



دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

: عنوان

پروژه درس طراحی پایگاه داده‌ها

Database-Design Course Project

نگارش

ملیکا علیزاده - ثمین اکبری - معین آعلی

استاد

مهردی آخی

نیمسال دوم ۰۲-۰۳

تقسیم‌بندی پروژهتسک‌های مربوط به عضو اول: [ملیکا علیزاده](#)

۱. ساختن جداول در در دیتابیس

۲. نرم‌ال سازی

۳. تولید داده فیک

۴. اصلاح کوئری‌ها

۵. ساختن index

تسک‌های مربوط به عضو دوم: [شمین اکبری](#)

۱. تغییر دادن روابط ERD

۲. نرم‌ال سازی

۳. ساختن index

۴. امتیازی بخش Mongo

۵. اصلاح کوئری‌ها

تسک‌های مربوط به عضو سوم: [معین آعلی](#)

۱. نوشتمن مستندات در قالب LaTeX

۲. بخش امتیازی مربوط به API

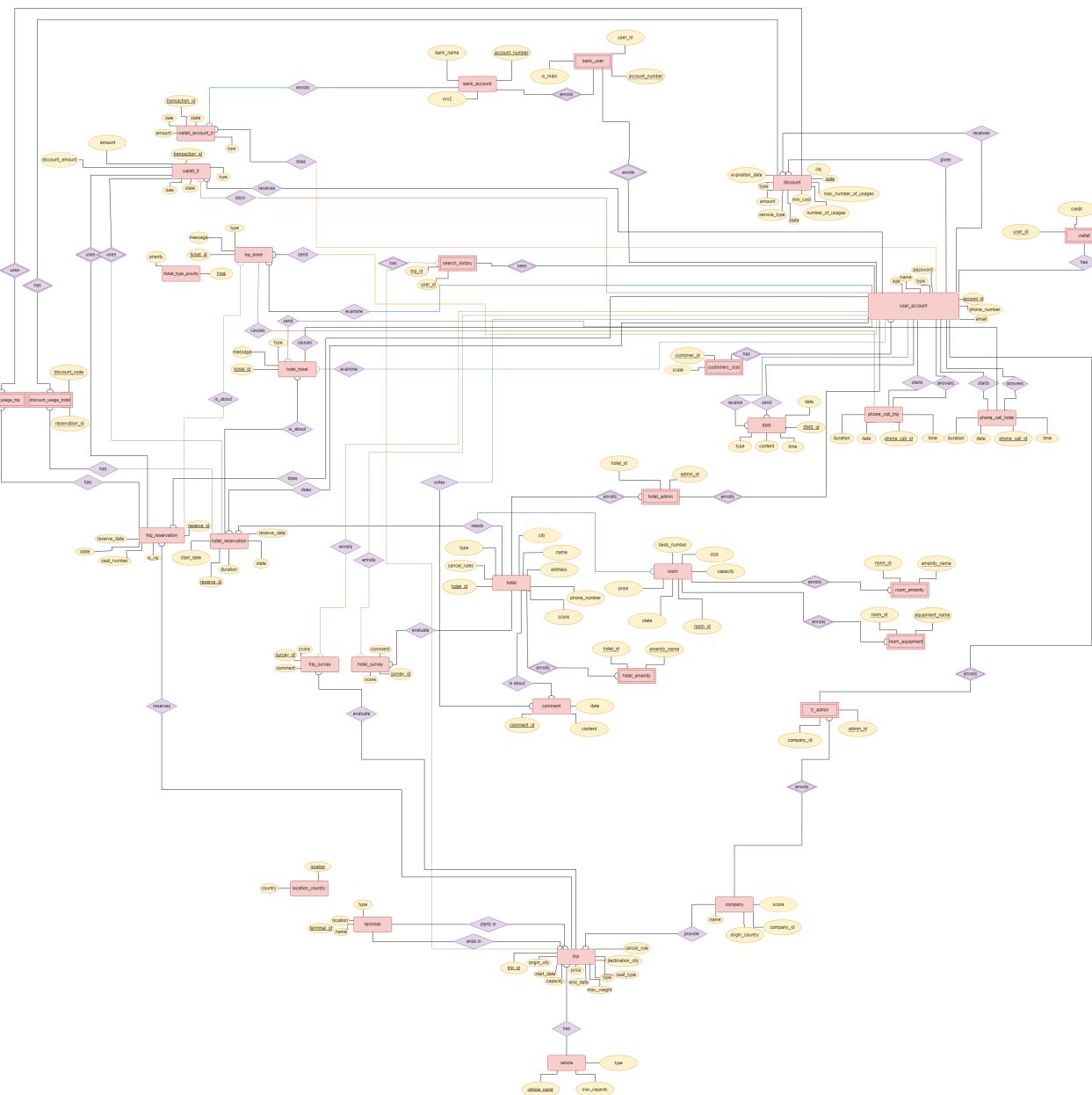
۳. نوشتمن کوئری‌ها

۴. تولید داده فیک

جزئیات تقسیم‌بندی کارها داخل این [Github Issue](#) موجود است.

نمودار رابطه - موجودیت

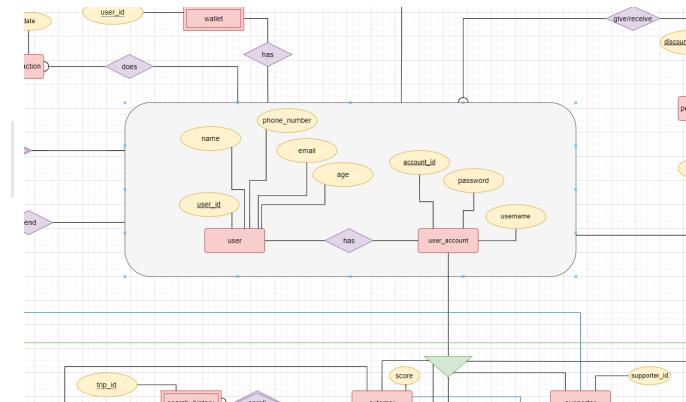
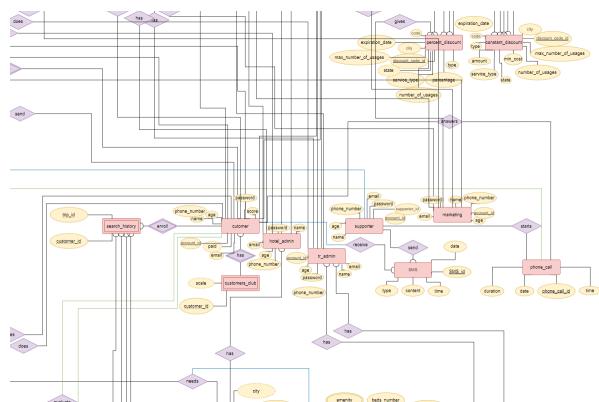
نمودار مربوط به فاز دوم پس از اعمال تغییرات مورد نیاز بر روی نمودار فاز ۱ به این صورت است:



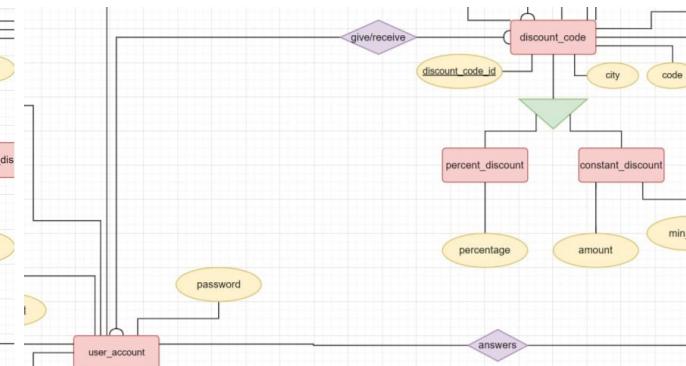
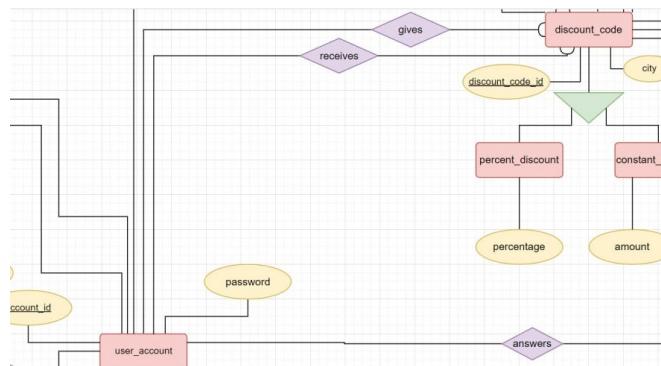
فایل Draw.io مربوط به ERD فاز دوم پروژه، داخل پیوست قرار دارد.

تغییرات نمودار در راستای انطباق بر SQL و نرم‌مال‌تر سازی

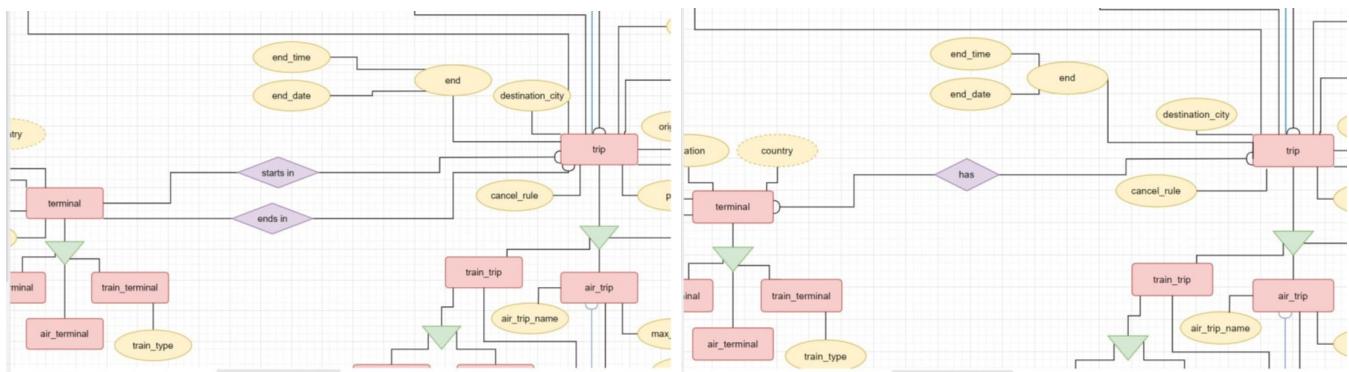
در تمامی عکس‌های این بخش، عکس سمت راست معادل پیش از تغییرات است و عکس سمت چپ معادل پس از تغییرات است. دلیل برخی از تغییرات در ادامه به دلیل تکراری بودن ذکر نشده است.



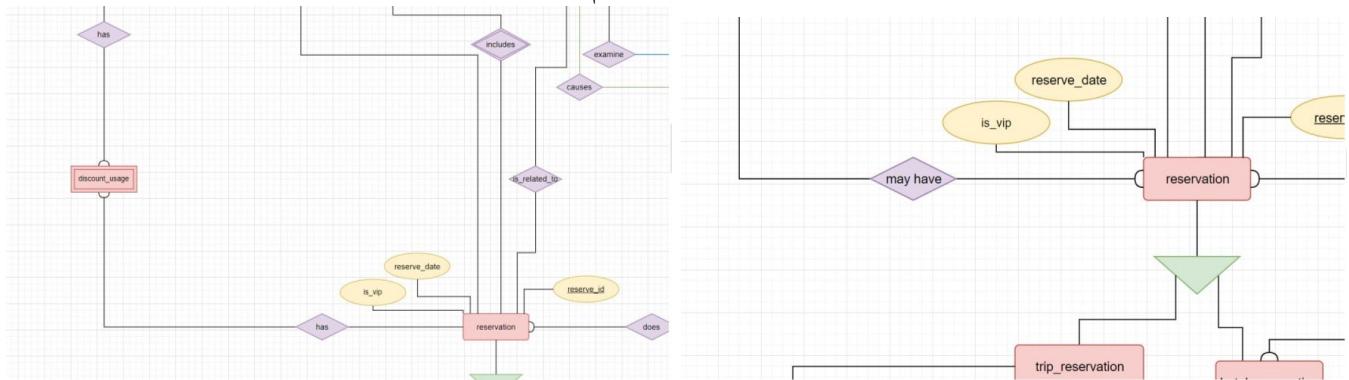
اگریگیشن در SQL وجود ندارد. بنابراین باید شکسته شود. در اینجا تصمیم بر این شد که کاربر و حساب کاربری به دلیل داشتن رابطه ۱:۱ و ویژگی‌های تقریباً مرتبط تبدیل به یک موجودیت شوند و تمام روابطی که به این اگریگیشن متصل است، به userAccount وصل شود.



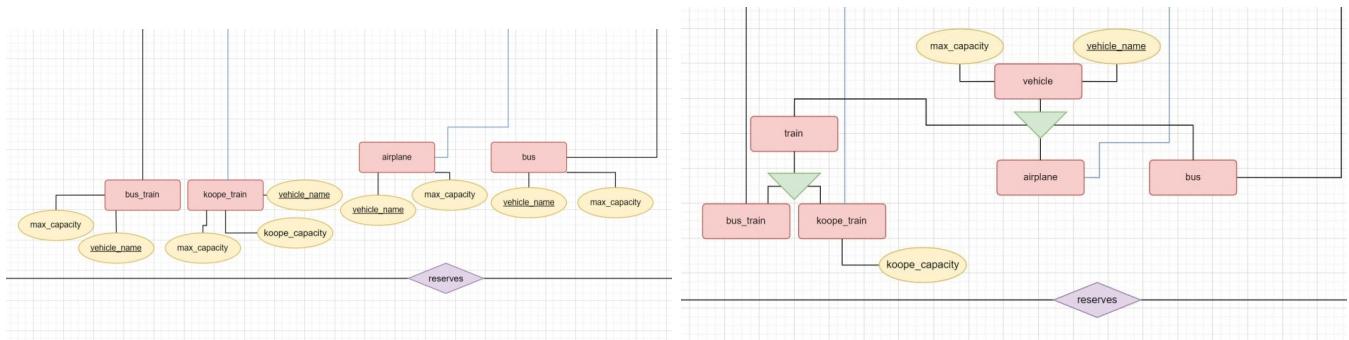
در گام بعد باید همه روابط manyToMany شکسته شوند. در اینجا رابطه بین userAccount و discountCode باید شکسته شود. برای این کار این رابطه را به دو رابطه تبدیل می‌کنیم که یکی از آنها مربوط به دریافت‌کننده و دیگری مربوط به اهداکننده کد تخفیف است.



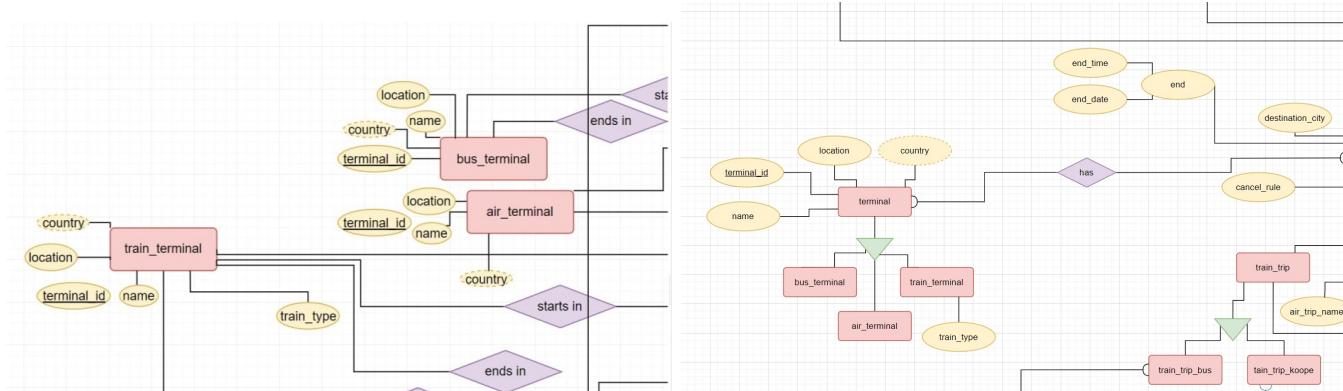
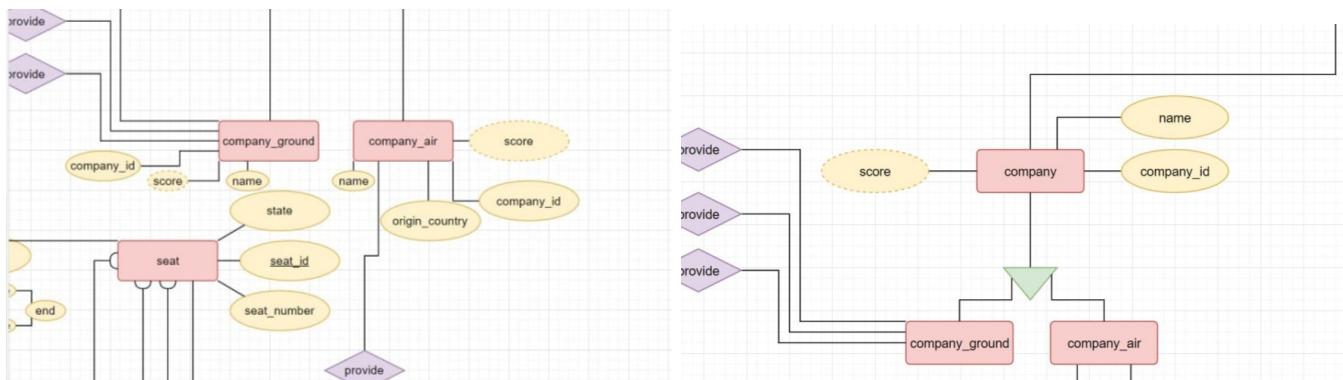
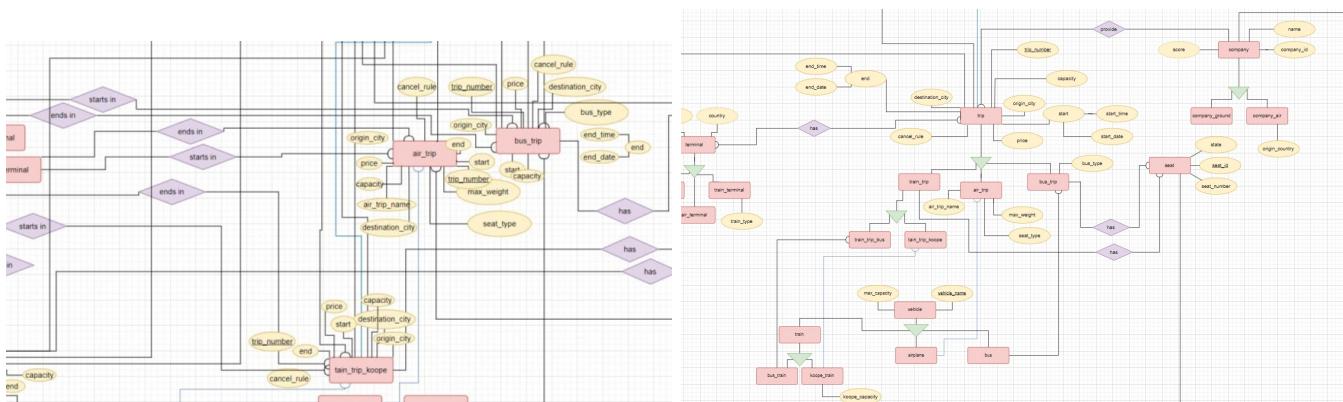
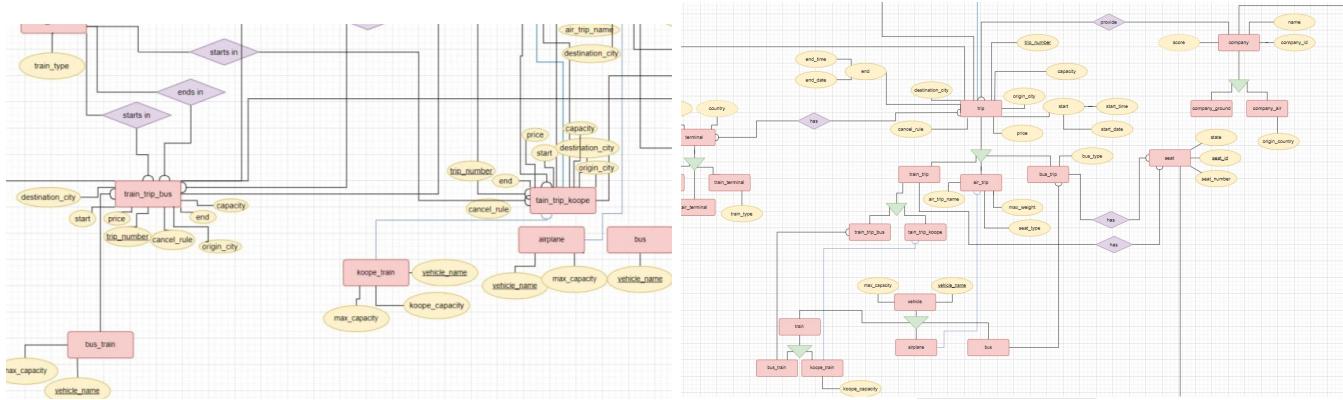
در اینجا بین **terminal** و **trip** رابطه m:n برقرار بود زیرا از هر ترمینال چند سفر انجام می‌شود و هر سفر یک ترمینال مبدا و یک ترمینال مقصد دارد. حالا باید این رابطه را تبدیل به دو رابطه اکنیم که یکی مبدا را مشخص می‌کند و یکی مقصد را.

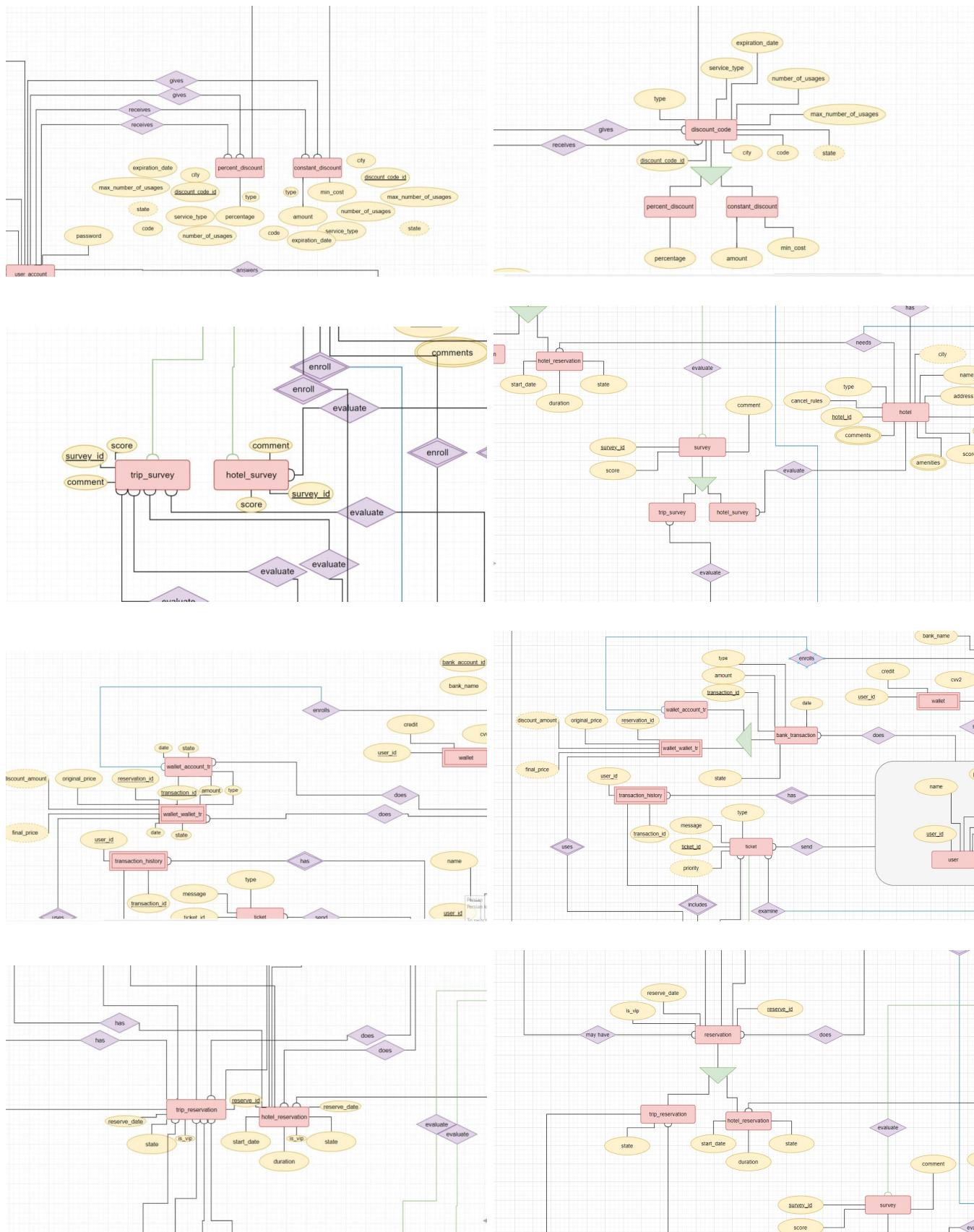


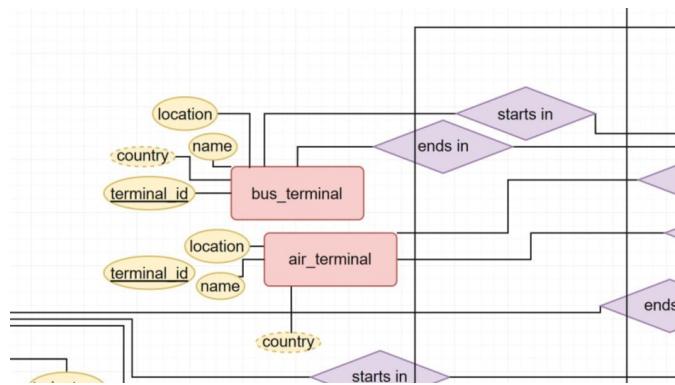
رابطه بین **discountCode** و **reservation** هم m:n بود چون هر کد تخفیف ممکن است بر چند رزرو اعمال شود و هر رزرو چند کد تخفیف داشته باشد. برای حل مشکل می‌توانیم یک موجودیت ضعیف **discountUsage** داشته باشیم که کلید خارجی به این دو موجودیت داشته باشد و مشخص کند هر استفاده‌ای از کد تخفیف مربوط به چه کد تخفیفی و برای چه رزروی است.



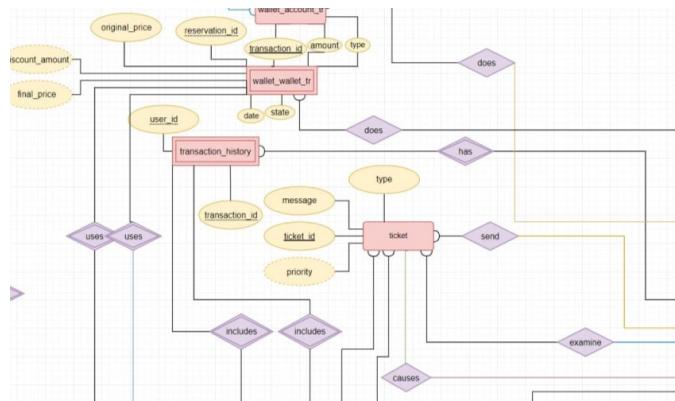
مورد دیگری که در SQL وجود ندارد ولی در نمودار ما بود، specialization است. برای حل این مشکل موجودیت کلی‌تر را حذف می‌کنیم، و تمام **attribute** روابط آن را به همه موجودیت‌های جزئی آن نسبت می‌دهیم.



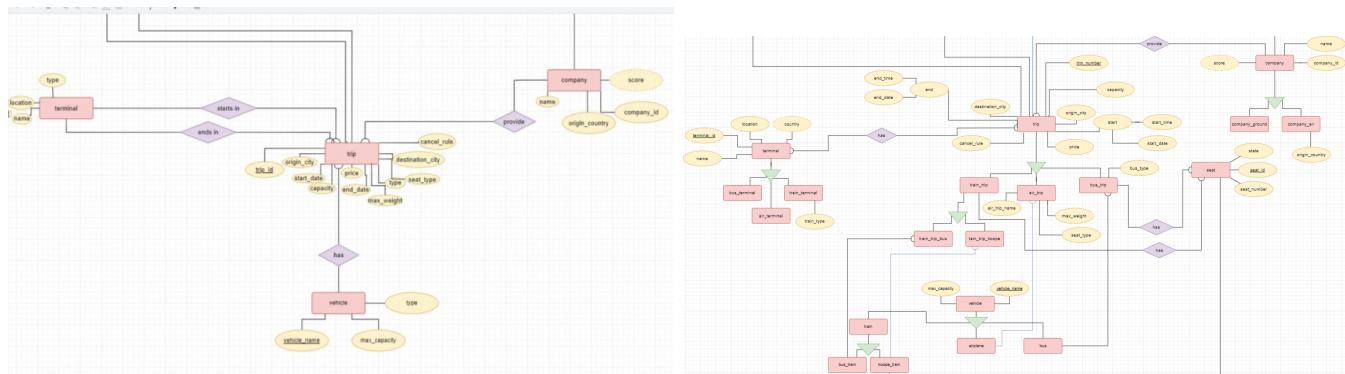




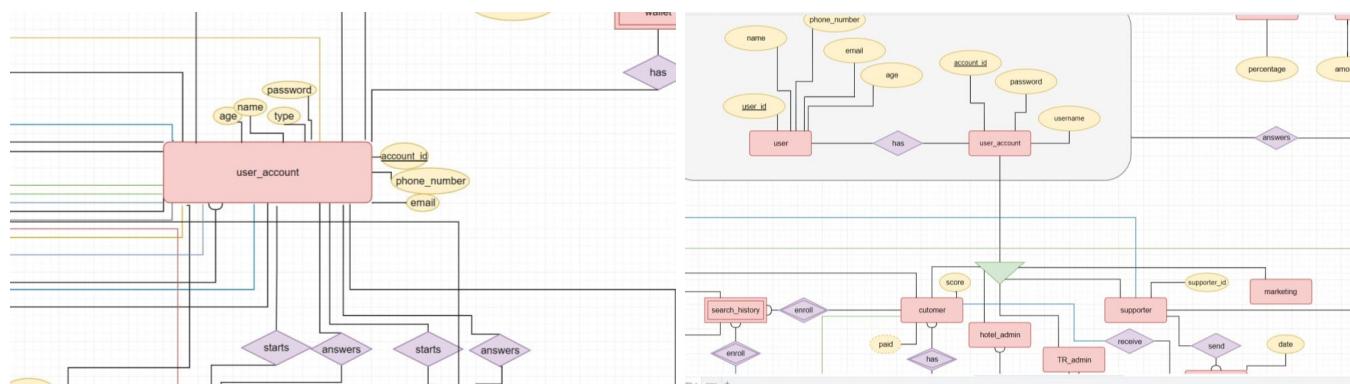
در همه انواع ترمینال لوکیشن میتواند کشور را مشخص کند، و لوکیشن یک ۳nf باید این مورد رفع شود. یک موجودیت ضعیف برای این مورد میسازیم و ویژگی کشور را از ترمینال حذف میکنیم.



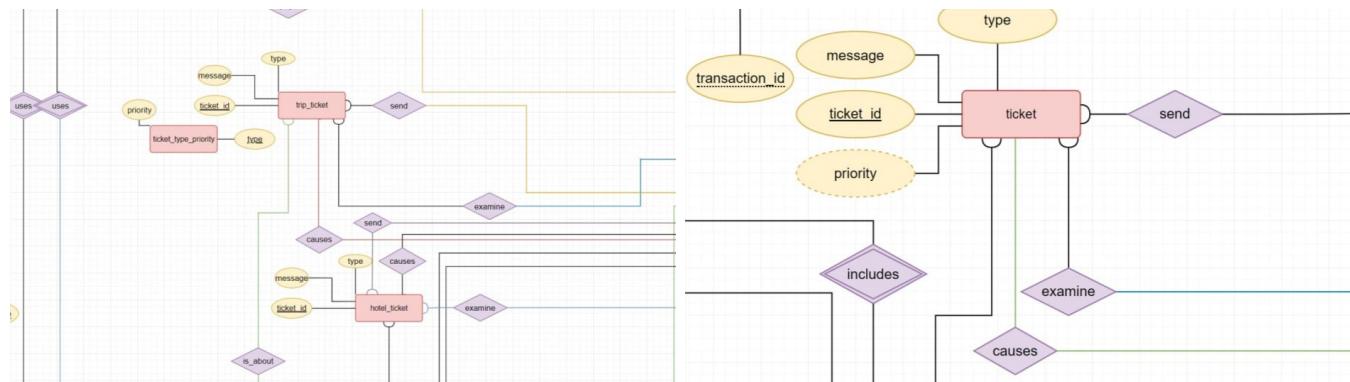
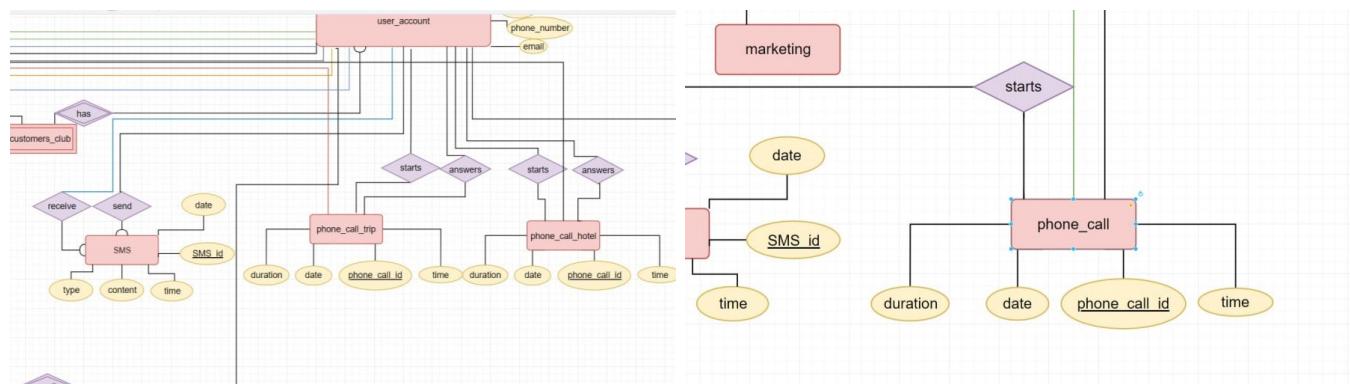
در اینجا هم بین type و priority وجود دارد. به همین دلیل این دو را در جدولی جدا درج کردیم که اصول ۳nf برقرار باشد.



در حین پیاده‌سازی جداول متوجه شدیم که به صورت انواع مختلفی از یک موجودیت جدا کردیم (specialization)، میتوانند همگی یک موجودیت باشند که نوعشان در اtributیتی به نام type ذخیره شود:



همچنین دریافتیم که نیاز است برخی از موجودیت‌ها به دو موجودیت جداگانه شکسته شوند. دلیل این موضوع، **foreign key** بود که مشخص نبود باید دقیقاً به کدام موجودیت اشاره کنند (دو حالت داشتند).



ساخت جداول در Postgres

جداول دیتابیس ما از روی ERD اصلاح شده، مطابق دستورات زیر ساخته شدند:
فایل sql مربوط به create-table.sql داخل پوست قرار دارد.

```

1 CREATE TABLE user_account
2 (
3     account_id    SERIAL PRIMARY KEY,
4     type          TEXT,
5     name          TEXT,
6     password      TEXT,
7     age           INT,
8     phone_number TEXT,
9     email         TEXT
10 );
11
12 CREATE TABLE customer_club
13 (
14     customer_id  INT not null PRIMARY KEY,
15     scale         INT,
16     CONSTRAINT fk_customer
17         FOREIGN KEY (customer_id)
18             REFERENCES user_account (account_id)
19 );
20
21 CREATE TABLE SMS
22 (
23     SMS_id        SERIAL not null PRIMARY KEY,
24     sender_id     INT    not null,
25     receiver_id   INT    not null,
26     date          DATE,
27     type          TEXT,
28     content       TEXT,
29     time          TIME,
30     CONSTRAINT fk_sender
31         FOREIGN KEY (sender_id)
32             REFERENCES user_account (account_id),
33     CONSTRAINT fk_receiver
34         FOREIGN KEY (receiver_id)
35             REFERENCES user_account (account_id)
36 );
37
38
39 CREATE TABLE wallet
40 (
41     user_id       INT not null PRIMARY KEY,
42     credit        DECIMAL,
43     CONSTRAINT fk_user
44         FOREIGN KEY (user_id)
45             REFERENCES user_account (account_id)
46 );
47
48 CREATE TABLE discount
49 (
50     code          VARCHAR(255) not null PRIMARY KEY,
51     user_id       INT          not null,
52     city          TEXT,
53     expiration_date DATE,
54     max_number_of_usages INT,
55     service_type  TEXT,
56     amount        DECIMAL,
57     type          TEXT,
58     min_cost      DECIMAL,
59     number_of_usage INT,
60     CONSTRAINT fk_user
61         FOREIGN KEY (user_id)
62             REFERENCES user_account (account_id)
63 );
64
65
66

```

```

66 CREATE TABLE bank_account
67 (
68     account_number INT not null PRIMARY KEY,
69     bank_name      TEXT,
70     CVV2          INT
71 );
72
73 CREATE TABLE bank_user
74 (
75     account_number INT not null PRIMARY KEY,
76     user_id        INT,
77     is_main         boolean default false,
78     CONSTRAINT fk_account_number
79         FOREIGN KEY (account_number)
80             REFERENCES bank_account (account_number),
81     CONSTRAINT fk_user
82         FOREIGN KEY (user_id)
83             REFERENCES user_account (account_id)
84 );
85
86 CREATE TABLE hotel
87 (
88     hotel_id      SERIAL PRIMARY KEY,
89     type          INT          not null,
90     name          TEXT         not null,
91     address       TEXT         not null,
92     city          TEXT         not null,
93     cancel_rule   TEXT         not null,
94     score         float default 0 not null,
95     phone_number  TEXT         not null
96 );
97
98 CREATE TABLE hotel_admin
99 (
100    hotel_id INT not null,
101    admin_id INT not null,
102    PRIMARY KEY (hotel_id, admin_id),
103    CONSTRAINT fk_hotel
104        FOREIGN KEY (hotel_id)
105            REFERENCES hotel (hotel_id),
106    CONSTRAINT fk_admin
107        FOREIGN KEY (admin_id)
108            REFERENCES user_account (account_id)
109 );
110
111 CREATE TABLE hotel_amenity
112 (
113     hotel_id      INT          not null,
114     amenity_name varchar(255) not null,
115     PRIMARY KEY (hotel_id, amenity_name),
116     CONSTRAINT fk_hotel
117         FOREIGN KEY (hotel_id)
118             REFERENCES hotel (hotel_id)
119 );
120
121 CREATE TABLE room
122 (
123     room_id      SERIAL PRIMARY KEY,
124     hotel_id     INT NOT NULL,
125     state         boolean default False,
126     capacity      INT not null,
127     size          INT not null,
128     beds_number  INT not null,
129     price         DECIMAL,
130     CONSTRAINT fk_hotel
131         FOREIGN KEY (hotel_id)
132             REFERENCES hotel (hotel_id)
133 );
134
135
136

```

```

137 CREATE TABLE room_amenity
138 (
139     room_id      INT          not null,
140     amenity_name varchar(255) not null,
141     PRIMARY KEY (room_id, amenity_name),
142     CONSTRAINT fk_room
143         FOREIGN KEY (room_id)
144             REFERENCES room (room_id)
145 );
146
147 CREATE TABLE room_equipment
148 (
149     room_id      INT          not null,
150     equipment_name varchar(255) not null,
151     PRIMARY KEY (room_id, equipment_name),
152     CONSTRAINT fk_room
153         FOREIGN KEY (room_id)
154             REFERENCES room (room_id)
155 );
156
157 CREATE TABLE company
158 (
159     company_id    SERIAL        not null PRIMARY KEY,
160     name          TEXT          not null,
161     score         float default 0 not null,
162     origin_country TEXT         not null
163 );
164
165 CREATE TABLE tr_admin
166 (
167     company_id INT not null,
168     admin_id   INT not null,
169     PRIMARY KEY (admin_id),
170     CONSTRAINT fk_admin
171         FOREIGN KEY (company_id)
172             REFERENCES company (company_id),
173     CONSTRAINT fk_admin
174         FOREIGN KEY (admin_id)
175             REFERENCES user_account (account_id)
176 );
177
178 CREATE TABLE comment
179 (
180     comment_id   SERIAL not null PRIMARY KEY,
181     hotel_id    INT    not null,
182     writer_id   INT    not null,
183     date        DATE,
184     content     TEXT,
185     CONSTRAINT fk_hotel
186         FOREIGN KEY (hotel_id)
187             REFERENCES hotel (hotel_id),
188     CONSTRAINT fk_writer
189         FOREIGN KEY (writer_id)
190             REFERENCES user_account (account_id)
191 );
192
193
194 CREATE TABLE terminal
195 (
196     terminal_id SERIAL not null PRIMARY KEY,
197     name          TEXT,
198     location     TEXT,
199     type         TEXT
200 );
201
202 CREATE TABLE vehicle
203 (
204     vehicle_name TEXT not null PRIMARY KEY,
205     max_capacity INT,
206     type         VARCHAR(255)
207 );

```

```

CREATE TABLE trip
(
    trip_id          VARCHAR(255) not null PRIMARY KEY,
    origin_terminal_id INT        not null,
    destination_terminal_id INT        not null,
    vehicle_name     TEXT       not null,
    origin_city      TEXT       not null,
    price            DECIMAL,
    capacity          INT,
    destination_city TEXT,
    end_date          DATE,
    start_date        DATE,
    cancel_rule      TEXT,
    max_weight        DECIMAL,
    seat_type         TEXT,
    CONSTRAINT fk_vehicle
        FOREIGN KEY (vehicle_name)
            REFERENCES vehicle (vehicle_name),
    CONSTRAINT fk_origin_terminal
        FOREIGN KEY (origin_terminal_id)
            REFERENCES terminal (terminal_id),
    CONSTRAINT fk_destination_terminal
        FOREIGN KEY (destination_terminal_id)
            REFERENCES terminal (terminal_id)
);
;

CREATE TABLE location_country
(
    location TEXT not null PRIMARY KEY,
    country  TEXT not null
);
;

CREATE TABLE wallet_account_tr
(
    transaction_id    SERIAL not null PRIMARY KEY,
    bank_account_number INT,
    user_id           INT,
    date              DATE,
    token             TEXT,
    amount            DECIMAL,
    type              TEXT,
    state             TEXT,
    CONSTRAINT fk_account_number
        FOREIGN KEY (bank_account_number)
            REFERENCES bank_account (account_number),
    CONSTRAINT fk_user
        FOREIGN KEY (user_id)
            REFERENCES user_account (account_id)
);
;

CREATE TABLE wallet_tr
(
    transaction_id    SERIAL not null PRIMARY KEY,
    giver_id          INT,
    receiver_id        INT,
    original_price    DECIMAL,
    discount_amount   DECIMAL,
    date              DATE,
    amount            DECIMAL,
    type              TEXT,
    state             TEXT,
    CONSTRAINT fk_giver
        FOREIGN KEY (giver_id)
            REFERENCES user_account (account_id),
    CONSTRAINT fk_receiver
        FOREIGN KEY (receiver_id)
            REFERENCES user_account (account_id)
);
;

```

```

779 CREATE TABLE trip_reservation
780 (
781     reserve_id      SERIAL not null PRIMARY KEY,
782     transaction_id INT,
783     trip_number    VARCHAR(255),
784     user_id        INT,
785     reserve_date   DATE,
786     is_vip         boolean default false,
787     state          TEXT,
788     seat_number    INT,
789     CONSTRAINT fk_trip
790         FOREIGN KEY (trip_number)
791             REFERENCES trip (trip_id),
792     CONSTRAINT fk_admin
793         FOREIGN KEY (user_id)
794             REFERENCES user_account (account_id),
795     CONSTRAINT fk_transaction
796         FOREIGN KEY (transaction_id)
797             REFERENCES wallet_tr (transaction_id)
798 );
799
800 CREATE TABLE hotel_reservation
801 (
802     reserve_id      SERIAL not null PRIMARY KEY,
803     transaction_id INT,
804     hotel_id       INT,
805     room_id        INT,
806     user_id        INT,
807     reserve_date   DATE,
808     duration       INTERVAL,
809     state          TEXT,
810     start_date    DATE,
811     CONSTRAINT fk_hotel
812         FOREIGN KEY (hotel_id)
813             REFERENCES hotel (hotel_id),
814     CONSTRAINT fk_room
815         FOREIGN KEY (room_id)
816             REFERENCES room (room_id),
817     CONSTRAINT fk_admin
818         FOREIGN KEY (user_id)
819             REFERENCES user_account (account_id),
820     CONSTRAINT fk_transaction
821         FOREIGN KEY (transaction_id)
822             REFERENCES wallet_tr (transaction_id)
823 );
824
825 CREATE TABLE discount_usage_trip
826 (
827     discount_code  VARCHAR(255) not null,
828     reservation_id INT          not null,
829     PRIMARY KEY (discount_code, reservation_id),
830     CONSTRAINT fk_discount
831         FOREIGN KEY (discount_code)
832             REFERENCES discount (code),
833     CONSTRAINT fk_reservation
834         FOREIGN KEY (reservation_id)
835             REFERENCES trip_reservation (reserve_id)
836 );
837
838 CREATE TABLE discount_usage_hotel
839 (
840     discount_code  VARCHAR(255) not null,
841     reservation_id INT          not null,
842     PRIMARY KEY (discount_code, reservation_id),
843     CONSTRAINT fk_discount
844         FOREIGN KEY (discount_code)
845             REFERENCES discount (code),
846     CONSTRAINT fk_reservation
847         FOREIGN KEY (reservation_id)
848             REFERENCES hotel_reservation (reserve_id)
849 );

```

```

۷۰+
۷۰۱ CREATE TABLE trip_survey
(
۷۰۲     survey_id    SERIAL not null PRIMARY KEY,
۷۰۳     trip_number  VARCHAR(255),
۷۰۴     user_id      INT,
۷۰۵     comment      TEXT,
۷۰۶     score        INT,
۷۰۷     CONSTRAINT fk_trip
۷۰۸         FOREIGN KEY (trip_number)
۷۰۹             REFERENCES trip (trip_id),
۷۱۰     CONSTRAINT fk_admin
۷۱۱         FOREIGN KEY (user_id)
۷۱۲             REFERENCES user_account (account_id)
);
۷۱۳
۷۱۴ CREATE TABLE hotel_survey
(
۷۱۵     survey_id    SERIAL not null PRIMARY KEY,
۷۱۶     hotel_id     INT,
۷۱۷     user_id      INT,
۷۱۸     comment      TEXT,
۷۱۹     score        INT,
۷۲۰     CONSTRAINT fk_hotel
۷۲۱         FOREIGN KEY (hotel_id)
۷۲۲             REFERENCES hotel (hotel_id),
۷۲۳     CONSTRAINT fk_admin
۷۲۴         FOREIGN KEY (user_id)
۷۲۵             REFERENCES user_account (account_id)
);
۷۲۶
۷۲۷ CREATE TABLE trip_ticket
(
۷۲۸     ticket_id     SERIAL not null PRIMARY KEY,
۷۲۹     trip_reservation_id INT,
۷۳۰     user_id       INT,
۷۳۱     examiner_id   INT,
۷۳۲     message       TEXT,
۷۳۳     type          TEXT,
۷۳۴     CONSTRAINT fk_trip
۷۳۵         FOREIGN KEY (trip_reservation_id)
۷۳۶             REFERENCES trip_reservation (reserve_id),
۷۳۷     CONSTRAINT fk_user
۷۳۸         FOREIGN KEY (user_id)
۷۳۹             REFERENCES user_account (account_id),
۷۴۰     CONSTRAINT fk_examiner
۷۴۱         FOREIGN KEY (examiner_id)
۷۴۲             REFERENCES user_account (account_id)
);
۷۴۳
۷۴۴ CREATE TABLE hotel_ticket
(
۷۴۵     ticket_id     SERIAL not null PRIMARY KEY,
۷۴۶     hotel_reservation_id INT,
۷۴۷     user_id       INT,
۷۴۸     examiner_id   INT,
۷۴۹     message       TEXT,
۷۵۰     type          TEXT,
۷۵۱     CONSTRAINT fk_hotel
۷۵۲         FOREIGN KEY (hotel_reservation_id)
۷۵۳             REFERENCES hotel_reservation (reserve_id),
۷۵۴     CONSTRAINT fk_user
۷۵۵         FOREIGN KEY (user_id)
۷۵۶             REFERENCES user_account (account_id),
۷۵۷     CONSTRAINT fk_examiner
۷۵۸         FOREIGN KEY (examiner_id)
۷۵۹             REFERENCES user_account (account_id)
);
۷۶۰
۷۶۱

```

```

۴۲۱ CREATE TABLE phone_call_hotel
۴۲۲ (
۴۲۳     phone_call_id SERIAL not null PRIMARY KEY,
۴۲۴     caller_id      INT    not null,
۴۲۵     callee_id     INT    not null,
۴۲۶     ticket_id     int     not null,
۴۲۷     duration       INTERVAL,
۴۲۸     date          DATE,
۴۲۹     time          TIME,
۴۳۰     CONSTRAINT fk_caller
۴۳۱         FOREIGN KEY (caller_id)
۴۳۲             REFERENCES user_account (account_id),
۴۳۳     CONSTRAINT fk_callee
۴۳۴         FOREIGN KEY (callee_id)
۴۳۵             REFERENCES user_account (account_id),
۴۳۶     CONSTRAINT fk_ticket
۴۳۷         FOREIGN KEY (ticket_id)
۴۳۸             REFERENCES hotel_ticket (ticket_id)
۴۳۹ );
۴۴۰
۴۴۱ CREATE TABLE phone_call_trip
۴۴۲ (
۴۴۳     phone_call_id SERIAL not null PRIMARY KEY,
۴۴۴     caller_id      INT    not null,
۴۴۵     callee_id     INT    not null,
۴۴۶     ticket_id     int     not null,
۴۴۷     duration       INTERVAL,
۴۴۸     date          DATE,
۴۴۹     time          TIME,
۴۵۰     CONSTRAINT fk_caller
۴۵۱         FOREIGN KEY (caller_id)
۴۵۲             REFERENCES user_account (account_id),
۴۵۳     CONSTRAINT fk_callee
۴۵۴         FOREIGN KEY (callee_id)
۴۵۵             REFERENCES user_account (account_id),
۴۵۶     CONSTRAINT fk_ticket
۴۵۷         FOREIGN KEY (ticket_id)
۴۵۸             REFERENCES trip_ticket (ticket_id)
۴۵۹ );
۴۶۰
۴۶۱
۴۶۲ CREATE TABLE ticket_type_priority
۴۶۳ (
۴۶۴     type      TEXT not null primary key,
۴۶۵     priority  INT
۴۶۶ );

```

ساخت index

```

1 CREATE INDEX idx_hotel_name ON hotel (name);
2 CREATE INDEX idx_user_name ON user_account (name);
3 CREATE INDEX idx_hotel_type ON hotel (type);
4 CREATE INDEX idx_company_score ON company (score);

```

برای هتل، علاوه بر ایندکس اصلی که id آن است، یک ایندکس هم بر اساس نام هتل‌ها گذاشتیم. این باعث تسريع در اجرای کوئری‌ها می‌شود. برای مثال اگر میخواهیم به دنبال نام یک هتل بگردیم، با این ایندکس راحت‌تر میتوانیم این کار را بکنیم.

برای هتل همچنین یک ایندکس دیگر برای type (تعداد ستاره‌ها) تعريف کردیم. اگر کاربری بخواهد بهترین هتل‌ها را ببیند، میتوانیم بر اساس این ایندکس به راحتی این هتل‌ها را به ترتیب خروجی دهیم.

برای شرکت‌ها هم بر اساس امتیازشان ایندکس گذاشتیم که اگر بخواهیم در جایی شرکت‌ها را با بالاترین امتیاز یا در رنج خاصی از امتیاز نمایش دهیم، به راحتی قابل انجام باشد.

برای کاربران هم یک ایندکس بر اساس اسمای آنها گذاشتیم. اگر بخواهیم اطلاعات یک کاربر را با داشتن نامش به دست آوریم، با استفاده از این ایندکس به راحتی امکان‌پذیر است.

با توجه به کوئری‌های بخش SQL باقی Index‌ها به این صورت ساخته شدند:

```

-- more indexes

CREATE INDEX idx_trip_start_date_vehicle_name ON trip (start_date, vehicle_name);

CREATE INDEX idx_vehicle_type_vehicle_name ON vehicle (type, vehicle_name);

CREATE INDEX idx_room_beds_number_room_id ON room (beds_number, room_id);

CREATE INDEX idx_hotel_amenity_name_hotel_id ON hotel_amenity (amenity_name, hotel_id);

CREATE INDEX idx_discount_code_type ON discount (code, type);

CREATE INDEX idx_discount_usage_trip_discount_code ON discount_usage_trip (discount_code);

CREATE INDEX idx_discount_usage_hotel_discount_code ON discount_usage_hotel (discount_code);

CREATE INDEX idx_trip_reservation_state_trip_number ON trip_reservation (state, trip_number);

CREATE INDEX idx_trip_start_date_vehicle_name ON trip (start_date, vehicle_name);

CREATE INDEX idx_trip_reservation_state_trip_number ON trip_reservation (state, trip_number);

CREATE INDEX idx_hotel_city_hotel_id ON hotel (city, hotel_id);

CREATE INDEX idx_trip_reservation_user_id_trip_number ON trip_reservation (user_id, trip_number);

CREATE INDEX idx_hotel_reservation_hotel_id_start_date ON hotel_reservation (hotel_id, start_date);

```

ساخت سرویس برای Rest API

برای زدن این بخش مجبور شدیم که با انواع ریکوئست‌های HTTP آشنا بشویم. در اینجا نیاز بود یک فریمورک برای پیاده‌سازی وب‌سرویس بک‌اندی خودمون انتخاب کنیم.

ما از زبان برنامه‌نویسی JS و فریمورک Fastify استفاده کردیم و وب‌سرویس را پیاده‌سازی کردیم.

همچنین برای زدن ریکوئست‌های Post از ابزار Postman استفاده کردیم که در ادامه به آن اشاره خواهیم کرد.

```

1 const fastify = require("fastify")();
2
3 fastify.register(require("@fastify/postgres"), {
4   connectionString: "postgres://postgres:1111@localhost:5432/db_project"
5 });

```

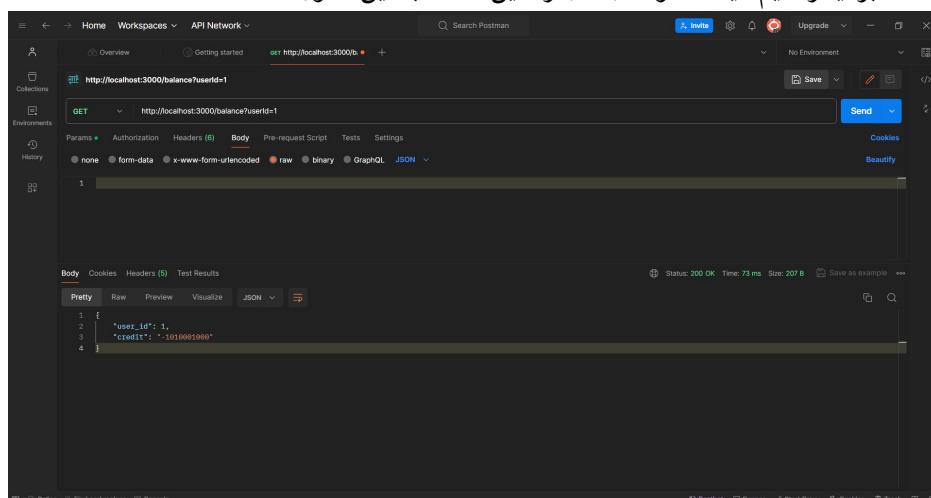
در اینجا کتابخانه Fastify را فراخوانی می‌کنیم و سپس با استفاده از آن به دیتابیس خود که روی localhost و پورت ۵۴۳۲ بالا است وصل می‌شویم.

```

1 fastify.get("/balance", async (req, res) => {
2   const userId = req.query?.userId;
3
4   if (!userId) {
5     return res.status(400).send("Please Enter userId ...");
6   }
7
8   const query = `
9     SELECT user_id, credit
10    FROM user_account
11   JOIN wallet ON wallet.user_id = user_account.account_id
12 WHERE user_account.account_id = $1
13 `;
14
15   try {
16     const result = await fastify.pg.query(query, [userId]);
17
18     if (result.rowCount === 0) {
19       return res.status(404).send("User not found");
20     }
21
22     res.send(result.rows[0]);
23   } catch (err) {
24     console.error("Database query error: ", err);
25     res.status(500).send("Internal Server Error");
26   }
27 });

```

در این بخش اولین ریکوئست ما هندل شده است. این ریکوئست از نوع Get است و با استفاده از userID ما میزان موجودی آن کاربر را به همراه Id برミگردانیم. یک نمونه از کارکرد این API به این صورت است:

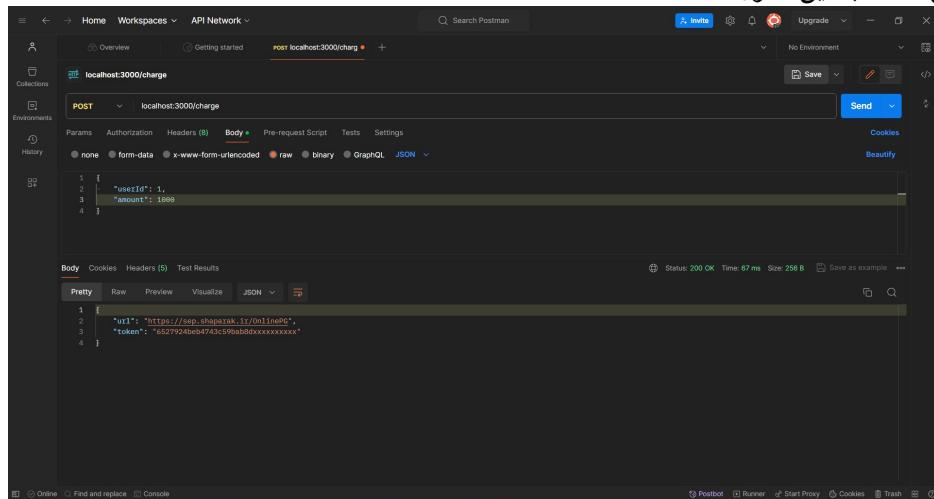


```

1 fastify.post("/charge", async (req, res) => {
2   const userId = req.body?.userId;
3   const amount = req.body?.amount;
4
5   const query = `
6     SELECT ACCOUNT_NUMBER
7       FROM bank_user
8      WHERE user_id = $1;
9
10  const query2 = `
11    INSERT INTO wallet_account_tr (token, bank_account_number, user_id, date, amount, type, state)
12    VALUES ('6527924beb4743c59bab8dxxxxxxxxx', $1, $2, '2024-06-01', $3, 'Deposit', 'Pending');
13
14  if (!(userId && amount)) res.send("Please enter userId and amount ...");
15  try {
16    let result = await fastify.pg.query(query, [userId])
17    const accountNumber = result.rows[0].account_number
18    result = await fastify.pg.query(query2, [accountNumber, userId, amount])
19    res.send({ "url": "https://sep.shaparak.ir/OnlinePG", "token": "6527924beb4743c59bab8dxxxxxxxxx"
20      });
21  } catch (error) {
22    console.error("Database query error: ", err);
23    res.status(500).send("Internal Server Error");
24  }
25});

```

ریکوئست این بخش از نوع Post است که تراکنش ایجاد کرده و لینک درگاه پرداخت به همراه یک توکن به وی برگردانده میشود. کارکرد این API به این صورت است:



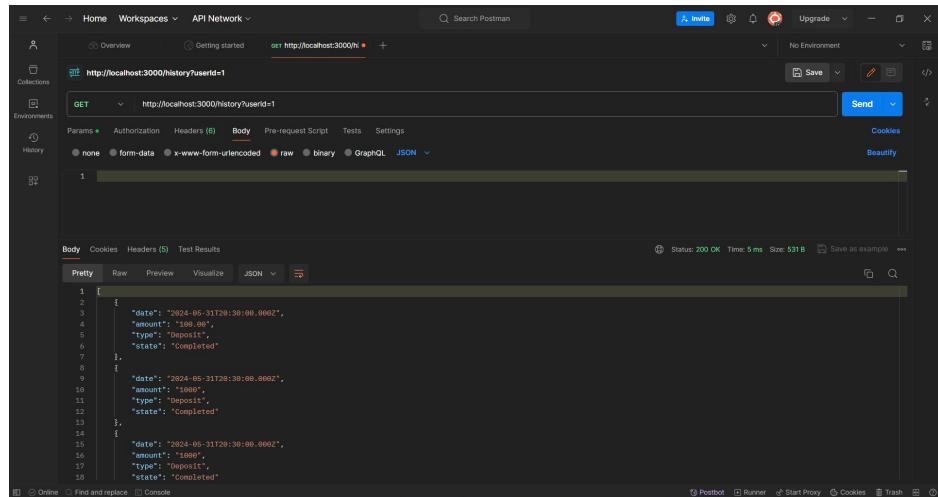
البته مقدار توکن را به صورت هاردکد قرار دادیم چون تولید یک توکن یکتا و ذخیره آن جزوی از سرویس ما نیست و باید داخل یک سرویس دیگر پیاده سازی شود و ما از آن استفاده کنیم.

```

1 fastify.post("/verify", async (req, res) => {
2   const userId = req.body?.userId;
3   const token = req.body?.token;
4   const amount = req.body?.amount;
5   if (!token) res.send("Please enter token ...");
6   const query = `
7     UPDATE wallet_account_tr
8     SET state = 'Completed'
9     WHERE token = $1 AND user_id = $2;
10    `;
11
12   const query2 = `
13     SELECT credit
14     FROM wallet
15     WHERE user_id = $1
16    `;
17
18   const query3 = `
19     UPDATE wallet
20     SET credit = $2
21     WHERE user_id = $1;
22    `;
23
24   try {
25     let result = await fastify.pg.query(query, [token, userId])
26     let credit_old = await fastify.pg.query(query2, [userId])
27     result = await fastify.pg.query(query3, [userId, credit_old.rows[0].credit + amount])
28     res.send({ "status": "verified" });
29   } catch (error) {
30     console.error("Database query error: ", err);
31     res.status(500).send("Internal Server Error");
32   }
33 });

```

این بخش همانند API قبلي از نوع Post است و باید یک ریکوئست بزنیم و تراکنش قبلی را تایید و ثبت کنیم. کارکرد این API به این صورت است:

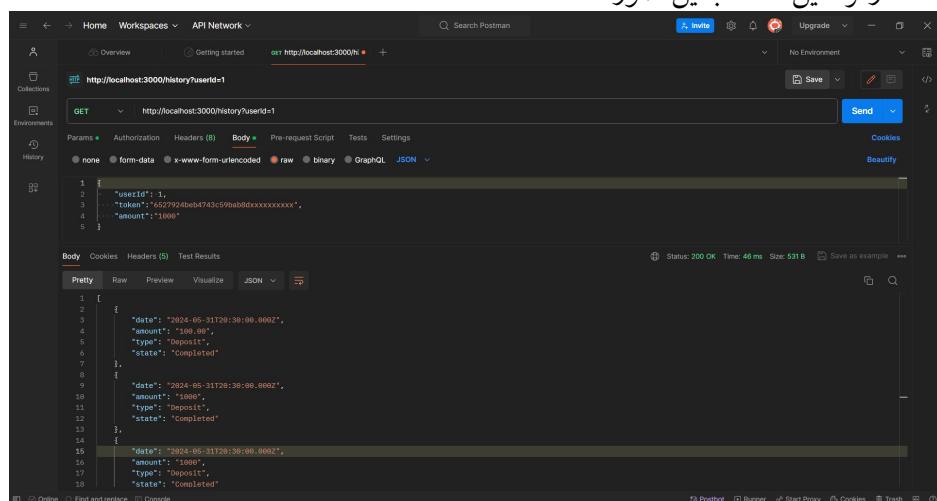


```

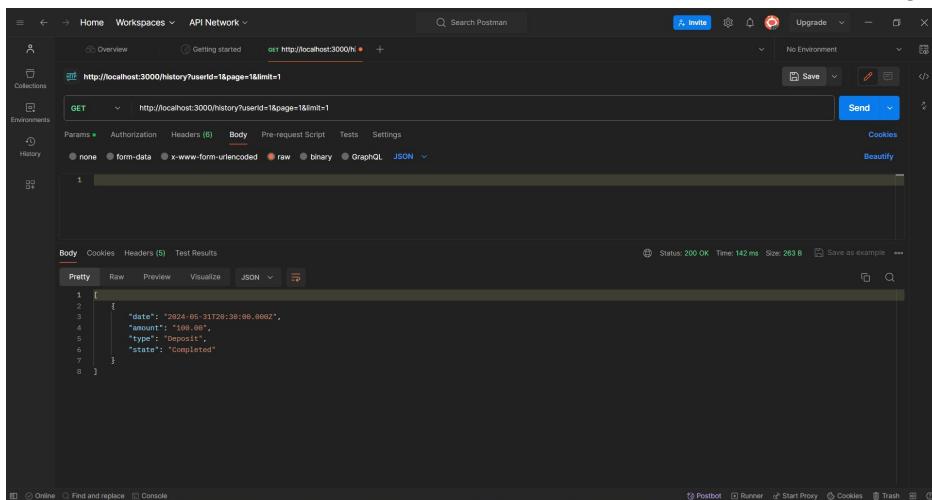
1 fastify.get("/history", async (req, res) => {
2   const userId = req.query?.userId;
3   if (!userId) return res.send("Please Enter userId ...");
4
5   const page = req.query?.page;
6   const limit = req.query?.limit;
7   const offset = (page - 1) * limit;
8
9   const query_offset = ` 
10    SELECT date, amount, type, state
11    FROM wallet_account_tr
12   WHERE user_id = $1
13   AND state = 'Completed'
14  ORDER BY date
15  LIMIT $2 OFFSET $3
16 `;
17
18   const query = ` 
19    SELECT date, amount, type, state
20    FROM wallet_account_tr
21   WHERE user_id = $1
22   AND state = 'Completed'
23  order by date
24 `;
25
26   if (page && limit) {
27     try {
28       const result = await fastify.pg.query(query_offset, [userId, limit, offset]);
29       res.send(result.rows);
30     } catch (err) {
31       console.error("Database query error: ", err);
32       res.status(500).send("Internal Server Error");
33     }
34   } else {
35     try {
36       const result = await fastify.pg.query(query, [userId]);
37       res.send(result.rows);
38     } catch (err) {
39       console.error("Database query error: ", err);
40       res.status(500).send("Internal Server Error");
41     }
42   }
43 }
44 );

```

در این بخش که آخرین API است یک ریکوئست Get داریم که به دو صورت می‌تواند باشد، یکی اینکه فقط با Rیکوئست را بزنند و یکی این که علاوه بر آن به ما page و limit هم بدهد تا کوئری ای بزنیم که مناسب در صفحات وب است. کارکرد این API بدین صورت است:



می بینید که همه تراکنش های موفق آن کاربر به ترتیب تاریخ برای ما برگردانده شده است. حال اگر ریکوئست را برای Pagi nation بزنیم، به این صورت است:



```

\ fastify.listen({ port: 3000, host: "0.0.0.0" }, (err) => {
  if (err) throw err;
  console.log(`server listening on ${fastify.server.address().port}`);
});

```

در آخر این وب سرویس را روی پورت ۳۰۰۰ اجرا کرده و ریکوئست ها را با استفاده از postman می زنیم.

همچنین در وب سرویس ما تا جای ممکن ارور را هندل کردیم، به عنوان مثال اگر یوزری پیدا نمی‌شود ارور مخصوص میدهیم:

The screenshot shows a POST request to `http://localhost:3000/balance?userId=22`. The response status is 404 Not Found, with the message "User not found".

یا اینکه اگر یک سری از فیلدها خالی باشد موقع ریکوئست زدن، ما ارور مناسب برمی‌گردانیم:

The screenshot shows a POST request to `http://localhost:3000/charge`. The response status is 200 OK, but it includes an error message: "Please enter userId and amount ...".

همچنین در همه جا از `try` و `catch` استفاده کردیم تا ارورهای احتمالی دیتابیس باعث توقف سرویس ننشود.

NoSQL دیتابیس

در این بخش باید قسمت های مربوط به CRM را که تعدادی از آن ها را داخل Postgres پیاده سازی کرده بودیم، این بار داخل MongoDB پیاده سازی کنیم.

```
1 db.createCollection("trip_ticket", {
2   validator: {
3     $jsonSchema: {
4       bsonType: "object",
5       required: ["trip_reservation_id", "user_id", "examiner_id", "message", "type"],
6       properties: {
7         trip_reservation_id: {
8           bsonType: "int",
9           description: "must be an integer and is required"
10      },
11      company_id: {
12        bsonType: "int",
13        description: "must be an integer and is required"
14      },
15      user_id: {
16        bsonType: "int",
17        description: "must be an integer and is required"
18      },
19      examiner_id: {
20        bsonType: "int",
21        description: "must be an integer and is required"
22      },
23      message: {
24        bsonType: "string",
25        description: "must be a string and is required"
26      },
27      type: {
28        enum: ["text", "not satisfied"],
29        description: "can only be 'text' or 'not satisfied' and is required"
30      }
31    }
32  }
33 });
34
35
36 db.createCollection("hotel_ticket", {
37   validator: {
38     $jsonSchema: {
39       bsonType: "object",
40       required: ["hotel_reservation_id", "user_id", "examiner_id", "message", "type"],
41       properties: {
42         hotel_reservation_id: {
43           bsonType: "int",
44           description: "must be an integer and is required"
45      },
46         user_id: {
47           bsonType: "int",
48           description: "must be an integer and is required"
49      },
50         examiner_id: {
51           bsonType: "int",
52           description: "must be an integer and is required"
53      },
54         message: {
55           bsonType: "string",
56           description: "must be a string and is required"
57      },
58         type: {
59           enum: ["text", "cancel"],
60           description: "can only be 'text' or 'cancel' and is required"
61         }
62       }
63     }
64   }
65 });
```

```

95     }
96 });
97
98 db.createCollection("ticket_type_priority", {
99   validator: {
100     $jsonSchema: {
101       bsonType: "object",
102       required: ["type", "priority"],
103       properties: {
104         type: {
105           bsonType: "string",
106           description: "must be a string and is required"
107         },
108         priority: {
109           bsonType: "int",
110           description: "must be an integer and is required"
111         }
112       }
113     }
114   }
115 });
116
117 db.createCollection("phone_call_trip", {
118   validator: {
119     $jsonSchema: {
120       bsonType: "object",
121       required: ["caller_id", "callee_id", "ticket_id", "duration", "date", "time"],
122       properties: {
123         caller_id: {
124           bsonType: "int",
125           description: "must be an integer and is required"
126         },
127         callee_id: {
128           bsonType: "int",
129           description: "must be an integer and is required"
130         },
131         ticket_id: {
132           bsonType: "int",
133           description: "must be an integer and is required"
134         },
135         duration: {
136           bsonType: "string",
137           description: "must be a string and is required"
138         },
139         date: {
140           bsonType: "date",
141           description: "must be a date and is required"
142         },
143         time: {
144           bsonType: "string",
145           description: "must be a string and is required"
146         },
147         content: {
148           bsonType: "string",
149           description: "must be a string and is required"
150         }
151       }
152     }
153   }
154 });
155
156 db.createCollection("phone_call_hotel", {
157   validator: {
158     $jsonSchema: {
159       bsonType: "object",
160       required: ["caller_id", "callee_id", "ticket_id", "duration", "date", "time"],
161       properties: {
162         caller_id: {
163           bsonType: "int",
164           description: "must be an integer and is required"
165         },
166       }
167     }
168   }
169 });
170

```

```
136         callee_id: {
137             bsonType: "int",
138             description: "must be an integer and is required"
139         },
140         ticket_id: {
141             bsonType: "int",
142             description: "must be an integer and is required"
143         },
144         duration: {
145             bsonType: "string",
146             description: "must be a string and is required"
147         },
148         date: {
149             bsonType: "date",
150             description: "must be a date and is required"
151         },
152         time: {
153             bsonType: "string",
154             description: "must be a string and is required"
155         },
156         content: {
157             bsonType: "string",
158             description: "must be a string and is required"
159         }
160     }
161 }
162 });
163 );
```