Εθνικό Μετσόβιο Πολυτεχνείο

Βάσεις Δεδομένων

Αναφορά Εξαμηνιαίο Project 2013-2014

Ηλίας Φωτόπουλος
03109106
Θανάσης Βράτιμος
03110769

1 Λεπτομέρειες Υλοποίησης

Για την κατασκευή του project έγινε χρήση των παρακάτω τεχνολογιών:

- Ruby on Rails (Framework)
- HTML Sass (Syntactically Awesome Style Sheets)
- Bootstrap (Sass Framework)
- CoffeeScript
- Git (Version Control)
- PostgreSQL
- Heroku

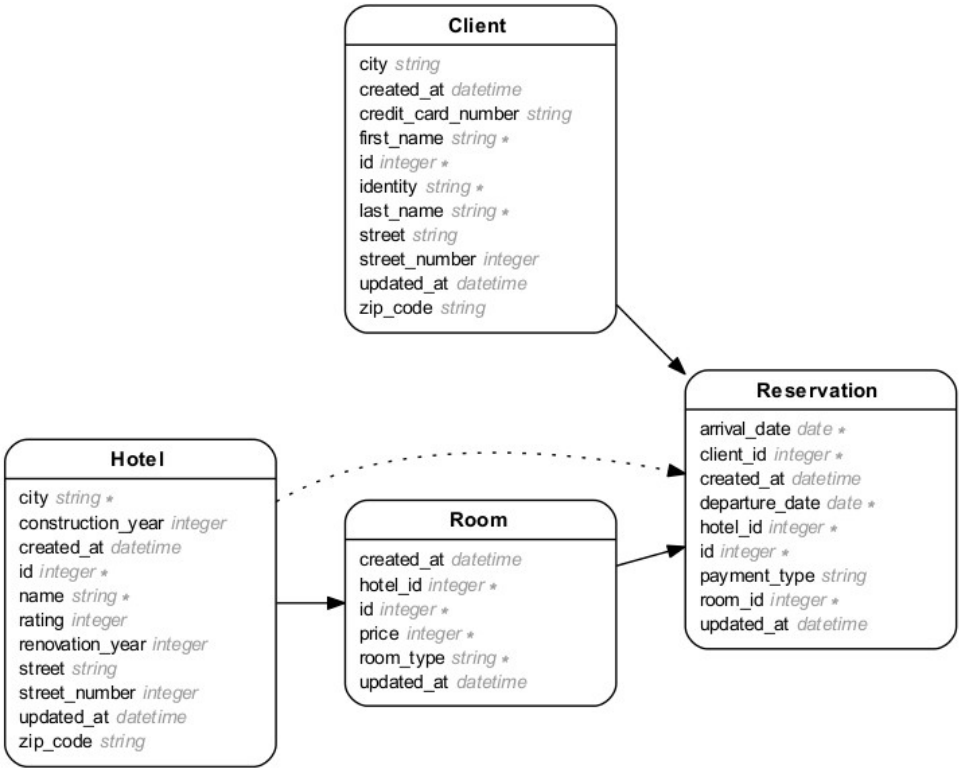
Κατά την φάση του σχεδιασμού αποφασίστηκε η χρήση του RoR framework λόγω των παρακάτω χαρακτηριστικών της:

- Αρχιτεκτονική MVC
- Convention over Configuration (CoC)
- REST (Representational state transfer)

Έτσι το project αναλύθηκε σε models, οπου εμπεριέχεται όλη η λογική επικοινωνίας με τη βάση, views, όπου γίνεται η παραγωγή της διεπαφής με το χρήστη, και controllers, όπου αποφασίζεται ποιο view και model θα κληθεί, και υλοποιεί τη σωστή επικοινωνία μεταξύ τους.

Με τον τρόπο αυτό, καθίσταται ευκολότερη και ταχύτερη η ανάπτυξη και η αποσφαλμάτωση του λογισμικού, ενώ ο κώδικας είναι πιο ευέλικτος, αφού αποκρύπτονται οι λεπτομέρειες υλοποίησης μεταξύ των συνιστωσών (αγνωστικό μοντέλο - αποσύνδεση)

2 Relational model



3 Constraints

Χρησιμοποιήσαμε περιορισμούς σε επίπεδο model διότι:

- Είναι Database Agnostic
- Γράφονται, διαβάζονται, συντηρούνται εύκολα

Υπάρχουν πολλοί διαφορετικοί τρόποι που με τους οποίους μπορούσαμε να εφαρμόσουμε constraints και validations στην εφαρμογή μας μερικοί από τους οποίους είναι οι παρακάτω:

- Database Constaints (Υποστηρίζονται από την PostgreSQL)
- Client-side validations
- Controller-level validations

Σε κάθε φόρμα για update ή edit μιας σχέσης γίνονται κάποια validations. Έτσι ώστε να εξασφαλιστεί data integrity στην βάση δεδομένων μας.

3.1 Hotel Validations

```
validates :name, :city, presence: true
validates :street_number, numericality: true
validates :rating, numericality: true,
inclusion: {in: 0..5, message: "rating should be 0-5"}

validates :construction_year, :renovation_year,
inclusion: { in: 1800..Date.today.year, message: "Year should be over 1800"},
  allow_nil: true,
  format:
  {
    with: /(18|19|20)\d{2}/i,
    message: "should be a four-digit year"
  }
```

3.2 Room Validations

```
validates :price, :room_type, :hotel_id, presence: true
validates :price, numericality: {greater_than: 0}
```

3.3 Client Validations

```
validates :identity, :first_name, :last_name, presence: true
validates :identity, uniqueness: {message: "This identity already exists. Identity must be unique!"}
validates :street_number, numericality: true
```

3.4 Reservation Validations

Ειδικά για τις Reservation φτιάξαμε μια Service με όνομα ReservationService, η οποία ουσιαστικά ελέγχει να δει αν ένα δωμάτιο είναι ελεύθερο κάποιες ημερομηνίες που της δίνουμε. Η βασική της λογική φαίνεται στην παρακάτω συνάρτηση:

```ruby
def room_available?
 reservations = UpcomingReservationUpdatable.where("room_id = ?", @room.id)
 arrival_date = Date.parse @arrival
 departure_date = Date.parse @departure

 reservations.each do |r|
   before = (arrival_date < r.arrival_date) && (departure_date < r.departure_date)
   after = (arrival_date > r.arrival_date) && (departure_date > r.departure_date)
   if before || after
     next
   else
     return false
   end
 end
 return true
end
```

Και τα γενικότερα validation του model Reservation:

```ruby
belongs_to :client
validates :client_id, presence: true

belongs_to :room
validates :room_id, presence: true

has_one :hotel, through: :room
validates :hotel_id, presence: true

validates :arrival_date, :departure_date, presence: true

validate :arrival_before_departure, :arrival_date_cannot_be_in_the_past, :departure_date_cannot_be_in_the_past

def arrival_before_departure
errors.add(:arrival_date, "must be before departure date") if arrival_date >= departure_date
  end

  def arrival_date_cannot_be_in_the_past
```

```
19  errors.add(:arrival_date, "can't be in the past") if ↩
      arrival_date < Date.today
20    end
21
22    def departure_date_cannot_be_in_the_past
23  errors.add(:departure_date, "can't be in the past") if ↩
      departure_date < Date.today
24    end
```

4 SQL Queries

Σε όλες τις σχέσεις δημιουργήθηκαν queries για προβολή, εισαγωγή, ενημέρωση και διαγραφή εγγραφών, τα οποία δεν περιλαμβάνονται την παρούσα αναφορά αφού δεν χρειάζονται κάποια εξήγηση.

4.1 Join queries

Γίνεται συχνή χρήση ερωτημάτων Join στην εφαρμογή μας όπως για παράδειγμα στο Reservations index:

```
1  SELECT reservations.id, reservations.arrival_date, ↩
     reservations.departure_date, reservations.created_at, ↩
     reservations.updated_at, clients.first_name, clients.↩
     last_name, hotels.name, rooms.room_type
2  FROM reservations
3  INNER JOIN clients
4  ON reservations.client_id = clients.id
5  INNER JOIN hotels
6  ON reservations.hotel_id = hotels.id
7  INNER JOIN rooms
8  ON reservations.room_id = rooms.id
9  ORDER BY arrival_date
```

Άλλο ένα παράδειγμα χρήσης Join είναι στο show client, όπου εμφανίζονται και οι reservations του κάθε πελάτη:

```
1  SELECT reservations.id, reservations.arrival_date, ↩
     reservations.departure_date, reservations.created_at, ↩
     reservations.updated_at, hotels.name, rooms.room_type
2  FROM reservations
3  INNER JOIN hotels
```

```
4 ON reservations.hotel_id = hotels.id
5 INNER JOIN rooms
6 ON reservations.room_id = rooms.id
7 ORDER BY arrival_date
```

Ενώ μπορείτε να δείτε και άλλο παράδειγμα χρήσης join στα Views που δημιουργήσαμε για την εφαρμογή.

4.2 Aggregate queries

Στην εφαρμογή μας κάναμε κυρίως χρήση του count για να μετράμε τον αριθμό των reservations ενός πελάτη ή τον αριθμό των δωματίων ενός ξενοδοχείου.

Εύρεση αριθμού δωματίων ενός ξενοδοχείου:

```
1 SELECT COUNT(*)
2 FROM rooms
3 WHERE rooms.hotel_id = hotel_id
```

Εύρεση αριθμού κρατήσεων ενός πελάτη:

```
1 SELECT COUNT(*)
2 FROM reservations
3 WHERE reservations.client_id = client_id
```

4.3 Group By

Στην ενότητα Reports της εφαρμογής γίνεται χρήση της εντολής Group By σε συνδυασμό με την aggregate εντολή count.

Το παρακάτω ερώτημα βρίσκει τον αριθμό των κρατήσεων κάθε ξενοδοχειακής μονάδας.

```
1 SELECT COUNT(reservations.id) AS count, hotels.name AS ↩
     name
2   FROM
3     reservations
4   INNER JOIN
5     hotels
6   ON
7     reservations.hotel_id = hotels.id
8   GROUP BY
```

```
9       hotels.id
```

Ενώ το παρακάτω βρίσκει τον αριθμό των κρατήσεων ανά τύπο δωματίου.

```
1  SELECT COUNT(reservations.id) AS count, rooms.room_type ↩
       AS room_type
2    FROM
3      reservations
4    INNER JOIN
5      rooms
6    ON
7      reservations.room_id = rooms.id
8    GROUP BY
9      rooms.room_type
```

4.4 Order By

Έχει γίνει χρήση Order By στα περισσότερα ερωτήματα της εφαρμογής μας. Μερικά παραδείγματα υπάρχουν στα views που εμφανίζονται στο παρακάτω section της παρούσας αναφοράς.

4.5 Group By - Having

Το παρακάτω ερώτημα επιστρέφει τα ξενοδοχεία που έχουν αριθμό δωματίων μεγαλύτερου του 5.

```
1  SELECT COUNT(rooms.id) AS count, hotels.name AS name
2    FROM
3      rooms
4    INNER JOIN
5      hotels
6    ON
7      rooms.hotel_id = hotels.id
8    GROUP BY
9      hotels.id
10   HAVING
11     COUNT(rooms.id) > 5
```

5 Views

Δημιουργήσαμε τα παρακάτω δύο views. Το view upcoming reservations περιέχει όλες τις κρατήσεις για τις οποίες η μέρα άφιξης είναι μεταγενέστερη της σημερινής μέρας. Επίσης εκτελεί inner join, με τα tables: clients,hotels,rooms έτσι ώστε να εμφανίσει όνομα πελάτη, ξενοδοχείου αλλά και τον τύπο του δωματίου. Η χρησιμότητα του view είναι η παρουσίαση όλων των κρατήσεων για τις οποίες ο πελάτης δεν έχει φτάσει ακόμη.

```
1  CREATE OR REPLACE VIEW upcoming_reservations AS
2  SELECT reservations.id, reservations.arrival_date, ↩
       reservations.departure_date, reservations.created_at, ↩
       reservations.updated_at, clients.first_name, clients.↩
       last_name, hotels.name, rooms.room_type
3  FROM reservations
4  INNER JOIN clients
5  ON reservations.client_id = clients.id
6  INNER JOIN hotels
7  ON reservations.hotel_id = hotels.id
8  INNER JOIN rooms
9  ON reservations.room_id = rooms.id
10 WHERE arrival_date > CURDATE()
11 ORDER BY arrival_date
```

Το view upcoming reservations updatable επιτελεί τις ίδιες λειτουργίες με το παραπάνω με δύο βασικές διαφορές:

- Δεν χρησιμοποιεί inner join για να πάρει τα names από τα άλλα tables.
- Περιέχει κρατήσεις όπου η μέρα αναχώρησης (όχι άφιξης) είναι μεταγενέστερη της σημερινής

Το εν λόγω view παίζει σημαντικό ρόλο στην εσωτερική αρχιτεκτονική και λογική της εφαρμογής μας. Χρησιμοποιείται από το ReservationService το οποίο ελέγχει αν ένα δωμάτιο είναι διαθέσιμο για κράτηση.

```
1  CREATE OR REPLACE VIEW upcoming_reservations_updatable AS
2  SELECT *
3  FROM reservations
4  WHERE departure_date > CURDATE()
5  ORDER BY arrival_date
```

6 Triggers

Δημιουργήσαμε τα παρακάτω δύο triggers.
Το trigger delete hotel rooms, το οποίο πριν διαγράψει ένα ξενοδοχείο διαγράφει και όλα τα δωμάτια του. Τα δωμάτια ανήκουν στο αδύναμο σύνολο οντοτήτων "Δωμάτιο" και εξαρτιούνται πλήρως από το ξενοδοχείο στο οποίο ανήκουν, οπότε και δεν έχει νόημα να υφίστανται αν διαγραφεί το ξενοδοχείο τους.

```
1 CREATE FUNCTION delete_rooms() RETURNS TRIGGER AS $_$
2         BEGIN
3             DELETE FROM rooms WHERE OLD.id = hotel_id;
4         RETURN OLD;
5         END $_$ LANGUAGE 'plpgsql';
6
7 REATE TRIGGER delete_hotel_rooms
8         BEFORE DELETE ON hotels
9         FOR EACH ROW EXECUTE PROCEDURE delete_rooms()
```

Παρομοίως με παραπάνω δημιουργήσαμε ένα trigger που διαγράφει όλες τις κρατήσεις ενός πελάτη πριν διαγραφεί ο πελάτης.

```
1 CREATE FUNCTION delete_reservations() RETURNS TRIGGER AS ↩
     $_$
2         BEGIN
3             DELETE FROM reservations WHERE OLD.id = ↩
                 client_id;
4         RETURN OLD;
5         END $_$ LANGUAGE 'plpgsql';
6
7 CREATE TRIGGER delete_client_reservations
8         BEFORE DELETE ON clients
9         FOR EACH ROW EXECUTE PROCEDURE ↩
             delete_reservations()
```

Στο σημείο αυτό αξίζει να αναφερθεί ότι το παραπάνω θα μπορούσε να υλοποιηθεί εύκολα και μέσα από το περιβάλλον του RoR Framework (χρήση Rails ActiveRecord callbacks). Αρκεί να προστεθούν τα παρακάτω στα μοντέλα hotel και client αντίστοιχα.

```
1 has_many :rooms, dependent: :destroy
```

```
1 has_many :reservations, dependent: :destroy
```

6.1 Callback vs Trigger

Πλεονεκτήματα Callback:

- Όλη η Business Logic του μοντέλου βρίσκεται στην Rails κάτι το οποίο κάνει πιο εύκολη την συντήρηση και ανάπτυξη του.
- Εύκολο Debug
- Ο κώδικας Ruby γράφετε και διαβάζεται πιο εύκολα
- Η εφαρμογή μας δεν εξαρτάται από την υλοποίηση (και συντακτικό) της βάσης δεδομένων.
- Database Agnostic

Πλεονεκτήματα Trigger:

- Είναι πιο γρήγορο από ένα callback. (Στο callback χρειάζεται να συνδεθείς στην βάση δεδομένων, ενώ στο trigger είσαι ήδη στο layer της βάσης δεδομένων)

Βάση των παραπάνω και δεδομένου το μέγεθος της εφαρμογής το ιδανικό θα ήταν να γίνει χρήση Rails ActiveRecord callbacks.

7 Ευρετήρια

Δημιουργήσαμε ευρετήρια στα foreign keys.

```
1 CREATE INDEX index_reservations_on_client_id ON ↩
    reservations USING btree (client_id);
2
3 CREATE INDEX index_reservations_on_hotel_id ON ↩
    reservations USING btree (hotel_id);
4
5 CREATE INDEX index_reservations_on_room_id ON ↩
    reservations USING btree (room_id);
6
7 CREATE INDEX index_rooms_on_hotel_id ON rooms USING btree↩
     (hotel_id);
8
9 CREATE UNIQUE INDEX unique_schema_migrations ON ↩
    schema_migrations USING btree (version);
```

8 SQL Dump

```
1  CREATE FUNCTION delete_reservations() RETURNS trigger
2      LANGUAGE plpgsql
3      AS $$
4          BEGIN
5              DELETE FROM reservations WHERE OLD.id = ↩
                   client_id;
6          RETURN OLD;
7          END $$;
8
9  CREATE FUNCTION delete_rooms() RETURNS trigger
10     LANGUAGE plpgsql
11     AS $$
12         BEGIN
13             DELETE FROM rooms WHERE OLD.id = hotel_id;
14         RETURN OLD;
15         END $$;
16
17 CREATE TABLE clients (
18     id integer NOT NULL,
19     identity character varying(255),
20     first_name character varying(255),
21     last_name character varying(255),
22     street character varying(255),
23     street_number integer,
24     zip_code character varying(255),
25     city character varying(255),
26     credit_card_number character varying(255),
27     created_at timestamp without time zone,
28     updated_at timestamp without time zone
29 );
30
31 CREATE SEQUENCE clients_id_seq
32     START WITH 1
33     INCREMENT BY 1
34     NO MINVALUE
35     NO MAXVALUE
36     CACHE 1;
37
38
39 CREATE TABLE hotels (
```

```
40      id integer NOT NULL,
41      name character varying(255),
42      street character varying(255),
43      street_number integer,
44      zip_code character varying(255),
45      city character varying(255),
46      rating integer,
47      construction_year integer,
48      renovation_year integer,
49      created_at timestamp without time zone,
50      updated_at timestamp without time zone
51  );
52
53
54  CREATE SEQUENCE hotels_id_seq
55      START WITH 1
56      INCREMENT BY 1
57      NO MINVALUE
58      NO MAXVALUE
59      CACHE 1;
60
61
62
63  CREATE TABLE reservations (
64      id integer NOT NULL,
65      hotel_id integer,
66      room_id integer,
67      client_id integer,
68      arrival_date date,
69      departure_date date,
70      payment_type character varying(255),
71      created_at timestamp without time zone,
72      updated_at timestamp without time zone
73  );
74
75  CREATE SEQUENCE reservations_id_seq
76      START WITH 1
77      INCREMENT BY 1
78      NO MINVALUE
79      NO MAXVALUE
80      CACHE 1;
81
82
```

```
83  CREATE TABLE rooms (
84      id integer NOT NULL,
85      hotel_id integer,
86      room_type character varying(255),
87      price integer,
88      created_at timestamp without time zone,
89      updated_at timestamp without time zone
90  );
91
92  CREATE SEQUENCE rooms_id_seq
93      START WITH 1
94      INCREMENT BY 1
95      NO MINVALUE
96      NO MAXVALUE
97      CACHE 1;
98
99
100 CREATE TABLE schema_migrations (
101     version character varying(255) NOT NULL
102 );
103
104 CREATE VIEW upcoming_reservations AS
105  SELECT reservations.id,
106     reservations.arrival_date,
107     reservations.departure_date,
108     reservations.created_at,
109     reservations.updated_at,
110     clients.first_name,
111     clients.last_name,
112     hotels.name,
113     rooms.room_type
114    FROM (((reservations
115    JOIN clients ON ((reservations.client_id = clients.id)↩
           ))
116    JOIN hotels ON ((reservations.hotel_id = hotels.id)))
117    JOIN rooms ON ((reservations.room_id = rooms.id)))
118   WHERE (reservations.arrival_date > ('now'::text)::date)
119   ORDER BY reservations.arrival_date;
120
121 CREATE VIEW upcoming_reservations_updatable AS
122  SELECT reservations.id,
123     reservations.hotel_id,
124     reservations.room_id,
```

```
125     reservations.client_id,
126     reservations.arrival_date,
127     reservations.departure_date,
128     reservations.payment_type,
129     reservations.created_at,
130     reservations.updated_at
131    FROM reservations
132   WHERE (reservations.departure_date > ('now'::text)::↩
          date)
133   ORDER BY reservations.arrival_date;
134
135
136
137 ALTER TABLE ONLY clients
138     ADD CONSTRAINT clients_pkey PRIMARY KEY (id);
139
140 ALTER TABLE ONLY hotels
141     ADD CONSTRAINT hotels_pkey PRIMARY KEY (id);
142
143 ALTER TABLE ONLY reservations
144     ADD CONSTRAINT reservations_pkey PRIMARY KEY (id);
145
146 ALTER TABLE ONLY rooms
147     ADD CONSTRAINT rooms_pkey PRIMARY KEY (id);
148
149
150 CREATE TRIGGER delete_client_reservations BEFORE DELETE ↩
       ON clients FOR EACH ROW EXECUTE PROCEDURE ↩
       delete_reservations();
151
152 CREATE TRIGGER delete_hotel_rooms BEFORE DELETE ON hotels↩
       FOR EACH ROW EXECUTE PROCEDURE delete_rooms();
```