T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER ENGINEERING

PAYMENT PLAN COMPANY
DATABASE & INTERFACE

CSE 414 DATABASES
TERM PROJECT REPORT

STUDENT
Şeyda Nur DEMİR
12 10 44 042
LECTURER
Yrd. Doç. Dr. Burcu YILMAZ
ASSISTANT
-

KOCAELİ, 2021

Project Details:

- Suppose that you have a company.
  You want to create the database of your company with user interfaces of it.
- Describe the user requirements of your company.
- Draw the E-R diyagram of the company.
- Do normalization.
- List the functional dependencies.
- List your tables.
- Create the database including everything (tables, relationships, keys etc.).
  You may use any database management system.
- Create the user interface of at least 5 modules using any programming language.
  Choose the interfaces according to the questions below.
  Create 1 outer right, 1 outer left, full outer querry on user interface.
- Create and use at least 5 different triggers making completely different tasks.
  Show how they are working in your interfaces.
- Create and use at least 5 different views that are completely different from each other.
  Show how they are working in your interface.

Additional:

- Give and create at least 3 atomic transactions.
- For at least 3 tasks solve concurrency of transactions using any techniques in the book.
- Add additional details about the database if there is any.
- Use inheritance for the tables.

Note:

- Your E-R diagram should be as complex as the E-R diagram of student managements system example in the book. Which means, it should be have many entities and relations.
- You have to prepare your project individually. This project is not a group Project.

Deadline:

- 06 Jun 2021 23:55

Submission:

- On Moodle

CONTENTS

1. COMPANY DEFINITON

The company brings together service providers and subscribers to reconcile bill payments.

The company ensures that service providers get paid for difficult customers. It also prevents subscribers from being penalized by paying later if they promise.

2. PROBLEM DEFINITION

Service providers may sometimes have problems collecting billing amount from subscribers. Every month they reflect a certain delay penalty due to late payment. Subscribers may not be able to pay. Both service providers should be able to charge on time and subscribers should be able to pay their amounts later without penalty. In this case, Payment Plan Corp. is mediating for both users.

To solve the problem, we need to define users and competencies. In the system, there must be an administrator, a staff member at a company branch, a service provider and a subscriber. Then, tables are created according to their competencies. The table stores the user id and time of the transaction. Table relationships are established. Triggers are added to tables based on usage. The interface is developed. Authentication is added. Authorization is set. Users are created. Test data is added and tested. Observe that queries and views are running.

3. PROBLEM SOLUTION APPROACHES

MySql version 10.5.9-MariaDB was used as RDBMS in this project. Node.js is preferred for backend services. It was developed with the MVC approach. Npm packages such as express, hbs, mysql are used in the project. User interface is included in the same project. Service communication is provided by JavaScript. Template engine named handlebar.js is used in views. Bootstrap 4 was used for the interface style.

4. USER REQUIREMENTS

First, all users log on to the system.

The admin registers users in the system. It manages system parameters, branches, and employees.

The employee serves the service providers in the town where he is located. Adds providers to the system.

The service provider displays its own subscribers that have been added in the system. Monitors debt status. Adds billing information. It gets paid into your wallet. Receives notifications about bills, subscribers, and payments.

The subscribers add their subscriptions. Displays your invoices. Adds a payment promise. Manages your payments. If he keeps his promise, he pays the bill without penalty.

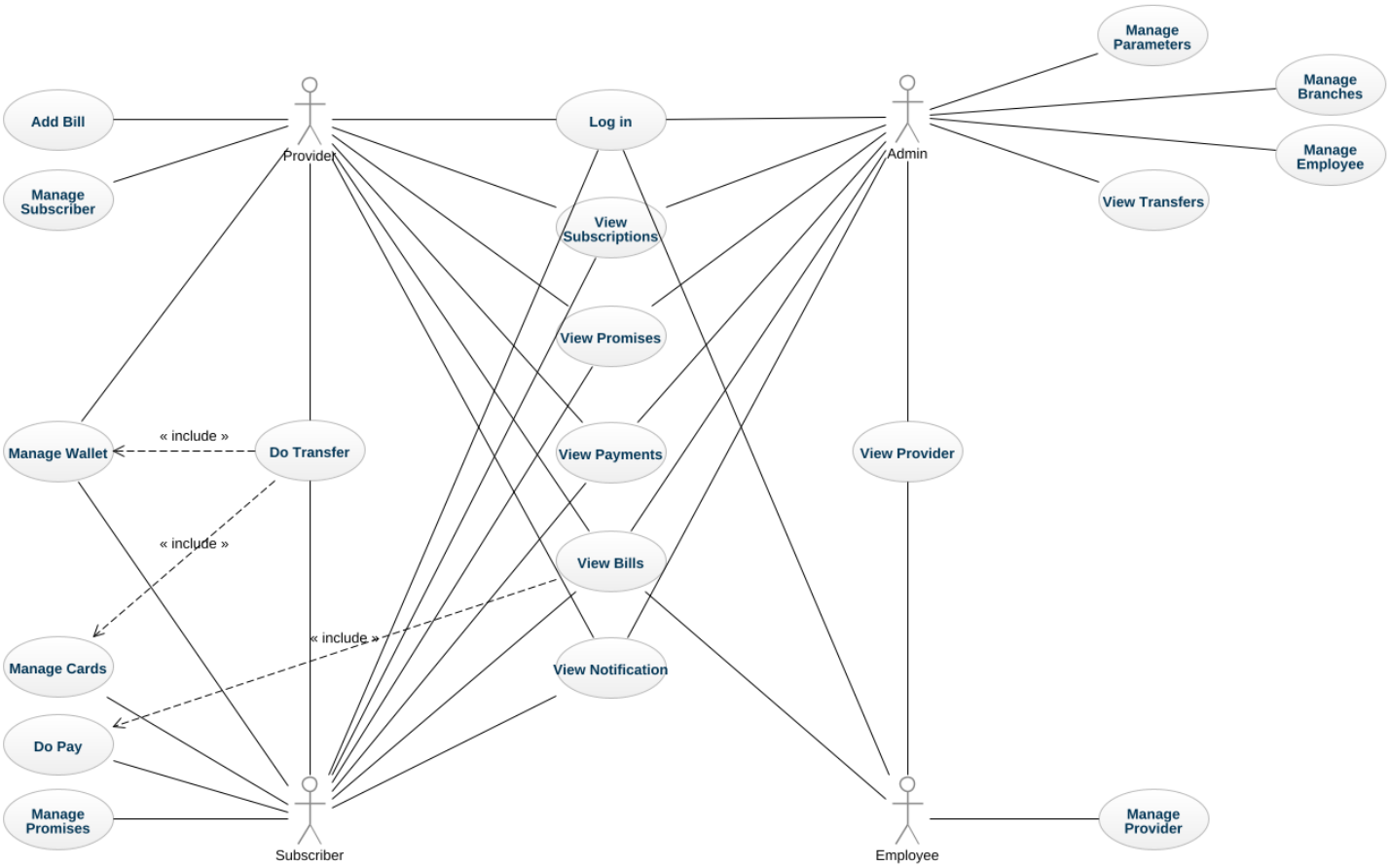
5. USERS OF THE SYSTEM

There are four user types in the system: admin, employee, service provider and subscriber.

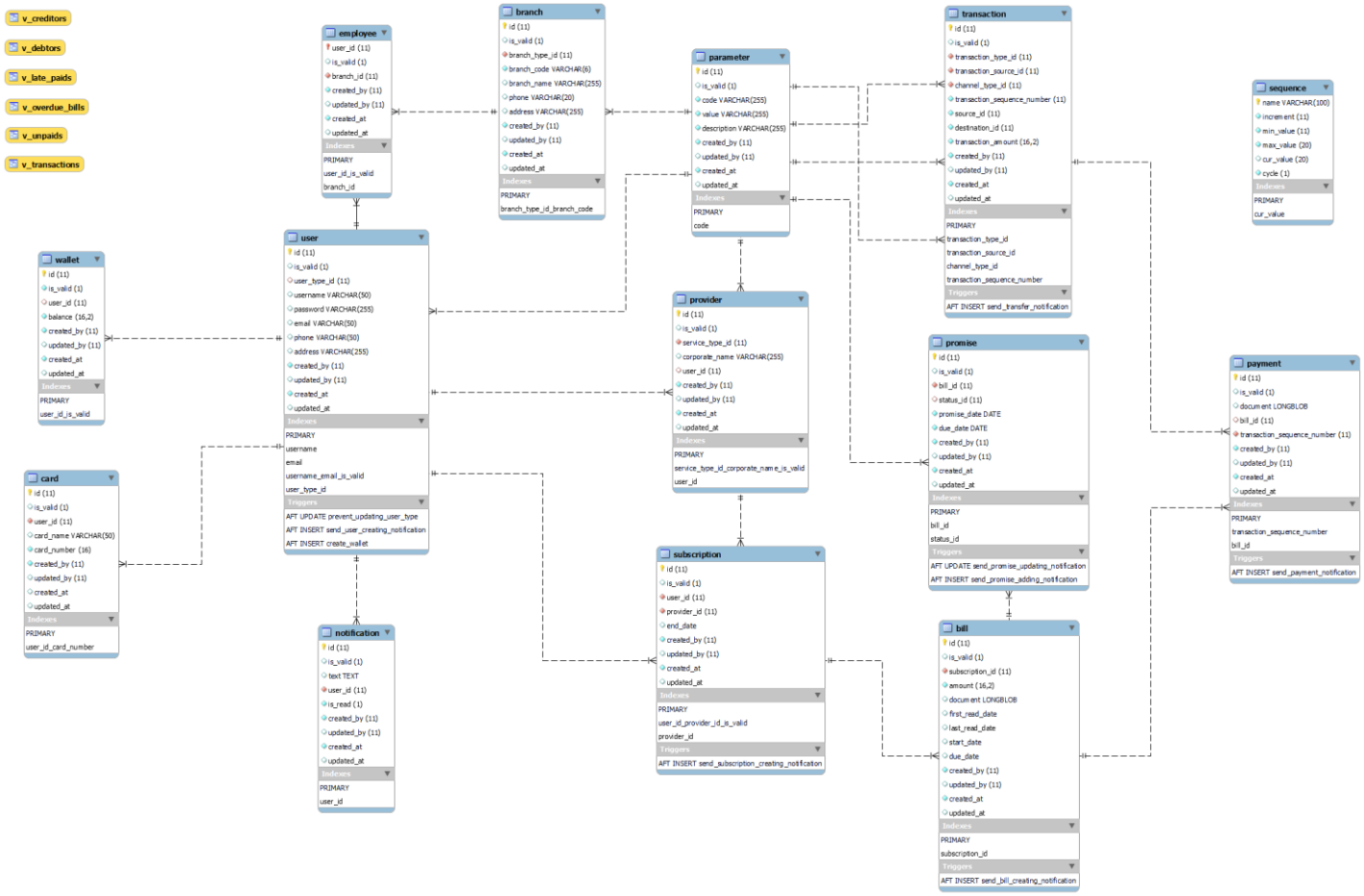
- The admin is the administrator of the system. Manages system parameters.
- The employee is the user who adds the service providers located in the city to the system.
- The service provider is the user in the system in order to receive late payment by entering the bills of the subscribers who cannot pay.
- The subscribers are users who can make late payments by promising to pay their bills.

6. DIAGRAMS

6.1. USE CASE DIAGRAM



6.2. ENTITY RELATIONSHIP DIAGRAM



7. NORMALIZATION

Normalization rules are applied.

8. FUNCTIONAL DEPENDENCIES

9. TABLES

9.1. Bill

It keeps the bills created by the service providers for the subscribers.

Columns

#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
1	**id**	int(11)			Hayır	*Yok*	AUTO_INCREMENT
2	**is_valid**	tinyint(1)			Evet	1	
3	**subscription_id**	int(11)			Hayır	*Yok*	
4	**amount**	decimal(16,2)			Hayır	*Yok*	
5	**document**	longblob			Evet	*NULL*	
6	**first_read_date**	datetime			Evet	*NULL*	
7	**last_read_date**	datetime			Evet	*NULL*	
8	**start_date**	datetime			Evet	*NULL*	
9	**due_date**	datetime			Evet	*NULL*	
10	**created_by**	int(11)			Hayır	*Yok*	
11	**updated_by**	int(11)			Evet	*NULL*	
12	**created_at**	datetime			Hayır	current_timestamp()	
13	**updated_at**	datetime			Evet	*NULL*	ON UPDATE CURRENT_TIMESTAMP()

id

It keeps the row number of the record. Also it is primary key.

is_valid

It keeps whether the record is valid. 1 = Active, 0 = Passive.

subscription_id

It keeps the subscription for which the record was created.

amount

It keeps the amount information.

document

It keeps the document such as image or file, if any.

first_read_date

It keeps the first reading date of the bill.

last_read_date

It keeps the last reading date of the bill.

start_date

It keeps the first payable date of the bill.

due_date

It keeps the due date of the bill.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```
CREATE TABLE `bill` (
  `id` int(11) NOT NULL,
  `is_valid` tinyint(1) DEFAULT 1,
  `subscription_id` int(11) NOT NULL,
  `amount` decimal(6,2) NOT NULL,
  `document` longblob NOT NULL,
  `first_read_date` datetime DEFAULT NULL,
  `last_read_date` datetime DEFAULT NULL,
  `start_date` datetime DEFAULT NULL,
  `due_date` datetime DEFAULT NULL,
  `created_by` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
  `updated_by` int(11) DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP()
) ;
```

```
ALTER TABLE `bill`
  ADD PRIMARY KEY (`id`),
  ADD INDEX `subscription_id` (`subscription_id` ASC),
  ADD CONSTRAINT `bill_ibfk_1`
    FOREIGN KEY (`subscription_id`)
    REFERENCES `paymentplandb`.`subscription` (`id`);
```

```
ALTER TABLE `bill`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;
```

9.2. Branch

It keeps the branches of the Payment Plan company.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**id**	int(11)			Hayır	*Yok*	AUTO_INCREMENT
☐	2	**is_valid**	tinyint(1)			Evet	1	
☐	3	**branch_type_id**	int(11)			Hayır	*Yok*	
☐	4	**branch_code**	varchar(6)			Hayır	*Yok*	
☐	5	**branch_name**	varchar(255)			Evet	*NULL*	
☐	6	**phone**	varchar(20)			Evet	*NULL*	
☐	7	**address**	varchar(255)			Evet	*NULL*	
☐	8	**created_by**	int(11)			Hayır	*Yok*	
☐	9	**updated_by**	int(11)			Evet	*NULL*	
☐	10	**created_at**	datetime			Hayır	current_timestamp()	
☐	11	**updated_at**	datetime			Evet	*NULL*	ON UPDATE CURRENT_TIMESTAMP()

id

It keeps the row number of the record. Also it is primary key.

is_valid

It keeps whether the record is valid. 1 = Active, 0 = Passive.

branch_type_id

It keeps the type of the branch.

branch_code

It keeps the unique code of the branch.

branch_name

It keeps the name of the branch.

phone

It keeps the phone number.

address

It keeps the address information.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```
CREATE TABLE `branch` (
  `id` int(11) NOT NULL,
  `is_valid` tinyint(1) DEFAULT 1,
  `branch_type_id` int(11) NOT NULL,
  `branch_code` varchar(6) COLLATE utf8_unicode_ci NOT NULL,
  `branch_name` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
  `phone` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,
  `address` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
  `created_by` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
  `updated_by` int(11) DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP()
) ;
```

```
INSERT INTO `branch` (`id`, `is_valid`, `branch_type_id`, `branch_code`, `br
anch_name`, `phone`, `address`, `created_by`, `updated_by`, `created_at`, `u
pdated_at`) VALUES
(1, 1, 19, 'PAYONL', 'Online', '', '', 1, NULL, '2021-05-
13 20:06:28', NULL),
(2, 1, 20, 'PAYIST', 'İstanbul', '', '', 1, NULL, '2021-05-
13 20:06:28', NULL),
(3, 1, 20, 'PAYANK', 'Ankara', '', '', 1, NULL, '2021-05-
13 20:06:28', NULL),
(4, 1, 20, 'PAYKOC', 'Kocaeli', '', '', 1, NULL, '2021-05-
13 20:06:28', NULL);
```

```
ALTER TABLE `branch`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE INDEX `branch_type_id_branch_code` (`branch_type_id` ASC, `bran
ch_code` ASC),
  ADD CONSTRAINT `branch_ibfk_1`
    FOREIGN KEY (`branch_type_id`)
    REFERENCES `paymentplandb`.`parameter` (`id`);
```

```
ALTER TABLE `branch`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;
```

9.3. Card

It keeps the credit cards of the subscriptions.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**id**	int(11)			Hayır	*Yok*	AUTO_INCREMENT
☐	2	**is_valid**	tinyint(1)			Evet	1	
☐	3	**user_id**	int(11)			Hayır	*Yok*	
☐	4	**card_name**	varchar(50)			Evet	*NULL*	
☐	5	**card_number**	bigint(16)			Hayır	*Yok*	
☐	6	**created_by**	int(11)			Hayır	*Yok*	
☐	7	**updated_by**	int(11)			Evet	*NULL*	
☐	8	**created_at**	datetime			Evet	current_timestamp()	
☐	9	**updated_at**	datetime			Evet	*NULL*	ON UPDATE CURRENT_TIMESTAMP()

id

It keeps the row number of the record. Also it is primary key.

is_valid

It keeps whether the record is valid. 1 = Active, 0 = Passive.

user_id

It keeps the user for which the record was created.

card_name

It keeps the card name.

card_number

It keeps the 16 digit number.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```sql
CREATE TABLE `card` (
  `id` int(11) NOT NULL,
  `is_valid` tinyint(1) DEFAULT 1,
  `user_id` int(11) NOT NULL,
  `card_name` varchar(50) COLLATE utf8_unicode_ci DEFAULT NULL,
  `card_number` int(16) NOT NULL,
  `created_by` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
  `updated_by` int(11) DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP()
) ;
```

```sql
ALTER TABLE `card`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE INDEX `user_id_card_number` (`user_id` ASC, `card_number` ASC),
  ADD CONSTRAINT `card_ibfk_1`
    FOREIGN KEY (`user_id`)
    REFERENCES `paymentplandb`.`user` (`id`);
```

```sql
ALTER TABLE `card`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;
```

9.4. Employee

It keeps the employees in branches. Also it is primary key.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**user_id**	int(11)			Hayır	*Yok*	
☐	2	**is_valid**	tinyint(1)			Evet	1	
☐	3	**branch_id**	int(11)			Hayır	*Yok*	
☐	4	**created_by**	int(11)			Hayır	*Yok*	
☐	5	**updated_by**	int(11)			Evet	*NULL*	
☐	6	**created_at**	datetime			Hayır	current_timestamp()	
☐	7	**updated_at**	datetime			Evet	*NULL*	ON UPDATE CURRENT_TIMESTAMP()

user_id

It keeps the user for which the record was created.

is_valid

It keeps whether the record is valid. 1 = Active, 0 = Passive.

branch_id

It keeps the branch for which the record was created.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```sql
CREATE TABLE `employee` (
  `user_id` int(11) NOT NULL,
  `is_valid` tinyint(1) DEFAULT 1,
  `branch_id` int(11) NOT NULL,
  `created_by` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
  `updated_by` int(11) DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP()
) ;
```

```sql
ALTER TABLE `employee`
  ADD PRIMARY KEY (`user_id`, `branch_id`),
  ADD CONSTRAINT `employee_ibfk_1`
    FOREIGN KEY (`user_id`)
    REFERENCES `paymentplandb`.`user` (`id`),
  ADD CONSTRAINT `employee_ibfk_2`
    FOREIGN KEY (`branch_id`)
    REFERENCES `paymentplandb`.`branch` (`id`);
```

9.5. Notification

It keeps the notifications created by the system after the transactions.

Columns

#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
1	**id**	int(11)			Hayır	*Yok*	AUTO_INCREMENT
2	**is_valid**	tinyint(1)			Evet	1	
3	**text**	text			Evet	*NULL*	
4	**user_id**	int(11)			Hayır	*Yok*	
5	**is_read**	tinyint(1)			Hayır	0	
6	**created_by**	int(11)			Hayır	*Yok*	
7	**updated_by**	int(11)			Evet	*NULL*	
8	**created_at**	datetime			Hayır	current_timestamp()	
9	**updated_at**	datetime			Evet	*NULL*	ON UPDATE CURRENT_TIMESTAMP()

id

It keeps the row number of the record. Also it is primary key.

is_valid

It keeps whether the record is valid. 1 = Active, 0 = Passive.

text

It keeps text to be displayed.

user_id

It keeps which user the notification is sent to.

is_read

It keeps whether the notification is shown or not.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```sql
CREATE TABLE `notification` (
  `id` int(11) NOT NULL,
  `is_valid` tinyint(1) DEFAULT 1,
  `text` text COLLATE utf8_unicode_ci DEFAULT NULL,
  `user_id` int(11) NOT NULL,
  `is_read` tinyint(1) NOT NULL DEFAULT 0,
  `created_by` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
  `updated_by` int(11) DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP()
) ;
```

```sql
ALTER TABLE `notification`
  ADD PRIMARY KEY (`id`),
  ADD INDEX `user_id` (`user_id` ASC),
  ADD CONSTRAINT `notification_ibfk_1`
    FOREIGN KEY (`user_id`)
    REFERENCES `paymentplandb`.`user` (`id`);
```

```sql
ALTER TABLE `notification`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;
```

9.6. Parameter

It keeps the parameters used for system.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**id**	int(11)			Hayır	*Yok*	AUTO_INCREMENT
☐	2	**is_valid**	tinyint(1)			Evet	1	
☐	3	**code**	varchar(255)			Hayır	*Yok*	
☐	4	**value**	varchar(255)			Hayır	*Yok*	
☐	5	**description**	varchar(255)			Hayır	*Yok*	
☐	6	**created_by**	int(11)			Hayır	*Yok*	
☐	7	**updated_by**	int(11)			Evet	*NULL*	
☐	8	**created_at**	datetime			Hayır	current_timestamp()	
☐	9	**updated_at**	datetime			Evet	*NULL*	ON UPDATE CURRENT_TIMESTAMP()

id

It keeps the row number of the record. Also it is primary key.

is_valid

It keeps whether the record is valid. 1 = Active, 0 = Passive.

code

It keeps the code information for grouping parameters.

value

It keeps a value that is used to separate parameters.

description

It keeps the description of the parameters.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```sql
CREATE TABLE `parameter` (
  `id` int(11) NOT NULL,
  `is_valid` tinyint(1) DEFAULT 1,
  `code` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `value` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `description` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `created_by` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
  `updated_by` int(11) DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP()
) ;
```

```sql
INSERT INTO `parameter` (`id`, `is_valid`, `code`, `value`, `description`, `
created_by`, `updated_by`, `created_at`, `updated_at`) VALUES
(1, 1, 'user_type', 'ADM', 'Administrator', 1, NULL, '2021-05-
12 23:16:57', NULL),
(2, 1, 'user_type', 'EMP', 'Employee', 1, NULL, '2021-05-
12 23:16:57', NULL),
(3, 1, 'user_type', 'PRO', 'Provider', 1, NULL, '2021-05-
12 23:16:57', NULL),
(4, 1, 'user_type', 'SUB', 'Subscriber', 1, NULL, '2021-05-
12 23:16:57', NULL),
(5, 1, 'subscriber_type', 'IND', 'Individual', 1, NULL, '2021-05-
12 23:16:57', NULL),
(6, 1, 'subscriber_type', 'BUS', 'Business', 1, NULL, '2021-05-
12 23:16:57', NULL),
(7, 1, 'service_type', 'ELC', 'Electric', 1, NULL, '2021-05-
12 23:16:57', NULL),
(8, 1, 'service_type', 'WAT', 'Water', 1, NULL, '2021-05-
12 23:16:57', NULL),
(9, 1, 'service_type', 'GAS', 'Gas', 1, NULL, '2021-05-12 23:16:57', NULL),
(10, 1, 'service_type', 'TEL', 'Telecommunication', 1, NULL, '2021-05-
12 23:16:57', NULL),
(11, 1, 'service_type', 'GSM', 'Cell', 1, NULL, '2021-05-
12 23:16:57', NULL),
(12, 1, 'service_type', 'TAX', 'Government', 1, NULL, '2021-05-
12 23:16:57', NULL),
(13, 1, 'promise_status', 'WAI', 'Waiting', 1, NULL, '2021-05-
24 11:14:33', NULL),
(14, 1, 'promise_status', 'SUC', 'Succeeded', 1, NULL, '2021-05-
12 23:16:57', NULL),
(15, 1, 'promise_status', 'REP', 'Repeated', 1, NULL, '2021-05-
12 23:16:57', NULL),
(16, 1, 'promise_status', 'CAN', 'Canceled', 1, NULL, '2021-05-
12 23:16:57', NULL),
(17, 1, 'promise_status', 'UNS', 'Unsucceeded', 1, NULL, '2021-05-
12 23:16:57', NULL),
```

```
(18, 1, 'channel_type', 'BRA', 'Branch', 1, NULL, '2021-05-
12 23:16:57', NULL),
(19, 1, 'channel_type', 'OTH', 'Other', 1, NULL, '2021-05-
12 23:16:57', NULL),
(20, 1, 'branch_type', 'ONL', 'Online', 1, NULL, '2021-05-
12 23:16:57', NULL),
(21, 1, 'branch_type', 'INS', 'In store', 1, NULL, '2021-05-
12 23:16:57', NULL),
(22, 1, 'transaction_type', 'CRE', 'Credit', 1, NULL, '2021-05-
12 23:16:57', NULL),
(23, 1, 'transaction_type', 'DEB', 'Debit', 1, NULL, '2021-05-
12 23:16:57', NULL),
(24, 1, 'transaction_source', 'INT', 'Internal', 1, NULL, '2021-05-
14 18:30:48', NULL),
(25, 1, 'transaction_source', 'EXT', 'External', 1, NULL, '2021-05-
12 23:16:57', NULL),
(26, 1, 'log_type', 'INS', 'Insert', 1, NULL, '2021-05-12 23:16:57', NULL),
(27, 1, 'log_type', 'UPD', 'Update', 1, NULL, '2021-05-12 23:16:57', NULL);
```

```
ALTER TABLE `parameter`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE INDEX `code` (`code` ASC, `value` ASC);
```

```
ALTER TABLE `parameter`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=28;
```

9.7. Payment

It keeps the bill payment information.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**id**	int(11)			Hayır	*Yok*	AUTO_INCREMENT
☐	2	**is_valid**	tinyint(1)			Evet	1	
☐	3	**document**	longblob			Evet	*NULL*	
☐	4	**bill_id**	int(11)			Evet	*NULL*	
☐	5	**transaction_sequence_number**	int(11)			Hayır	*Yok*	
☐	6	**created_by**	int(11)			Hayır	*Yok*	
☐	7	**updated_by**	int(11)			Evet	*NULL*	
☐	8	**created_at**	datetime			Hayır	current_timestamp()	
☐	9	**updated_at**	datetime			Evet	*NULL*	ON UPDATE CURRENT_TIMESTAMP()

id

It keeps the row number of the record. Also it is primary key.

is_valid

It keeps whether the record is valid. 1 = Active, 0 = Passive.

document

It keeps the document such as image or file, if any.

bill_id

It keeps the bill for which the record was created.

transaction_sequence_number

If any transaction is occured, database creates sequence_number for both of credit & debit.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```sql
CREATE TABLE `payment` (
  `id` int(11) NOT NULL,
  `is_valid` tinyint(1) DEFAULT 1,
  `document` longblob NOT NULL,
  `bill_id` int(11) DEFAULT NULL,
  `transaction_sequence_number` int(11) NOT NULL,
  `created_by` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
  `updated_by` int(11) DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP()
) ;
```

```sql
ALTER TABLE `payment`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE INDEX `transaction_sequence_number` (`transaction_sequence_numb
er` ASC)
  ADD CONSTRAINT `payment_ibfk_1`
    FOREIGN KEY (`transaction_sequence_number`)
    REFERENCES `paymentplandb`.`transaction` (`transaction_sequence_number`)
  ADD CONSTRAINT `payment_ibfk_2`
    FOREIGN KEY (`bill_id`)
    REFERENCES `paymentplandb`.`bill` (`id`);
```

```sql
ALTER TABLE `payment`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;
```

9.8. Promise

It keeps the payment promise information created by the subscriber for delayed bill payments.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**id**	int(11)			Hayır	*Yok*	AUTO_INCREMENT
☐	2	**is_valid**	tinyint(1)			Evet	1	
☐	3	**bill_id**	int(11)			Hayır	*Yok*	
☐	4	**status_id**	int(11)			Evet	13	
☐	5	**promise_date**	date			Hayır	current_timestamp()	
☐	6	**due_date**	date			Hayır	*Yok*	
☐	7	**created_by**	int(11)			Hayır	*Yok*	
☐	8	**updated_by**	int(11)			Evet	*NULL*	
☐	9	**created_at**	datetime			Hayır	current_timestamp()	
☐	10	**updated_at**	datetime			Evet	*NULL*	ON UPDATE CURRENT_TIMESTAMP()

id

It keeps the row number of the record. Also it is primary key.

is_valid

It keeps whether the record is valid. 1 = Active, 0 = Passive.

bill_id

It keeps the bill for which the record was created.

status_id

It keeps the status of the promise.

promise_date

It keeps the promised date.

due_date

If keeps the due date.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```sql
CREATE TABLE `promise` (
  `id` int(11) NOT NULL,
  `is_valid` tinyint(1) DEFAULT 1,
  `bill_id` int(11) NOT NULL,
  `status_id` int(11) NOT NULL,
  `promise_date` date NOT NULL,
  `due_date` date NOT NULL,
  `created_by` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
  `updated_by` int(11) DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP()
) ;
```

```sql
ALTER TABLE `promise`
  ADD PRIMARY KEY (`id`),
  ADD INDEX `bill_id` (`bill_id` ASC),
  ADD INDEX `status_id` (`status_id` ASC),
  ADD CONSTRAINT `promise_ibfk_1`
    FOREIGN KEY (`bill_id`)
    REFERENCES `paymentplandb`.`bill` (`id`),
  ADD CONSTRAINT `promise_ibfk_2`
    FOREIGN KEY (`status_id`)
    REFERENCES `paymentplandb`.`parameter` (`id`);
```

```sql
ALTER TABLE `promise`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;
```

9.9. Provider

It keeps the service providers.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**id**	int(11)			Hayır	*Yok*	AUTO_INCREMENT
☐	2	**is_valid**	tinyint(1)			Evet	1	
☐	3	**service_type_id**	int(11)			Hayır	*Yok*	
☐	4	**corporate_name**	varchar(255)			Evet	*NULL*	
☐	5	**user_id**	int(11)			Evet	*NULL*	
☐	6	**created_by**	int(11)			Hayır	*Yok*	
☐	7	**updated_by**	int(11)			Evet	*NULL*	
☐	8	**created_at**	datetime			Hayır	current_timestamp()	
☐	9	**updated_at**	datetime			Evet	*NULL*	ON UPDATE CURRENT_TIMESTAMP()

id

It keeps the row number of the record. Also it is primary key.

is_valid

It keeps whether the record is valid. 1 = Active, 0 = Passive.

service_type_id

It keeps the service type of the provider.

corporate_name

It keeps the corporate name of the provider.

user_id

It keeps the user for which the record was created.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```sql
CREATE TABLE `provider` (
  `id` int(11) NOT NULL,
  `is_valid` tinyint(1) DEFAULT 1,
  `user_id` int(11) DEFAULT NULL,
  `service_type_id` int(11) NOT NULL,
  `corporate_name` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
  `created_by` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
  `updated_by` int(11) DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP()
) ;
```

```sql
ALTER TABLE `provider`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE INDEX `service_type_id_corporate_name` (`service_type_id` ASC,
`corporate_name` ASC),
  ADD INDEX `user_id` (`user_id` ASC),
  ADD CONSTRAINT `provider_ibfk_1`
    FOREIGN KEY (`user_id`)
    REFERENCES `paymentplandb`.`user` (`id`),
  ADD CONSTRAINT `provider_ibfk_2`
    FOREIGN KEY (`service_type_id`)
    REFERENCES `paymentplandb`.`parameter` (`id`);
```

```sql
ALTER TABLE `provider`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;
```

9.10. Sequence

Generates unique numbers for each transaction.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**name**	varchar(100)			Hayır	*Yok*	
☐	2	**increment**	int(11)			Hayır	1	
☐	3	**min_value**	int(11)			Hayır	1	
☐	4	**max_value**	bigint(20)			Hayır	9223372036854775807	
☐	5	**cur_value**	bigint(20)			Evet	1	
☐	6	**cycle**	tinyint(1)			Hayır	0	

name

It keeps a unique name for the function that returns a value.. Also it is primary key.

increment

It keeps the increment value. We use as 1.

min_value

It keeps the initial value.

max_value

It keeps the maximum value that can be reached.

cur_value

It keeps the last generated value.

cycle

It is the flag field used to initialize after the maximum value. We don't use.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```sql
CREATE TABLE `sequence` (
  `name` varchar(100) COLLATE utf8_unicode_ci NOT NULL,
  `increment` int(11) NOT NULL DEFAULT 1,
  `min_value` int(11) NOT NULL DEFAULT 1,
  `max_value` bigint(20) DEFAULT 1,
  `cur_value` bigint(20) DEFAULT 1,
  `cycle` tinyint(1) NOT NULL DEFAULT 0
) ;
```

```sql
INSERT INTO `sequence` (`name`, `increment`, `min_value`, `max_value`, `cur_
value`, `cycle`) VALUES
('trn_seq', 1, 1000000, 10000000, 1000001, 0);
```

9.11. Subscription

It keeps the subscriptions.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**id**	int(11)			Hayır	*Yok*	AUTO_INCREMENT
☐	2	**is_valid**	tinyint(1)			Evet	1	
☐	3	**user_id**	int(11)			Hayır	*Yok*	
☐	4	**provider_id**	int(11)			Hayır	*Yok*	
☐	5	**end_date**	datetime			Evet	*NULL*	
☐	6	**created_by**	int(11)			Hayır	*Yok*	
☐	7	**updated_by**	int(11)			Evet	*NULL*	
☐	8	**created_at**	datetime			Hayır	current_timestamp()	
☐	9	**updated_at**	datetime			Evet	*NULL*	ON UPDATE CURRENT_TIMESTAMP()

id

It keeps the row number of the record. Also it is primary key.

is_valid

It keeps whether the record is valid. 1 = Active, 0 = Passive.

user_id

It keeps the user for which the record was created.

provider_id

It keeps the provider for which the record was created.

end_date

It keeps the date the subscription was terminated.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```sql
CREATE TABLE `subscription` (
  `id` int(11) NOT NULL,
  `is_valid` tinyint(1) DEFAULT 1,
  `user_id` int(11) NOT NULL,
  `provider_id` int(11) NOT NULL,
  `end_date` datetime NOT NULL,
  `created_by` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
  `updated_by` int(11) DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP()
) ;
```

```sql
ALTER TABLE `subscription`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE INDEX `user_id_provider_id` (`user_id` ASC, `provider_id` ASC),
  ADD INDEX `provider_id` (`provider_id` ASC),
  ADD CONSTRAINT `subscription_ibfk_1`
    FOREIGN KEY (`user_id`)
    REFERENCES `paymentplandb`.`user` (`id`),
  ADD CONSTRAINT `subscription_ibfk_2`
    FOREIGN KEY (`provider_id`)
    REFERENCES `paymentplandb`.`provider` (`id`);
```

```sql
ALTER TABLE `subscription`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;
```

9.12. Transaction

It keeps the all transactions such as balance loading from card to wallet (subscriber), transfer from wallet to wallet (subscriber => provider).

Columns

#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
1	**id**	int(11)			Hayır	*Yok*	AUTO_INCREMENT
2	**is_valid**	tinyint(1)			Evet	1	
3	**transaction_type_id**	int(11)			Hayır	*Yok*	
4	**transaction_source_id**	int(11)			Hayır	*Yok*	
5	**channel_type_id**	int(11)			Hayır	*Yok*	
6	**transaction_sequence_number**	int(11)			Hayır	*Yok*	
7	**source_id**	int(11)			Hayır	*Yok*	
8	**destination_id**	int(11)			Hayır	*Yok*	
9	**transaction_amount**	decimal(16,2)			Hayır	*Yok*	
10	**created_by**	int(11)			Hayır	*Yok*	
11	**updated_by**	int(11)			Evet	*NULL*	
12	**created_at**	datetime			Hayır	current_timestamp()	
13	**updated_at**	datetime			Evet	*NULL*	ON UPDATE CURRENT_TIMESTAMP()

id

It keeps the row number of the record. Also it is primary key.

is_valid

It keeps whether the record is valid. 1 = Active, 0 = Passive.

transaction_type_id

It keeps the transaction type (Credit, Debit).

transaction_source_id

It keeps the transaction source. (Internal, External)

channel_type_id

It keeps the channel type of the transaction. (Branch, Other)

transaction_sequence_number

If any transaction is occured, database creates sequence_number for both of credit & debit.

source_id

It keeps the source card or wallet id by transaction type and source.

destination_id

It keeps the destination card or wallet id by transaction type and source.

transaction_amount

It keeps the amount information.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```
CREATE TABLE `transaction` (
  `id` int(11) NOT NULL,
  `is_valid` tinyint(1) DEFAULT 1,
  `transaction_type_id` int(11) NOT NULL,
  `transaction_source_id` int(11) NOT NULL,
  `channel_type_id` int(11) NOT NULL,
  `transaction_sequence_number` int(11) NOT NULL,
  `source_id` int(11) NOT NULL,
  `destination_id` int(11) NOT NULL,
  `transaction_amount` decimal(16,2) NOT NULL,
  `created_by` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
  `updated_by` int(11) DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP()
) ;
```

```
ALTER TABLE `transaction`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;
```

9.13. User

It keeps four types of users in the system as admin, employee, provider, and subscriber.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**id**	int(11)			Hayır	*Yok*	AUTO_INCREMENT
☐	2	**is_valid**	tinyint(1)			Evet	1	
☐	3	**user_type_id**	int(11)			Evet	*NULL*	
☐	4	**username**	varchar(50)			Evet	*NULL*	
☐	5	**password**	varchar(255)			Evet	*NULL*	
☐	6	**email**	varchar(50)			Evet	*NULL*	
☐	7	**phone**	varchar(50)			Evet	*NULL*	
☐	8	**address**	varchar(255)			Evet	*NULL*	
☐	9	**created_by**	int(11)			Hayır	*Yok*	
☐	10	**updated_by**	int(11)			Evet	*NULL*	
☐	11	**created_at**	datetime			Hayır	current_timestamp()	
☐	12	**updated_at**	datetime			Evet	*NULL*	ON UPDATE CURRENT_TIMESTAMP()

id

It keeps the row number of the record. Also it is primary key.

is_valid

It keeps whether the record is valid. 1 = Active, 0 = Passive.

user_type_id

It keeps the type of user. (Admin, Employee, Provider, Subscriber)

username

It keeps the username to login.

password

It keeps the password to login.

email

It keeps the unique email to sign up.

phone

It keeps the phone number.

address

It keeps the address information.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```
CREATE TABLE `user` (
  `id` int(11) NOT NULL,
  `is_valid` tinyint(1) DEFAULT 1,
  `user_type_id` int(11) DEFAULT NULL,
  `username` varchar(50) COLLATE utf8_unicode_ci DEFAULT NULL,
  `password` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
  `email` varchar(50) COLLATE utf8_unicode_ci DEFAULT NULL,
  `phone` varchar(50) COLLATE utf8_unicode_ci DEFAULT NULL,
  `address` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
  `created_by` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
  `updated_by` int(11) DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP()
) ;
```

```
INSERT INTO `user` (`id`, `is_valid`, `user_type_id`, `username`, `password`
, `email`, `phone`, `address`, `created_by`, `updated_by`, `created_at`, `up
dated_at`) VALUES
(1, 1, 1, 'admin', '123456', 'admin@admin.com', '543-210-
1122', 'test', 1, NULL, '2021-05-13 20:06:50', NULL);
```

```
ALTER TABLE `user`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE INDEX `username` (`username` ASC),
  ADD UNIQUE INDEX `email` (`email` ASC),
  ADD UNIQUE INDEX `username_email` (`username` ASC, `email` ASC),
  ADD INDEX `user_type_id` (`user_type_id` ASC),
  ADD CONSTRAINT `user_ibfk_1`
    FOREIGN KEY (`user_type_id`)
    REFERENCES `paymentplandb`.`parameter` (`id`);
```

```
ALTER TABLE `user`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;
```

9.14. Wallet

It keeps the wallets of subscriber or provider users for payment transactions.

Columns

#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
1	**id**	int(11)			Hayır	*Yok*	AUTO_INCREMENT
2	**is_valid**	tinyint(1)			Evet	1	
3	**user_id**	int(11)			Hayır	*Yok*	
4	**card_name**	varchar(50)			Evet	*NULL*	
5	**card_number**	bigint(16)			Hayır	*Yok*	
6	**created_by**	int(11)			Hayır	*Yok*	
7	**updated_by**	int(11)			Evet	*NULL*	
8	**created_at**	datetime			Evet	current_timestamp()	
9	**updated_at**	datetime			Evet	*NULL*	ON UPDATE CURRENT_TIMESTAMP()

id

It keeps the row number of the record. Also it is primary key.

is_valid

It keeps whether the record is valid. 1 = Active, 0 = Passive.

user_id

It keeps the user for which the record was created.

balance

It keeps the balance of the wallet.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

SQL

```sql
CREATE TABLE `wallet` (
  `id` int(11) NOT NULL,
  `is_valid` tinyint(1) DEFAULT 1,
  `user_id` int(11) NOT NULL,
  `balance` decimal(6,2) DEFAULT NULL,
  `created_by` int(11) NOT NULL,
  `created_at` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP(),
  `updated_by` int(11) DEFAULT NULL,
  `updated_at` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP()
);
```

```sql
ALTER TABLE `wallet`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE INDEX `user_id` (`user_id` ASC),
  ADD CONSTRAINT `wallet_ibfk_1`
    FOREIGN KEY (`user_id`)
    REFERENCES `paymentplandb`.`user` (`id`);
```

```sql
ALTER TABLE `wallet`
  MODIFY `id` int(11) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;
```

10. TRIGGERS

10.1. create_wallet

It creates a wallet when a user is created as a provider or subscriber..

SQL

```
CREATE TRIGGER `create_wallet` AFTER INSERT ON `user` FOR EACH ROW BEGIN
    SET @sub_type_id := (SELECT id FROM parameter WHERE code = 'user_type' A
ND value = 'SUB');
    SET @prv_type_id := (SELECT id FROM parameter WHERE code = 'user_type' A
ND value = 'PRO');

    IF NEW.user_type_id = @sub_type_id OR NEW.user_type_id = @prv_type_id TH
EN
    INSERT INTO wallet(`user_id`, `created_by`) VALUES (NEW.id, NEW.created_
by);
    END IF;
END
```

10.2. prevent_updating_user_type

It prevents updating the user type.

SQL

```
CREATE TRIGGER `prevent_updating_user_type` AFTER UPDATE ON `user` FOR EACH
ROW BEGIN
  IF NEW.user_type_id != OLD.user_type_id THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Kullanıcı tipi değiştirileme
z!';
  END IF;
END
```

10.3. send_bill_creating_notification

It sends a notification for the created bill.

SQL

```
CREATE TRIGGER `send_bill_creating_notification` AFTER INSERT ON `bill` FOR
EACH ROW BEGIN
    SET @subscription_number := (SELECT CONCAT(LPAD(sub.provider_id, 4, '000
0'), LPAD(sub.user_id, 4, '0000')) FROM subscription AS sub WHERE sub.id = N
EW.subscription_id);
    SET @bill_number := (SELECT CONCAT('BILL', LPAD(NEW.subscription_id, 4,
'0000'), LPAD(NEW.id, 4, '0000')));
    SET @user_id := (SELECT user_id FROM subscription WHERE id = NEW.subscri
ption_id);
    SET @username := (SELECT username FROM user WHERE id = @user_id);

    INSERT INTO notification (text, user_id, created_by) VALUES (CONCAT(@sub
scription_number, ' numaralı aboneliğiniz için yeni faturanız oluşturuldu. F
atura No: ', @bill_number, ', Tutar: ', NEW.amount), @user_id, NEW.created_b
y);
END
```

10.4. send_payment_notification

It sends a notification for the paid bill.

SQL

```
CREATE TRIGGER `send_payment_notification` AFTER INSERT ON `payment` FOR EAC
H ROW BEGIN
    SET @bill_number := (SELECT CONCAT('BILL', LPAD(bill.subscription_id, 7,
 '0000000'), LPAD(bill.id, 4, '0000')) FROM bill WHERE id = NEW.bill_id);
    SET @subscriber_user_id := (SELECT sub.user_id FROM bill INNER JOIN subs
cription AS sub ON sub.id = bill.subscription_id WHERE bill.id = NEW.bill_id
);
    SET @provider_user_id := (SELECT provider.user_id FROM bill INNER JOIN s
ubscription AS sub ON sub.id = bill.subscription_id INNER JOIN provider ON p
rovider.id = sub.provider_id WHERE bill.id = NEW.bill_id);
    SET @succeeded_status_id := (SELECT id FROM parameter WHERE code = 'prom
ise_status' AND value = 'SUC');

    UPDATE promise SET status_id = @succeeded_status_id WHERE bill_id = NEW.
bill_id;

    INSERT INTO notification (text, user_id, created_by) VALUES (CONCAT(@bil
l_number, ' numaralı faturanızın ödemesi yapılmıştır. Teşekkür ederiz.'), @s
ubscriber_user_id, NEW.created_by);

    INSERT INTO notification (text, user_id, created_by) VALUES (CONCAT(@bil
l_number, ' numaralı faturanız abone tarafından ödenmiştir.'), @provider_use
r_id, NEW.created_by);
END
```

10.5. send_promise_adding_notification

It sends a detail notification for the promised bill.

SQL

```
CREATE TRIGGER `send_promise_adding_notification` AFTER INSERT ON `promise`
FOR EACH ROW BEGIN
    SET @bill_number := (SELECT CONCAT('BILL', LPAD(bill.subscription_id, 7,
 '0000000'), LPAD(bill.id, 4, '0000')) FROM bill WHERE id = NEW.bill_id);
    SET @user_id := (SELECT sub.user_id FROM bill INNER JOIN subscription AS
 sub ON sub.id = bill.subscription_id WHERE bill.id = NEW.bill_id);

    INSERT INTO notification (text, user_id, created_by) VALUES (CONCAT('Öde
me sözünüz eklenmiştir. Detaylar; Fatura No: ', @bill_number, ', Söz Verilen
 Ödeme Tarihi: ', NEW.due_date, ', Söz Verilen Tarihi: ', NEW.promise_date),
 @user_id, NEW.created_by);
END
```

10.6. send_promise_updating_notification

It sends a notification for the updated bill promise.

SQL

```
CREATE TRIGGER `send_promise_updating_notification` AFTER UPDATE ON `promise
` FOR EACH ROW BEGIN
    SET @bill_number := (SELECT CONCAT('BILL', LPAD(bill.subscription_id, 7,
 '0000000'), LPAD(bill.id, 4, '0000')) FROM bill WHERE id = NEW.bill_id);
    SET @user_id := (SELECT sub.user_id FROM bill INNER JOIN subscription AS
 sub ON sub.id = bill.subscription_id WHERE bill.id = NEW.bill_id);

    INSERT INTO notification (text, user_id, created_by) VALUES (CONCAT('Öde
me sözünüz güncellenmiştir. Detaylar; Fatura No: ', @bill_number, ', Söz Ver
ilen Ödeme Tarihi: ', NEW.due_date, ', Söz Verilen Tarihi: ', NEW.promise_da
te), @user_id, NEW.created_by);
END
```

10.7. send_subscription_creating_notification

It sends a detailed notification to both the provider and subscriber when a subscription is added.

SQL

```
CREATE TRIGGER `send_subscription_creating_notification` AFTER INSERT ON `su
bscription` FOR EACH ROW BEGIN
    SET @subscription_number := (SELECT CONCAT(LPAD(NEW.provider_id, 4, '000
0'), LPAD(NEW.user_id, 4, '0000')));
    SET @provider_user_id := (SELECT user_id FROM provider WHERE id = NEW.pr
ovider_id);
    SET @corporate_name := (SELECT corporate_name FROM provider WHERE id = N
EW.provider_id);
    SET @username := (SELECT username FROM user WHERE id = NEW.user_id);

    INSERT INTO notification (text, user_id, created_by) VALUES (CONCAT(@cor
porate_name, ' şirketi için aboneliğiniz eklenmiştir. Abone No: ', @subscrip
tion_number), NEW.user_id, NEW.created_by);
    INSERT INTO notification (text, user_id, created_by) VALUES (CONCAT(@use
rname, ' adlı kullanıcı aboneniz olarak eklenmiştir. Abone No: ', @subscript
ion_number), @provider_user_id, NEW.created_by);
END
```

10.8. send_transfer_notification

It sends a detailed notification to the receiver or sender according to transfer type.

SQL

```
CREATE TRIGGER `send_transfer_notification` AFTER INSERT ON `transaction` FO
R EACH ROW BEGIN
    SET @source := (SELECT CASE
    WHEN NEW.transaction_source_id = 24 AND NEW.transaction_type_id = 22 THE
N
      (SELECT LPAD(`sw`.`user_id`, 11, '10000000000') FROM `wallet` AS sw WH
ERE sw.id = NEW.source_id)
    WHEN NEW.transaction_source_id = 25 AND NEW.transaction_type_id = 22 THE
N
      (SELECT CONCAT(LEFT(sc.card_number, 4), '-
', SUBSTR(sc.card_number, 6, 2),'**-****-
**', RIGHT(sc.card_number, 4)) FROM `card` AS sc WHERE sc.id = NEW.source_id
)
  END);
    SET @destination := (SELECT CASE
    WHEN NEW.transaction_source_id = 24 AND NEW.transaction_type_id = 23 THE
N
      (SELECT LPAD(`dw`.`user_id`, 11, '10000000000') FROM `wallet` AS dw WH
ERE dw.id = NEW.destination_id)
    WHEN NEW.transaction_source_id = 25 AND NEW.transaction_type_id = 23 THE
N
      (SELECT CONCAT(LEFT(dc.card_number, 4), '-
', SUBSTR(dc.card_number, 6, 2),'**-****-
**', RIGHT(dc.card_number, 4)) FROM `card` AS dc WHERE dc.id = NEW.destinati
on_id)
  END);
    SET @source_user_id := (SELECT CASE
    WHEN NEW.transaction_source_id = 24 AND NEW.transaction_type_id = 22 THE
N
      (SELECT user_id FROM `wallet` AS sw WHERE sw.id = NEW.source_id)
    WHEN NEW.transaction_source_id = 25 AND NEW.transaction_type_id = 22 THE
N
      (SELECT user_id FROM `card` AS sc WHERE sc.id = NEW.source_id)
  END);
    SET @destination_user_id := (SELECT CASE
    WHEN NEW.transaction_source_id = 24 AND NEW.transaction_type_id = 23 THE
N
      (SELECT user_id FROM `wallet` AS dw WHERE dw.id = NEW.destination_id)
    WHEN NEW.transaction_source_id = 25 AND NEW.transaction_type_id = 23 THE
N
      (SELECT user_id FROM `card` AS dc WHERE dc.id = NEW.destination_id)
  END);
```

```
    IF NEW.transaction_type_id = 22 THEN
    INSERT INTO notification (text, user_id, created_by) VALUES (CONCAT(@sou
rce, ' cüzdan/kart-
ınızdan ', NEW.transaction_amount, ' ₺ tutarında para gönderilmiştir.'), @so
urce_user_id, NEW.created_by);
  ELSE IF NEW.transaction_type_id = 23 THEN
    INSERT INTO notification (text, user_id, created_by) VALUES (CONCAT(@des
tination, ' cüzdanınıza ', NEW.transaction_amount, ' ₺ tutarında para geldi.
'), @destination_user_id, NEW.created_by);
   END IF;
    END IF;
END
```

10.9. send_user_creating_notification

It sends a notification when a user signs up.

SQL

```
CREATE TRIGGER `send_user_creating_notification` AFTER INSERT ON `user` FOR
EACH ROW BEGIN
   INSERT INTO notification (text, user_id, created_by) VALUES (CONCAT('Sn.
', NEW.username, ', kullanıcı hesabınız oluşturulmuştur.'), NEW.id, NEW.crea
ted_by);
END
```

11. VIEWS

11.1. v_creditors

It returns creditor service providers & total amounts information from bills.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**provider_id**	int(11)			Hayır	*Yok*	
☐	2	**corporate_name**	varchar(255)			Evet	*NULL*	
☐	3	**total_count**	bigint(21)			Hayır	0	
☐	4	**total_amount**	decimal(38,2)			Evet	*NULL*	

provider_id

It keeps the provider's record id.

corporate_name

It keeps the corporate name of the provider.

total_count

It returns the total number of unpaid bills.

total_amount

It returns the total amount of unpaid bills.

SQL

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `v_creditors` AS
    SELECT
        `subscription`.`provider_id` AS `provider_id`,
        MAX(`provider`.`corporate_name`) AS `corporate_name`,
        COUNT(`bill`.`id`) AS `total_count`,
        SUM(`bill`.`amount`) AS `total_amount`
    FROM
        (((`bill`
        JOIN `subscription` ON (`subscription`.`id` = `bill`.`subscription_i
d`))
        JOIN `provider` ON (`provider`.`id` = `subscription`.`provider_id`))
        LEFT JOIN `payment` ON (`payment`.`bill_id` = `bill`.`id`))
    WHERE
```

```
        `payment`.`id` IS NULL
    GROUP BY `subscription`.`provider_id`;
```

11.2. v_debtors

It returns debtor subscribers & total amounts information from bills.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**user_id**	int(11)			Hayır	*Yok*	
☐	2	**username**	varchar(50)			Evet	*NULL*	
☐	3	**total_count**	bigint(21)			Hayır	0	
☐	4	**total_amount**	decimal(38,2)			Evet	*NULL*	

user_id

It keeps the user's record id.

username

It keeps the subscriber's username.

total_count

It returns the total number of unpaid bills.

total_amount

It returns the total amount of unpaid bills.

SQL

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `v_debtors` AS
    SELECT
        `subscription`.`user_id` AS `user_id`,
        MAX(`user`.`username`) AS `username`,
        COUNT(`bill`.`id`) AS `total_count`,
        SUM(`bill`.`amount`) AS `total_amount`
    FROM
        (((`bill`
        JOIN `subscription` ON (`subscription`.`id` = `bill`.`subscription_i
d`))
        JOIN `user` ON (`user`.`id` = `subscription`.`user_id`))
        LEFT JOIN `payment` ON (`payment`.`bill_id` = `bill`.`id`))
```

```
    WHERE
        `payment`.`id` IS NULL
    GROUP BY `subscription`.`user_id`;
```

11.3. v_late_paids

It returns late paid bill informations.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**payment_id**	int(11)			Hayır	0	
☐	2	**bill_id**	int(11)			Evet	*NULL*	
☐	3	**bill_number**	varchar(12)			Evet	*NULL*	
☐	4	**transaction_sequence_number**	int(11)			Hayır	*Yok*	
☐	5	**subscription_id**	int(11)			Hayır	*Yok*	
☐	6	**subscription_number**	varchar(8)			Evet	*NULL*	
☐	7	**provider_id**	int(11)			Hayır	*Yok*	
☐	8	**corporate_name**	varchar(255)			Evet	*NULL*	
☐	9	**transaction_amount**	decimal(16,2)			Hayır	*Yok*	

payment_id

It keeps the payment's record id.

bill_id

It keeps the bill's record id.

bill_number

It keeps the number of the bill to display.

transaction_sequence_number

It keeps the sequence number of the payment.

subscription_id

It keeps the subscription's record id.

subscription_number

It keeps the number of the subscription to display.

provider_id

It keeps the provider's record id.

corporate_name

It keeps the corporate name of the provider.

transaction_amount

It returns the total amount.

SQL

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `v_late_paids` AS
    SELECT
        `payment`.`id` AS `payment_id`,
        `payment`.`bill_id` AS `bill_id`,
        CONCAT('BILL',
                LPAD(`bill`.`subscription_id`, 4, '0000'),
                LPAD(`bill`.`id`, 4, '0000')) AS `bill_number`,
        `transaction`.`transaction_sequence_number` AS `transaction_sequence
_number`,
        `bill`.`subscription_id` AS `subscription_id`,
        CONCAT(LPAD(`sub`.`provider_id`, 4, '0000'),
                LPAD(`sub`.`user_id`, 4, '0000')) AS `subscription_number`,
        `sub`.`provider_id` AS `provider_id`,
        `provider`.`corporate_name` AS `corporate_name`,
        `transaction`.`transaction_amount` AS `transaction_amount`
    FROM
        ((((`payment`
        JOIN `transaction` ON (`transaction`.`transaction_sequence_number` =
 `payment`.`transaction_sequence_number`))
        JOIN `bill` ON (`bill`.`id` = `payment`.`bill_id`))
        JOIN `subscription` `sub` ON (`bill`.`subscription_id` = `sub`.`id`)
)
        JOIN `provider` ON (`provider`.`id` = `sub`.`provider_id`))
    WHERE
        `bill`.`due_date` < `transaction`.`created_at`
            AND `payment`.`is_valid` = 1
    GROUP BY `payment`.`bill_id`;
```

11.4. v_overdue_bills

It returns overdue unpaid bill informations.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**bill_id**	int(11)			Hayır	0	
☐	2	**bill_number**	varchar(12)			Evet	*NULL*	
☐	3	**first_read_date**	datetime			Evet	*NULL*	
☐	4	**last_read_date**	datetime			Evet	*NULL*	
☐	5	**start_date**	datetime			Evet	*NULL*	
☐	6	**due_date**	datetime			Evet	*NULL*	
☐	7	**subscription_id**	int(11)			Hayır	*Yok*	
☐	8	**subscription_number**	varchar(8)			Evet	*NULL*	
☐	9	**provider_id**	int(11)			Hayır	*Yok*	
☐	10	**corporate_name**	varchar(255)			Evet	*NULL*	
☐	11	**user_id**	int(11)			Hayır	*Yok*	
☐	12	**username**	varchar(50)			Evet	*NULL*	
☐	13	**bill_amount**	decimal(16,2)			Hayır	*Yok*	

bill_id

It keeps the bill's record id.

bill_number

It keeps the number of the bill to display.

first_read_date

It keeps the first reading date of the bill.

last_read_date

It keeps the last reading date of the bill.

start_date

It keeps the first payable date of the bill.

due_date

It keeps the due date of the bill.

subscription_id

It keeps the subscription's record id.

subscription_number

It keeps the number of the subscription to display.

provider_id

It keeps the provider's record id.

corporate_name

It keeps the corporate name of the provider.

user_id

It keeps the user's record id.

username

It keeps the subscriber's username.

bill_amount

It returns the total amount of unpaid bills.

SQL

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW  `v_overdue_bills` AS
SELECT
   `bill`.`id` AS `bill_id`,
   CONCAT('BILL', LPAD(`bill`.`subscription_id`, 4, '0000'), LPAD(`bill`.`id
`, 4, '0000')) AS `bill_number`,
   `bill`.`first_read_date` AS `first_read_date`,
   `bill`.`last_read_date` AS `last_read_date`,
   `bill`.`start_date` AS `start_date`,
   `bill`.`due_date` AS `due_date`,
   `bill`.`subscription_id` AS `subscription_id`,
   CONCAT(LPAD(`sub`.`provider_id`, 4, '0000'), LPAD(`sub`.`user_id`, 4, '00
00')) AS `subscription_number`,
   `sub`.`provider_id` AS `provider_id`,
   `provider`.`corporate_name` AS `corporate_name`,
   `sub`.`user_id` AS `user_id`,
   `USER`.`username` AS `username`,
   `bill`.`amount` AS `bill_amount`
FROM
   ((((`bill`
      JOIN `subscription` `sub` ON(`sub`.`id` = `bill`.`subscription_id`))
      JOIN `provider` ON(`provider`.`id` = `sub`.`provider_id`))
      JOIN `user` ON(`user`.`id` = `sub`.`user_id`))
      LEFT JOIN `payment` ON(`payment`.`bill_id` = `bill`.`id`))
WHERE `payment`.`id` IS NULL AND `bill`.`due_date` < CURRENT_TIMESTAMP() ;
```

11.5. v_transactions

It returns a single record, including the reverse operation by groups sequence number.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**transaction_sequence_number**	int(11)			Hayır	*Yok*	
☐	2	**transaction_amount**	decimal(16,2)			Hayır	*Yok*	
☐	3	**channel_type_id**	int(11)			Hayır	*Yok*	
☐	4	**source_id**	int(11)			Hayır	*Yok*	
☐	5	**source**	varchar(19)			Evet	*NULL*	
☐	6	**destination_id**	int(11)			Hayır	*Yok*	
☐	7	**destination**	varchar(19)			Evet	*NULL*	
☐	8	**created_by**	int(11)			Hayır	*Yok*	
☐	9	**updated_by**	int(11)			Evet	*NULL*	
☐	10	**created_at**	datetime			Hayır	current_timestamp()	
☐	11	**updated_at**	datetime			Evet	*NULL*	
☐	12	**row_num**	bigint(21)			Hayır	0	

transaction_sequence_number

It keeps the sequence number of the duo-transactions.

transaction_amount

It keeps the amount information.

channel_type_id

It keeps the channel type of the transaction. (Branch, Other)

source_id

It keeps the source card or wallet record id by transaction type and source.

source

It keeps the source card or wallet display text by transaction type and source.

destination_id

It keeps the destination card or wallet record id by transaction type and source.

destination

It keeps the destination card or wallet display text by transaction type and source.

created_by

It keeps which user the record was created by.

updated_by

It keeps which user the record was updated by.

created_at

It keeps when user the record was created.

updated_at

It keeps when user the record was updated.

row_num

It keeps the row numbers.

SQL

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW  `v_transactions` AS
  SELECT   `t1`.`transaction_sequence_number` AS `transaction_sequence_numbe
r`,
           `t1`.`transaction_amount`          AS `transaction_amount`,
           `t1`.`channel_type_id`             AS `channel_type_id`,
           `t1`.`source_id`                   AS `source_id`,
           CASE
                    WHEN `t1`.`transaction_source_id` = 24
                    AND      `t1`.`transaction_type_id` = 22 THEN
                             (
                                    SELECT LPAD(`sw`.`user_id`,11,'100000000
00')
                                    FROM   `wallet` `sw`
                                    WHERE  `sw`.`id` = `t1`.`source_id`)
                    WHEN `t1`.`transaction_source_id` = 25
                    AND      `t1`.`transaction_type_id` = 22 THEN
                             (
                                    SELECT CONCAT(LEFT(`sc`.`card_number`,4)
,'-****-****-**',RIGHT(`sc`.`card_number`,2))
                                    FROM   `card` `sc`
                                    WHERE  `sc`.`id` = `t1`.`source_id`)
           END                   AS `source`,
           `t2`.`destination_id` AS `destination_id`,
           CASE
                    WHEN `t2`.`transaction_source_id` = 24
                    AND      `t2`.`transaction_type_id` = 23 THEN
                             (
                                    SELECT LPAD(`dw`.`user_id`,11,'100000000
00')
                                    FROM   `wallet` `dw`
```

```
                                        WHERE  `dw`.`id` = `t2`.`destination_id`
)
                    WHEN `t2`.`transaction_source_id` = 25
                    AND      `t2`.`transaction_type_id` = 23 THEN
                             (
                                        SELECT CONCAT(LEFT(`dc`.`card_number`,4)
,'-****-****-**',RIGHT(`dc`.`card_number`,2))
                                        FROM   `card` `dc`
                                        WHERE  `dc`.`id` = `t2`.`destination_id`
)
            END
    AS `destination`,
            `t1`.`created_by`
    AS `created_by`,
            `t1`.`updated_by`
    AS `updated_by`,
            `t1`.`created_at`
    AS `created_at`,
            `t1`.`updated_at`
    AS `updated_at`,
            row_number() OVER (PARTITION BY `t1`.`transaction_sequence_number
`) AS `row_num`
  FROM      (`transaction` `t1`
  JOIN      `transaction` `t2`)
  WHERE     `t1`.`transaction_sequence_number` = `t2`.`transaction_sequence_n
umber`
  AND       `t1`.`id` <> `t2`.`id`
  GROUP BY `t1`.`transaction_sequence_number`
  HAVING    `source` IS NOT NULL;
```

11.6. v_unpaids

It returns unpaid bill information.

Columns

	#	Adı	Türü	Karşılaştırma	Öznitelikler	Boş	Varsayılan	Ekstra
☐	1	**bill_id**	int(11)			Hayır	0	
☐	2	**bill_number**	varchar(12)			Evet	*NULL*	
☐	3	**first_read_date**	date			Evet	*NULL*	
☐	4	**last_read_date**	date			Evet	*NULL*	
☐	5	**start_date**	date			Evet	*NULL*	
☐	6	**due_date**	date			Evet	*NULL*	
☐	7	**subscription_id**	int(11)			Hayır	*Yok*	
☐	8	**subscription_number**	varchar(8)			Evet	*NULL*	
☐	9	**provider_id**	int(11)			Hayır	*Yok*	
☐	10	**corporate_name**	varchar(255)			Evet	*NULL*	
☐	11	**user_id**	int(11)			Hayır	*Yok*	
☐	12	**username**	varchar(50)			Evet	*NULL*	
☐	13	**bill_amount**	decimal(16,2)			Hayır	*Yok*	

bill_id

It keeps the bill's record id.

bill_number

It keeps the number of the bill to display.

first_read_date

It keeps the first reading date of the bill.

last_read_date

It keeps the last reading date of the bill.

start_date

It keeps the first payable date of the bill.

due_date

It keeps the due date of the bill.

subscription_id

It keeps the subscription's record id.

subscription_number

It keeps the number of the subscription to display.

provider_id

It keeps the provider's record id.

corporate_name

It keeps the corporate name of the provider.

user_id

It keeps the user's record id.

username

It keeps the subscriber's username.

bill_amount

It returns the total amount of unpaid bills.

SQL

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `v_unpaids` AS
    SELECT
        `bill`.`id` AS `bill_id`,
        CONCAT('BILL',
                LPAD(`bill`.`subscription_id`, 4, '0000'),
                LPAD(`bill`.`id`, 4, '0000')) AS `bill_number`,
        CAST(`bill`.`first_read_date` AS DATE) AS `first_read_date`,
        CAST(`bill`.`last_read_date` AS DATE) AS `last_read_date`,
        CAST(`bill`.`start_date` AS DATE) AS `start_date`,
        CAST(`bill`.`due_date` AS DATE) AS `due_date`,
        `bill`.`subscription_id` AS `subscription_id`,
        CONCAT(LPAD(`sub`.`provider_id`, 4, '0000'),
                LPAD(`sub`.`user_id`, 4, '0000')) AS `subscription_number`,
        `sub`.`provider_id` AS `provider_id`,
        `provider`.`corporate_name` AS `corporate_name`,
        `sub`.`user_id` AS `user_id`,
        `user`.`username` AS `username`,
        `bill`.`amount` AS `bill_amount`
    FROM
        ((((`bill`
        JOIN `subscription` `sub` ON (`bill`.`subscription_id` = `sub`.`id`)
)
        JOIN `provider` ON (`provider`.`id` = `sub`.`provider_id`))
        JOIN `user` ON (`user`.`id` = `sub`.`user_id`))
        LEFT JOIN `payment` ON (`bill`.`id` = `payment`.`bill_id`))
    WHERE `payment`.`id` IS NULL;
```

12. FUNCTIONS

12.1. next_value

It returns creditor service providers & total amounts information from bills.

Parameters

Yordam adı	next_value			
Türü	FUNCTION			
Parametreler	**Adı**	**Türü**	**Uzunluk/Değerler**	**Seçenekler**
	seq_name	VARCHAR	100	Karakter
	Kaldır			
	Parametre ekle			
Dönüş türü	BIGINT			
Dönüş uzunluğu/değerl				
Dönüş seçenekleri				

seq_name

For the sequence name to get the next value.

SQL

```
CREATE DEFINER=`root`@`localhost` FUNCTION `next_value` (`seq_name` VARCHAR(
100)) RETURNS BIGINT(20)
BEGIN
    DECLARE cur_val BIGINT;

    SELECT cur_value INTO cur_val FROM sequence WHERE name = seq_name;

    IF cur_val IS NOT NULL THEN
        UPDATE
            sequence
        SET
            cur_value = IF (
                (cur_value + increment) > max_value OR (cur_value + incremen
t) < min_value,
                IF (cycle = TRUE,
                    IF ( (cur_value + increment) > max_value,
                        min_value,
                        max_value
                    ),
                    NULL
                ),
```

```
                cur_value + increment
            )
        WHERE
            name = seq_name;
    END IF;
    RETURN cur_val;
END$$
```

Usage

```
await db.db("SELECT next_value('trn_seq') AS nextval;");
```

Table values after called

name	increment	min_value	max_value	cur_value	cycle
trn_seq	1	1000000	10000000	1000003	0

13. STORED PROSEDURES

13.1. create_transaction

It provides balance update of a transaction by type.

Parameters

Yordam adı	create_transaction					
Türü	PROCEDURE					
Parametreler	**Yön**	**Adı**	**Türü**	**Uzunluk/Değerler**	**Seçenekler**	
	IN	transaction_type_id	INT			Kaldır
	IN	transaction_source_ic	INT			Kaldır
	IN	channel_type_id	INT			Kaldır
	IN	source_id	INT			Kaldır
	IN	destination_id	INT			Kaldır
	IN	transaction_amount	DECIMAL	6,2		Kaldır
	IN	created_by	INT			Kaldır
	IN	transaction_sequence	INT			Kaldır
	OUT	result_val	TINYINT	1		Kaldır

transaction_type_id

For transaction type (Credit, Debit).

transaction_source_id

For transaction source. (Internal, External)

channel_type_id

For channel type of the transaction. (Branch, Other)

source_id

It is the source card or wallet id by transaction type and source.

destination_id

It is the destination card or wallet id by transaction type and source.

transaction_amount

For transfer amount.

transaction_sequence_number

It is the same sequence number to differentiate a transaction and its reverse transaction from others.

result_val

It is the output value of the operation. Returns 1 if no error occurs.

SQL

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `create_transaction` (IN `transa
ction_type_id` INT, IN `transaction_source_id` INT, IN `channel_type_id` INT
, IN `source_id` INT, IN `destination_id` INT, IN `transaction_amount` DECIM
AL(6,2), IN `created_by` INT, IN `transaction_sequence_number` INT, OUT `res
ult_val` TINYINT(1))
BEGIN
  DECLARE transaction_type VARCHAR(255);
  DECLARE transaction_source VARCHAR(255);
  DECLARE source_balance VARCHAR(255);
  DECLARE source_wallet_id INT;
  DECLARE credit_type_id INT;
  DECLARE debit_type_id INT;

  SELECT value INTO transaction_type FROM parameter WHERE id = transaction_t
ype_id LIMIT 1;
  SELECT value INTO transaction_source FROM parameter WHERE id = transaction
_source_id LIMIT 1;

  SELECT id INTO credit_type_id FROM parameter WHERE code = 'transaction_typ
e' AND value = 'CRE';
  SELECT id INTO debit_type_id FROM parameter WHERE code = 'transaction_type
' AND value = 'DEB';

  IF transaction_type = 'CRE' THEN -- Para çekme
    IF transaction_source = 'INT' THEN -
- İç kanal üzerinden yapılan transfer cüzdanlar arasında yapılabilir
      SELECT COALESCE(balance, 0) INTO source_balance FROM wallet WHERE id =
 source_id AND is_valid = 1;
      IF source_balance < transaction_amount THEN -- Yeterli bakiye kontrolü
        SET result_val = -1;
      END IF;

      UPDATE wallet SET balance = balance
- transaction_amount WHERE id = source_id;
    -- ELSE -- Dış kanal transferler karttan yapılır (bakiyeye bakamayız)
    END IF;

    INSERT INTO transaction (`transaction_type_id`, `transaction_source_id`,
 `channel_type_id`, `transaction_sequence_number`, `source_id`, `destination
_id`, `transaction_amount`, `created_by`, `created_at`)
    VALUES (credit_type_id, transaction_source_id, channel_type_id, transact
ion_sequence_number, source_id, destination_id, transaction_amount, created_
by, CURRENT_TIMESTAMP());
    ELSE -- Para yatırma
    IF transaction_source = 'INT' THEN
      UPDATE wallet SET balance = balance +
transaction_amount WHERE id = destination_id;
```

```
    END IF;

    INSERT INTO transaction (`transaction_type_id`, `transaction_source_id`,
 `channel_type_id`, `transaction_sequence_number`, `source_id`, `destination
_id`, `transaction_amount`, `created_by`, `created_at`)
    VALUES (debit_type_id, transaction_source_id, channel_type_id, transacti
on_sequence_number, source_id, destination_id, transaction_amount, created_b
y, CURRENT_TIMESTAMP());
  END IF;

    SET result_val = 1;
END$$
```

14. ATOMIC TRANSACTIONS

Money transfers must be atomic transactions. It is managed on the code side, as it is a reverse operation. The code first calls the ***create_transaction*** stored procedure for the first transaction and does not commit until the second transaction is finished. If the first operation is successful, the procedure is called a second time for the reverse transaction and commits if there is no error. It rollbacks in case of any error.

Therefore, there is no commit and rollback code in the ***create_transaction*** procedure. It provides *BEGIN* operation with *db.startTransaction()* method and defaults to auto_commit = 0. It will be commit or rollback depending on its status at the end of the transaction.

Code

```
// Begin transaction
    let transaction = await db.startTransaction();
    if (!transaction) {
        res.resultError("Bir hata oluştu!");
        return;
    }

    try {
        let credit_id = constants.transactionTypes[0].id;
        let debit_id = constants.transactionTypes[1].id;

        if ([credit_id, debit_id].find(x => x == req.body.transaction_type_i
d) == null) {
            res.resultError("Transfer tipi belirlenemedi!");
            return;
        }

        // Transaction Sequence Number
        let seq_num_results = await db.db("SELECT next_value('trn_seq') AS n
extval;");
        let seq_number = seq_num_results[0].nextval;

        // First Transaction
        await db.query(`CALL create_transaction (?, ?, ?, ?, ?, ?, ?, ?, @re
sult_val);`,
            [req.body.transaction_type_id == credit_id ? credit_id : debit_i
d,
            req.body.transaction_source_id,
            req.body.channel_type_id,
            req.body.source_id,
            req.body.destination_id,
            req.body.transaction_amount,
            req.auth_id,
                seq_number], function (err, results) {
```

```
                    if (err) throw err;

                    db.query("SELECT @result_val AS result;", function (errC
rd, resultsCrd) {
                        if (errCrd) throw errCrd;
                        if (resultsCrd[0].result < 1) {
                            if (err) throw err;
                            res.resultError("Limit yetersiz!");
                            return;
                        }
                    });

                    console.log("First Transaction: ", results);
                });

        // Reverse Transaction
        await db.query("CALL create_transaction (?, ?, ?, ?, ?, ?, ?, ?, @re
sult_val);",
            [req.body.transaction_type_id == credit_id ? debit_id : credit_i
d,
            req.body.transaction_source_id,
            req.body.channel_type_id,
            req.body.source_id,
            req.body.destination_id,
            req.body.transaction_amount,
            req.auth_id,
                seq_number], function (err, results) {
                    if (err) throw err;

                    console.log("Second Transaction: ", results);
                });
    } catch (e) {
        // Rollback
        await db.rollbackTransaction();
        console.log("Hata: ", String(e));
    }
    // Commit
    await db.commitTransaction();
```

15. TASKS

15.1. check_payments

Controls the payment of bills at the end of the day. It looks at the due date of the invoice and for which date if promised. If there is an delayed payment, it will applies a penalty on the bill. It also updates its status as unsucceeded for the promise.

SQL

```
CREATE EVENT check_payments
ON SCHEDULE EVERY 24 HOUR STARTS
DO
BEGIN
    SET @unsucceeded_id :=
      (SELECT id
       FROM PARAMETER
       WHERE code = 'promise_status'
         AND value = 'UNS');

    DECLARE cr_promises
    CURSOR
    FOR
    SELECT `o`.`bill_id`, COALESCE(`promise`.`id`, 0) AS `promise_id`
    FROM `v_overdue_bills` AS `o`
    LEFT JOIN `promise` ON `promise`.`bill_id` = `o`.`bill_id`
    WHERE (`promise`.`id` IS NULL
          AND `o`.`due_date` < CURRENT_TIMESTAMP())
      OR (`promise`.`id` IS NOT NULL
          AND `promise`.`due_date` < CURRENT_TIMESTAMP());

    OPEN cr_promises FETCH NEXT
    FROM cr_promises INTO @bill_id, @promise_id

    WHILE @@FETCH_STATUS = 0
        IF @promise_id > 0 THEN

            UPDATE `promise`
            SET `status_id` = @unsucceeded_id,
                `updated_by` = 1,
                `updated_at` = CURRENT_TIMESTAMP()
            WHERE `promise`.`id` = @promise_id;

        END IF;

        UPDATE `bill`
        SET `amount` += 0.1,
                        `updated_by` = 1,
```

```
                            `updated_at` = CURRENT_TIMESTAMP()
        WHERE `bill`.`id` = @bill_id;

    FETCH NEXT FROM cr_promises
    CLOSE cr_promises
    DEALLOCATE cr_promises
END
```

16. QUERIES

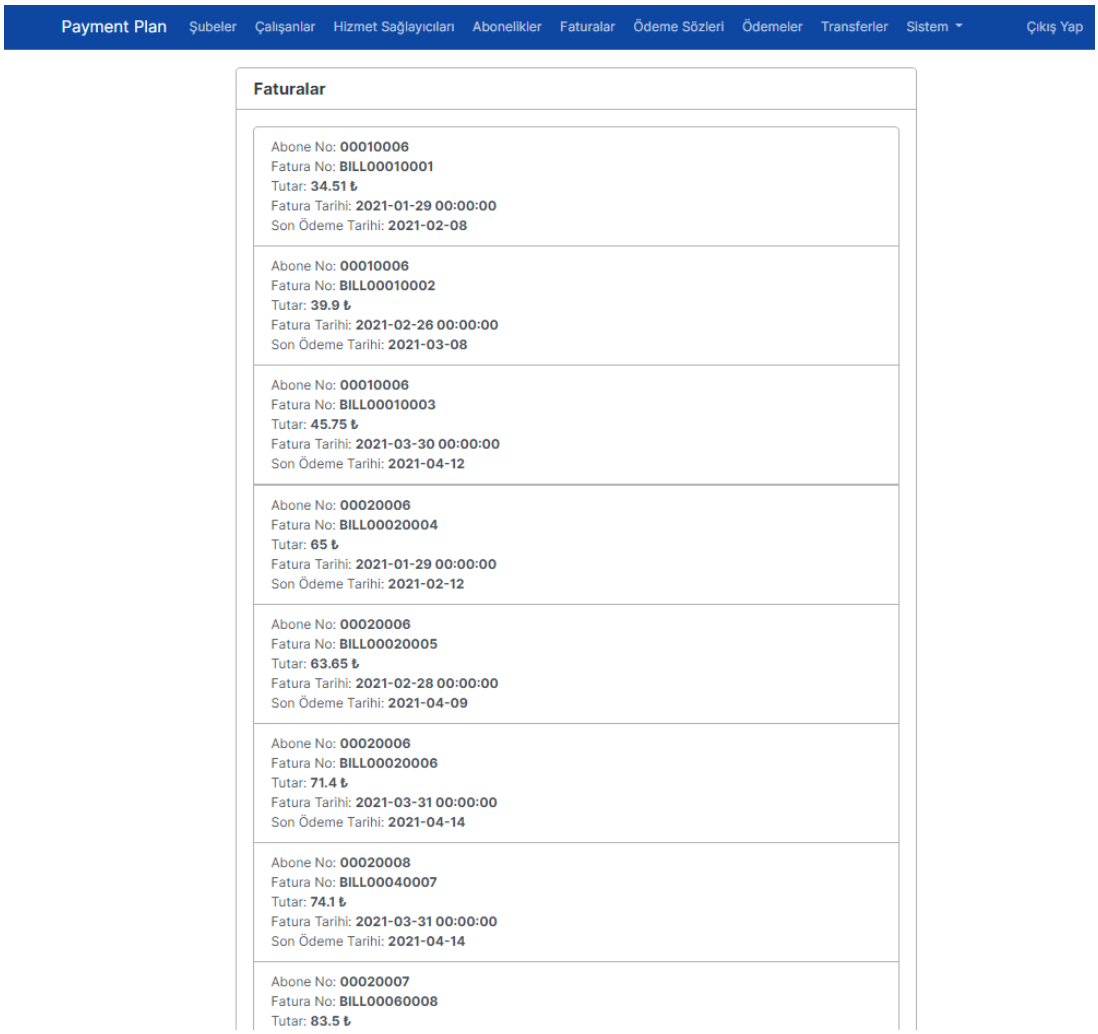
16.1. Bill List Screen

select_bills

For Admin

```
let bills = await db.db(`SELECT bill.id, bill.is_valid, bill.subscription_id
, bill.amount, bill.first_read_date, bill.last_read_date, bill.start_date, b
ill.due_date,
                                CONCAT('BILL', LPAD(bill.subscription_id, 4,
 '0000'), LPAD(bill.id, 4, '0000')) AS bill_number,
                                CONCAT(LPAD(sub.provider_id, 4, '0000'), LPA
D(sub.user_id, 4, '0000')) AS subscription_number
                              FROM bill
                              INNER JOIN subscription AS sub ON sub.id = bill
.subscription_id AND sub.is_valid = 1`);
```

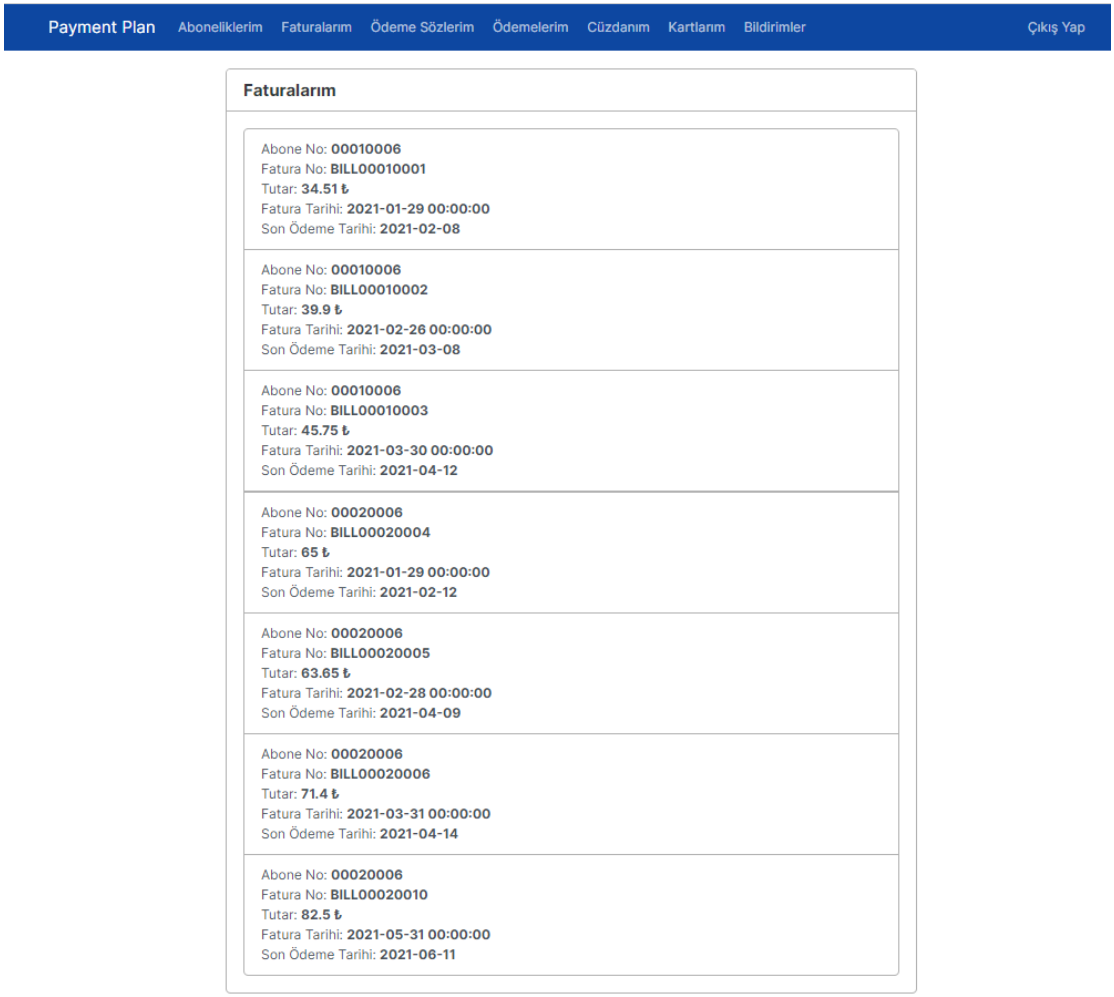
Screenshot



For Subscriber

```
let bills = await db.db(`SELECT bill.id, bill.is_valid, bill.subscription_id
, bill.amount, bill.first_read_date, bill.last_read_date, bill.start_date, b
ill.due_date,
                                CONCAT('BILL', LPAD(bill.subscription_id, 4,
 '0000'), LPAD(bill.id, 4, '0000')) AS bill_number,
                                CONCAT(LPAD(sub.provider_id, 4, '0000'), LPA
D(sub.user_id, 4, '0000')) AS subscription_number
                              FROM bill
                              INNER JOIN subscription AS sub ON sub.id = bill
.subscription_id AND sub.user_id = ? AND sub.is_valid = 1
                              WHERE bill.is_valid = 1`, [req.auth_id]);
```

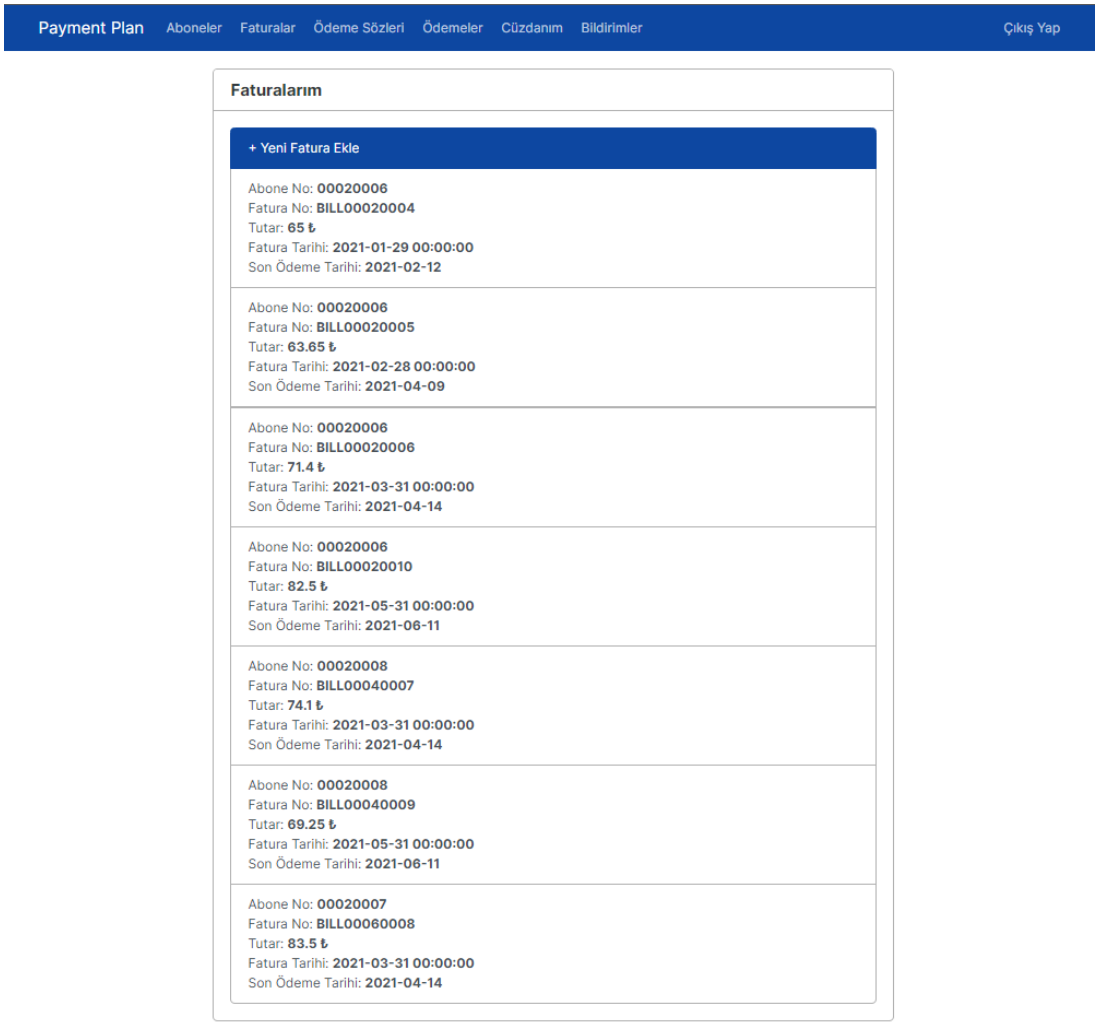
Screenshot



For Provider

```
let bills = await db.db(`SELECT bill.id, bill.is_valid, bill.subscription_id
, bill.amount, bill.first_read_date, bill.last_read_date, bill.start_date, b
ill.due_date,
                                CONCAT('BILL', LPAD(bill.subscription_id, 4,
 '0000'), LPAD(bill.id, 4, '0000')) AS bill_number,
                                CONCAT(LPAD(sub.provider_id, 4, '0000'), LPA
D(sub.user_id, 4, '0000')) AS subscription_number
                              FROM bill
                              INNER JOIN subscription AS sub ON sub.id = bill
.subscription_id AND sub.provider_id = ? AND sub.is_valid = 1
                              WHERE bill.is_valid = 1`, [provider.id]);
```

Screenshot



16.2. Add Bill Screen

select_subscriptions

```
let subscriptions = await db.db(`SELECT sub.id, sub.is_valid, CONCAT(LPAD(su
b.provider_id, 4, '0000'), LPAD(sub.user_id, 4, '0000')) AS subscription_num
ber
                                    FROM subscription AS sub
                                    INNER JOIN provider ON provider.id = su
b.provider_id AND provider.user_id = ? AND provider.is_valid = 1
                                    WHERE sub.is_valid = 1`, [req.auth_id])
;
```

select_bill

```
let bill = await db.dbGetFirst(`SELECT id, is_valid, subscription_id, amount
, first_read_date, last_read_date, start_date, due_date,
                                       CONCAT('BILL', LPAD(subscription_id,
 4, '0000'), LPAD(id, 4, '0000')) AS bill_number
                                   FROM bill
                                   WHERE id = ?`, [req.params.id]);
```

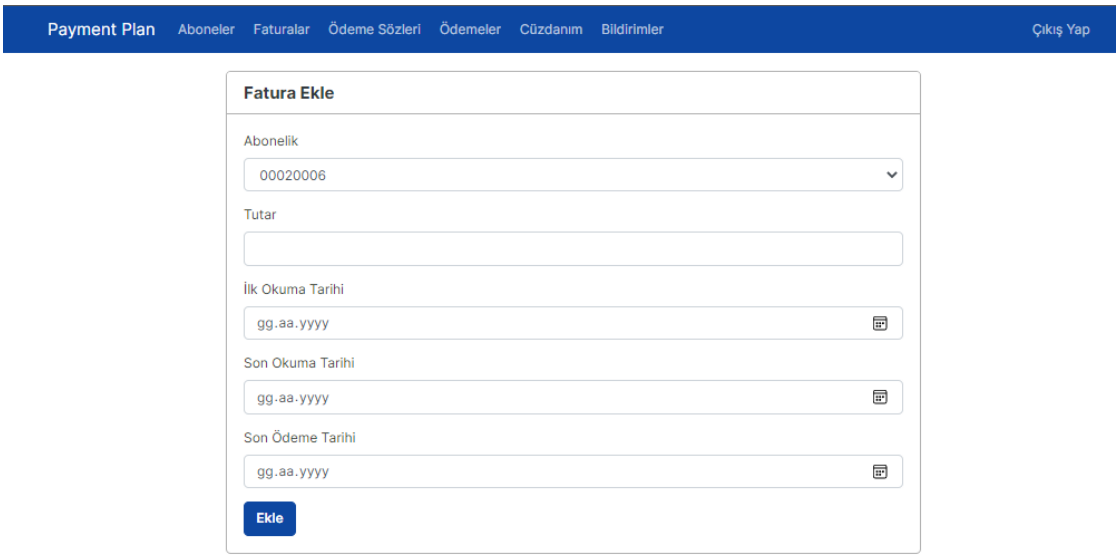
select_subscriptions

```
let subscriptions = await db.db(`SELECT sub.id, sub.is_valid, CONCAT(LPAD(su
b.provider_id, 4, '0000'), LPAD(sub.user_id, 4, '0000')) AS subscription_num
ber
                                    FROM subscription AS sub
                                    INNER JOIN provider ON provider.id = su
b.provider_id AND provider.user_id = ? AND provider.is_valid = 1
                                    WHERE sub.is_valid = 1`, [req.auth_id])
;
```

insert_bill

```
await db.dbInsert("INSERT INTO bill (subscription_id, amount, first_read_dat
e, last_read_date, start_date, due_date, created_by) VALUES (?, ?, ?, ?, ?,
?, ?)",
        [req.body.subscription_id, req.body.amount, req.body.first_read_date
, req.body.last_read_date, req.body.last_read_date, req.body.due_date, req.a
uth_id]);
```

Screenshot

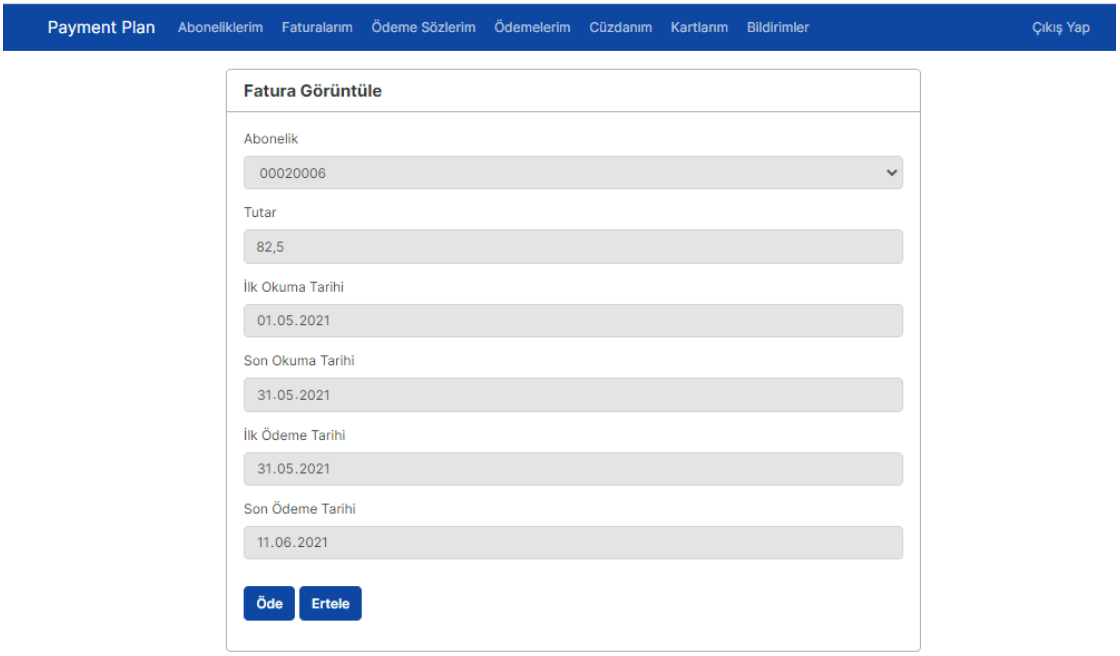

Payment Plan
Aboneler
Faturalar
Ödeme Sözleri
Ödemeler
Cüzdanım
Bildirimler
Çıkış Yap
Fatura Ekle
Abonelik
00020006
Tutar
İlk Okuma Tarihi
gg.aa.yyyy
Son Okuma Tarihi
gg.aa.yyyy
Son Ödeme Tarihi
gg.aa.yyyy
Ekle

16.3. View Bill Screen

select_bill

```
let bill = await db.dbGetFirst(`SELECT bill.id, bill.is_valid, bill.subscrip
tion_id, bill.amount, bill.first_read_date, bill.last_read_date, bill.start_
date, bill.due_date,
                                        CONCAT('BILL', LPAD(subscription_id,
 4, '0000'), LPAD(bill.id, 4, '0000')) AS bill_number,
                                        IFNULL(promise.id, 0) AS promise_id,
 IFNULL(payment.id, 0) AS payment_id
                                    FROM bill
                                    LEFT JOIN promise ON promise.bill_id = b
ill.id AND promise.is_valid = 1
                                    LEFT JOIN payment ON payment.bill_id = b
ill.id AND payment.is_valid = 1
                                    WHERE bill.id = ?`, [req.params.id]);
```

Screenshot

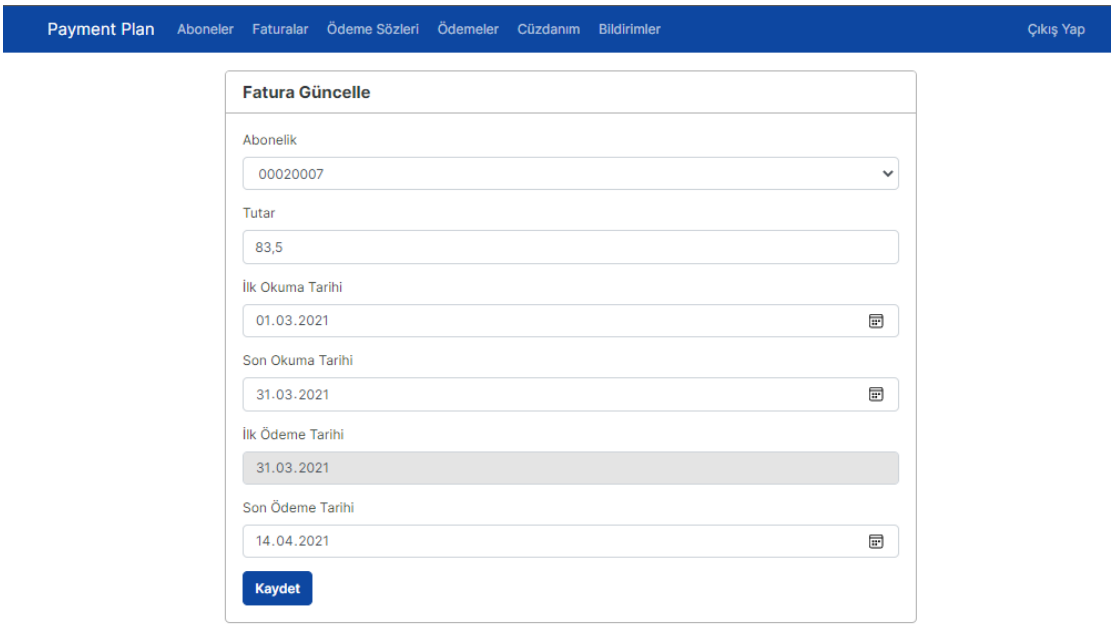


16.4. Edit Bill Screen

update_bill

```
await db.dbUpdate(`UPDATE bill SET subscription_id = ?, amount = ?, first_re
ad_date = ?, last_read_date = ?, start_date = ?, due_date = ?, updated_by =
?, updated_at = current_timestamp()
                        WHERE id = ?`, [req.body.subscription_id, req.body.am
ount, req.body.first_read_date, req.body.last_read_date, req.body.last_read_
date, req.body.due_date, req.auth_id, req.body.id]);
```

Screenshot



16.5. Branch List Screen

select_branches

```
let branchs = await db.db("SELECT id, is_valid, branch_type_id, branch_code,
 branch_name FROM branch");
```

Screenshot

16.6. Add Branch Screen

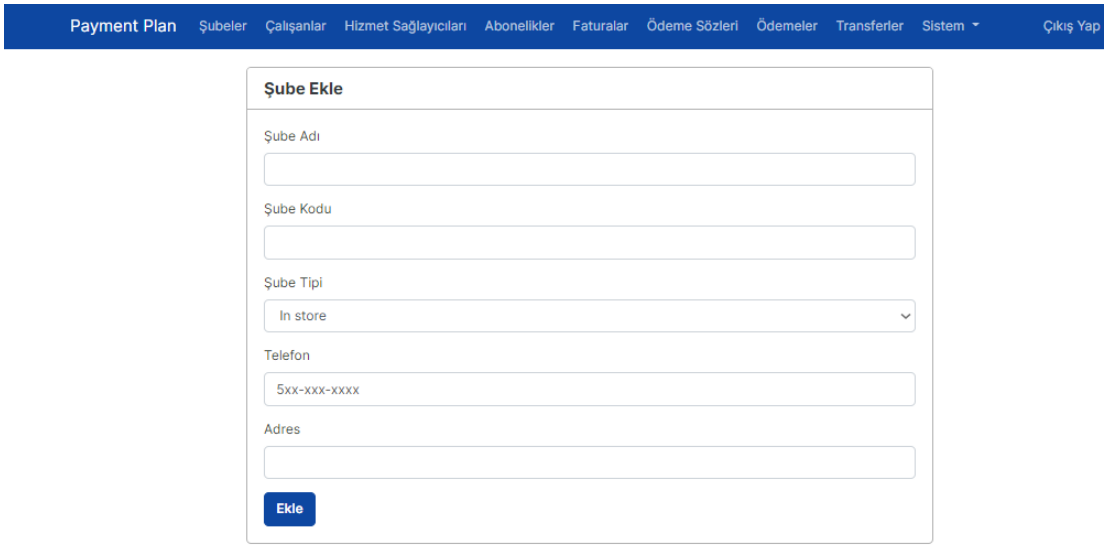
select_branch_types

```
let branchTypes = await db.db(`SELECT id, is_valid, code, value, description
                                FROM parameter
                                WHERE code = 'branch_type' AND is_valid =
 1`);
```

insert_branch

```
await db.dbInsert("INSERT INTO branch (branch_type_id, branch_code, branch_n
ame, phone, address, created_by) VALUES (?, ?, ?, ?, ?, ?)", [req.body.branc
h_type_id, req.body.branch_code, req.body.branch_name, req.body.phone, req.b
ody.address, req.auth_id]);
```

Screenshot


Payment Plan
Şubeler
Çalışanlar
Hizmet Sağlayıcıları
Abonelikler
Faturalar
Ödeme Sözleri
Ödemeler
Transferler
Sistem
Çıkış Yap
Şube Ekle
Şube Adı
Şube Kodu
Şube Tipi
In store
Telefon
5xx-xxx-xxxx
Adres
Ekle

16.7. Edit Branch Screen

select_branch

```
let branch = await db.dbGetFirst(`SELECT id, is_valid, branch_type_id, branc
h_code, branch_name, phone, address
                                    FROM branch
                                    WHERE id = ?`, [req.params.id]);
```

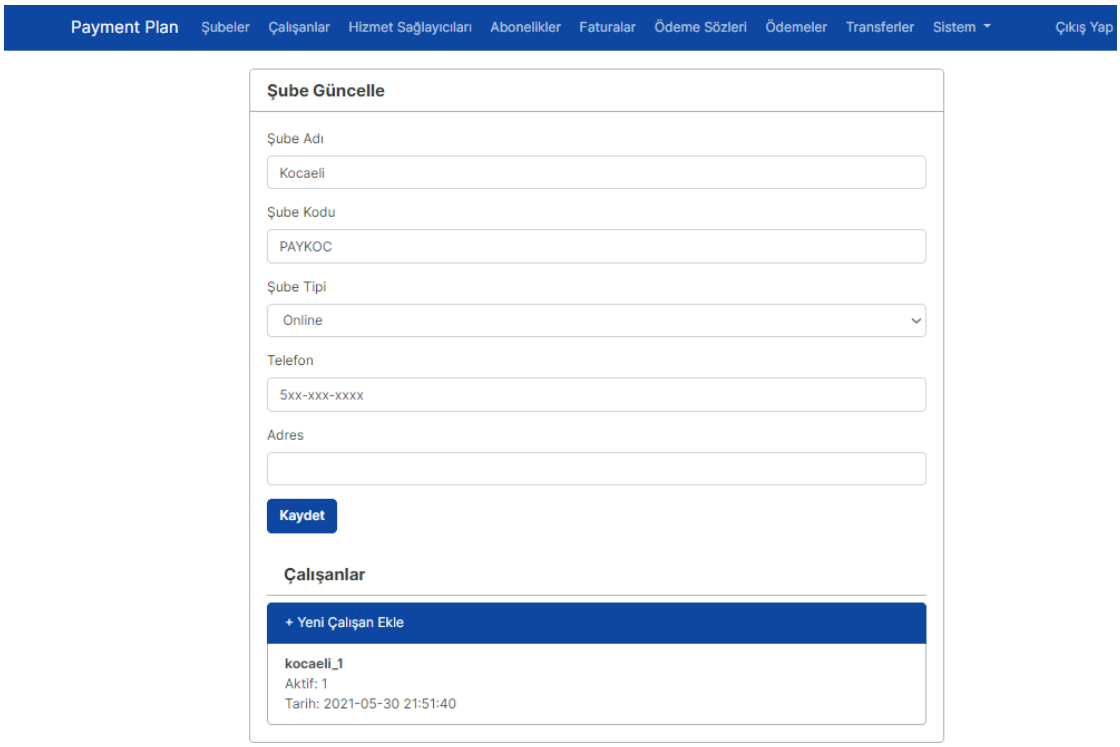
select_branch_types

```
let branchTypes = await db.db(`SELECT id, is_valid, code, value, description
                                FROM parameter
                                WHERE code = 'branch_type' AND is_valid =
 1`);
```

select_employees

```
let employees = await db.db(`SELECT e.user_id, e.is_valid, u.username, e.cre
ated_at
                            FROM employee AS e
                            INNER JOIN user AS u ON e.user_id = u.id AND u
.is_valid = 1
                            WHERE e.branch_id = ? AND e.is_valid = 1`, [br
anch.id]);
```

Screenshot

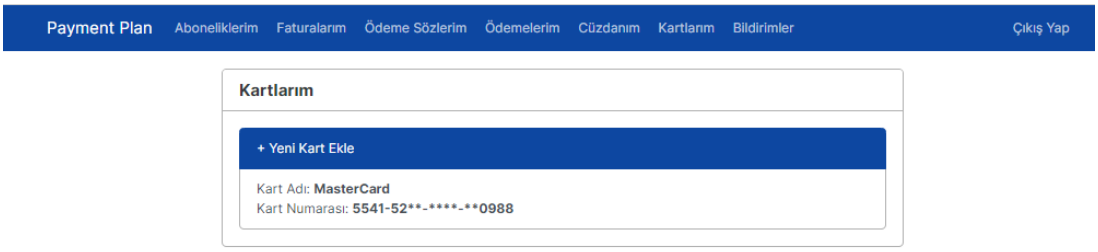


16.8. Card List Screen

select_cards

```
let cards = await db.db(`SELECT id, is_valid, user_id, CONCAT(LEFT(card_numb
er, 4), '-', SUBSTR(card_number, 6, 2),'**-****-
**', RIGHT(card_number, 4)) AS card_number, card_name
                              FROM card
                              WHERE user_id = ? AND is_valid = 1`, [req.auth_
id]);
```

Screenshot

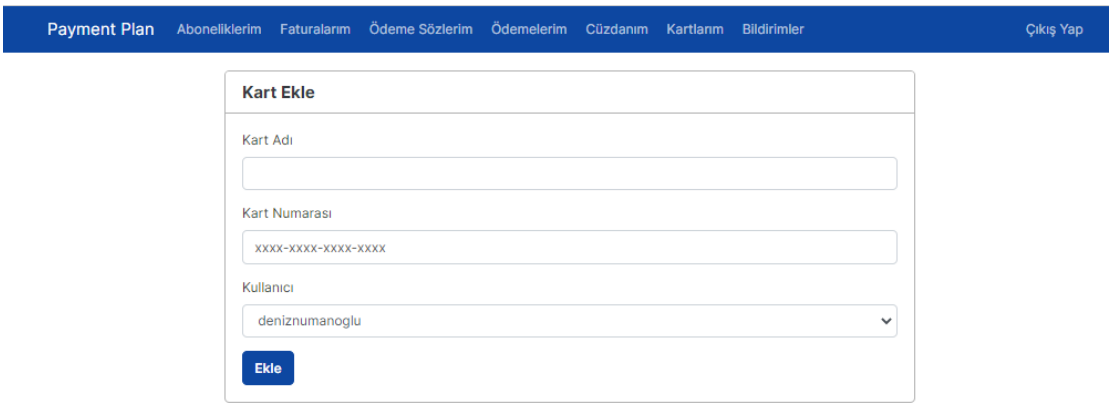


16.9. Add Card Screen

select_user

```
let user = await db.db(`SELECT id, is_valid, username
                              FROM user
                              WHERE id = ? AND is_valid = 1`, [req.auth_id]);
```

Screenshot



16.10. Edit Card Screen

select_card

```
let card = await db.dbGetFirst(`SELECT id, is_valid, user_id, card_number, c
ard_name
                                    FROM card
                                    WHERE id = ?`, [req.params.id]);
```

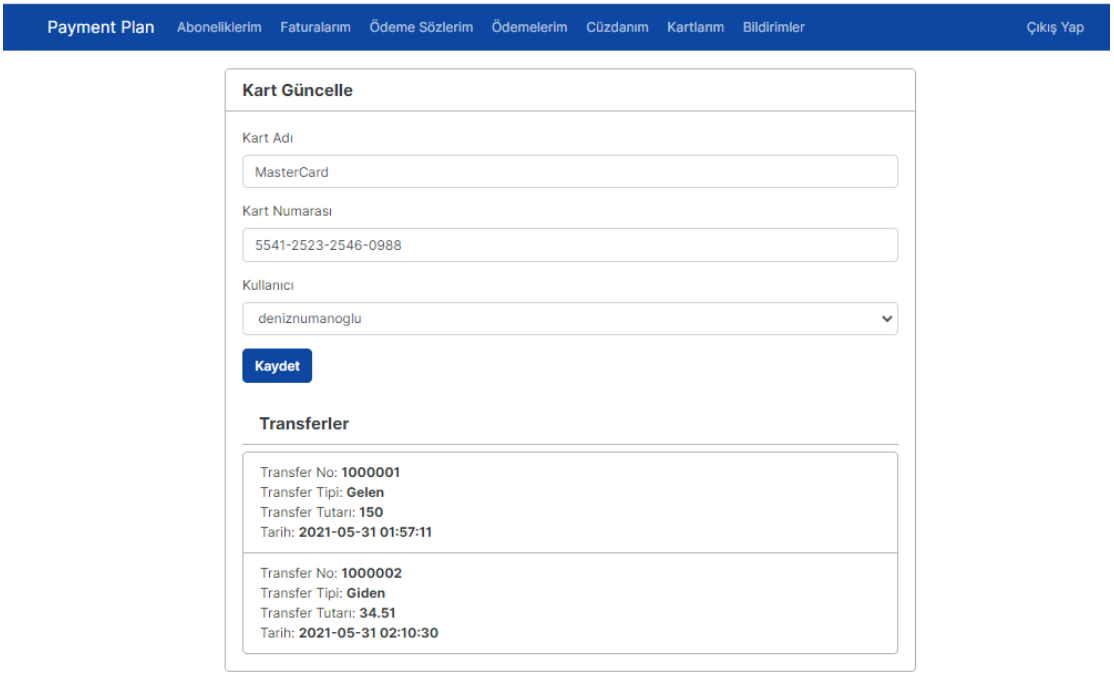
select_transactions

```
let transactions = await db.db(`SELECT transaction_sequence_number AS seq_nu
m, transaction_type_id, transaction_amount, channel_type_id, created_at, row
_number() over ( partition by transaction_sequence_number)
                                    FROM transaction
                                    WHERE transaction_source_id = 24 AND is_
valid = 1 AND (source_id = ? OR destination_id = ?)
                                    GROUP BY transaction_sequence_number`, [
card.id, card.id]);
```

update_card

```
await db.dbUpdate(`UPDATE card SET user_id = ?, card_number = ?, card_name =
 ?, updated_by = ?, updated_at = current_timestamp()
                        WHERE id = ?`, [req.body.user_id, req.body.card_numbe
r, req.body.card_name, req.auth_id, req.body.id]);
```

Screenshot

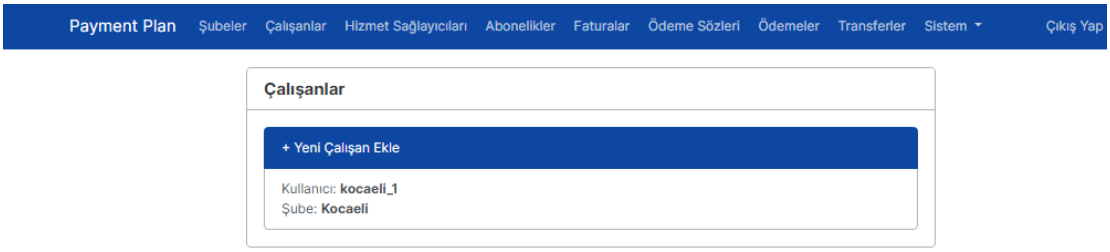


16.11. Employee List Screen

select_employees

```
let employees = await db.db(`SELECT e.user_id, u.username, b.branch_name, e.
created_at, e.updated_at
                                FROM employee AS e
                                INNER JOIN user AS u ON u.id = e.user_id AN
D u.is_valid = 1
                                LEFT JOIN branch AS b ON b.id = e.branch_id
 AND b.is_valid = 1`);
```

Screenshot



16.12. Add Employee Screen

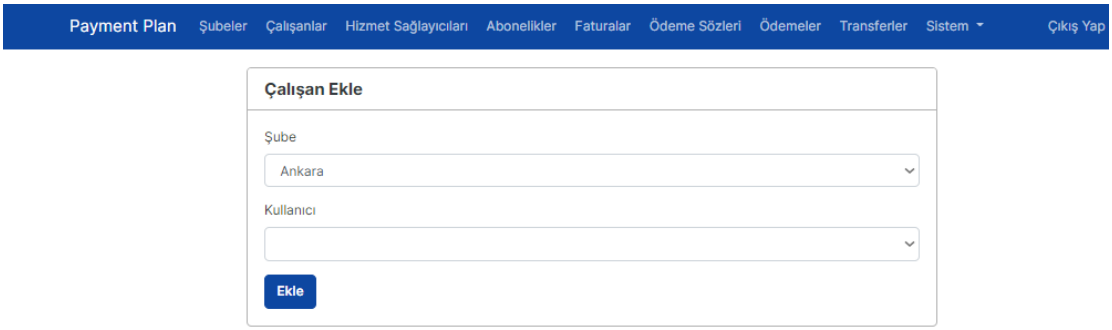
select_users_to_add

```
// Herhangi bir şubeyenin çalışanı olmayan kullanıcılar
    let users = await db.db(`SELECT u.id, u.is_valid, u.user_type_id, u.user
name
                            FROM user AS u
                            LEFT JOIN employee AS e ON e.user_id = u.id
                            WHERE e.user_id IS NULL AND u.user_type_id = 2`
);
```

insert_employee

```
await db.dbInsert("INSERT INTO employee (user_id, branch_id, created_by) VAL
UES (?, ?, ?)", [req.body.user_id, req.body.branch_id, req.auth_id]);
```

Screenshot



16.13. Edit Employee Screen

select_employee

```
let employee = await db.dbGetFirst(`SELECT user_id, is_valid, branch_id
                                    FROM employee
                                    WHERE user_id = ? AND is_valid = 1`, [req.pa
rams.user_id]);
```

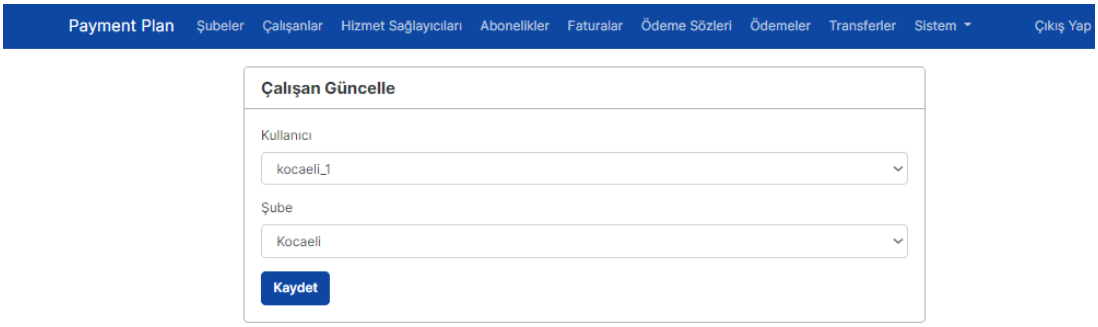
select_users

```
let users = await db.db(`SELECT id, is_valid, username
                                 FROM user
                                 WHERE user_type_id = 2 AND is_valid = 1`);
```

update_employee

```
await db.dbUpdate(`UPDATE employee SET branch_id = ?, updated_by = ?, update
d_at = current_timestamp()
                          WHERE user_id = ?`, [req.body.branch_id, req.auth_id,
 req.body.user_id]);
```

Screenshot



16.14. Notification List Screen

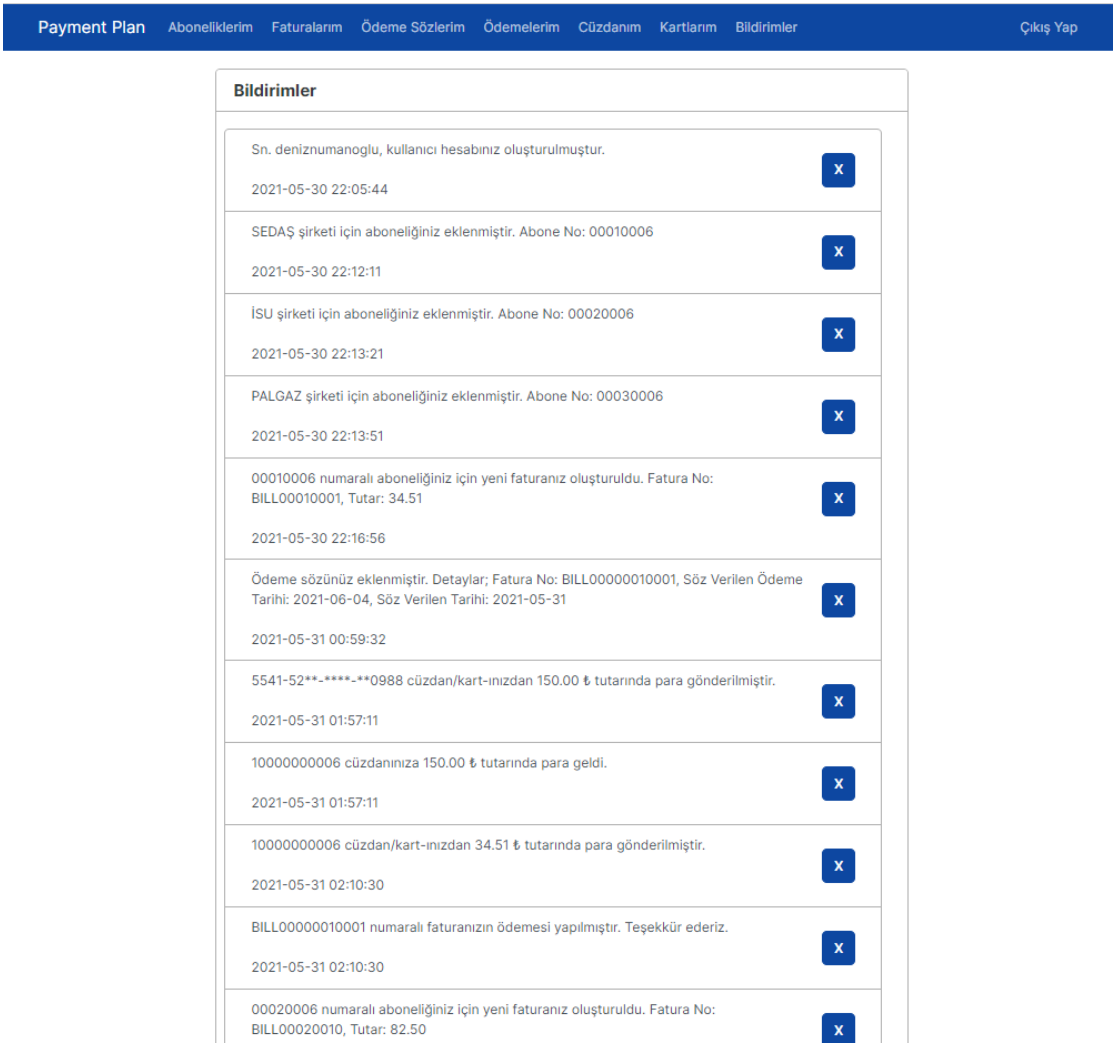
select_notifications

```
let notifications = await db.db(`SELECT *
                                FROM notification
                                WHERE is_read = 0 AND user_id = ?`, [req
.auth_id]);
```

mark_as_read

```
let result = await db.dbUpdate(`UPDATE notification SET is_read = 1, updated
_by = ?, updated_at = current_timestamp()
                                WHERE id = ?`, [req.auth_id, req.body.id
]);
```

Screenshot

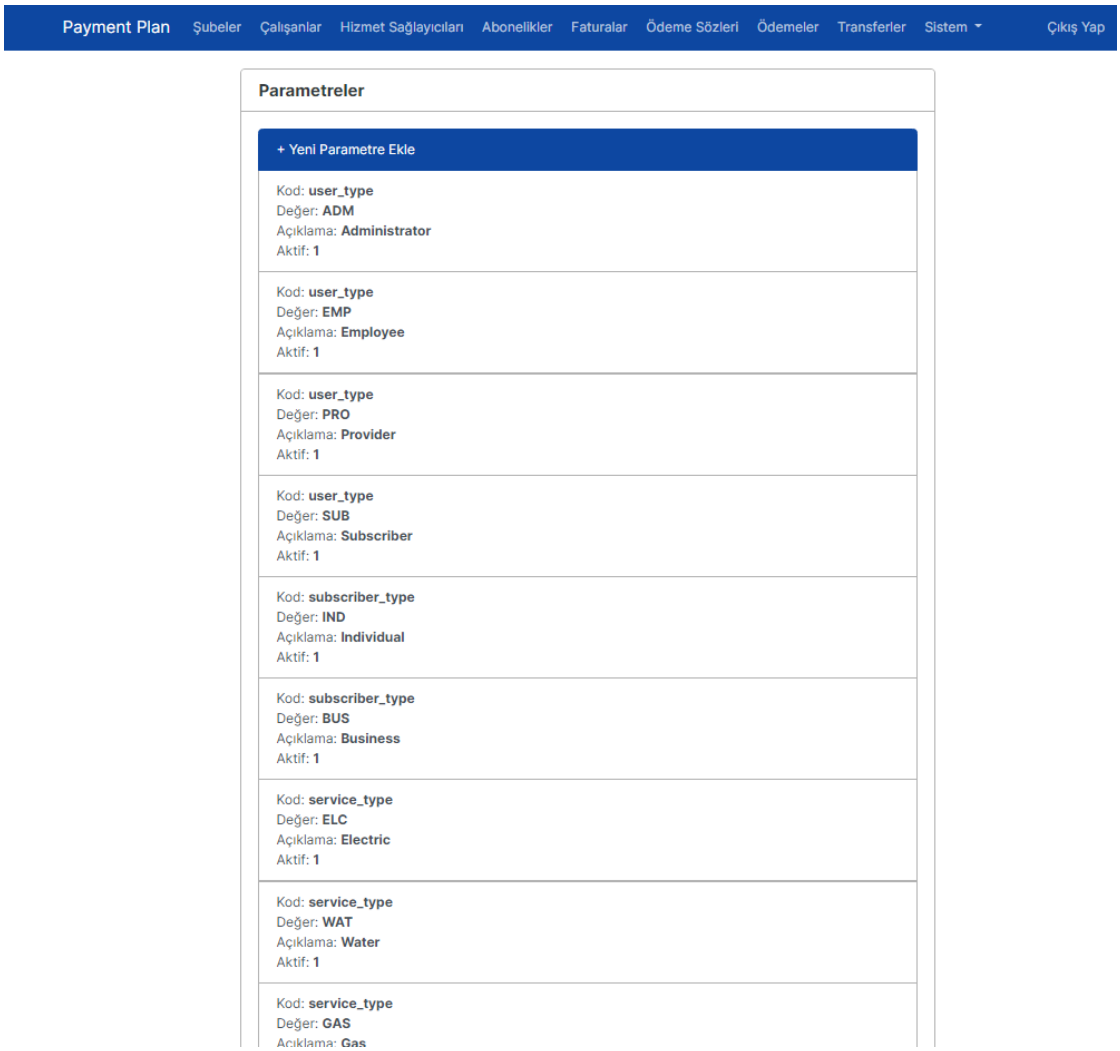


16.15. Parameter List Screen

select_parameters

```
let parameters = await db.db("SELECT id, is_valid, code, value, description
FROM parameter");
```

Screenshot

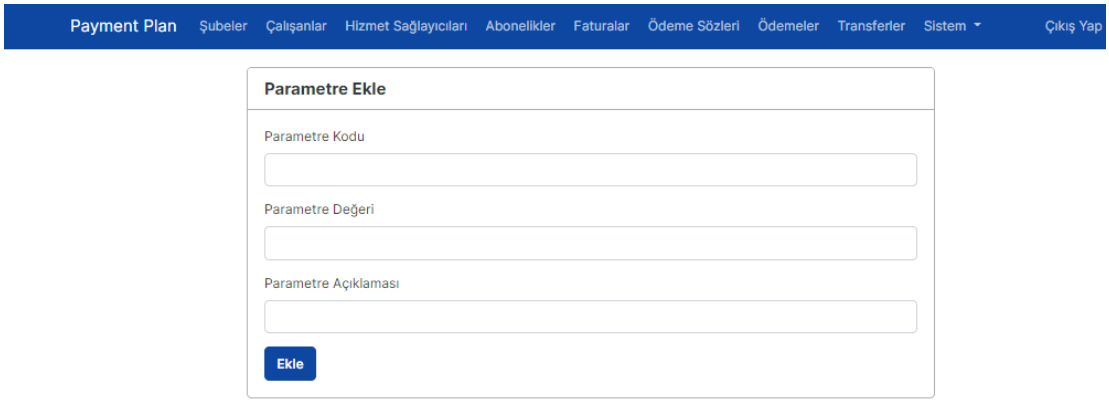


16.16. Add Parameter Screen

insert_parameter

```
await db.dbInsert("INSERT INTO parameter (code, value, description, created_
by) VALUES (?, ?, ?, ?, ?, ?)", [req.body.code, req.body.value, req.body.des
cription, req.auth_id]);
```

Screenshot



16.17. Edit Parameter Sreen

update_parameter

```
await db.dbUpdate(`UPDATE parameter SET code = ?, value = ?, description = ?
, updated_by = ?, updated_at = current_timestamp()
                      WHERE id = ?`, [req.body.code, req.body.value, req.bo
dy.description, req.auth_id, req.body.id]);
```

Screenshot

16.18. Payment List Screen

select_payments

For Admin

```
let payments = await db.db(`SELECT payment.transaction_sequence_number AS se
q_num, bill.amount AS bill_amount, bill.start_date, bill.due_date, payment.c
reated_at AS payment_date, provider.corporate_name,
                                    CONCAT('BILL', LPAD(bill.subscription_id
, 4, '0000'), LPAD(bill.id, 4, '0000')) AS bill_number,
                                    CONCAT(LPAD(sub.provider_id, 4, '0000'),
 LPAD(sub.user_id, 4, '0000')) AS subscription_number
                                FROM payment
                                INNER JOIN bill ON bill.id = payment.bill_id
                                INNER JOIN subscription AS sub ON sub.id = b
ill.subscription_id
                                INNER JOIN provider ON sub.provider_id = pro
vider.id`);
```

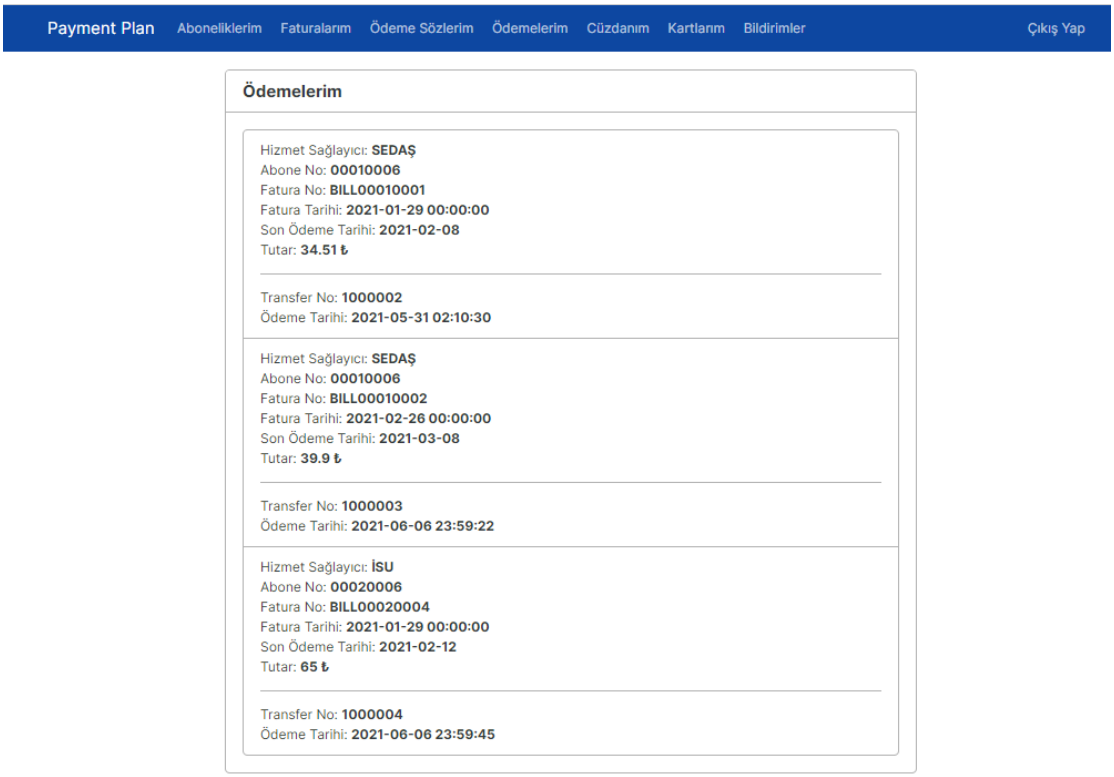
For Subscriber

```
// Abonenin ödemeleri
let payments = await db.db(`SELECT payment.transaction_sequence_number AS se
q_num, bill.amount AS bill_amount, bill.start_date, bill.due_date, payment.c
reated_at AS payment_date, provider.corporate_name,
                                    CONCAT('BILL', LPAD(bill.subscription_id
, 4, '0000'), LPAD(bill.id, 4, '0000')) AS bill_number,
                                    CONCAT(LPAD(sub.provider_id, 4, '0000'),
 LPAD(sub.user_id, 4, '0000')) AS subscription_number
                                FROM payment
                                INNER JOIN bill ON bill.id = payment.bill_id
                                INNER JOIN subscription AS sub ON sub.id = b
ill.subscription_id AND sub.user_id = ?
                                INNER JOIN provider ON sub.provider_id = pro
vider.id`, [req.auth_id]);
```

For Provider

```
// Sağlayıcının gelen ödemeler
let payments = await db.db(`SELECT payment.transaction_sequence_number AS se
q_num, bill.amount AS bill_amount, bill.start_date, bill.due_date, payment.c
reated_at AS payment_date,
                                CONCAT('BILL', LPAD(bill.subscription_id, 4,
 '0000'), LPAD(bill.id, 4, '0000')) AS bill_number,
                                 CONCAT(LPAD(sub.provider_id, 4, '0000'), LP
AD(sub.user_id, 4, '0000')) AS subscription_number
                              FROM payment
                              INNER JOIN bill ON bill.id = payment.bill_id
                              INNER JOIN subscription AS sub ON sub.id = bill
.subscription_id
                              INNER JOIN provider ON sub.provider_id = provid
er.id AND provider.user_id = ?`, [req.auth_id]);
```

Screenshot



16.19. Pay Screen

select_bills

```
// Abonenin ödenmemiş faturaları
    let bills = await db.db(`SELECT *
                              FROM v_unpaids
                              WHERE user_id = ?`, [req.auth_id]);
```

select_source_wallet

```
// Abonenin Cüzdanı
    let sourceWallet = await db.dbGetFirst(`SELECT id, is_valid, user_id, ba
lance, LPAD(user_id, 11, '10000000000') AS wallet_name
                                            FROM wallet
                                            WHERE user_id = ?`, [req.auth_id
]);
```

select_provider

```
// Sağlayıcı
    let provider = await db.dbGetFirst(`SELECT provider.id, wallet.id AS wal
let_id
                                        FROM provider
                                        INNER JOIN wallet ON wallet.user_id
= provider.user_id AND wallet.is_valid = 1
                                        WHERE provider.id = ?`, [req.body.pr
ovider_id]);
```

create_payment

call_create_credit_transaction

```
// First Transaction
        await db.query(`CALL create_transaction (?, ?, ?, ?, ?, ?, ?, ?, @re
sult_val);`,
            [credit_id,
                internal_id,
                other_channel_id,
                sourceWallet.id,
                provider.wallet_id,
                bill.amount,
                req.auth_id,
                seq_number], function (err, results) {
                    if (err) throw err;

                    db.query("SELECT @result_val AS result;", function (errC
rd, resultsCrd) {
                        if (errCrd) throw errCrd;
```

```
                        if (resultsCrd[0].result < 1) {
                            if (err) throw err;
                            res.resultError("Limit yetersiz!");
                            return;
                        }
                    });

                    console.log("First Transaction: ", results);
                });
```

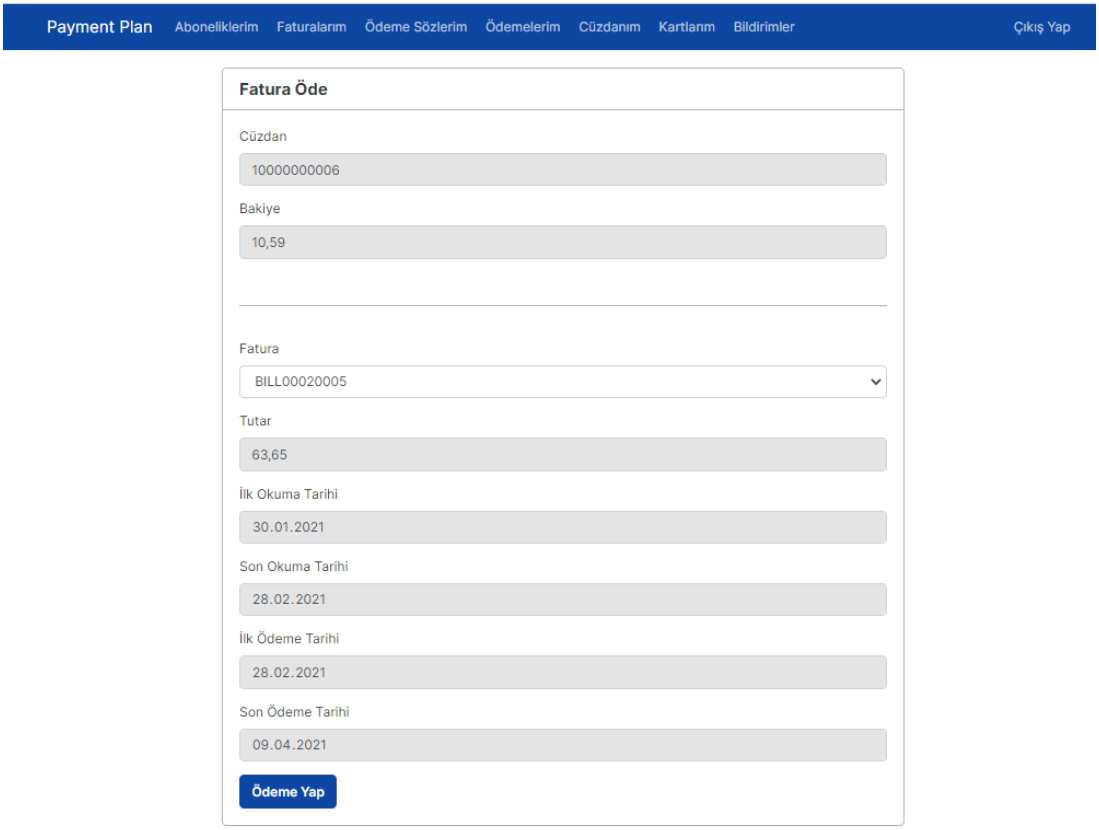
call_create_debit_transaction

```
// Second Transaction
        await db.query("CALL create_transaction (?, ?, ?, ?, ?, ?, ?, ?, @re
sult_val);",
            [debit_id,
                internal_id,
                other_channel_id,
                sourceWallet.id,
                provider.wallet_id,
                bill.amount,
                req.auth_id,
                seq_number], function (err, results) {
                    if (err) throw err;

                    console.log("Second Transaction: ", results);
                });
```

insert_payment

```
let payment_id = await db.dbInsert("INSERT INTO payment (bill_id, transactio
n_sequence_number, created_by) VALUES (?, ?, ?)",
            [req.body.bill_id, seq_number, req.auth_id]);
```

Screenshot



16.20. Promise List Screen

select_promises

For Admin

```
let promises = await db.db(`SELECT promise.id, promise.is_valid, promise.bil
l_id, promise.promise_date, promise.status_id, promise.due_date,
                                    CONCAT('BILL', LPAD(sub.id, 7, '0000000'
), LPAD(bill.id, 4, '0000')) AS bill_number
                                FROM promise
                                INNER JOIN bill AS bill ON bill.id = promise
.bill_id AND bill.is_valid = 1
                                INNER JOIN subscription AS sub ON sub.id = b
ill.subscription_id AND sub.is_valid = 1
                                WHERE bill.is_valid = 1`);
```

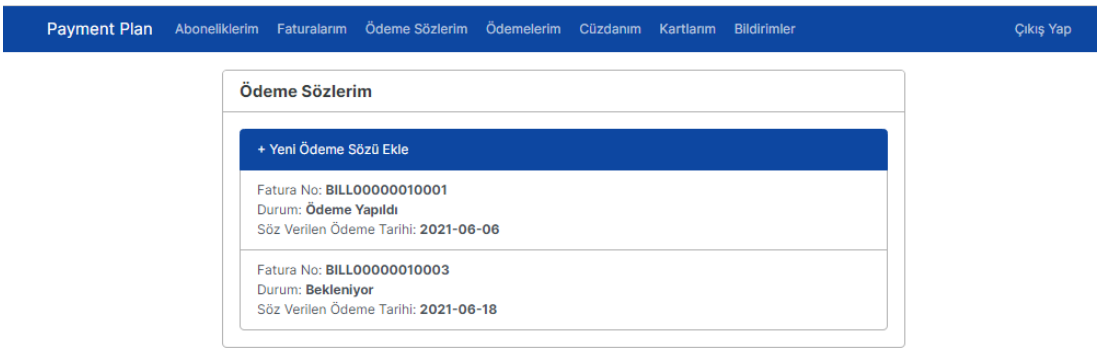
For Subscriber

```
// Abonenin verdiği ödeme sözleri
let promises = await db.db(`SELECT promise.id, promise.is_valid, promise.bil
l_id, promise.promise_date, promise.status_id, promise.due_date,
                                    CONCAT('BILL', LPAD(sub.id, 7, '0000000'
), LPAD(bill.id, 4, '0000')) AS bill_number
                                FROM promise
                                INNER JOIN bill AS bill ON bill.id = promise
.bill_id AND bill.is_valid = 1
                                INNER JOIN subscription AS sub ON sub.id = b
ill.subscription_id AND sub.user_id = ? AND sub.is_valid = 1
                                WHERE promise.is_valid = 1`, [req.auth_id]);
```

For Provider

```
// Sağlayıcının aldığı ödeme sözleri
let promises = await db.db(`SELECT promise.id, promise.is_valid, promise.bil
l_id, promise.promise_date, promise.status_id, promise.due_date,
                                    CONCAT('BILL', LPAD(sub.id, 7, '0000000'
), LPAD(bill.id, 4, '0000')) AS bill_number
                                FROM promise
                                INNER JOIN bill AS bill ON bill.id = promise
.bill_id AND bill.is_valid = 1
                                INNER JOIN subscription AS sub ON sub.id = b
ill.subscription_id AND sub.provider_id = ? AND sub.is_valid = 1
                                WHERE promise.is_valid = 1`, [provider.id]);
```

Screenshot



16.21. Add Promise Screen

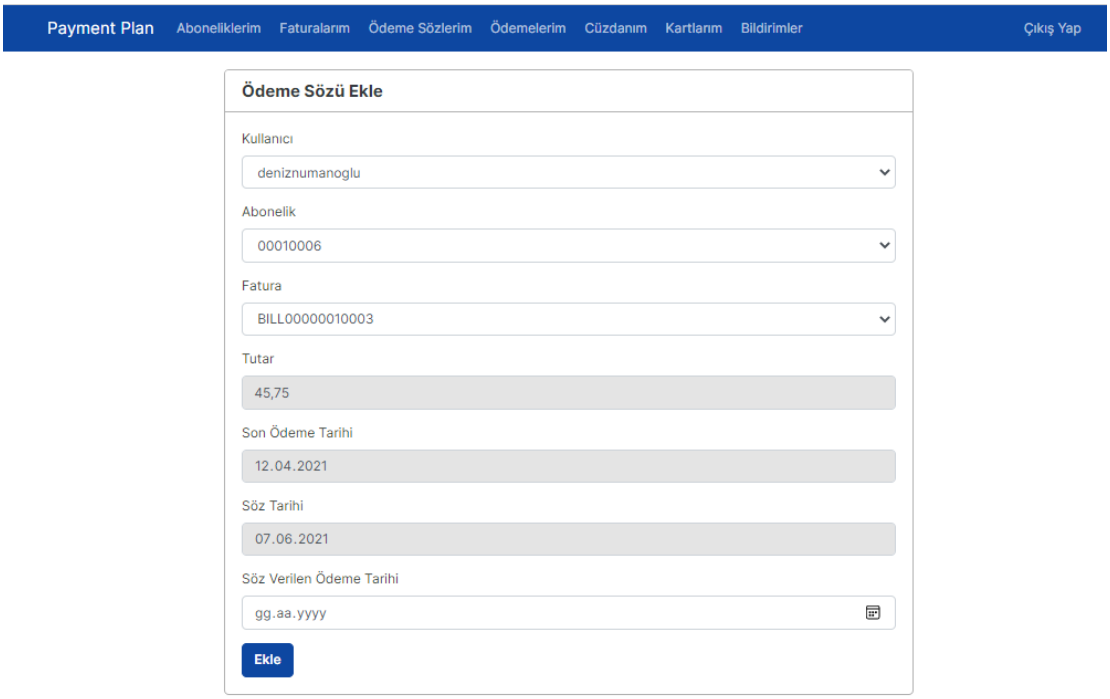
select_unpromised_bills

```
// Daha önce söz vermediği faturalar
    let bills = await db.db(`SELECT bill.id, bill.is_valid, bill.amount, CAS
T(bill.due_date AS DATE) as due_date,
                                CONCAT('BILL', LPAD(sub.id, 7, '0000000'),
LPAD(bill.id, 4, '0000')) AS bill_number,
                                sub.id AS subscription_id
                            FROM bill
                            INNER JOIN subscription AS sub ON sub.id = bill
.subscription_id AND sub.user_id = ? AND sub.is_valid = 1
                            LEFT JOIN promise ON promise.bill_id = bill.id
AND bill.is_valid = 1
                            WHERE bill.is_valid = 1 AND promise.id IS NULL`
, [req.auth_id]);
```

insert_promise

```
await db.dbInsert("INSERT INTO promise (bill_id, due_date, created_by) VALUE
S (?, ?, ?)",
        [req.body.bill_id, req.body.due_date, req.auth_id]);
```

Screenshot

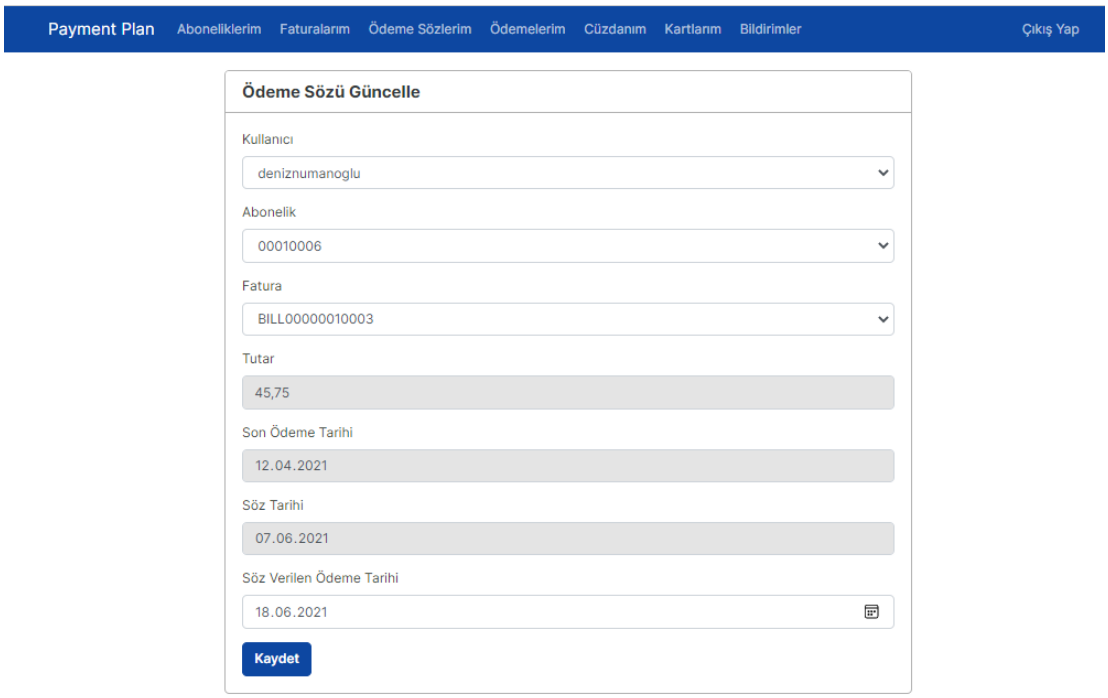


16.22. Edit Promise Screen

update_promise

```
await db.dbUpdate(`UPDATE promise SET bill_id = ?, due_date = ?, status_id =
 ?, updated_by = ?, updated_at = current_timestamp()
                        WHERE id = ?`, [req.body.bill_id, req.body.due_date,
15, req.auth_id, req.body.id]);
```

Screenshot

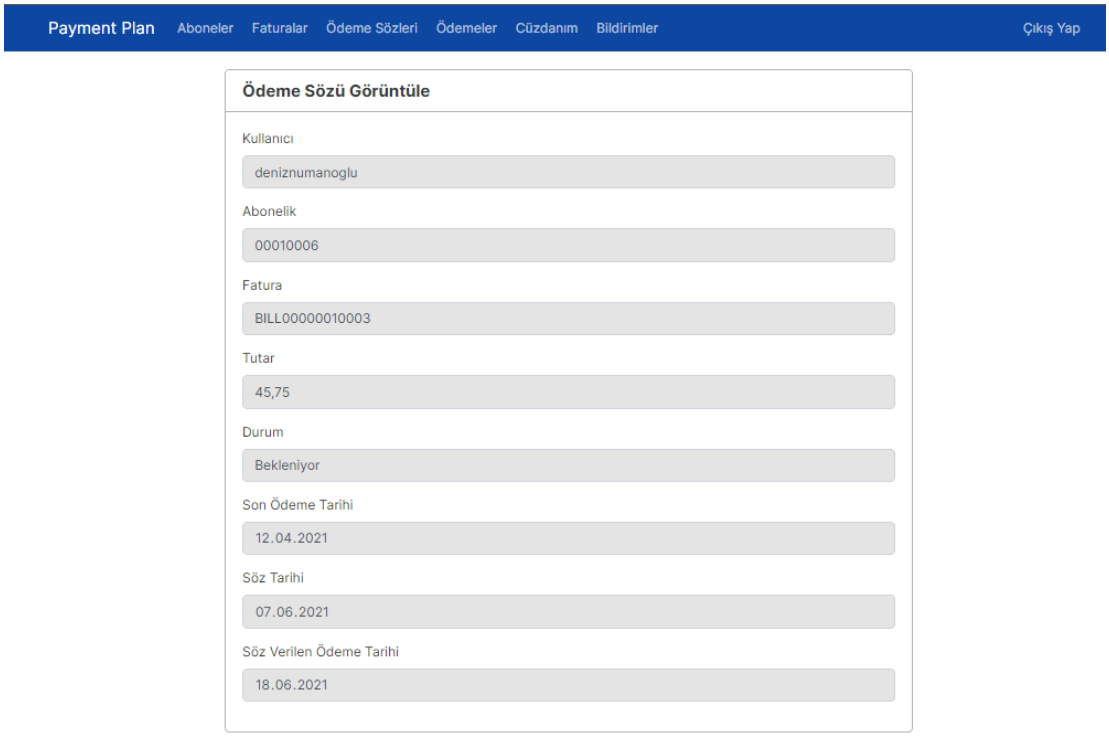


16.23. View Promise Screen

select_bill

```
let promise = await db.dbGetFirst(`SELECT id, is_valid, bill_id, promise_date, status_id, due_date
                                    FROM promise
                                    WHERE id = ?`, [req.body.id]);
```

Screenshot

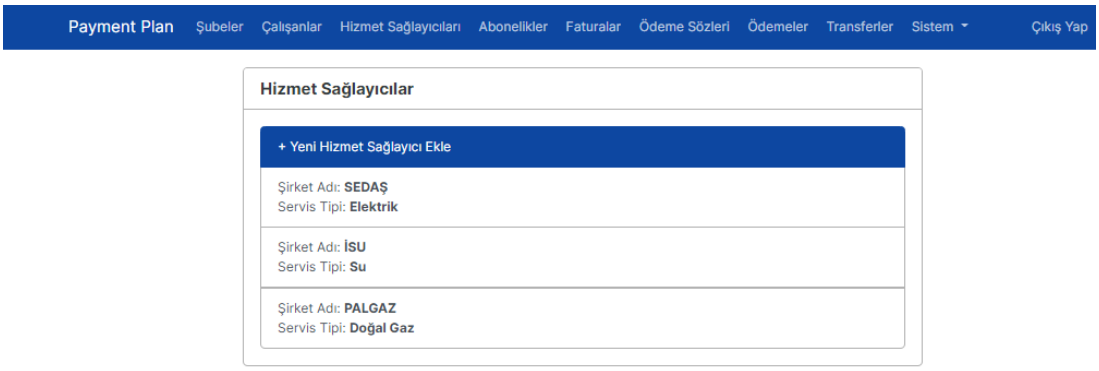


16.24. Provider List Screen

select_providers

```
// Sağlayıcılar
    let providers = await db.db(`SELECT id, is_valid, user_id, service_type_
id, corporate_name
                                FROM provider
                                ORDER BY service_type_id, corporate_name`);
```

Screenshot



16.25. Add Provider Screen

select_service_types

```
let serviceTypes = await db.db(`SELECT id, is_valid, code, value, descriptio
n
                                    FROM parameter
                                    WHERE code = 'service_type' AND is_valid
 = 1`);
```

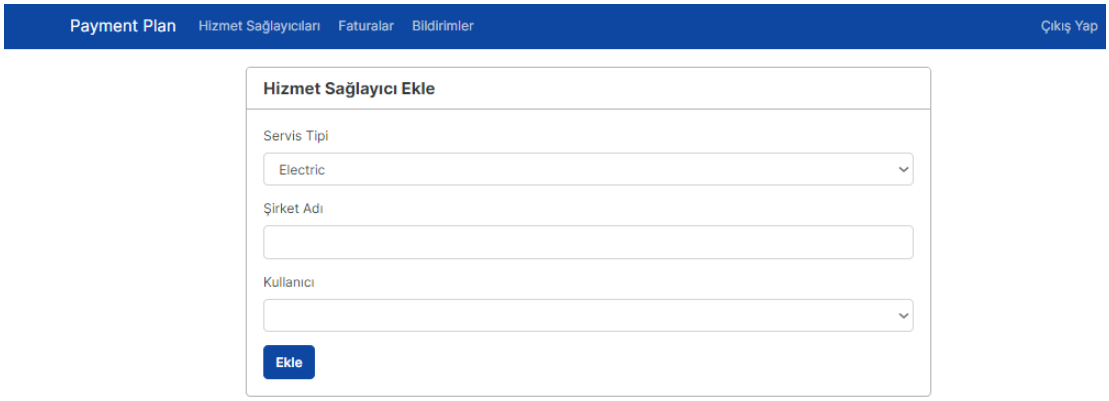
select_users

```
// Herhangi bir sağlayıcının çalışanı olmayan kullanıcılar
    let users = await db.db(`SELECT u.id, u.is_valid, u.user_type_id, u.user
name
                              FROM user AS u
                              LEFT JOIN provider AS p ON p.user_id = u.id  AN
D p.is_valid = 1
                              WHERE p.id IS NULL AND user_type_id = 3 AND u.i
s_valid = 1`);
```

update_provider

```
await db.dbUpdate(`UPDATE provider SET user_id = ?, service_type_id = ?, cor
porate_name = ?, updated_by = ?, updated_at = current_timestamp()
                        WHERE id = ?`, [req.body.user_id, req.body.service_ty
pe_id, req.body.corporate_name, req.auth_id, req.body.id]);
```

Screenshot



16.26. Edit Provider Screen

select_users

```
let users = await db.db(`SELECT id, is_valid, username
                          FROM user
                          WHERE user_type_id = 3 AND is_valid = 1`);
```

insert_provider

```
await db.dbInsert("INSERT INTO provider (user_id, service_type_id, corporate
_name, created_by) VALUES (?, ?, ?, ?)", [req.body.user_id, req.body.service
_type_id, req.body.corporate_name, req.auth_id]);
```

Screenshot

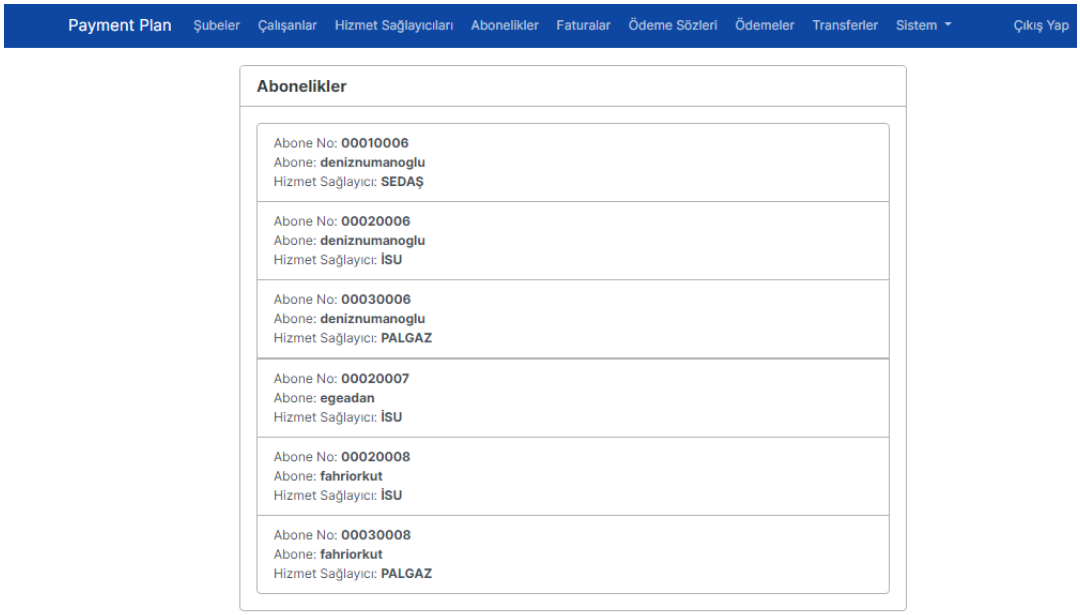
16.27. Subscription List Screen

select_subscriptions

For Admin

```
let subscriptions = await db.db(`SELECT sub.id, sub.is_valid, CONCAT(LPAD(su
b.provider_id, 4, '0000'), LPAD(sub.user_id, 4, '0000')) AS subscription_num
ber, sub.user_id, u.username, provider.corporate_name
                                    FROM subscription AS sub
                                    INNER JOIN user AS u ON u.id = sub.user_
id AND u.is_valid = 1
                                    INNER JOIN provider ON provider.id = sub
.provider_id AND provider.is_valid = 1
                                    WHERE sub.is_valid = 1
                                    ORDER BY sub.user_id, sub.id ASC`);
```

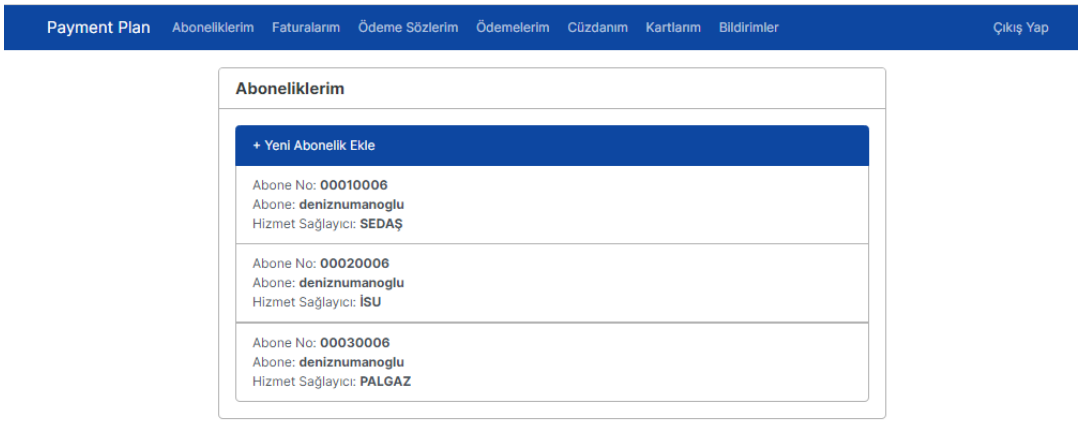
Screenshot



For Subscriber

```
// Abonenin abonelikleri
let subscriptions = await db.db(`SELECT sub.id, sub.is_valid, CONCAT(LPAD(su
b.provider_id, 4, '0000'), LPAD(sub.user_id, 4, '0000')) AS subscription_num
ber, sub.user_id, u.username, provider.corporate_name
                                        FROM subscription AS sub
                                        INNER JOIN user AS u ON u.id = sub.
user_id AND u.is_valid = 1
                                        INNER JOIN provider ON provider.id
= sub.provider_id AND provider.is_valid = 1
                                        WHERE sub.user_id = ? AND sub.is_va
lid = 1
                                        ORDER BY sub.user_id, sub.id ASC`,
[req.auth_id]);
```

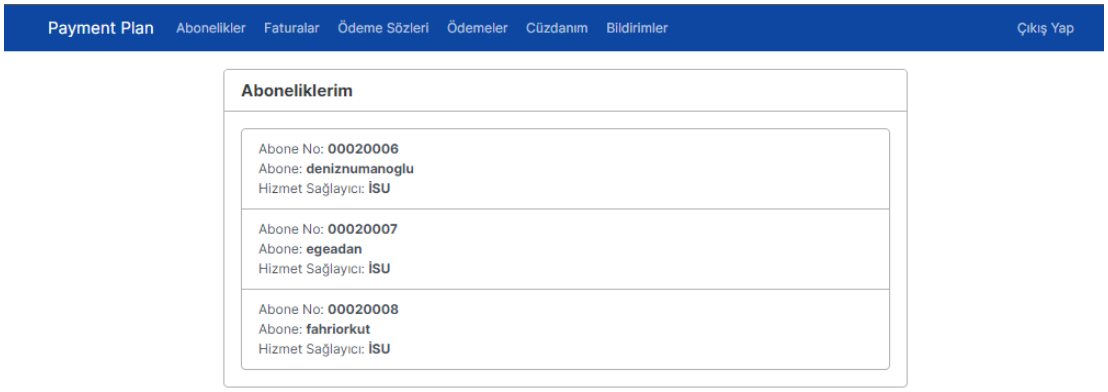
Screenshot



For Provider

```
// Sağlayıcının aboneleri
        subscriptions = await db.db(`SELECT sub.id, sub.is_valid, CONCAT(LPA
D(sub.provider_id, 4, '0000'), LPAD(sub.user_id, 4, '0000')) AS subscription
_number, sub.user_id, u.username, provider.corporate_name
                                    FROM subscription AS sub
                                    INNER JOIN user AS u ON u.id = sub.user
_id AND u.is_valid = 1
                                    INNER JOIN provider ON provider.id = su
b.provider_id AND provider.is_valid = 1
                                    WHERE sub.provider_id = ? AND sub.is_va
lid = 1
                                    ORDER BY sub.user_id, sub.id ASC`, [pro
vider.id]);
```

Screenshot



16.28. Add Subscription Screen

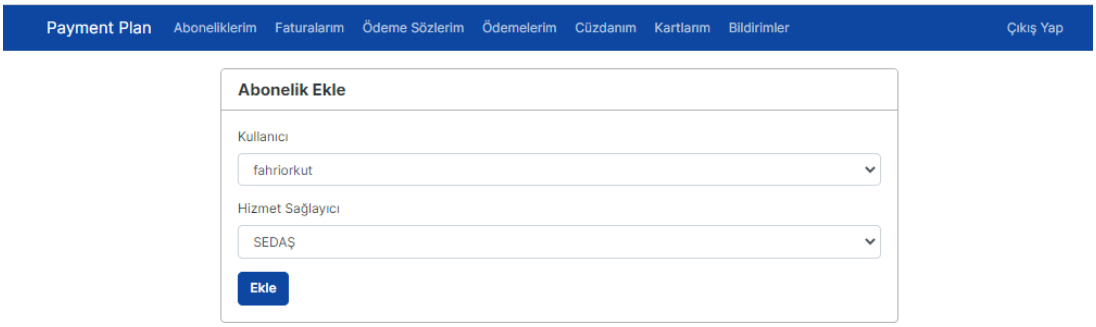
select_providers

```
// Abonenin aboneliği olmayan sağlayıcılar
    let providers = await db.db(`SELECT DISTINCT provider.id, provider.is_va
lid, corporate_name
                                FROM provider
                                LEFT JOIN subscription AS sub ON provider.i
d = sub.provider_id AND sub.user_id = ? AND sub.is_valid = 1
                                LEFT JOIN user ON user.id = sub.user_id AND
 user.is_valid = 1
                                WHERE provider.is_valid = 1 AND user.id IS
NULL
                                ORDER BY corporate_name ASC`, [req.auth_id]
);
```

insert_provider

```
await db.dbInsert("INSERT INTO subscription (user_id, provider_id, created_b
y) VALUES (?, ?, ?)",
        [req.body.user_id, req.body.provider_id, req.auth_id]);
```

Screenshot



16.29. Edit Subscription Screen

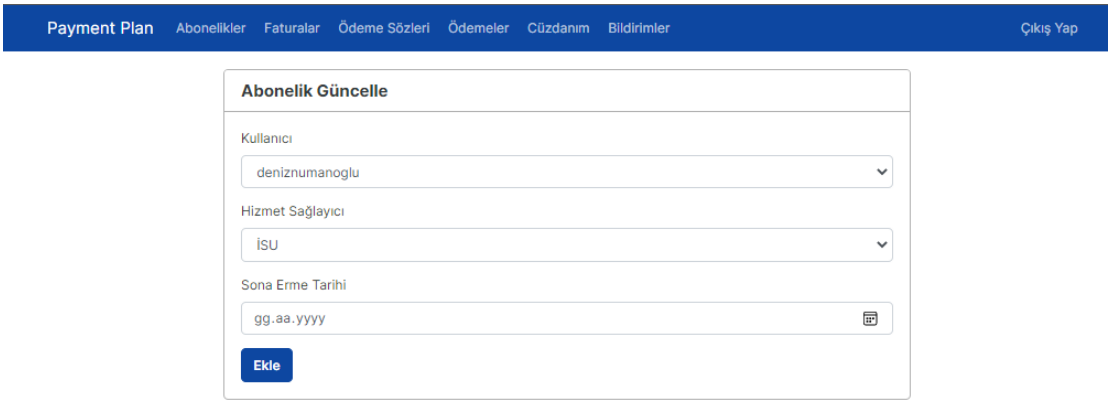
select_subscription

```
let subscription = await db.dbGetFirst(`SELECT sub.id, sub.is_valid, sub.use
r_id, CONCAT(LPAD(sub.provider_id, 4, '0000'), LPAD(sub.user_id, 4, '0000'))
 AS subscription_number, sub.user_id, sub.provider_id
                                          FROM subscription AS sub
                                          INNER JOIN provider ON provider.
id = sub.provider_id AND provider.is_valid = 1
                                          WHERE sub.id = ?`, [req.params.i
d]);
```

update_subscription

```
await db.dbUpdate(`UPDATE subscription SET user_id = ?, provider_id = ?, end
_date = ?, updated_by = ?, updated_at = current_timestamp()
                       WHERE id = ?`, [req.body.user_id, req.body.provider_i
d, req.body.end_date, req.auth_id, req.body.id]);
```

Screenshot



16.30. View Subscription Screen

select_subscription

```
let subscription = await db.dbGetFirst(`SELECT sub.id, sub.is_valid, CONCAT(
LPAD(sub.provider_id, 4, '0000'), LPAD(sub.user_id, 4, '0000')) AS subscript
ion_number, sub.user_id, sub.provider_id, sub.created_at
                                          FROM subscription AS sub
                                          INNER JOIN provider ON provider.
id = sub.provider_id AND provider.is_valid = 1
                                          WHERE sub.id = ?`, [req.params.i
d]);
```

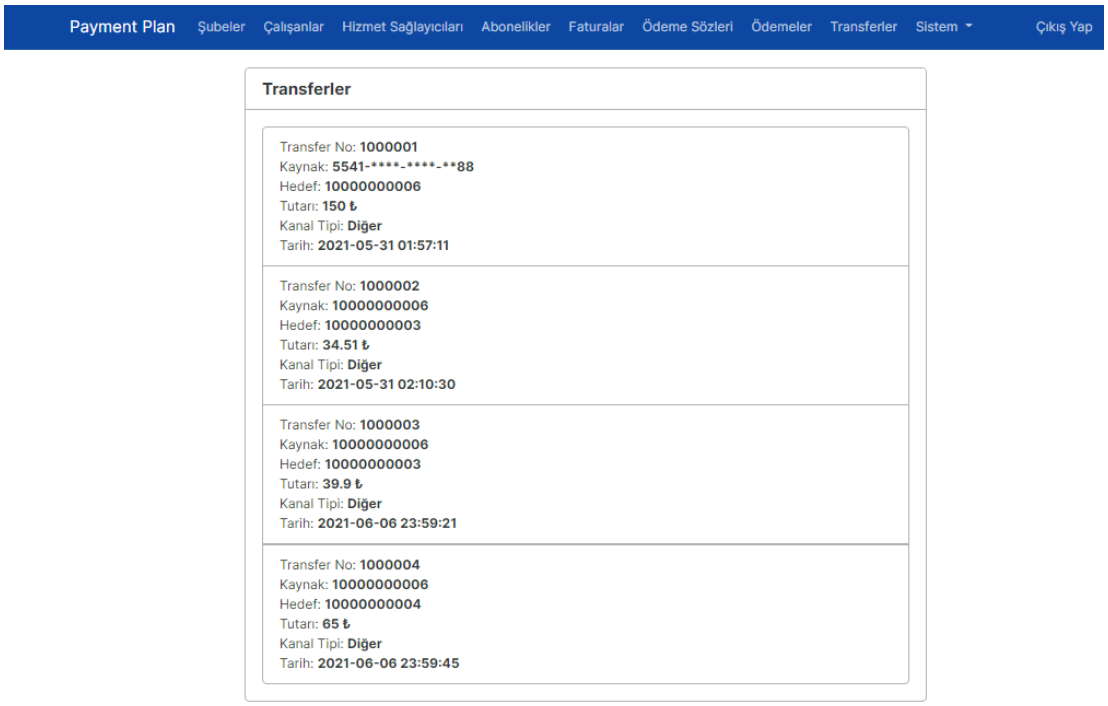
Screenshot

16.31. Transaction List Screen

select_transactions

```
let transactions = await db.db(`SELECT transaction_sequence_number AS seq_nu
m, source, destination, transaction_amount, channel_type_id, created_at
                                        FROM v_transactions`);
```

Screenshot

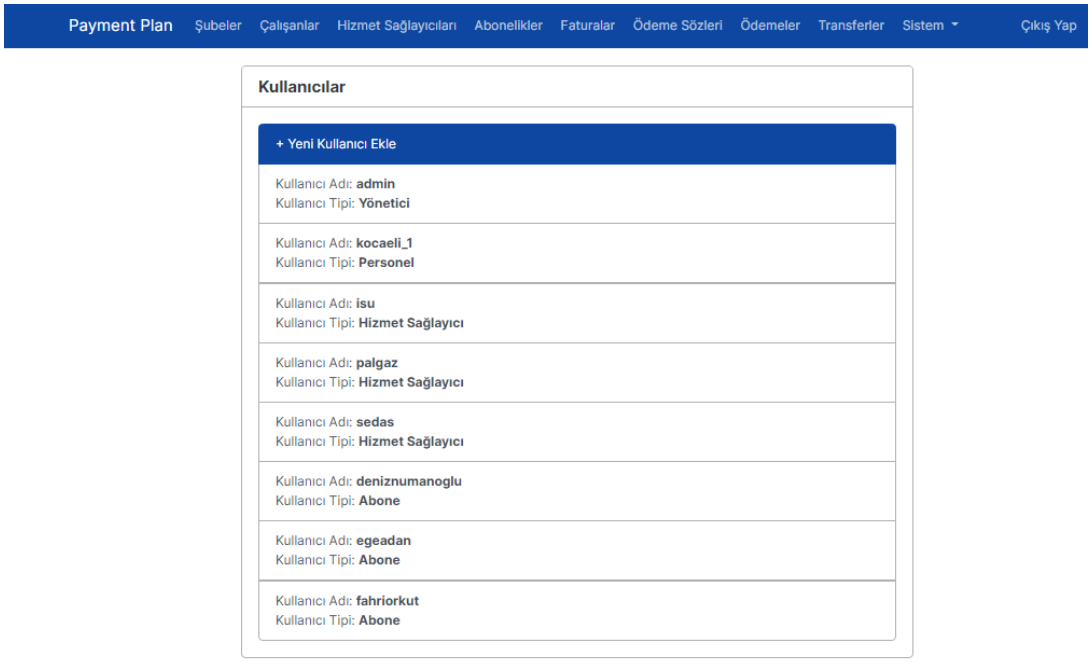


16.32. User List Screen

select_users

```
let users = await db.db(`SELECT id, is_valid, user_type_id, username
                         FROM user
                         ORDER BY user_type_id, username`);
```

Screenshot



16.33. Add User Screen

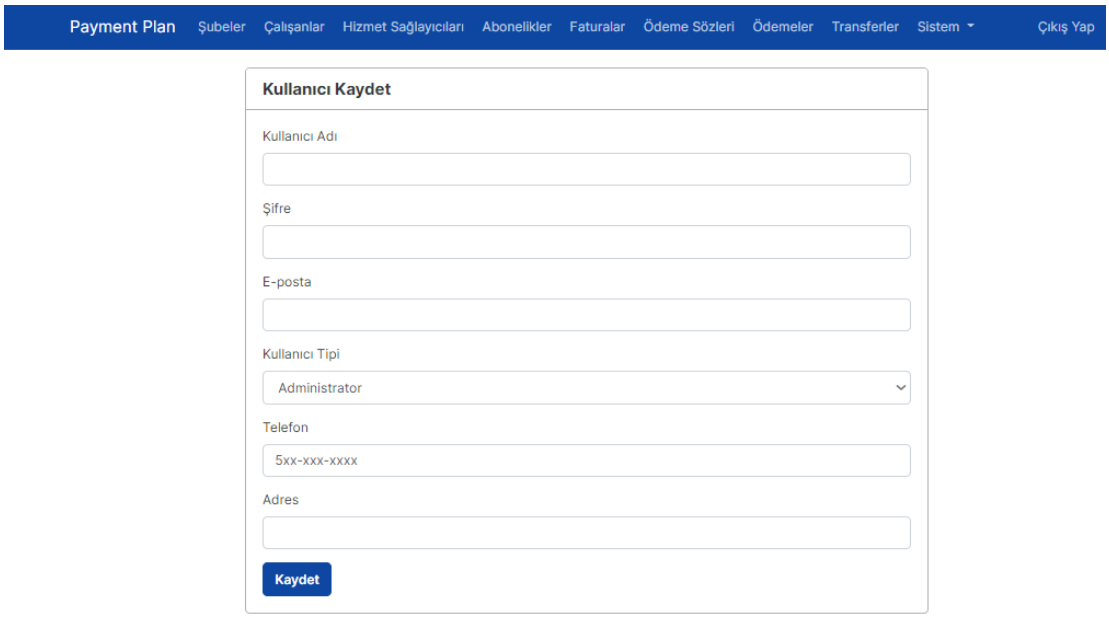
select_user_types

```
let userTypes = await db.db(`SELECT id, is_valid, code, value, description
                                FROM parameter
                                WHERE code = 'user_type' AND is_valid = 1`)
;
```

insert_user

```
await db.dbInsert(`INSERT INTO user (username, password, email, user_type_id
, phone, address, created_by)
                      VALUES (?, ?, ?, ?, ?, ?, ?)`, [req.body.username, re
q.body.password, req.body.email, req.body.user_type_id, req.body.phone, req.
body.address, req.auth_id]);
```

Screenshot

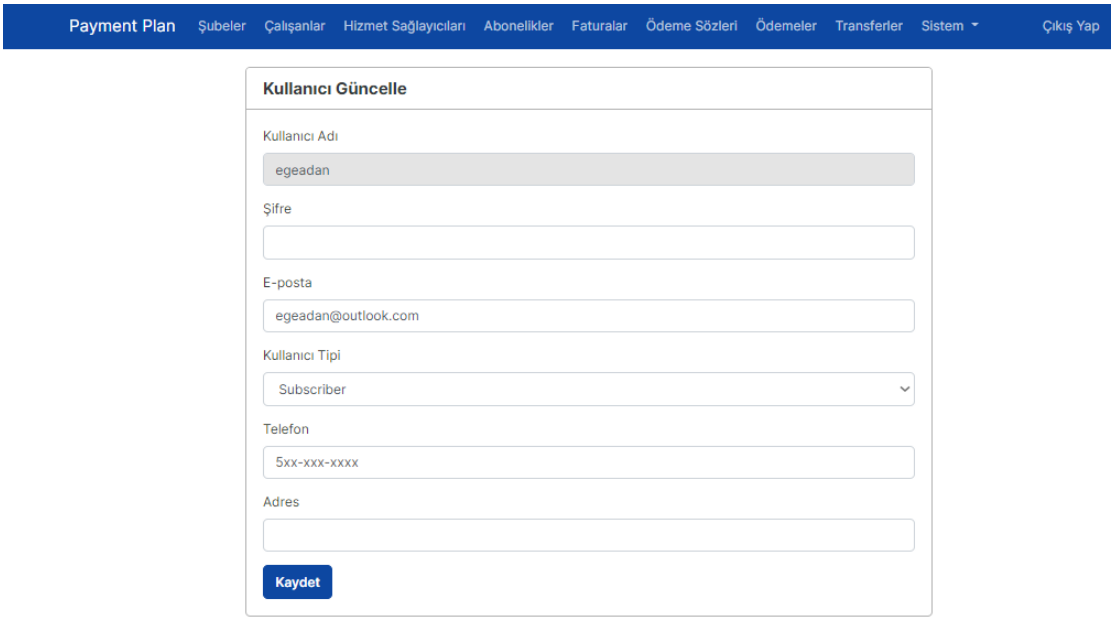


16.34. Edit User Screen

update_user

```
await db.dbUpdate(`UPDATE user SET password = ?, email = ?, user_type_id = ?
, phone = ?, address = ?, updated_by = ?, updated_at = current_timestamp()
                        WHERE id = ?`, [req.body.password, req.body.email, re
q.body.user_type_id, req.body.phone, req.body.address, req.auth_id, req.body
.id]);
```

Screenshot

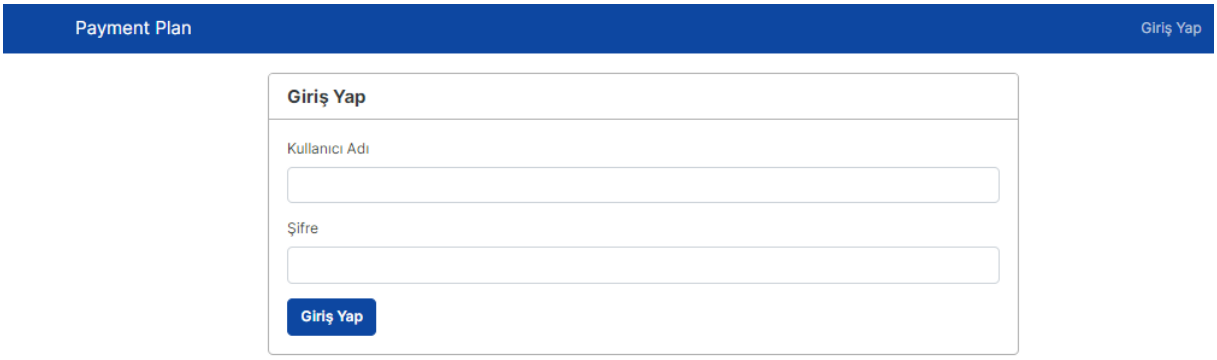


16.35. User Login Screen

login

```
let user = await db.dbGetFirst(`SELECT u.id, p.value AS user_type
                                FROM user AS u
                                LEFT JOIN parameter AS p ON p.id = u.use
r_type_id AND p.is_valid = 1
                                WHERE u.username = ? AND u.password = ?`
, [req.body.username, req.body.password]);
```

Screenshot



16.36. View Wallet Screen

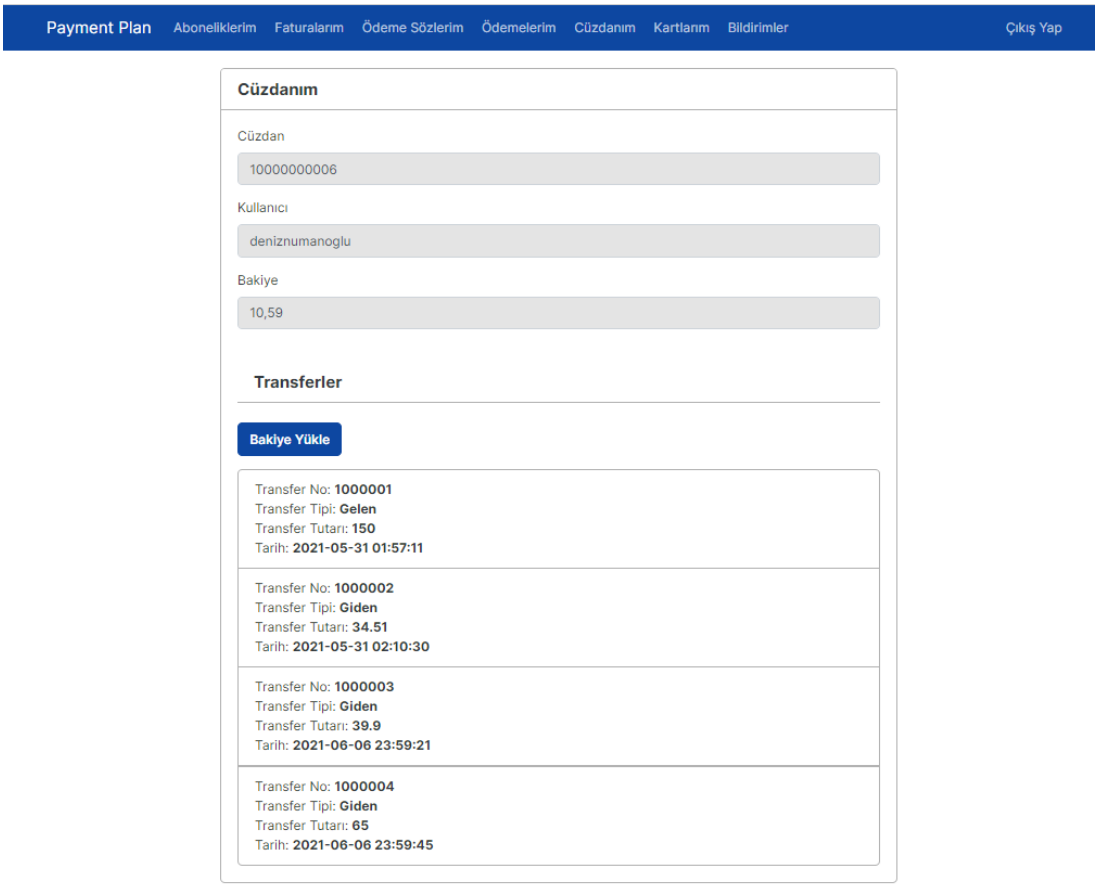
select_wallet

```
let wallet = await db.dbGetFirst(`SELECT wallet.id, wallet.is_valid, user.us
ername, wallet.balance, LPAD(wallet.user_id, 11, '10000000000') AS wallet_na
me
                                    FROM wallet
                                    INNER JOIN user ON user.id = wallet.us
er_id
                                    WHERE wallet.user_id = ?`, [req.auth_i
d]);
```

select_transactions

```
// Transferler
let transactions = await db.db(`SELECT t.* FROM
                                (SELECT transaction_sequence_number AS s
eq_num, transaction_type_id, transaction_amount, channel_type_id, created_at
                                                                    FROM
 transaction
                                                                    WHER
E transaction_source_id = 24 AND transaction_type_id = 22 AND is_valid = 1 A
ND source_id = ?
                                UNION
                                SELECT transaction_sequence_number AS se
q_num, transaction_type_id, transaction_amount, channel_type_id, created_at
                                                                    FROM
 transaction
                                                                    WHER
E transaction_source_id = 24 AND transaction_type_id = 23 AND is_valid = 1 A
ND destination_id = ?
                                ) AS t
                                ORDER BY t.created_at`, [wallet.id, wall
et.id]);
```

Screenshot



16.37. Upload to Wallet Screen

select_cards

```
let cards = await db.db(`SELECT id, is_valid, user_id, CONCAT(LEFT(card_numb
er, 4), '-', SUBSTR(card_number, 6, 2),'**-****-
**', RIGHT(card_number, 4)) AS card_number, card_name
                            FROM card
                            WHERE user_id = ? AND is_valid = 1`, [req.auth_
id]);
```

select_wallet

```
let wallet = await db.dbGetFirst(`SELECT id, is_valid, LPAD(user_id, 11, '10
000000000') AS wallet_name
                                    FROM wallet
                                    WHERE user_id = ? AND is_valid = 1`, [
req.auth_id]);
```

upload

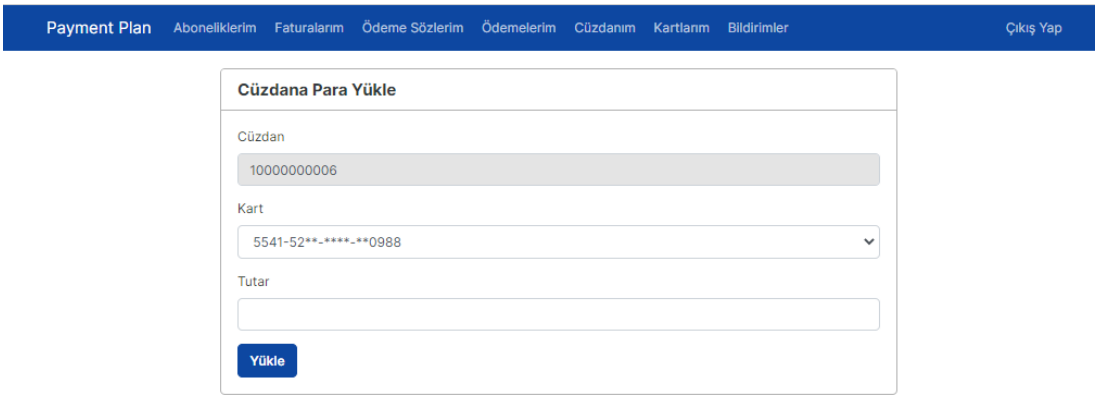
call_create_credit_transaction

```
// First Transaction
        await db.query(`CALL create_transaction (?, ?, ?, ?, ?, ?, ?, ?, @re
sult_val);`,
            [credit_id,
                external_id,
                other_channel_id,
                req.body.source_id,
                req.body.destination_id,
                req.body.transaction_amount,
                req.auth_id,
                seq_number], function (err, results) {
                    if (err) throw err;

                    db.query("SELECT @result_val AS result;", function (errC
rd, resultsCrd) {
                        if (errCrd) throw errCrd;
                        if (resultsCrd[0].result < 1) {
                            if (err) throw err;
                            res.resultError("Limit yetersiz!");
                            return;
                        }
                    });

                    console.log("First Transaction: ", results);
                });
```

call_create_debit_transaction

```
// Second Transaction
        await db.query("CALL create_transaction (?, ?, ?, ?, ?, ?, ?, ?, @re
sult_val);",
            [debit_id,
                internal_id,
                other_channel_id,
                req.body.source_id,
                req.body.destination_id,
                req.body.transaction_amount,
                req.auth_id,
                seq_number], function (err, results) {
                    if (err) throw err;

                    console.log("Second Transaction: ", results);
                });
```

Screenshot


Payment Plan
Aboneliklerim
Faturalarım
Ödeme Sözlerim
Ödemelerim
Cüzdanım
Kartlanım
Bildirimler
Çıkış Yap
Cüzdana Para Yükle
Cüzdan
10000000006
Kart
5541-52**-****-**0988
Tutar
Yükle

END OF THE REPORT

LAST UPDATE: 07.06.2021 03:30

STUDENT
Şeyda Nur DEMİR
12 10 44 042
LECTURER
Yrd. Doç. Dr. Burcu YILMAZ
ASSISTANT
-

KOCAELİ, 2021