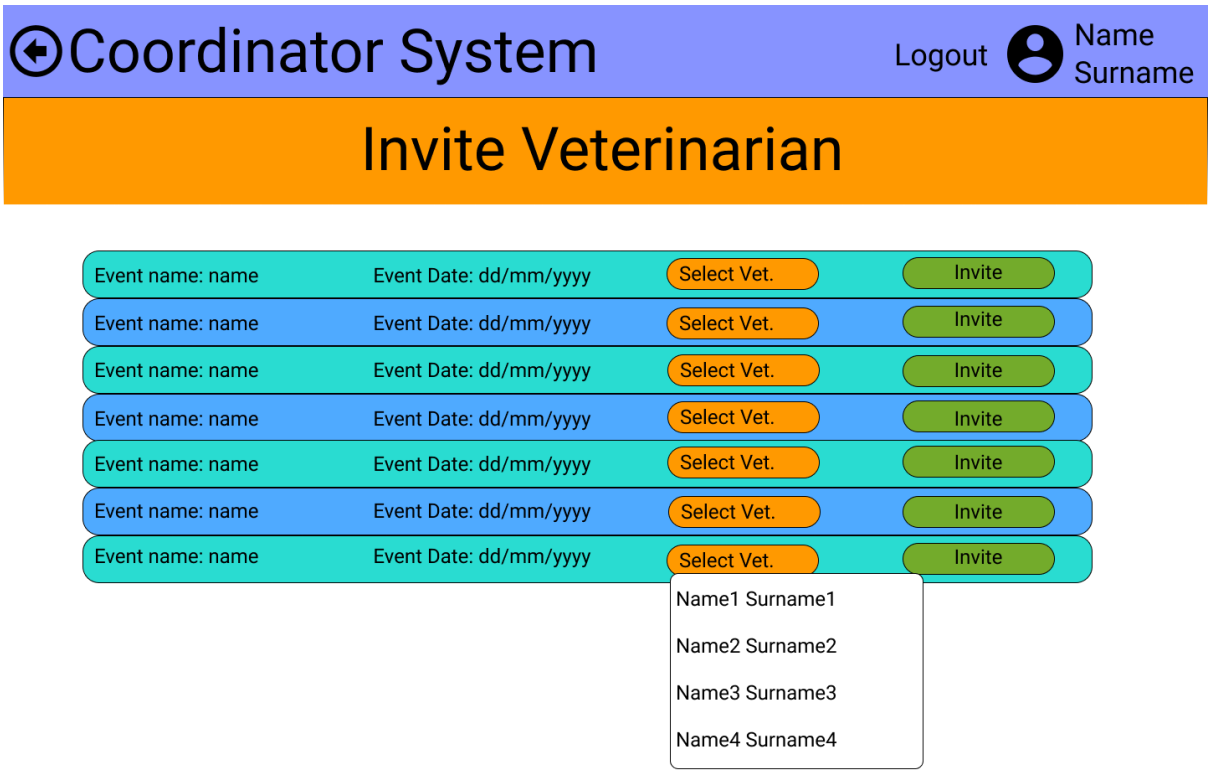


Inputs: @respondText
Process: This is the page when see complaint forms button is clicked by the coordinator. Here, the coordinator can see all the complaint forms made by users for a specific event. The coordinator can also respond to these complaints.
SQL Statements:

View Complaint Forms
SELECT username, event_name, complaint_type
FROM g_has_f NATURAL JOIN Complaint_Form

Respond to Complaint
INSERT INTO Complaint_Form
VALUES(form_id, message, date, complaint_type, @respondText)

Invite Veterinarian Screen

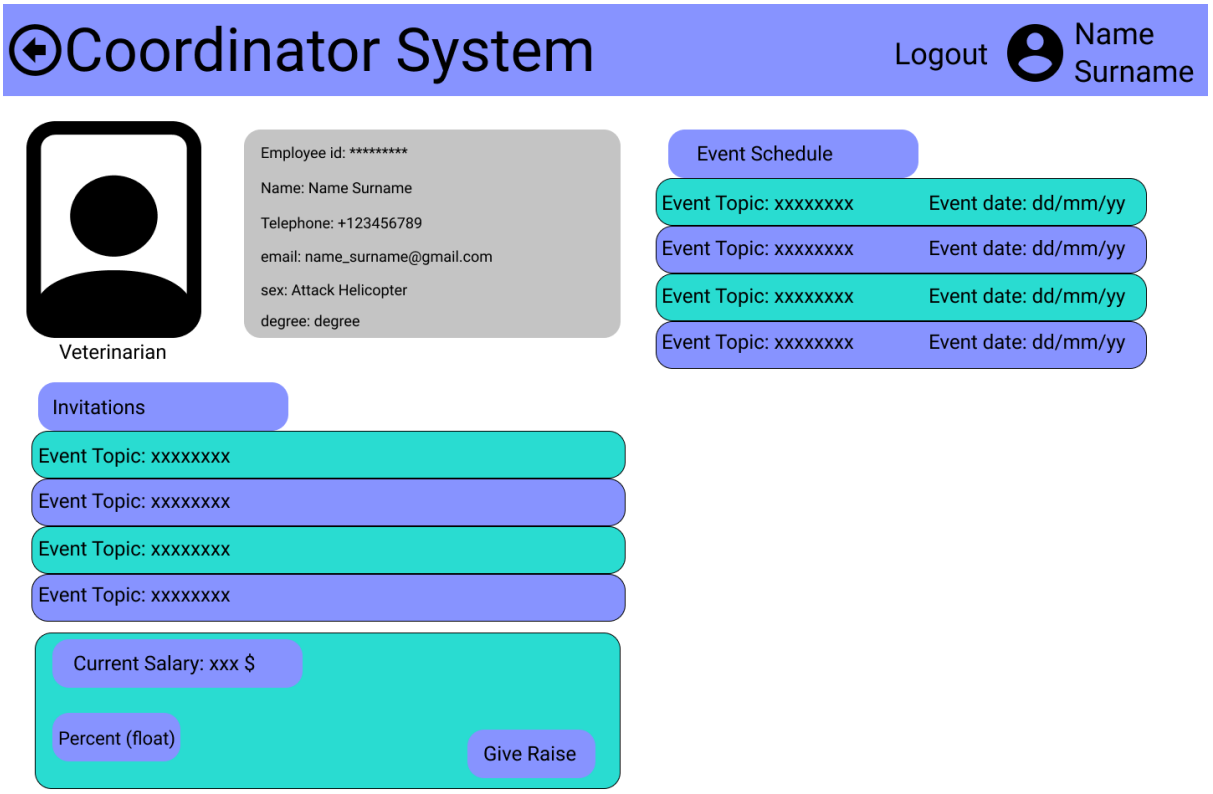


Inputs:
Process: This is the page when invite veterinarian button is clicked by the coordinator. The coordinator can see all the educational events to invite a veterinarian. They can select a veterinarian for each event and send their invitation.
SQL Statements:

Display Educational Events
SELECT event_name, date
FROM Educational_Events

Select Veterinarians
SELECT name, surname
FROM Veterinarian

View Veterinarian Profile Screen



Inputs: @raise
Process: This is the page when the coordinator clicks on a veterinarian on the “see employees” page. The coordinator can see the veterinarian’s profile as above. The coordinator can see the invitations that the veterinarian had, the scheduled events for the veterinarian, and the coordinator can give the veterinarian a raise in their salary by entering the raise percent.
SQL Statements:

View Veterinarian Info
SELECT *
FROM Veterinarian
WHERE username = @vetUsername

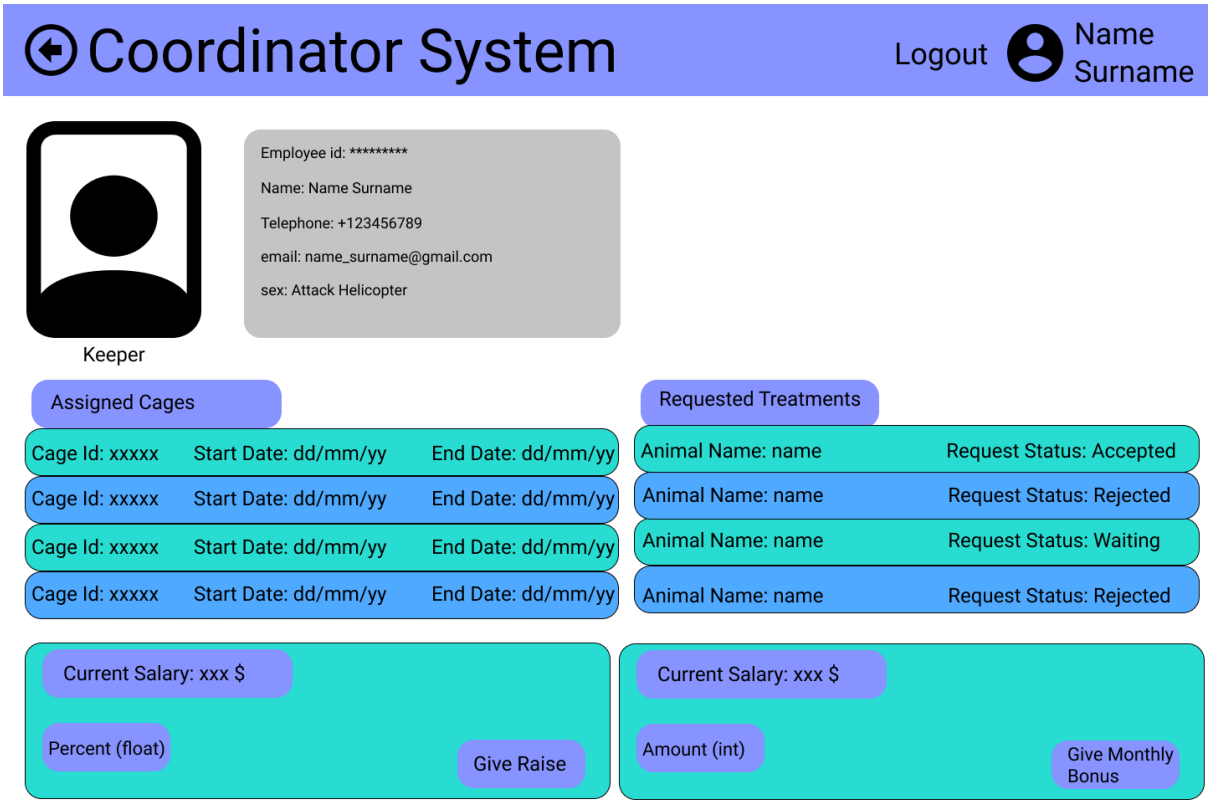
View Invitations of Veterinarian
SELECT topic
FROM invite NATURAL JOIN Educational_event
WHERE username = @vetUsername

View Event Schedule
SELECT topic, date
FROM invite NATURAL JOIN Educational_event
WHERE username = @vetUsername and request_status <> ‘R’

Give Raise
UPDATE Employee

SET salary = salary + @raise
WHERE username = @vetUsername

View Keeper Profile Screen



Inputs: @raise, @bonus
Process: This is the profile page of the keeper when a coordinator wants to see their profile. The coordinator can see the cages that the keeper is assigned to and the treatments that the keeper requested. The coordinator can give raise to the keeper by entering the percent or the coordinator can give the keeper a monthly bonus by entering the amount of the bonus.
SQL Statements:

Display Keeper Info
SELECT *
FROM Keeper
WHERE username = @keeperUsername

Display Assigned Cages
SELECT cage_id, start_date, end_date
FROM assigns
WHERE username = @keeperUsername

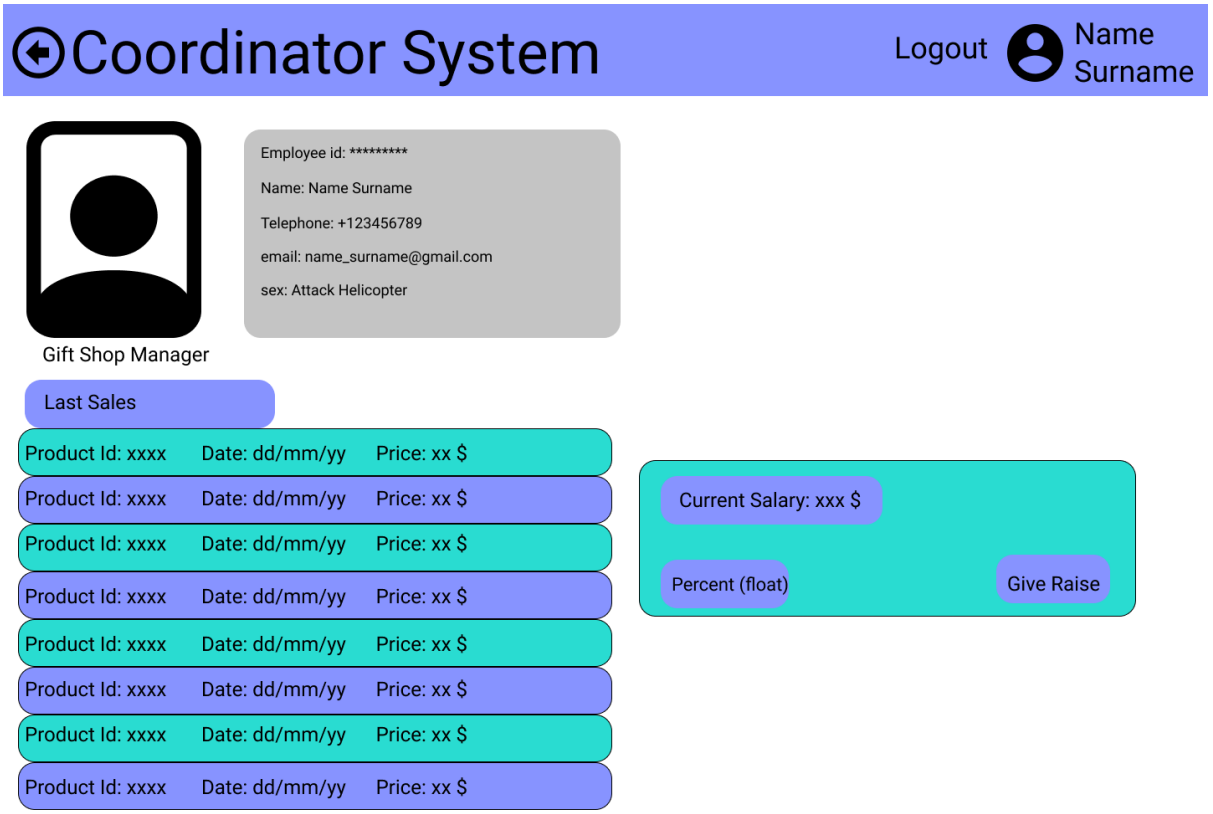
Display Requested Treatments
SELECT name, request_status
FROM treatment

WHERE username = @keeperUsername

Give Raise
UPDATE Employee
SET salary = salary + @raise
WHERE username = @keeperUsername

Give Monthly Bonus
INSERT INTO give_bonus
VALUES(username, @bonus, @keeperUsername)

View Gift Shop Manager Profile



Inputs: @raise
Process: This is the profile page of a gift shop manager when the coordinator wants to see it. The coordinator can see the last sales of the gift shop that the manager is responsible for and the coordinator can give the manager a raise.
SQL Statements:

Display GiftshopManager Info
SELECT *
FROM GiftShopManager
WHERE username = @giftShopManagerUsername

Display Sales
SELECT product_code, date, price

FROM Gift NATURAL JOIN Giftshop_Manager
ON shop = giftshop_name
WHERE username = @giftShopManagerUsername

Give Raise
UPDATE Employee
SET salary = salary + @raise
WHERE username = @giftShopManagerUsername

6. Advanced Database Components

a. Reports

i. List all animals by cage

SELECT name, type
FROM Animal
GROUPED BY cage_id

ii. Animals by Food

SELECT name, type, food
FROM Animal NATURAL JOIN eats
GROUPED BY food_id

iii. Best Keeper

SELECT username
FROM (select * from User where type = “Keeper”), SUM(amount) as m_bonus from give_bonus grouped by k_username
HAVING m_bonus = MAX(m_bonus))

b. Triggers

i. When a gift is sold the animal curator who bought the animal and the keeper of the animal gets 3% of the gifts sale value as monthly_bonus.
ii. When adding a new Donation if target_money is met explanation of the Conservation organization should be updated by starting target met.
iii. When giving a raise to an employe, if the salary is higher than the minimum salary of next job_title change job_title of the Employee
iv. When a gift is sold the animal curator who bought the animal should get a stipend increase by 20% of the gift sale price.
v. When the capacity of the group tour is increased, price should increase accordingly.
vi. When assigning a new keeper current work of the keeper must be inserted to previous work of Keeper.

c. Constraints

i. When adding an user type can only be “Visitor”, “Keeper”, “Coordinator”, “Veterinarian”, “Giftshop Manager” or “Animal Curator”

7. Implementation Plan

Our system will be created using the technologies; Postgresql for the database, Spring-boot for the backend and HTML,CSS and Thymeleaf for the Front-end. Backend calls will be made using JQuery.

8. Website

Our project information website link: https://atayurtsever.github.io/Zoo/