CS353 - Database Systems

Final Report

Group 4

Online Ticket Purchasing App for Transportation

Project Link: https://github.com/aemirbosnak/SuBilet

Ahmet Emir Boşnak - Section 2 - 22002398

Hasan Alp Caferoğlu - Section 1 - 22203991

Ozan Can Gülbaz - Section 1 - 22002019

Yağız Özkarahan - Section 1 - 22002276

4 June 2023

1. System Description

- Our system, ŞuBilet, is an online ticket purchasing application for various travel types in Turkey. Through our system, travelers are able to reserve and purchase transportation tickets, write reviews for their trips, create journeys and purchase/reserve the trips in their journeys, and use coupons to make discounted transactions. Our other user type, travel companies, are able to create travels for travelers to purchase, and also manage their travels as they desire. Admins of our system are able to manage the system, validate companies, delete users, and also generate various reports about the statistics of the website.

- ŞuBilet relies on a MySQL database and exclusively manually written SQL queries to retrieve and manage its information. Backend of the project is written in Python with Flask and the frontend functionalities are realised through Python's Jinja template.

- The GitHub repository of ŞuBilet also comes with a built Dockerfile and docker-compose to ensure that it runs the same on the devices on which it may be tested. Therefore, it is recommended to run the system through Docker to ensure a reliable experience. The repository of ŞuBilet is located in this link: **https://github.com/aemirbosnak/SuBilet**

2. Contributions of Each Member

2.1. Yağız Özkarahan

- Proposal Report:
  - I have written the non-functional requirements.
  - I have written the introduction segment.
  - The E/R diagram was made collaboratively with all members of the group, on a live session. I also was an active collaborator in this process.
- Design Report:
  - Revised E/R diagram was again made collaboratively with all members of the group on a live session. I also was an active collaborator in this process.
  - I wrote the main SQL queries of the system, in the “Part 4: UI Design and Corresponding SQL Queries” part of the report.
  - I wrote the implementation plan segment of the report.
  - I wrote the algorithms and data structures segments of the report, although it was a short segment.
  - Did the final polishing and structuring of the report (page margins, titles etc.)
- Implementation:
  - I have written all of the html/css files for the Traveler pages.
  - I have implemented the functionality for creating journeys and adding trips to journeys for the Traveler user.
- Final Report:
  - I have written the system description segment.
  - I have written the implementation details segment.
  - I have also added the tables and E/R diagrams, although most of them have not changed since the design report
  - I have written the user’s manual for docker build and traveler user.
  - I have written my own segment in this part.

2.2. Ahmet Emir Boşnak

- Proposal Report:
    - Written functional requirements.
    - Polished and finalized the E/R diagram after the structure (tables, relations, etc.) was decided collectively.
- Design Report:
    - Created mock-up UI designs for the use cases.Updated and revised the relevant SQL queries for he corresponding UI pages, supported with details about the implementation specifics.
- Implementation:
    - Implemented the coupon system.
    - Refined the travel search page with a date range option together with Ozan. Added checks for displaying travel search results.
    - Implemented the ticket purchase/reservation functionality
        - Implemented pnr and seat number generation for ticket purchases/reservations.
        - Connected coupon system with the purchase/reservation system.
        - Connected the journey system with the purchase/reservation system accordingly.
        - Implemented purchasing previously reserved ticket within the travels and purchase pages.
    - Implemented filtering system for traveler's past and upcoming travels.
    - Implemented purchase/reservation functionality for travels in journeys
        - Implemented buy all/reserve all functionality for journeys
- Final Report:
    - Written my contributions part.

2.3. Hasan Alp Caferoğlu

- Proposal Report:
    - I have written the limitations part of the report.
    - I actively participated in the design of the E/R diagram. In the online session which is conducted for determining the tables and their relations, I provide different solutions.
    - I partially contributed to the drawing of the E/R diagram by using online tools.
- Design Report:
    - During the construction of the revised E/R diagram with all teammates, I contributed well.
    - I wrote changes in the E/R diagram in this step detailly.
    - I wrote functional dependencies, candidate keys and primary keys for each relation schema. Besides that, I wrote explanation which shows schemas are in either BCNF or 3NF.
    - I wrote use cases/scenarios for all users in the applications.
    - I worked with Ozan to construct advanced database components.
    - I have checked the written SQL queries in the UI design part and made necessary corrections.
- Implementation:
    - I contributed to the revised version of the E/R.
    - I have constructed the initial schema.sql file to have initial tuples to use in the application.
    - I constructed the views.
    - Login/register functionality is done by me.
    - I have implemented change/edit profile pages for both travelers and companies.
    - I have implemented the rating and commenting on travel scenario.
    - I have implemented all use cases/scenarios for the company except sorting lists of things like travels. The implementations made by me for the company are written below.
        - Listing company's all travels.
        - Registering a new travel.
        - Viewing details of a travel.
        - Updating an upcoming travel.
        - Deleting a travel.
        - Deleting a purchase and refunding.
        - Deleting purchase and generating new bookings
    - I have implemented almost all use cases/scenarios for the Admin except sorting and searching companies. The implementations made by me for the admin are written below.
        - Generating application report and printing it.

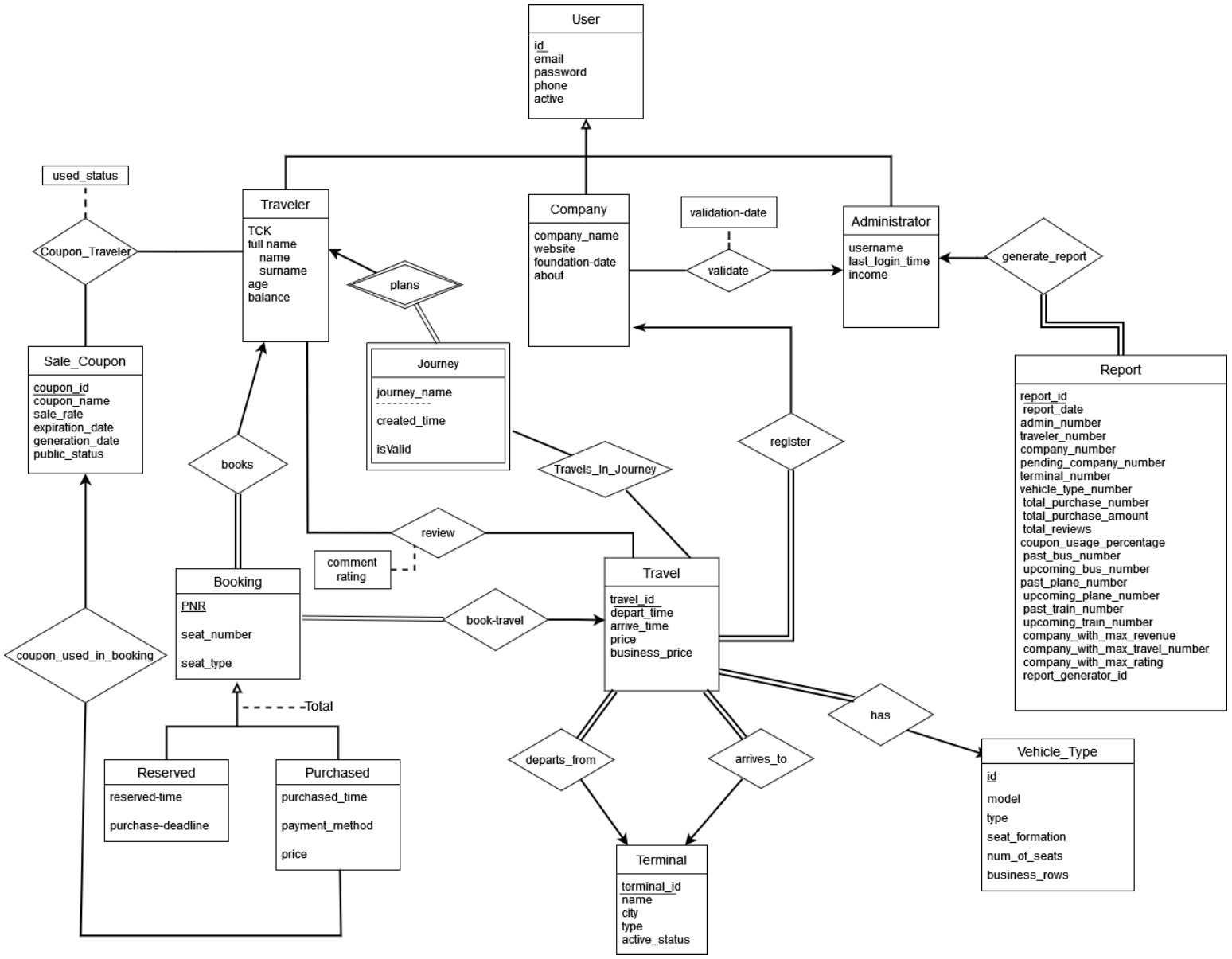
        - Listing companies. Filtering companies such as listing validated, unvalidated or inactive companies.
        - Deactivation of a company. Sub Functions of deactivation scenario.
        - Deletion of a company.Sub Functions of deletion scenario.
        - Validating a company.
        - Generating new coupons.
        - Deleting a coupon.
        - Listing vehicle types.
        - Adding new vehicle types.
        - Listing all terminals.
        - Adding new terminals.
        - Deleting a purchase without a refund. Deleting a purchase with a refund. Deleting a purchase and then making a new booking for the same traveler.
        - Deleting a reservation.
        - Adding a purchase with money deduction or without money deduction from the traveler.
        - Deleting a review.
- Final Report:
    - Written my contributions.

2.4. Ozan Can Gülbaz

- Proposal Report:
    - Took active part in database design.
    - Took part in the E/R diagram drawing with other group members.
    - Written the project description.
- Design Report:
    - Took part in revision of the E/R diagram and tha database design.
    - Contributed to relation schemes, refining the design and decomposing some relations.
    - Contributed to SQL queries in UI design, particularly coupons.
    - Written the advanced database components with Alp.
- Implementation:
    - Did general bug fixing over the project.
    - Implemented seat choosing system and UI.
    - Refined purchase and reserve functionalities.
    - Implemented sorting in searches and range search for travel date.
    - Implemented flexible company search for admin.
    - Fixed visual and functional issues of journeys.
- Final Report:
    - Written personal contributions.
    - Small contribution to the implementation details part.
    - Written Admin user's manual.

3. Final E/R Diagram

Difference from the design report: Report entity has changed. More system indicators are added as attributes.



4. Final List of Tables

Difference from the design report: Report table has changed

4.1. User

Table Schema: `User(id, email, password, phone, active)`

Candidate Keys: id, email, phone

Primary Key: id

Functional Dependencies: F = {
id → email, password, phone, active,
email → id,
phone → id }

Normal Form: Attribute email functionally determines the id (email → id) and also attribute phone functionally determines the id (phone → id). Since attribute id is selected as primary key which means functionally determines all attributes, email and phone are superkey for the User relation. So, the relation satisfies the condition of BCNF.

4.2. Traveler

Table Schema: `Traveler(id, TCK, name, surname, age, balance)`

Candidate Keys: { id, TCK }

Primary Key: { id }

Foreign Keys:

```
FOREIGN KEY(id) REFERENCES User(id)
```

Functional Dependencies: F = {
id → TCK, name, surname, age, balance,
TCK → id }

Normal Form: Attribute TCK functionally determines the id (TCK → id) Since attribute id is the primary key which means functionally determines all attributes, TCK is a super key for the Traveler relation. So, the relation satisfies the BCNF conditions. TCK is a candidate key because TCK is unique (superkey) and minimal.

4.3. Company

Table Schema: `Company(id, company_name, website, foundation_date, about, validator_id, validation_time)`

Candidate Keys: { id, company_name, website }

Primary Key: id

Foreign Keys:

```
FOREIGN KEY(id) REFERENCES User
FOREIGN KEY(validator_id) REFERENCES Administrator(id)
```

Functional Dependencies: F = {
id → company_name, website, foundation_date, about, validator_id, validation_time,
company_name → id,
website → id }

Normal Form: Attribute company_name functionally determines the id and website functionally determines the id. Since attribute id is the primary key which functionally determines all attributes, company_name and website are super keys. So, the relation satisfies the BCNF conditions. company_name and website are also candidate keys because both attributes are unique (superkey) and minimal.

4.4. Administrator

Table Schema: `Administrator(id, username, last_login_time, income)`

Candidate Keys: { id, username }

Primary Key: id

Foreign Keys:

```
FOREIGN KEY(id) REFERENCES User
```

Functional Dependencies: F = {
id → username, last_login_time, income,
username → id }

Normal Form: Attribute username functionally determines the id (username → id) Since attribute id is the primary key which means functionally determines all attributes, username is a super key. So, the relation satisfies the BCNF conditions. Attribute username is a candidate key because it is unique (superkey) and minimal.

4.5. Report

Table Schema: Report(report_id, report_date, admin_number, traveler_number, company_number, pending_company_number, terminal_number, vehicle_type_number, total_purchase_number, total_purchase_amount, total_reviews, coupon_usage_percentage, past_bus_number, upcoming_bus_number, past_plane_number, upcoming_plane_number, past_train_number, upcoming_train_number, company_with_max_revenue, company_with_max_travel_number, company_with_max_rating, report_generator_id)

Candidate Keys: report_id, report_date

Primary Key: report_id

Foreign Keys:

FOREIGN KEY(report_generator_id) REFERENCES Administrator(id),
FOREIGN KEY(company_with_max_revenue) REFERENCES Company(company_name),
FOREIGN KEY(company_with_max_travel_number) REFERENCES Company(company_name)
FOREIGN KEY(company_with_max_rating) REFERENCES Company(company_name)

Functional Dependencies: F = {
report_id → report_date, total_sales, total_reviews, total_travels, total_company, pending_company, total_travelers, total_bus, total_train, total_plane, report_generator_id,
report_date → report_id }

Normal Form: The functional dependency report_date → report_id satisfies the BCNF conditions.

4.6. Sale_Coupon

Table Schema: Sale_Coupon(coupon_id, coupon_name, sale_rate, expiration_date, generation_date, public_status)

Candidate Keys: coupon_id

Primary Key: coupon_id

Functional Dependencies: F = {
coupon_id → coupon_name, sale_rate, expration_date, generation_date, puclic_status }

Normal Form: Attribute coupon_id functionally determines all attributes of the relation. For that reason it is super key. This indicates that the relation is in BCNF. Since coupon_id is unique and minimal, coupon_id is a candidate key. The only candidate key is coupon_id and it is selected as the primary key.

4.7. Coupon_Traveler

Table Schema: Coupon_Traveler(coupon_id, user_id, used_status)

Candidate Keys: (coupon_id, user_id)

Primary Key: (coupon_id, user_id)

Foreign Keys:

```
FOREIGN KEY(coupon_id) REFERENCES Sale_Coupon(coupon_id),
FOREIGN KEY(user_id) REFERENCES Traveler(id)
```

Functional Dependencies: F = { coupon_id, user_id → used_status}

Normal Form: Both coupon_id and user_id functionally determine the used_status. This set of attributes (coupon_id, user_id) is super key for the relation. For that reason, the relation is in BCNF. Additionally this set of attributes is minimal which makes it only candidate key.

4.8. Vehicle_Type

Table Schema: `Vehicle_Type(id, model, type, seat_formation, num_of_seats, business_rows)`

Candidate Keys: `id, model`

Primary Key: `id`

Functional Dependencies: F = {
id → model, type, seat_formation, num_of_seats, business_rows
model → id }

Normal Form: Attribute model functionally determines the id (model → id) Since attribute id is the primary key which means functionally determines all attributes, model is a super key. So, the relation satisfies the BCNF conditions. Attribute model is also a candidate key because it is unique (superkey) and minimal.

4.9. Terminal

Table Schema: `Terminal(terminal_id, name, city, type, activate_status)`

Candidate Keys: `terminal_id, name`

Primary Key: `terminal_id`

Functional Dependencies: F = {
terminal_id → name, city, type, activate_status
name → terminal_id }

Normal Form: Attributes name functionally determines the terminal_id. Since terminal_id is the primary key which means functionally determines all attributes, name is a super key. So, the relation satisfies the BCNF conditions. Attributes name is also a candidate key because it is unique (superkey) and minimal.

4.10. Travel

Table Schema: Travel(travel_id, travel_company_id, departure_terminal_id, arrival_terminal_id, depart_time, arrive_time, price, business-price, vehicle_type_id)

Candidate Keys: travel_id

Primary Key: travel_id

Foreign Keys:

```
FOREIGN KEY(travel_company_id) REFERENCES Company(id)
FOREIGN KEY(departure_terminal_id) REFERENCES
          Terminal(terminal_id)
FOREIGN KEY(arrival_terminal_id) REFERENCES
          Terminal(terminal_id)
FOREIGN KEY(vehicle_type_id) REFERENCES Vehicle_Type(id)
```

Functional Dependencies: F = {
travel_id → travel_company_id, departure_terminal_id, arrival_terminal_id, depart_time, travel_duration, arrive_time, price, business-price, vehicle_type_id
}

Normal Form: Attribute travel_id is super key which functionally determines all the attributes of the relation. Attribute travel_id is also candidate key since it is unique and minimal. It is also selected as the primary key.

4.11. Booking

Table Schema: Booking(PNR, travel_id, seat_number, seat_type traveler_id)

Candidate Keys: PNR

Primary Key: PNR

Foreign Keys:

```
FOREIGN KEY (travel_id) REFERENCES Travel
FOREIGN KEY (traveler_id) REFERENCES Traveler
```

Functional Dependencies: F = { PNR → travel_id, seat_number, seat_type, traveler_id }

Normal Form: Attribute PNR functionally determines all the attributes of the relation, so it is a super key. PNRis also a candidate key since it is unique and minimal. It is also selected as the primary key.

4.12. Reserved

Table Schema: `Reserved(PNR, reserved_time, purchase_deadline)`

Candidate Keys: PNR

Primary Key: PNR

Foreign Keys:

```
FOREIGN KEY (PNR) REFERENCES Booking(PNR)
```

Functional Dependencies: F = { PNR → reserved_time, purchase_deadline }

Normal Form: Attribute PNR functionally determines all the attributes of the relation, so it is a super key. PNR is also a candidate key since it is unique and minimal. It is also selected as the primary key.

4.13. Purchased

Table Schema: Purchased(PNR, purchased_time, payment_method, price, coupon_id)

Candidate Keys: PNR

Primary Key: PNR

Foreign Keys:

```
FOREIGN KEY (PNR) REFERENCES Booking(PNR)
     ON DELETE CASCADE,
FOREIGN KEY (coupon_id) REFERENCES Sale_Coupon(coupon_id)
```

Functional Dependencies: F = { PNR → purchased_time, payment_method, price, coupon_id)

Normal Form: Attribute PNR functionally determines all the attributes of the relation, so it is a super key. PNRis also a candidate key since it is unique and minimal. It is also selected as the primary key.

4.14. Journey

Table Schema: `Journey(journey_name, traveler_id, created_time, isValid)`

Candidate Keys: {journey_name, traveler_id}

Primary Key: { journey_name, traveler_id }

Foreign Keys:

```
FOREIGN KEY (traveler_id) REFERENCES Traveler(id)
```

Functional Dependencies: F = { journey_name, traveler_id → created_time, isValid }

Normal Form: The set of attributes journey_name and traveler_id functionally determine all the attributes of the relation, so this set of attributes is a super key. It is also a candidate key since it is unique and minimal. It is also selected as the primary key.

4.15. Travels_In_Journey

Table Schema: `Travels_In_Journey(journey_name, traveler_id, travel_id)`

Candidate Keys: { journey_name, traveler_id, travel_id }

Primary Key: { journey_name, traveler_id, travel_id }

Foreign Keys:

```
FOREIGN KEY (journey_name) REFERENCES Journey(journey_name)
FOREIGN KEY (traveler_id) REFERENCES Traveler(id)
FOREIGN KEY (travel_id) REFERENCES Travel(travel_id)
```

Functional Dependencies: F = {
journey_name, traveler_id, travel_id → journey_name, traveler_id, travel_id)

Normal Form: Functional dependency written above is trivial. The relation is in BCNF.

4.16. Review

Table Schema: Review(travel_id, traveler_id, comment, rating)

Candidate Keys: { travel_id, traveler_id }

Primary Key: { travel_id, traveler_id }

Foreign Keys:

```
FOREIGN KEY (travel_id) REFERENCES Travel(travel_id)
FOREIGN KEY (traveler_id) REFERENCES Traveler(id)
```

Functional Dependencies: F = {
travel_id, traveler_id → comment, rating)

Normal Form: The set of attributes travel_id and traveler_id functionally determine all the attributes of the relation, so this set of attributes is a super key. It is also a candidate key since it is unique and minimal. It is also selected as the primary key.

5. Implementation Details

5.1. Database

- For the implementation of the database, MySQL was used.
- First, the “schema.sql” file was filled with “create table” entries of our tables and these tables were populated with sample data. After deploying the database with Docker (see Part 7.1. of this report for instructions), the schema.sql was connected to MySQL client. To do this:
    - Open the schema.sql file with the MySQL client (assuming it is already installed on the computer)
    - Select “Database->Connect to database” or simply Ctrl+U
    - From the “Parameters” tab, select hostname as “localhost” and **Port as 3307.** Select username as “root”.
    - Proceed and enter password. The password is “password”.
    - The connection is now established.
- After the connection was established, we were able to view our created tables and run our queries on our database.
- Through the project, MySQL was used to test our written queries on the database and verify their results.
- Additionally, views are generated with create view statements in order to create another security layer and decrease the repetition.
- MySql library for Python allows us to connect with our database, execute queries and obtain results of these queries.
- Referential integrity is ensured with foreign key constraints and with options when a tuple is deleted. These foreign key constraints and options are stated in create table statements.
- MySql 5.7 doesn't support the WITH clause. For that reason, we had to write longer queries occasionally.

5.2. Backend

- For the implementation of Backend, Python was used as the programming language.
- To establish the connection with our MySQL client, Flask was used as a microframework for Python. With Flask, we were able to access, read and update our database, and add or delete data.
    - To do these operations, **we have exclusively used handwritten SQL queries.**
    - We **have not used** any methods or technologies that would automatically access and manipulate the database without writing queries.
- The backend, with Flask, attaches the Python functions we have written to our specified website routes. For a rather simple example, login function is run when the localhost:5000/login page is entered.
    - Our specified html files are also rendered in these routes to display the page.
- We have written the entire backend logic inside one singular app.py file. Although this is not an object oriented approach and it is not recommended for group projects of large scale, it was not an obstacle for a project of this scale as we were working on different parts of the file to implement different functionalities. Therefore, we avoided conflicts.

5.3. Frontend

- For the design of the frontend, Bootstrap was used as the main library alongside HTML and CSS.
- The structuring and placing of the components on the page were done using Bootstrap's container, row and column system.
- To implement the dynamic functionalities of the pages, Flask's Jinja2 template was used. Our main necessity for frontend functionality was about displaying our data (for example displaying the list of travels received from the backend and making it look reactive), so Jinja2 template was useful enough with its simplicity.
    - Although when we needed extra functionalities, we have also occasionally used JavaScript with inline <script></script> tags.

6. Advanced Database Components

6.1. Views

- **Company_traveler_info view:**

```
CREATE VIEW company_traveler_info_view AS
SELECT TCK, name, surname, age, email, phone
FROM User JOIN Traveler ON User.id = Traveler.id;
```

- **Travel_with_vehicle_detail view:**

```
/*
travel_with_vehicle_detail_view is for obtaining detail
information about both travel and vehicle
*/
CREATE VIEW travel_with_vehicle_detail_view AS
SELECT *
FROM Travel T
JOIN Vehicle_Type V ON V.id = T.vehicle_type_id;
```

- **Travel_detail view:**

```
/*
travel_detail_view is for obtaining detail
information about both travel, terminal and vehicle
*/
CREATE VIEW travel_detail_view AS
SELECT
T.travel_id AS travel_id,
T.travel_company_id AS travel_company_id,
T.depart_time AS depart_time,
T.arrive_time AS arrival_time,
T.price AS price,
T.business_price AS business_price,
Dep.name AS departure_terminal_name,
Dep.city AS departure_city,
Ar.name AS arrival_terminal_name,
Ar.city AS arrival_city,
V.model AS vehicle_model,
V.type AS vehicle_type
FROM Travel T
JOIN Terminal Dep ON T.departure_terminal_id = Dep.terminal_id
JOIN Terminal Ar ON T.arrival_terminal_id = Ar.terminal_id
JOIN Vehicle_Type V ON V.id = T.vehicle_type_id;
```

- **Companies_travels_detail view:**

```
/*
companies_travels_detail_view is for obtaining detail
information about travel of a company
*/
CREATE VIEW companies_travels_detail_view AS
SELECT
C.id AS company_id,
C.company_name AS company_name,
TDW.travel_id AS travel_id,
TDW.depart_time AS depart_time,
TDW.arrival_time AS arrival_time,
TDW.price AS price,
TDW.business_price AS business_price,
TDW.departure_terminal_name AS departure_terminal_name,
TDW.departure_city AS departure_city,
TDW.arrival_terminal_name AS arrival_terminal_name,
TDW.arrival_city AS arrival_city,
TDW.vehicle_model AS vehicle_model,
TDW.vehicle_type AS vehicle_type
FROM Company C
JOIN travel_detail_view TDW ON C.id = TDW.travel_company_id;
```

7. User's Manual

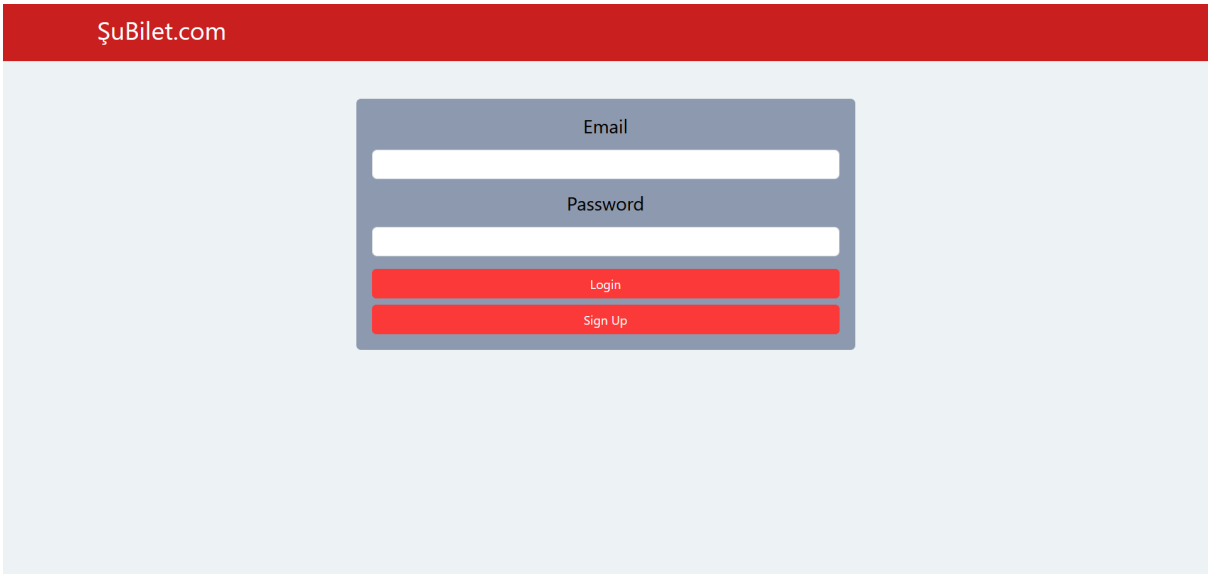
7.1. Docker Build

We recommend running the system through Docker to guarantee highest compatibility with our intended product. Our Docker build is very similar to the build in Homework 4 so it should be easy to build and run. After downloading the source code from our github repository (https://github.com/aemirbosnak/SuBilet), the steps are as follows:

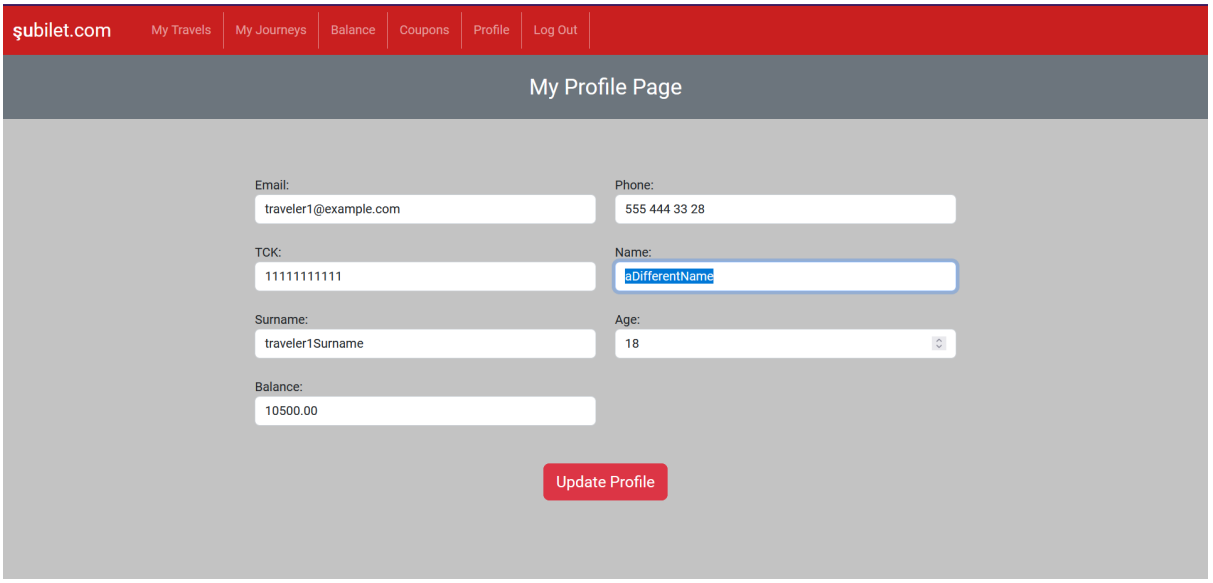
- Navigate to the folder "SuBilet". This is the folder with the Dockerfile and docker-compose.yaml
- Open the computer terminal.
- Enter: docker build -t subiletapp .
- After the build process is completed, you can verify it by entering: docker images.
- Then, enter: docker-compose up -d
- After the process is completed, you can view the services by entering: docker-compose ps
- This should output the database and web services that are currently running.
- Once the services are running, you can use Docker app to run/stop the Docker image and go to the application address. It should be running at **localhost:5000**

7.2. Traveler User

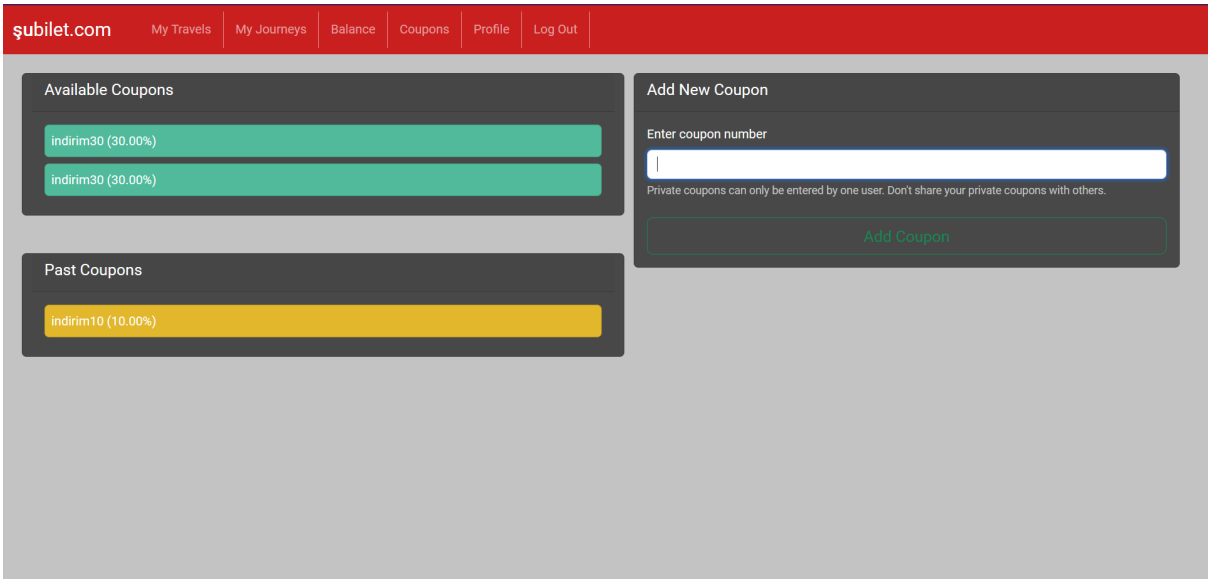
As the first step, navigate to the Login page from the navbar and login as a traveler.



From this page, you can use one of the travelers in our database, or sign up as a new traveler. One particular example credentials for a traveler is "traveler1@example.com" with the password "123456".

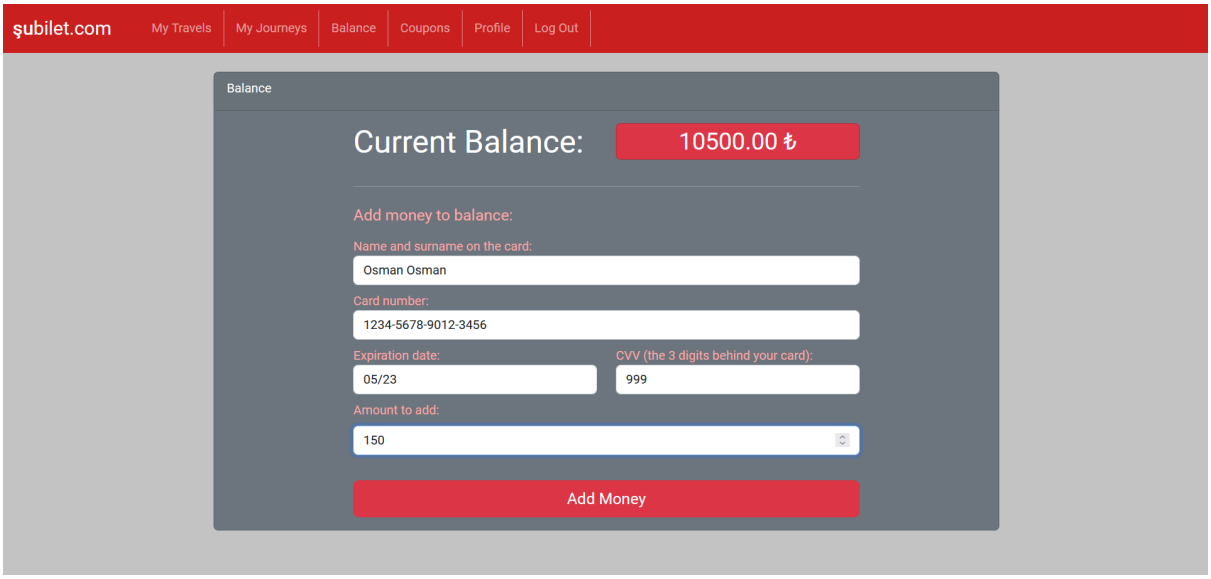


From the profile page, you can view the profile information of this user and update them as you wish (except the balance).

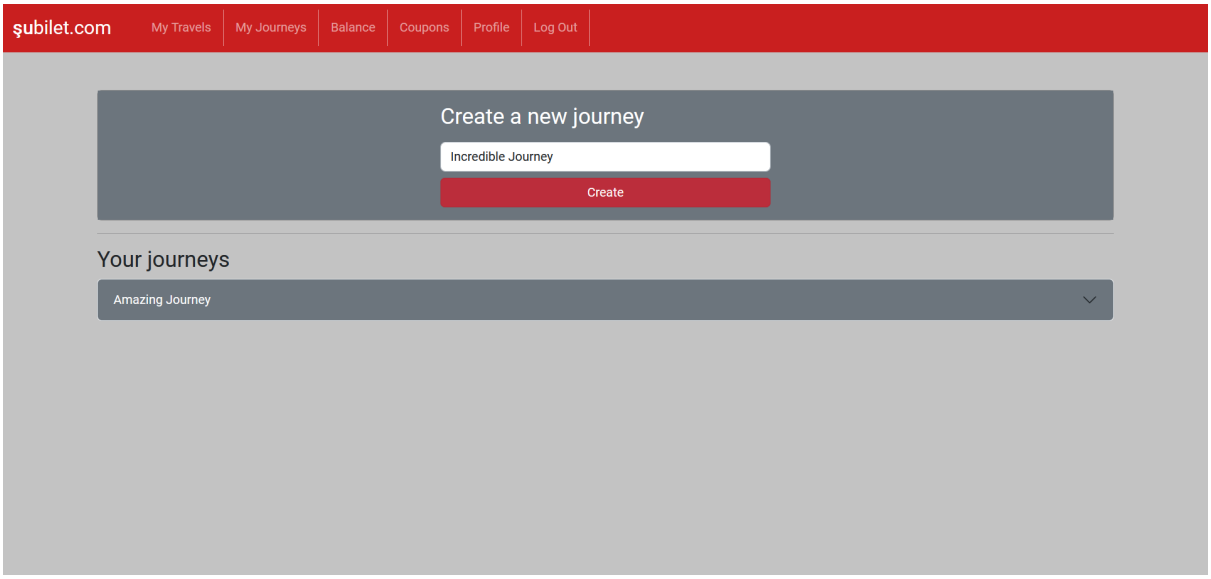


From the Coupons page, you can view the user's available and past coupons and add a new coupon. These coupons can be used while purchasing a travel. This traveler already has added both versions (public and private) of the indirim30 coupon, so trying to add it again will result in an error pop-up. You can try adding the "indirim20" coupon instead.

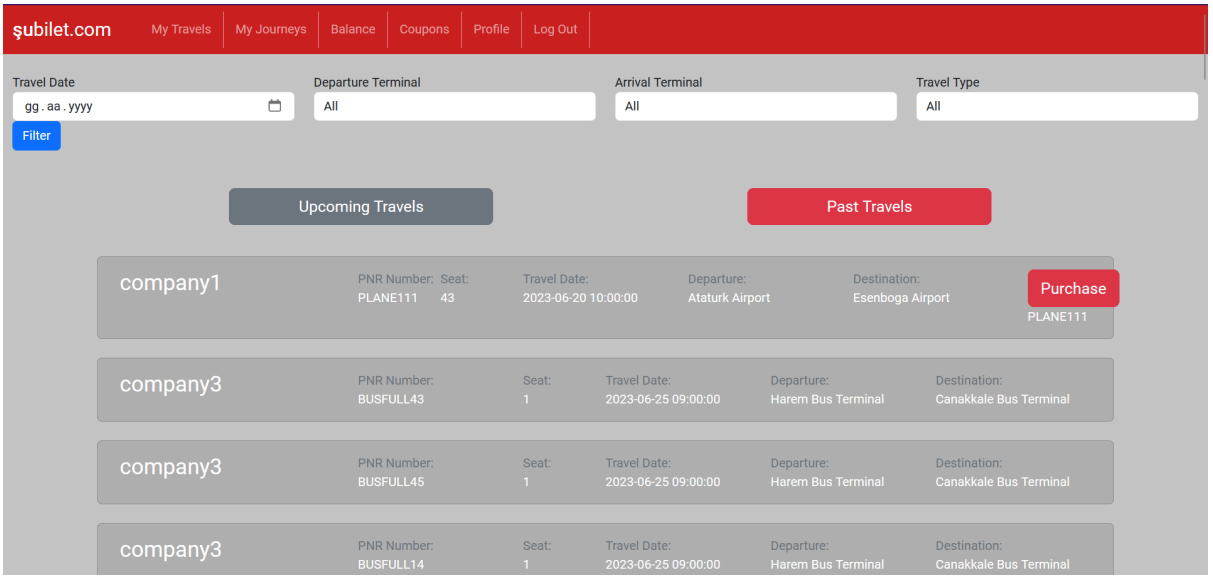
In ŞuBilet, public coupons can be added by all users; private coupons can only be added by one user in the system. However, one user cannot add the same coupon twice, regardless of it being public or private.



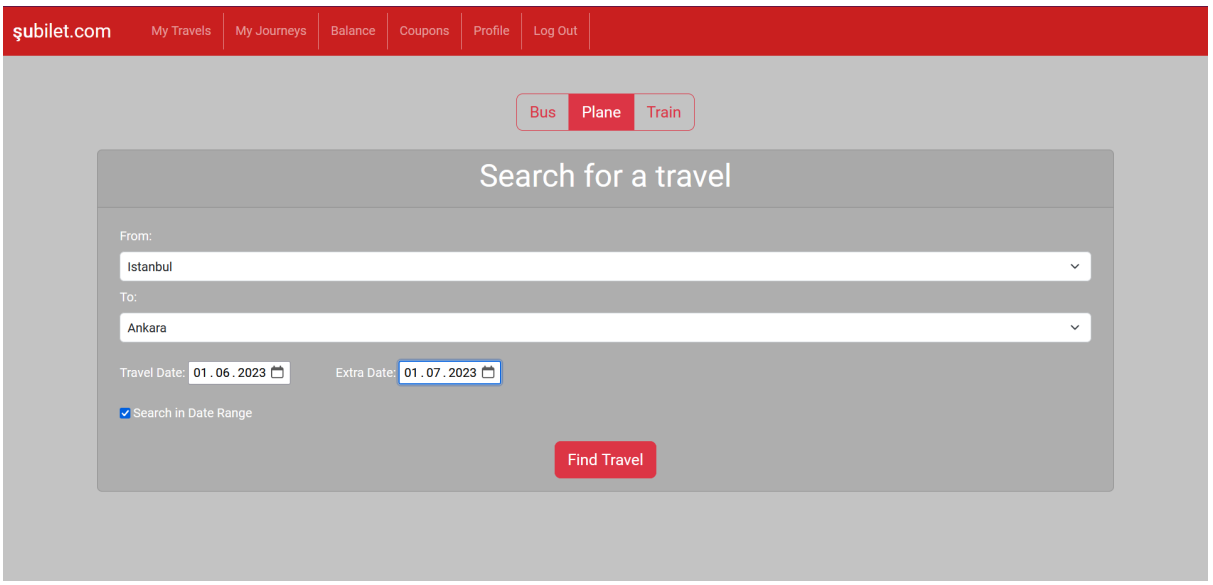
From the Balance page, you can add money to the user's balance by entering their credit card info and amount to be added. This project doesn't use a real-life credit card checking system, so you can enter any random information and still successfully add money for testing purposes.



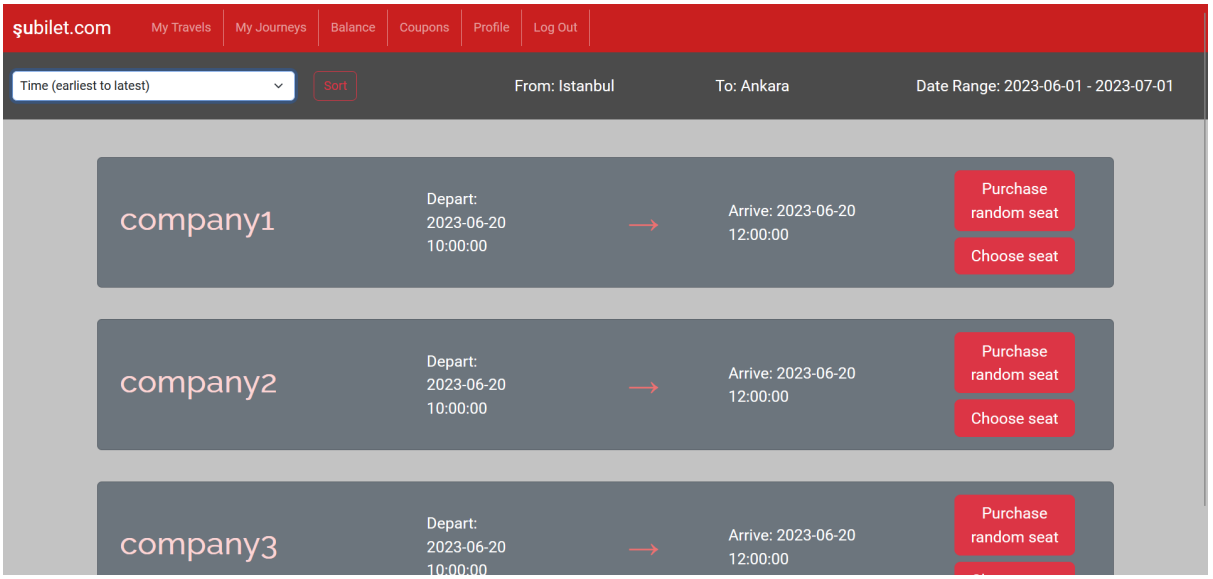
From the My Journeys page, you can view this user's journeys and create a new empty journey if you wish so. We will come back to this page later.



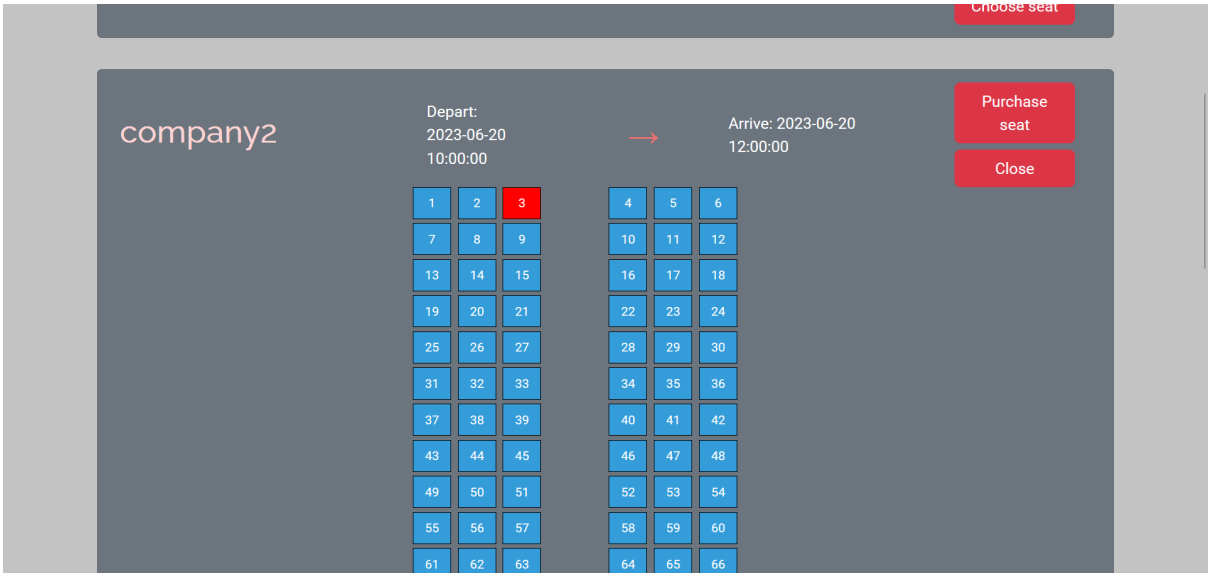
From the My Travels page, you can view the user's past and upcoming travels and filter them. In the upcoming travels, if the travel is reserved, you can purchase it. In the past travels, you can leave a comment and a rating for the travel. This user (the one that we logged in with traveler1@example.com) has already left a comment for all of their past travels, but another user from the database can be used to test the comment/rating function.



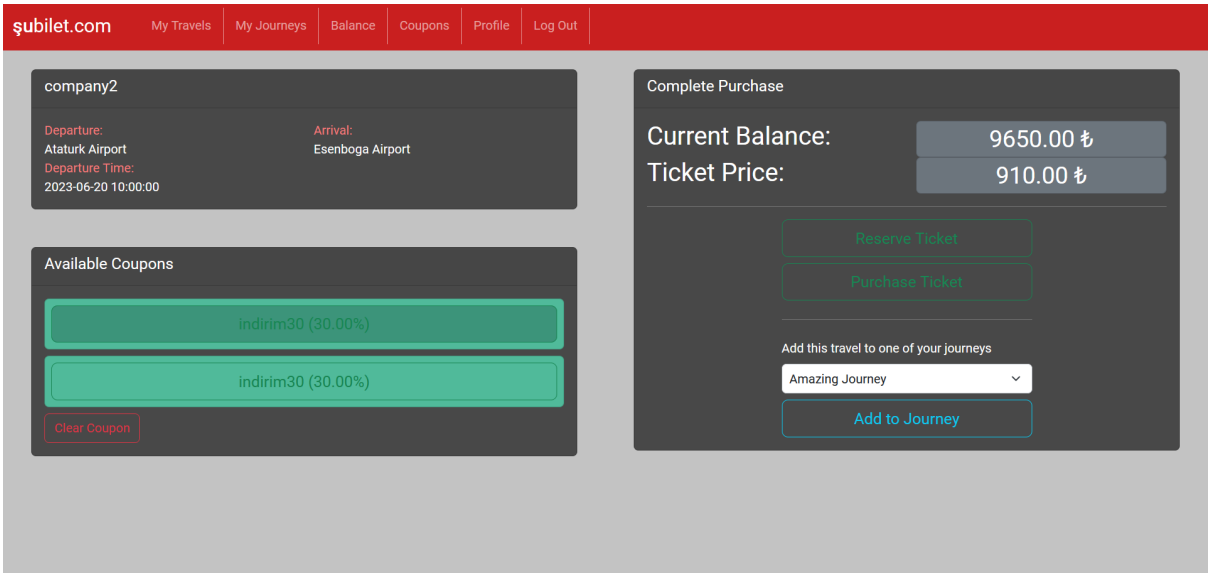
From the main page (accessed by clicking the şubilet.com logo in the navbar), you can search for travels. You can search for a specific date, or you can search in a date range. We will search all plane travels from İstanbul to Ankara in the date range 01.06.2023 - 01.07.2023.



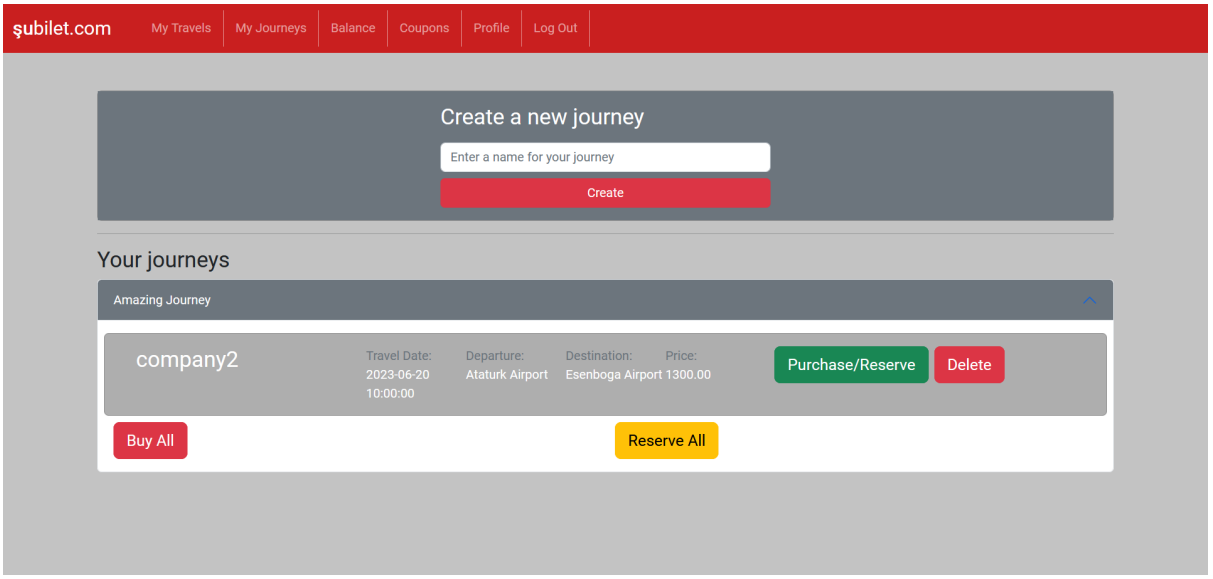
After the search, the travel options are listed. You can sort them by their time or by their price. You can purchase a travel with a random seat, or you can choose the seat yourself.



This is the display screen to choose a seat. Blue seats are available to choose and your choice (seat 3 in this instance) is displayed in red. We will proceed with seat 3.



This is the transaction page. You can choose a coupon for your purchase. You can reserve or purchase the ticket. Or instead, you can add this travel to one of your journeys (you can go back to the journeys page and create a journey if you don't have a journey). We already have one journey, called "Amazing Journey". We will add this ticket to the Amazing Journey.

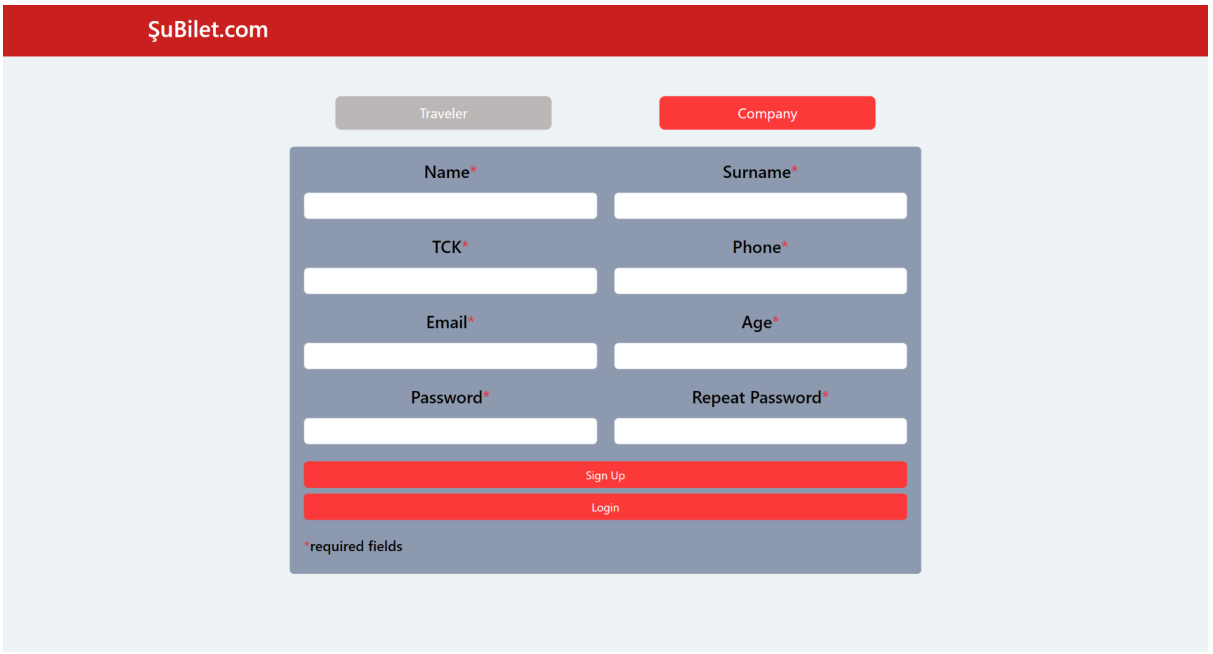


After adding this travel to Amazing Journey, we are navigated back to the Journeys page. As we can see, the travel is visible inside Amazing Journey. You can delete it, view its details, buy/reserve all the travels in the journey or buy/reserve specific travels in the journey. Note that deleting a travel from your journey doesn't refund or remove the travel itself; it just removes it from being a part of this journey.

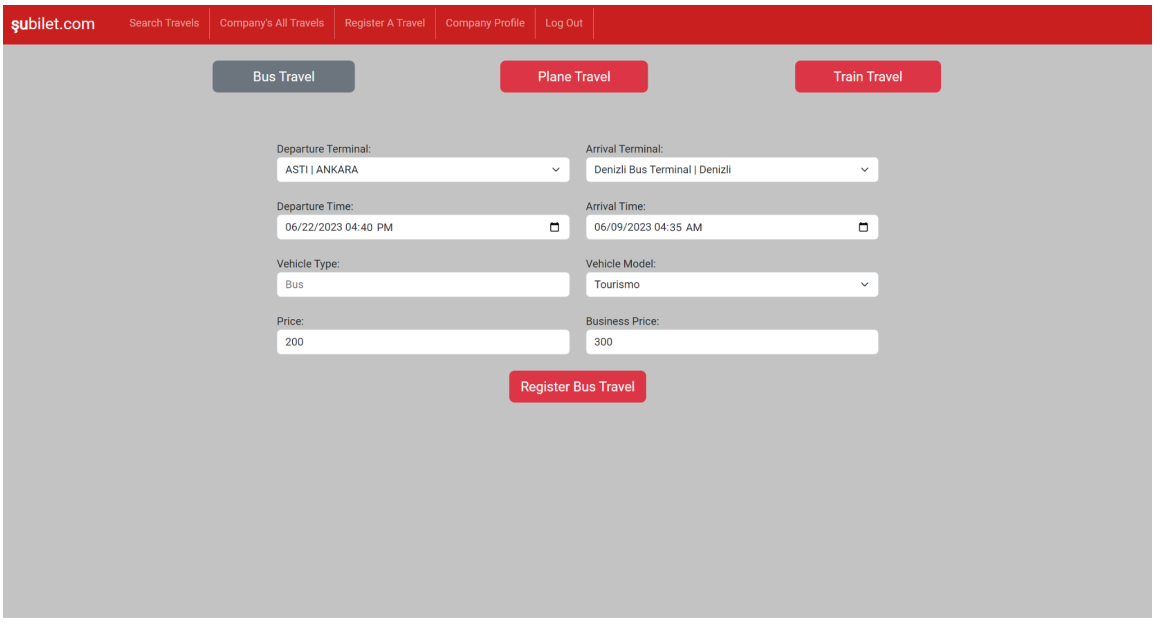
One corner case: If you add a travel which you already bought to one of your journeys, you can't buy/reserve it again from your journeys. Instead of the "Purchase/Reserve" button, there will be a "Details" button on display, which will lead to the details of the travel.

7.3. Company User

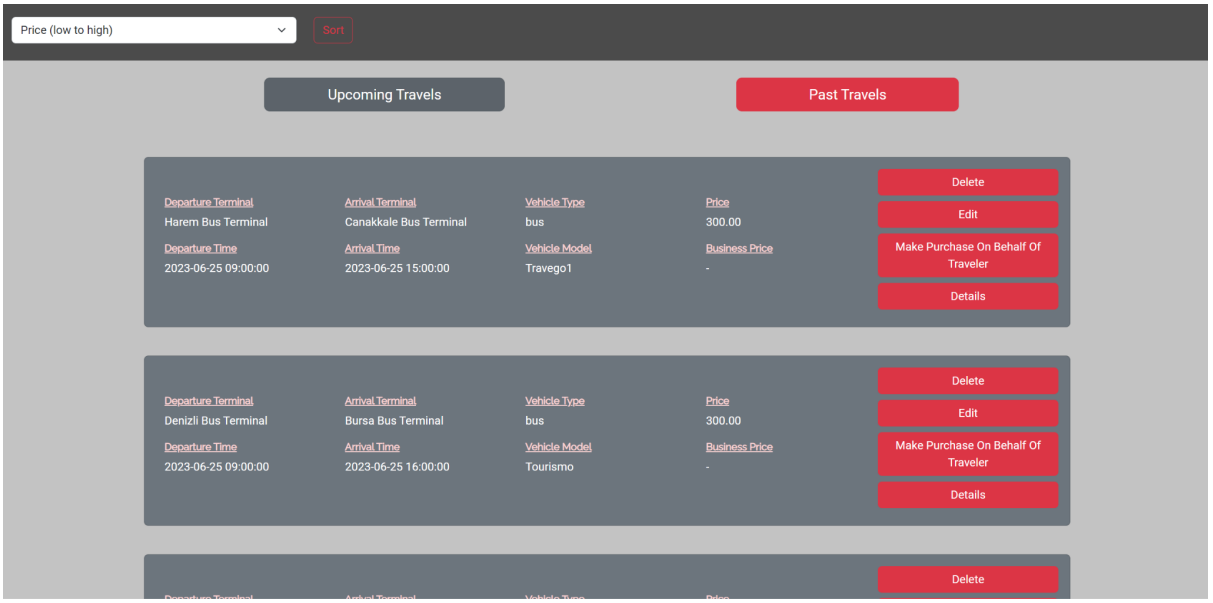
For companies, login process is the same as the traveler. From the login page, providing correct email and password for a registered company user is enough. "company1@example.com" with the password "123456" is an existing example company user in the system. For registering to the system, register button in the page can be used to register as a traveler user or a company user.



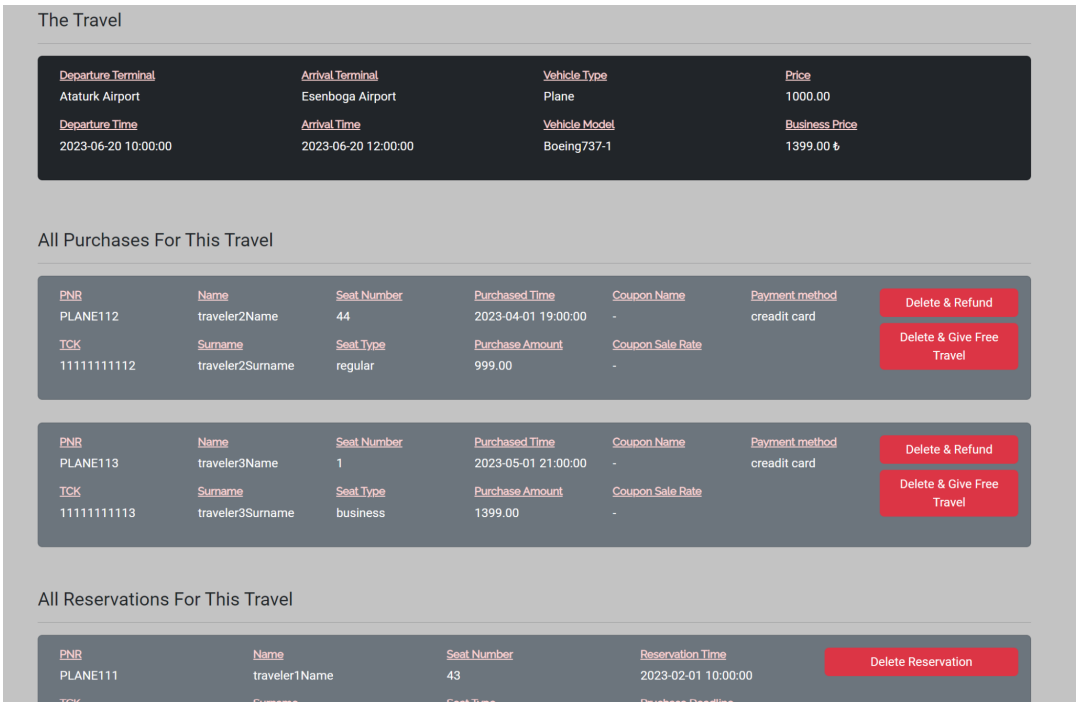
Companies can add travels by providing the required indormation. Any company can choose the transportation method and add the travels by specifying the terminals and vehicles that exist on the system.



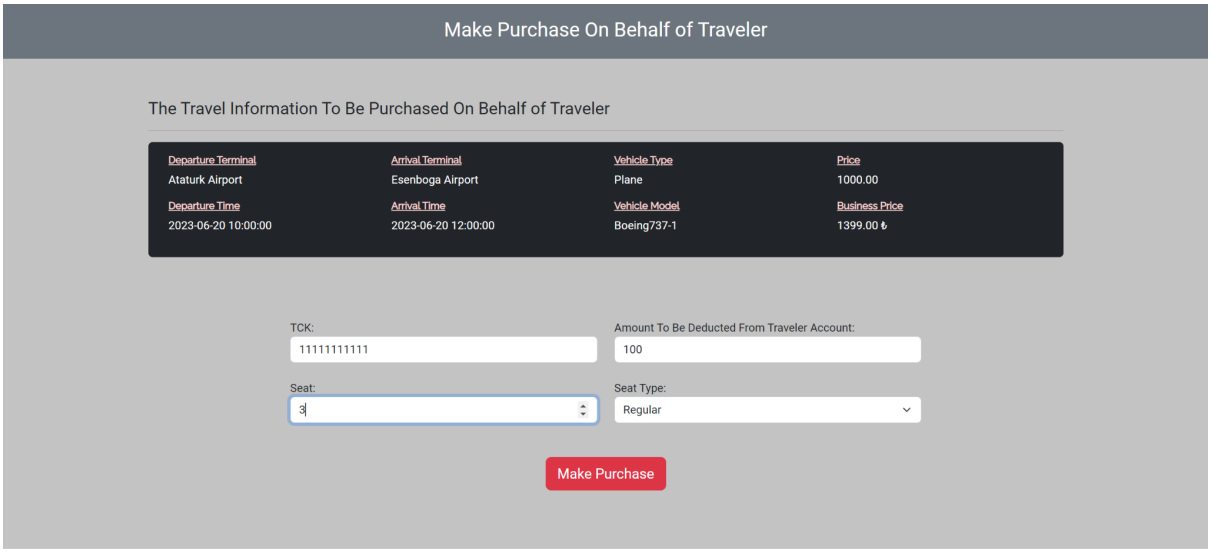
A company can see all of the travels they provide from the company's all travels window. Here a company can sort their travels by their costs, or departure times. Also a company might choose to see their past or upcoming travels. Using the buttons, a company can perform many actions on their travels.



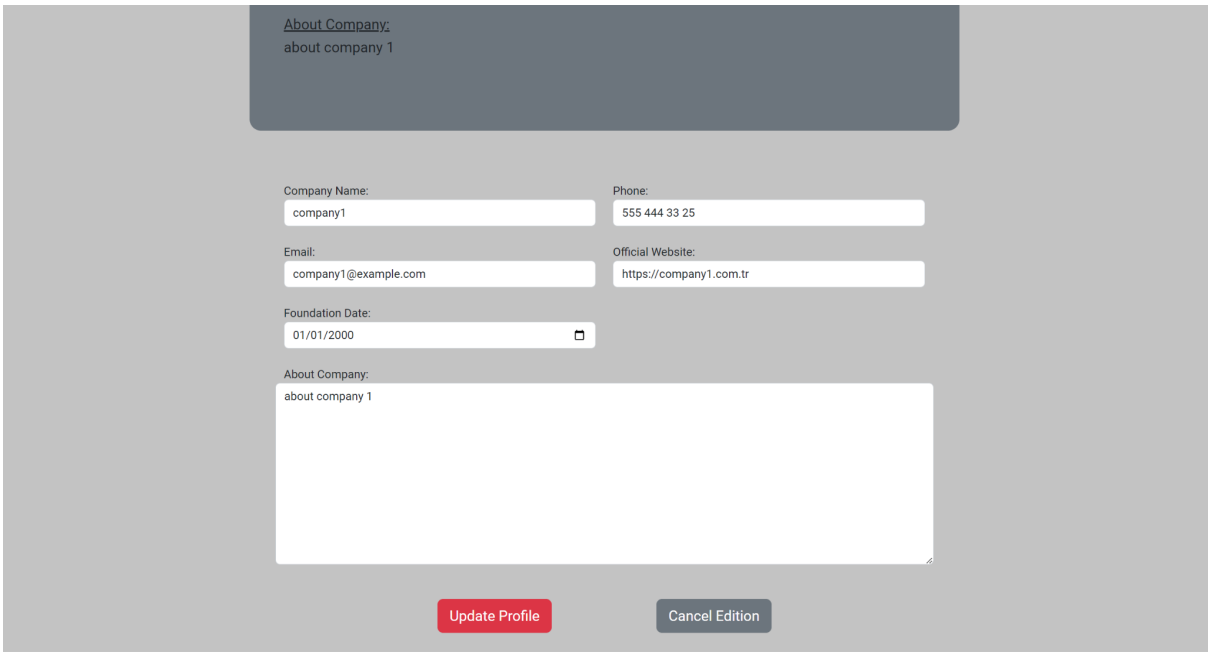
Deleting a travel first refunds all the purchase prices to the customers and deletes the travels from the system. Editing a travel is simple as attributes of the travel is simply changes and saved to the system. Details gives details on the travelers who purchased or reserved tickets from this travel. From there reservations can be deleted while purchases can either be refunded or be swapped for another ticket to a travel provided by the company.



Make purchase on behalf of a customer gives the companies to make a purchase for a customer, by reducing the balance of the traveler the given amount and assign that person a travel purchase. The use of this system allows the companies to change travels of travelers or give them travels in special cases. The companies can purchase a desired number amount of seats for the customer by specifying it.



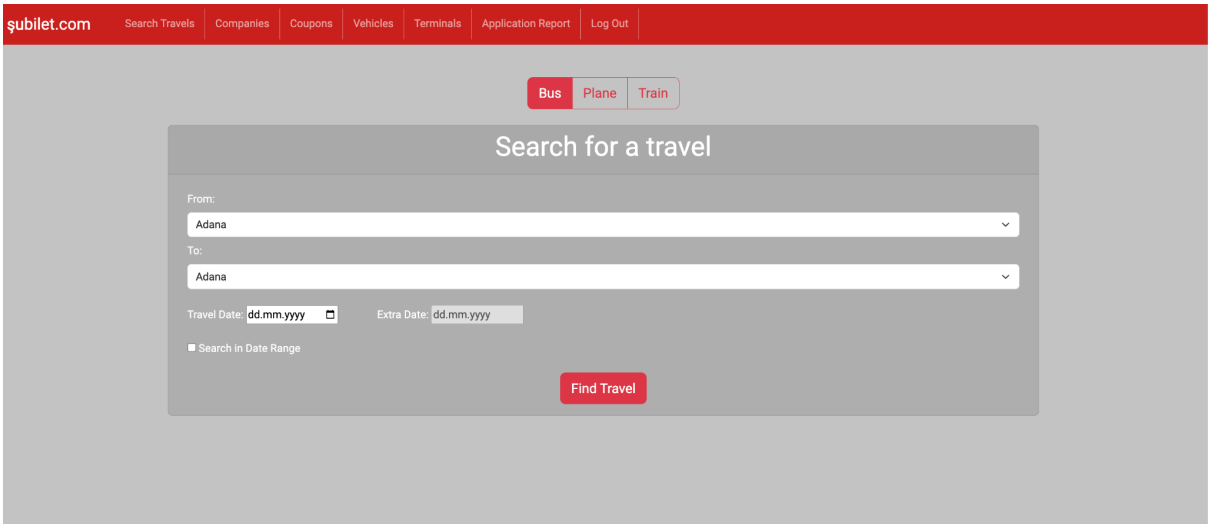
Finally companies can access and edit their profiles. This step is again simple as editing only requires compay user to press the edit button and changing the profile information before pressing update profile button.



7.4. Admin User

An admin can enter the system by writing his/her email address and password. Note that there is no signing up UI for admins.

After login, a search travel page is seen. As travelers and companies, any search is possible. To make a search vehicle type must be selected. Default value for vehicle type is bus. Destination city, arrival city and travel date are required for travel search.

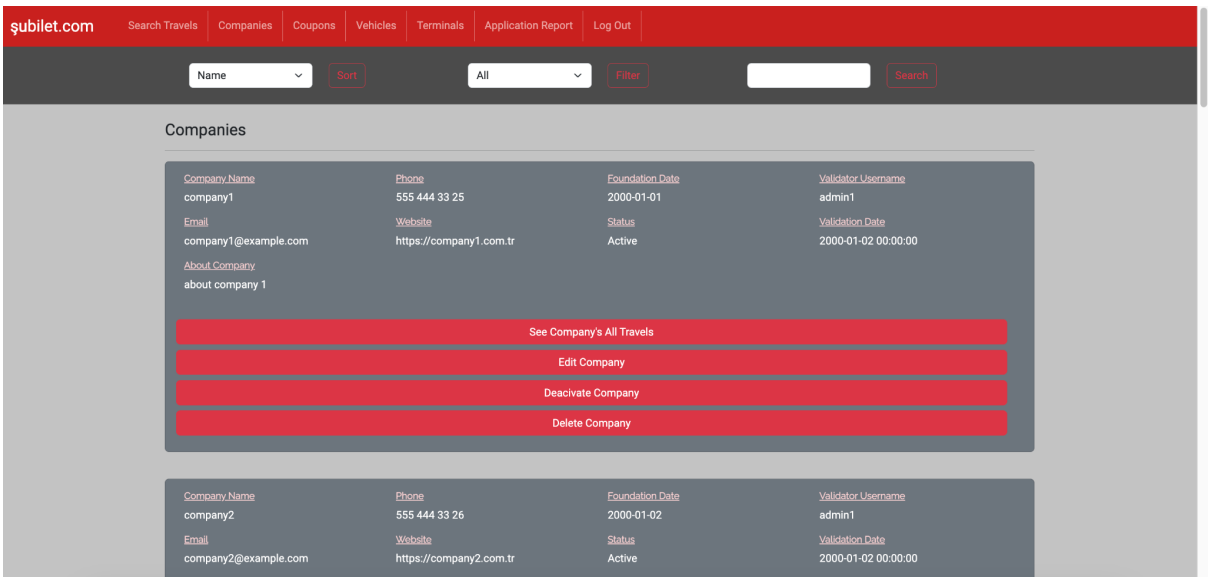


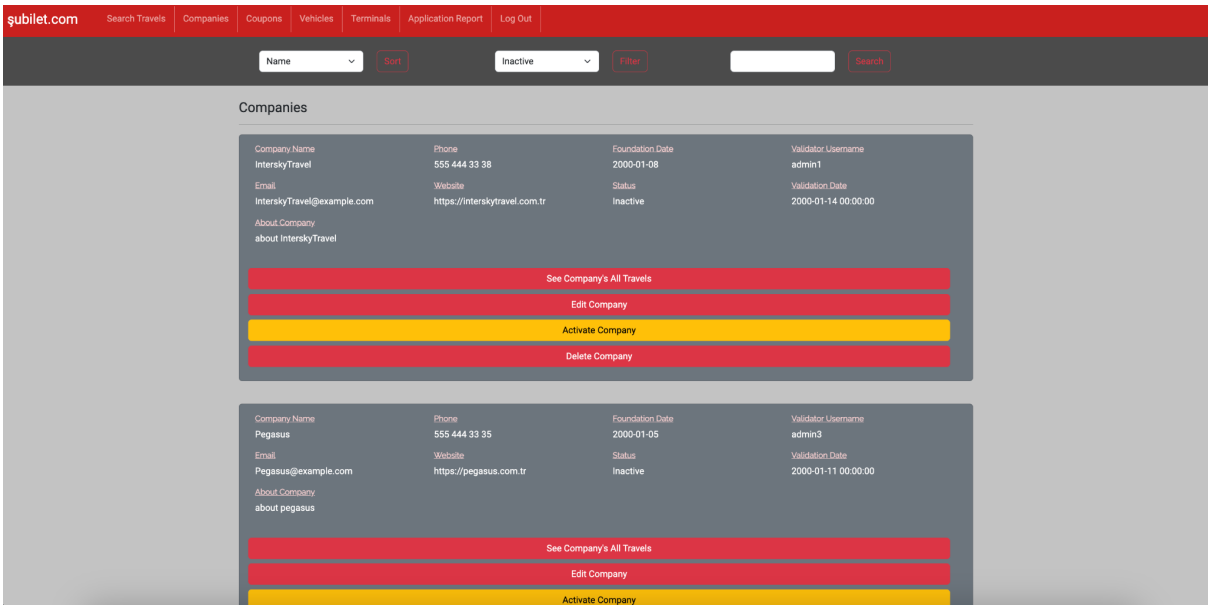
In the companies tab, admin can see all companies. Sorting, filtering and search in companies are available. With these, one can obtain desired company.
Sort options are as below:

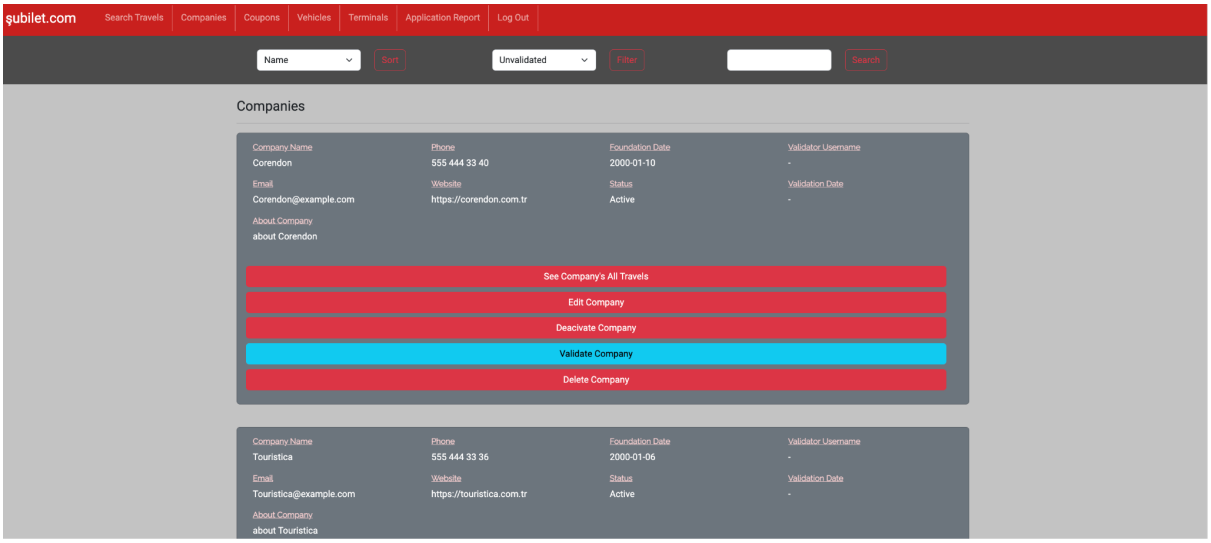
- Validation date (earliest to latest)
- Validation date (latest to earliest)
- Foundation date (earliest to latest)
- Foundation date (latest to earliest)

Filter options are as below:

- All
- Validated
- Active
- Inactive

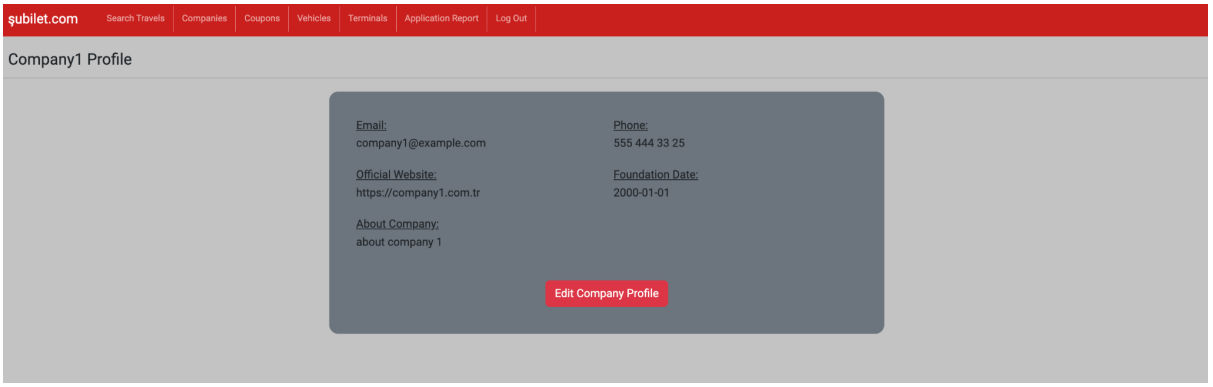


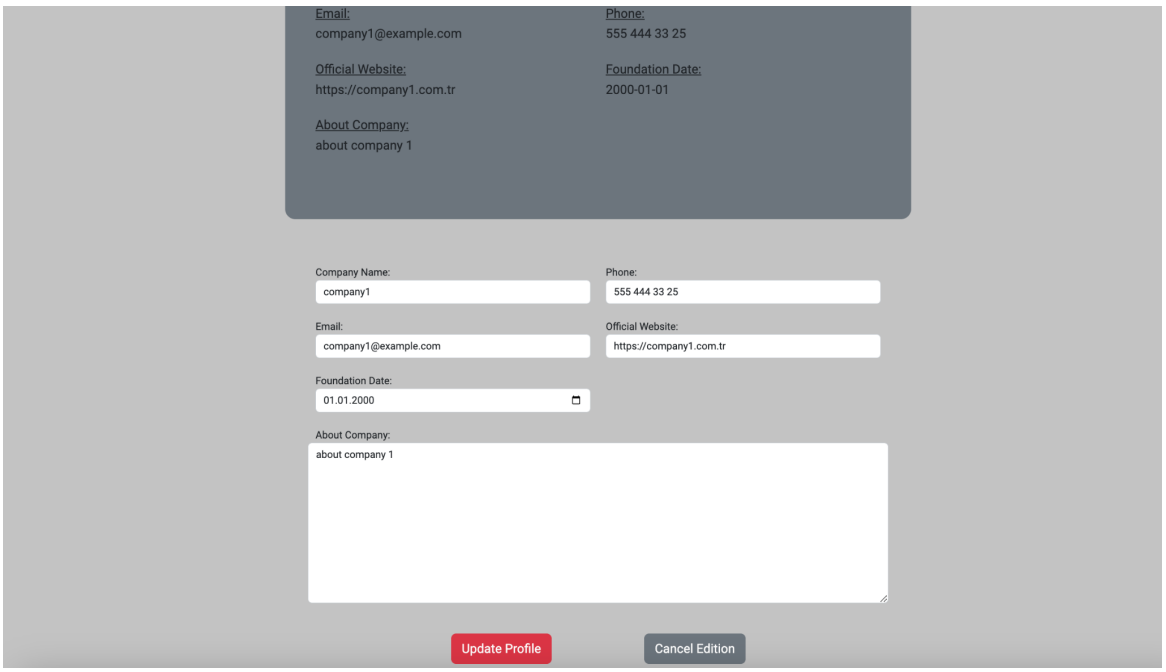




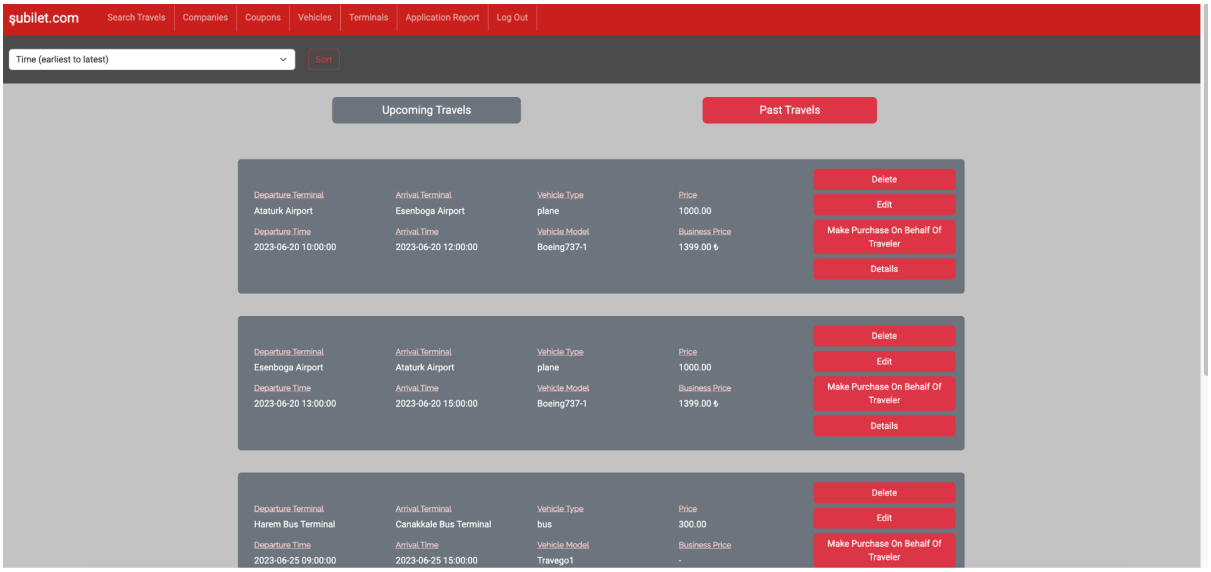
As it is seen in the figures above, an admin can easily validate a company, deactivate a company and activate a company with ease.

With the edit company button, properties of a company can be changed by an admin as seen figures below.

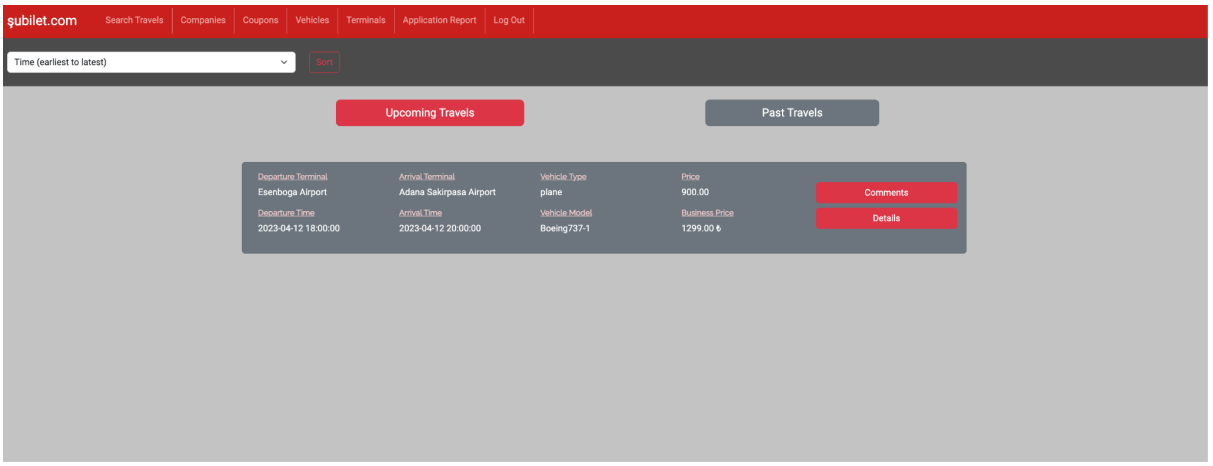




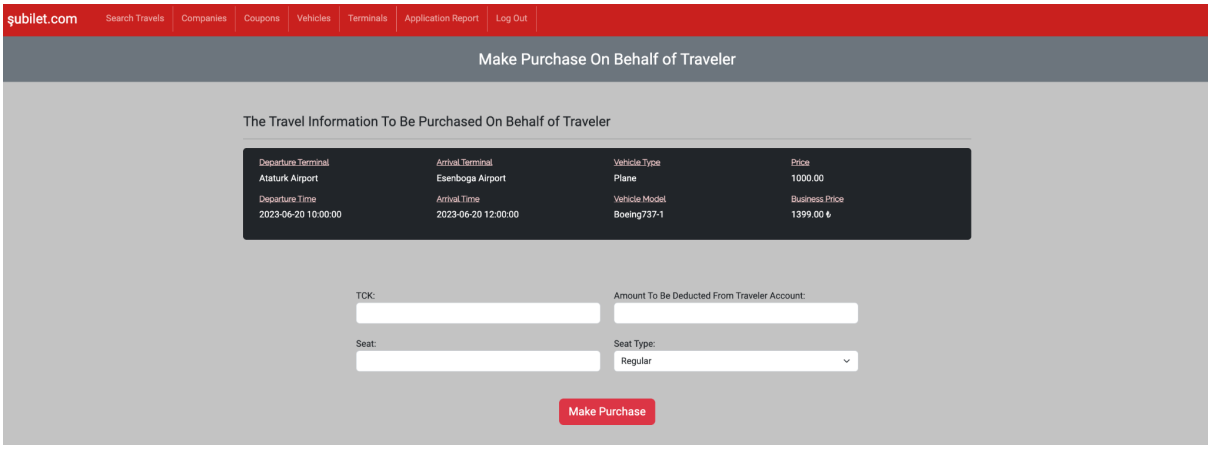
The "See Company's All Travels" allows an admin to see all upcoming and past travels of a company. As a default, upcoming travels are shown first but pressing the "Past Travels" button shows the past travels as its name indicates. The color of the buttons also gives a sign to the admin about whether upcoming or past travels are shown.



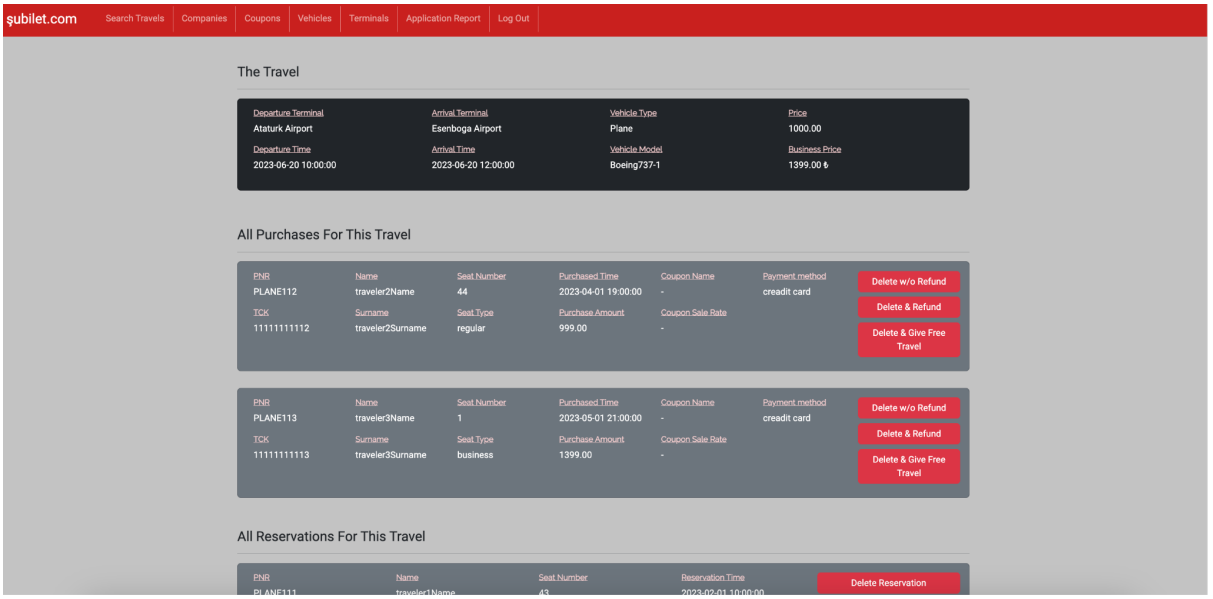
Note that the implementation of "Delete" and "Edit" buttons for travels are not completed. So, you won't be able to use these features of the application.



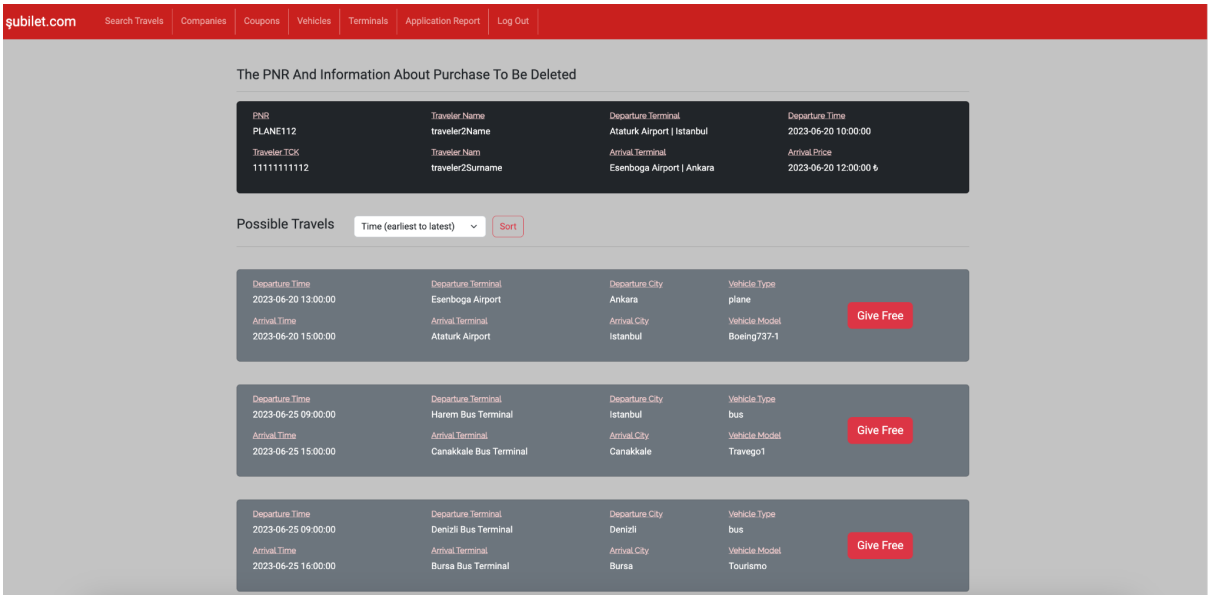
The button named as "Make Purchase On Behalf Of Traveler" allows admins to make purchases for a traveler by filling required fields. The amount of money that will be deducted from the traveler completely depend on the admin.



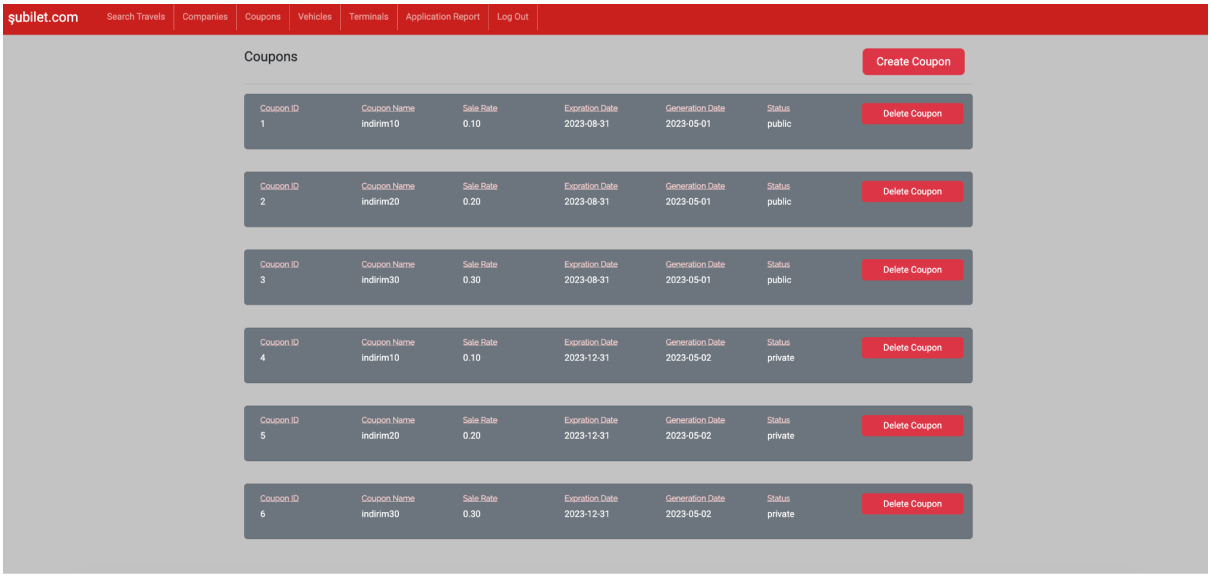
The "Details" button allows an admin to see all purchases and reservations on a trip. An admin can delete a reservation by pressing the "Delete Reservation" button. In addition to that as it is seen below, several choices are possible for purchases on a travel. An admin can delete a purchase without a refund or delete a purchase with refund or delete a purchase and give a free ticket for one of the company's travels..



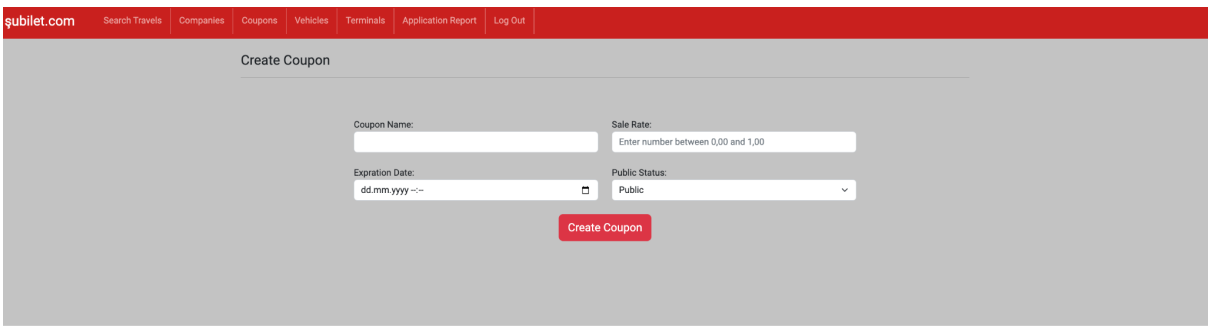
For the delete & give a free travel choice, purchase to be deleted is shown at the top of the page. When a travel of a company is selected and "Give Free" button is pressed, a new PNR is generated and a travel ticket is given to the user with random seat number.



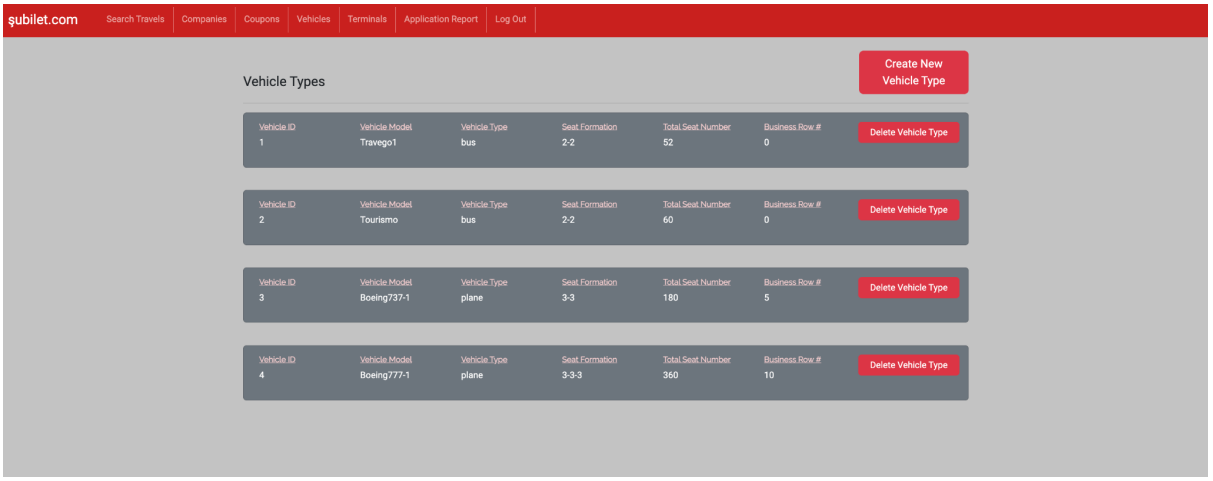
All coupons can be seen in the Coupons tab of the application. An admin can delete a coupon easily by pressing a "Delete Coupon" button next to it.

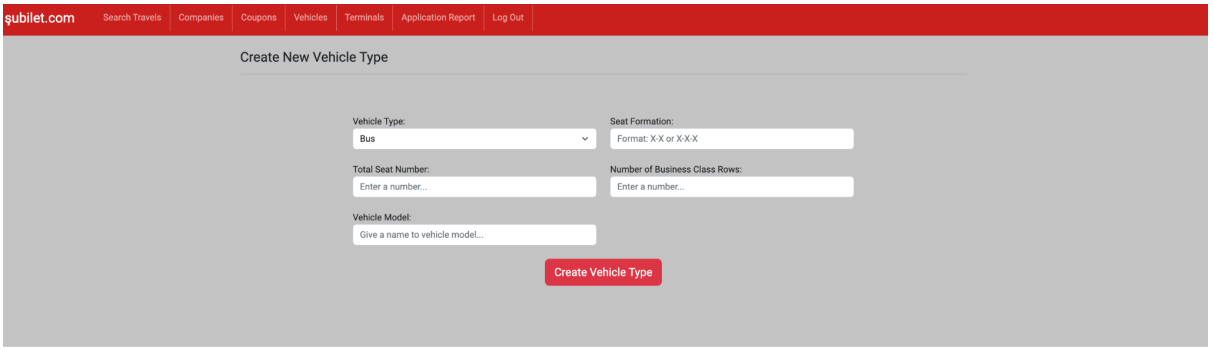


A new coupon can be created by only admins. After filling blanks, a new coupon is generated. If a coupon couldn't be generated, then the reason is written on the screen.

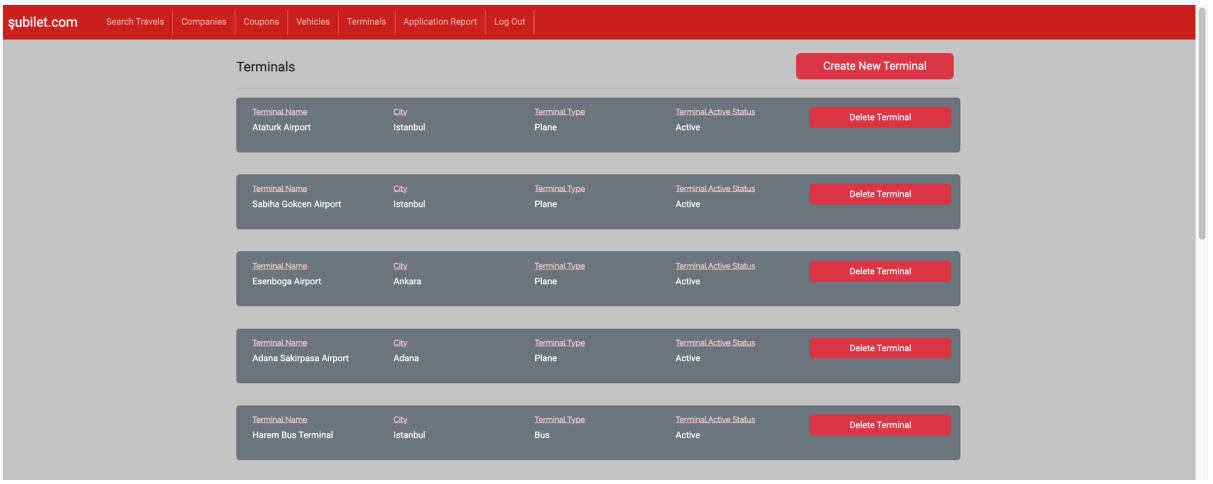


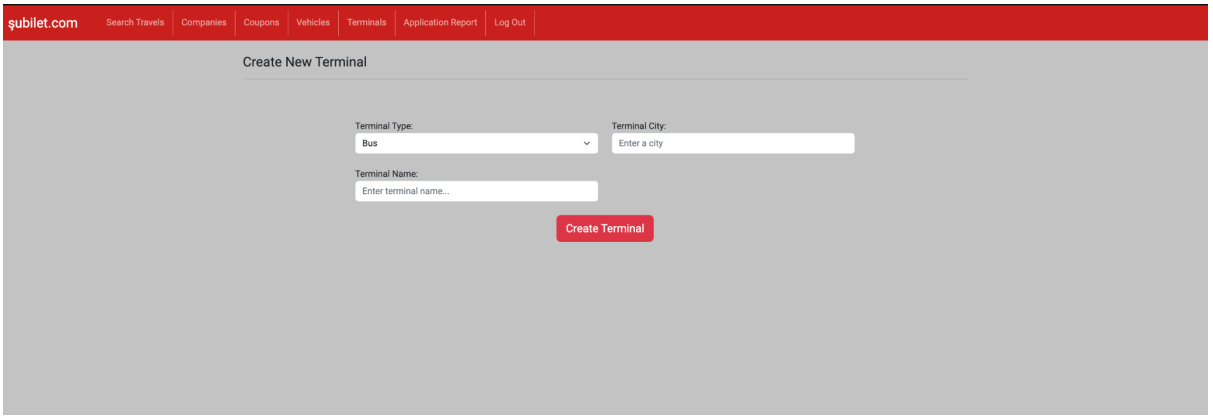
Similar to coupons, all vehicle types can be seen in the "Vehicles" tab. And a new vehicle type can be generated easily. Also, deletion of vehicle types is possible with one button.



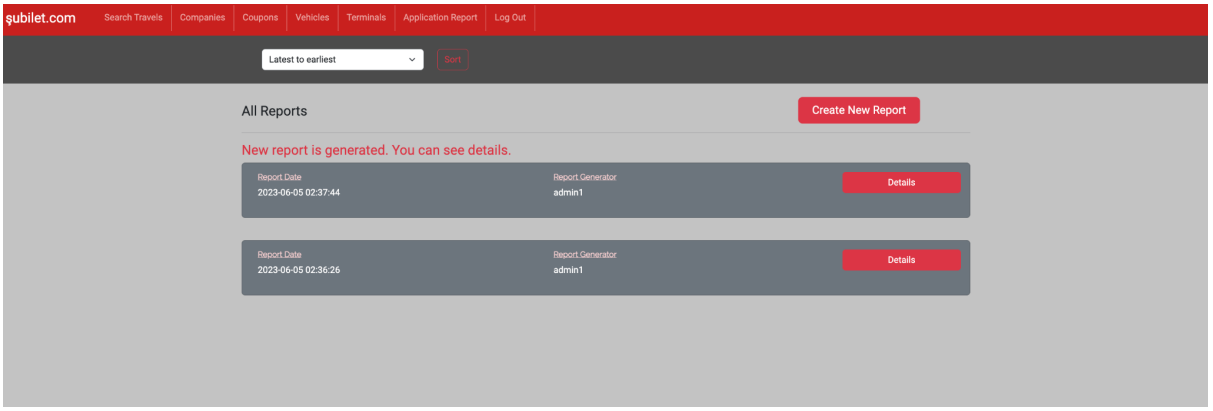


Similarly, "Terminals" tab allows admins to see all terminals, delete a terminal and create a new terminal.

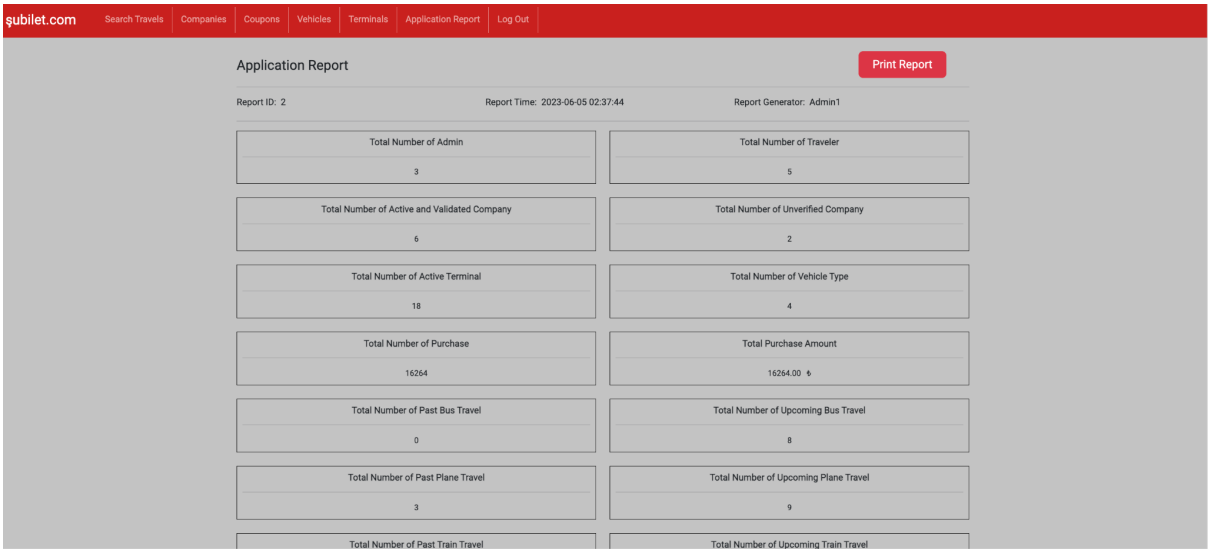




As it is seen in the below, all system reports can be seen in "Application Report" tab. A new report can be generated with "Create New Report" button.



Details of the reports are seen below.



An admin can print the report by clicking on "Print Report" buttons. After that a pdf file is downloaded to the admin's computer. The printed report format is shown in the figure below.

Application Report

Indicators	Value
Report ID	2
Report Date	2023-06-05 02:37:44
Total Number of Admin	3
Total Number of Traveler	5
Total Number of Active and Validated Company	6
Total Number of Unverified Company	2
Total Number of Active Terminal	18
Total Number of Vehicle Type	4
Total Number of Purchase	16264
Total Purchase Amount	16264.00
Total Number of Reviews	12
Coupon Usage Percentage	6
Total Number of Past Bus Travel	0
Total Number of Upcoming Bus Travel	8
Total Number of Past Plane Travel	3
Total Number of Upcoming Plane Travel	9
Total Number of Past Train Travel	0
Total Number of Upcoming Train Travel	0
Company With Max Revanue	Company1
Company With Max Travel Number	Company2
Company With Max Rating	Company1

This is the end of the user manual.