Nelson Batista

Abu Butt

Ichwan Palongengi

Adedamola Shomoye

Project 1 Progress Report

4/21/2017

1 GUI

The GUI for the program has been implemented in the form of a website where users may create new reservations or view old ones. This website is hosted on a machine running Python Flask's built-in webserver, which is not too fancy, but suits our needs quite well. Flask displays pages by using "templates" along with its builtin template engine, Jinja. The intent of a template is to allow an outside program to combine the template with data, perhaps from a database or a previous POST request, into a complete webpage tailored to the provided data.

For example, the template may have {{ `username` }} somewhere to display the username of the user viewing the page. Since this will, inevitably, depend on the specific user, the template is sent to the template engine along with the data, perhaps from a session cookie, necessary to replace {{ `username` }} with the literal username of the user through the Flask framework's `render_template` function. The result is an HTML document tailored specifically for the user, which is then rendered by the user's browser.

Thus, using templates and a Python connector to a SQL database, we are able to send HTML pages reflecting data in the database.

In order to make a reservation, the passengers must enter their desired source and destination stations, using dropdown menus on the main page, as well as the time of day (morning, afternoon, evening), and of course the date of the trip. The user must also indicate whether it is to be a round trip, and if so they must specify the date and time of their intended return trip. This is submitted through an HTML form which takes the user to the "search results" page.

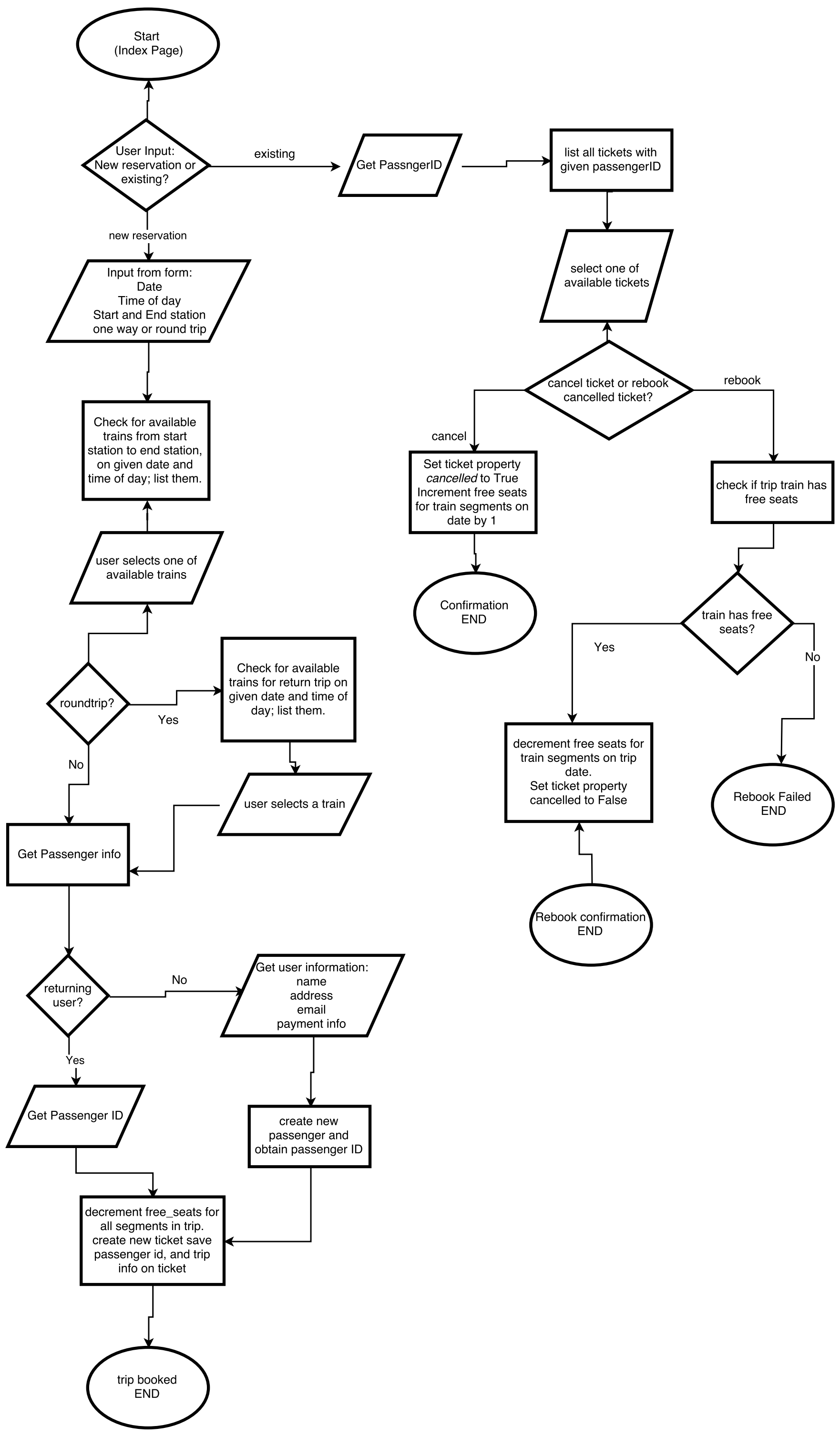
The "search results" page uses the data obtained from the POST request generated by submitting the form on the main page as arguments to a function used to query the databse for trains available under the parameters specified by the passenger. Specifically, the `get_trains_from_station` function is called, which performs a SQL `select` statement to query the database for all trains which stop at both the source and destination stations at the time of day of the trip date. This returns a dictionary of train ids, which is then filtered using the `check_free_seats` function to get only those trains which have at least one seat free for every segment between the two stations. The `check_free_seats` function takes a start station and an end station along with a train id and trip date as arguments, and returns `true` if the train has at least one seat free for that date on every segment between the start and end stations. This is done using another, somewhat more complex `select` SQL statement. The set of trains returned by `get_trains_from_station` is iterated over, and each train is appended to a `free_trains` array only if `check_free_seats` returns `true` with that train's id supplied as an argument. The search results template then uses the `free_trains` variable contents to display the trains available for the user to make reservations for.

On the search results page, the user may select any of the available trains to make a reservation for (or be told that there are no available trains). Upon reservation, we decrement the number of available free seats on the train for that date on every segment in between the start and end stations by 1, then write the new value back to the database.

This behavior and more is expressed in the flowchart on the following page.

CSC336

Proect 1 Team 3 4/20/2017



Start (Index Page)
User Input: New reservation or existing?
existing
Get PassngerID
list all tickets with given passengerID
select one of available tickets
cancel ticket or rebook cancelled ticket?
cancel
rebook
Set ticket property *cancelled* to True Increment free seats for train segments on date by 1
Confirmation END
check if trip train has free seats
train has free seats?
Yes
No
decrement free seats for train segments on trip date. Set ticket property cancelled to False
Rebook Failed END
Rebook confirmation END
new reservation
Input from form: Date Time of day Start and End station one way or round trip
Check for available trains from start station to end station, on given date and time of day; list them.
user selects one of available trains
roundtrip?
Yes
No
Check for available trains for return trip on given date and time of day; list them.
user selects a train
Get Passenger info
returning user?
No
Yes
Get user information: name address email payment info
Get Passenger ID
create new passenger and obtain passenger ID
decrement free_seats for all segments in trip. create new ticket save passenger id, and trip info on ticket
trip booked END

2 Database Layout

The layout of the database itself is shown in the ER diagram provided alongside this report. First, we have the `Passengers` table, which contains a `passenger_id` attribute, which serves as the primary key. Passengers purchase tickets, which represent reservations at certain times and dates, and which are themselves identified by the primary key `ticket_id`. The `Tickets` table is related to the `Passengers` table by a foreign key which references `Passengers (passenger_id)`.

All the relations, including foreign keys, are in the following Entity-Relation diagram:

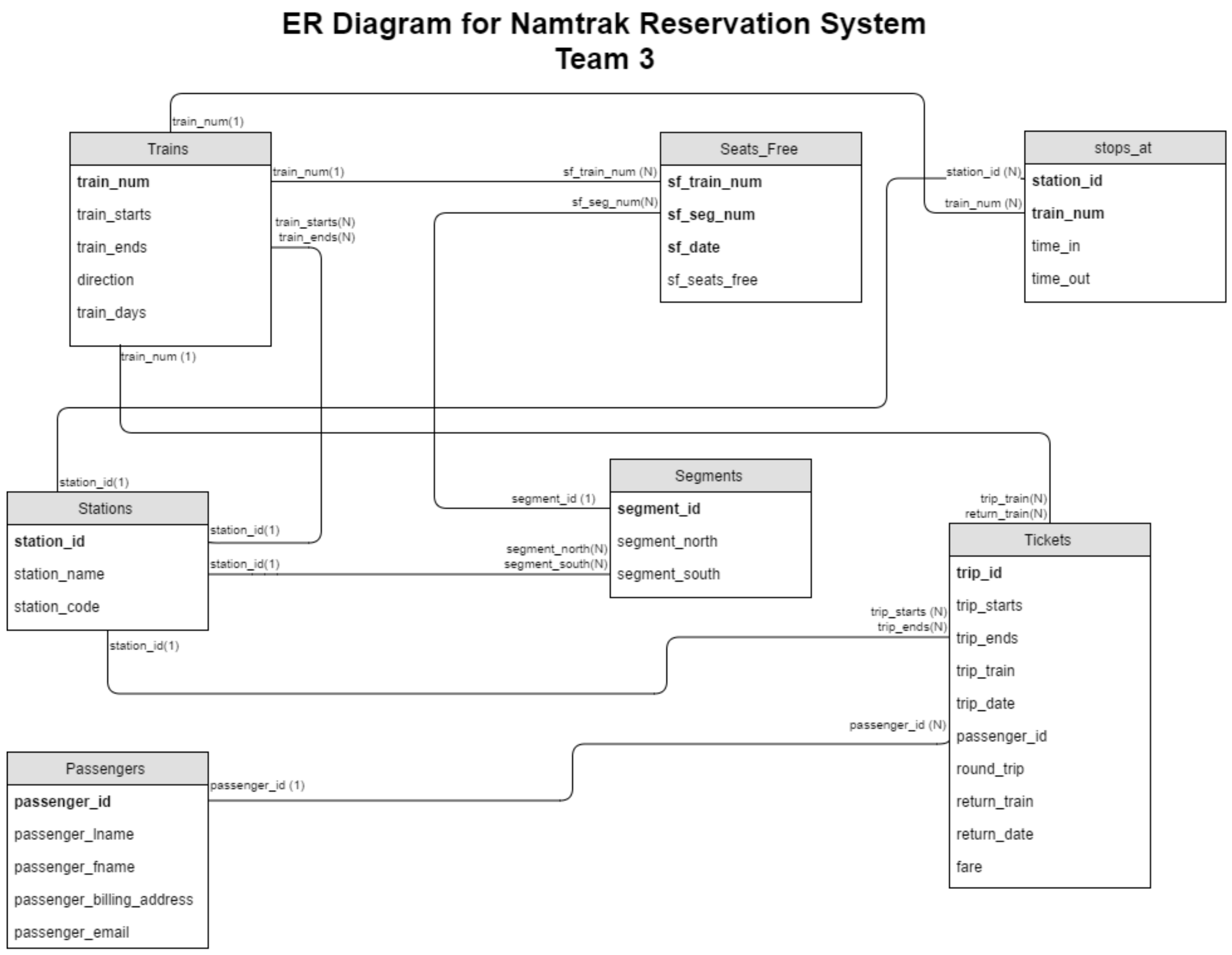


Figure 1: ER Diagram for our database.