Федеральное государственное автономное образовательное учреждение

высшего образования «Национальный исследовательский университет ИТМО»

*Факультет программной инженерии и компьютерной техники*

Отчет по Курсовой работе

Дисциплины “Информационные системы и базы данных”

Выполнила:

Е Хэн

Группа: Р33111

Преподаватель:

Санкт-Петербург, 2023г

**Этап 1**

Тема: Управление умным домом

**Описание предметной области**

На текущий момент, на рынке достаточно много приложений для управления умным домом. У всех компанией, которые создают умные дома, есть свои приложения для своих продуктов. Их процессы работы по сути похожи. Они позволяют пользователям самостоятельно модернизировать и создавать новые сценарии, которые будут удовлетворять потребности клиента. Например, увлажнитель начинает работать, если влажность воздуха ниже нормы или включается обогрев, когда температура опускается ниже заданной.

Но тут есть следующие проблемы. Во-первых, написание сценария может быть достаточно сложным, и пользователям трудно его создать, особенно, когда в доме достаточно много разнообразной умной мебели, и ее взаимодействие между собой сложное. Во-вторых, в существующих приложениях нет возможности делиться сценариями.

Поэтому мы хотим создать такое приложение, которое поддерживает общие сценарии и поставляется с готовыми, созданными программистами, чтобы помочь пользователям комфортно и эффективно управлять своим умным домом.

**Список сущностей**

|  |  |
| --- | --- |
| House | Где люди живут и хранят умную мебель |
| Furniture | Компоненты умного дома, такие как умные двери, умное освещение и т. д. |
| Problem | Умный дом – не панацея, в нем тоже будут bug или какие-то проблемы |
| Support\_man | Обслуживающий персонал, специалисты по ремонту умных домов |
| Action | Если пользователь предпримет соответствующие действия, умный дом даст обратную связь: например, если пользователь выйдет из комнаты, свет выключится сам. |
| User | Люди, которые используют эту систему приложений и облегчают с ее помощью свою жизнь |
| script | Целью скрипта (языка программирования), написанного Support\_man, является управление умным домом. |

**Бизнес-процесс**

1. Регистрация пользователя.
2. Авторизация пользователя.
3. Пользователь добавляет себя в дом в приложение. Пользователь просто присоединяется к другому дому.
4. Добавление комнат в дом. Пользователь добавляет комнаты в квартиру, и данные о них
5. Добавление умных вещей в помещение. Пользователь добавляет умные вещи в комнату.
6. Создание сценария по условию - умная вещь начинает работать, если выполняется какое-то условие (например, увлажнитель начинает работать, если влажность воздуха ниже нормы). Сценарий создан пользователем или программистом.
7. Создание сценария по расписанию - работа умных вещей по заданному расписанию. Сценарий создан пользователем или программистом.
8. Пользователь может поделиться сценариями с другими пользователями.
9. Пользователь управляет состоянием вещей при помощи сценариев.
10. Пользователь управляет состоянием вещей вручную.

**Этап 2**

# Описание предментной области:

|  |  |
| --- | --- |
| **Сушность** | **Классификация** |
| Script | Стержневая |
| House | Стержневая |
| Room | Стержневая |
| Furniture | Стержневая |
| Problem | Стержневая |
| Support\_man | Стержневая |
| Action | Стержневая |
| Contact | Характеристика |
| Address | Характеристика |
| User | Стержневая |
| Condition\_script | Характеристика |
| Schedule\_script | Характеристика |
| List\_Script\_User | Ассоциативная |
| List\_Action\_script | Ассоциативная |
| List\_User\_House | Ассоциативная |

## Стрежневая:

1. House
   1. id serial
   2. address text
   3. type\_house enum
2. Room
   1. id serial
   2. house\_id int
   3. area float
   4. height int
   5. Is\_filled boolean
   6. Type enum
3. Furniture
   1. id serial
   2. room\_id int
   3. type enum
   4. manufacture varchar(258)
   5. Available boolean
4. Script
   1. id serial
   2. Creator\_name varchar(64)
   3. Type enum
5. Action
   1. id serial
   2. Type\_furniture enum
   3. type enum
   4. description text
6. Support\_man
   1. Id serial
   2. name varchar(64)
   3. password varchar(256)
   4. is\_free boolean
7. Problem
   1. id serial
   2. description text
   3. user\_id int
   4. type enum
   5. support\_id int
   6. is\_finished boolean
8. User
   1. id int
   2. password varchar(256)
   3. Name varchar(64)
   4. Age int
   5. Address\_id int

## Характеристика:

1.Schedule\_script

* 1. script\_id int
  2. start\_time time
  3. end\_time time

1. Condition\_Script
   1. script\_id int
   2. condition text
2. Contact
   1. User\_id int
   2. Email varchar(128)
   3. Phone varchar(64)
3. Address
   1. id serial
   2. Contry enum
   3. City enum
   4. Street varchar(64)

## Ассоциативная:

1. List\_Action\_script
   1. id serial
   2. script\_id int
   3. action\_id int
2. List\_Script\_User
   1. id serial
   2. user\_id int
   3. script\_id int
3. List\_User\_house
   1. id serial
   2. user\_id int
   3. house\_id int

# Функциональные зависимости

## 1. User

id -> password

id -> name

id -> gender

id -> age

## 2.Contact

User\_id -> phone\_number

User\_id -> email

## 3. Problem

Id->description

Id-> user\_id

Id-> type

Id-> is\_finished

Id-> support\_id

## 4. House

Id->address\_id

Id->type\_house

## 5.Room

Id->area

Id->house\_id

Id->height

Id->is\_filled

Id -> type

## 6. Furniture

Id->room\_id

Id->type

Id->manufacture

Id->available

## 7.Action

Id->furniture\_id

Id->type

Id->Description

## 8. Support\_man

Id -> name

Id->password

Id->is\_free

## 9. Script

Id->creator\_name

Id->type

## 10. Schedule\_script

Script\_id -> start\_time

Script\_id -> end\_time

## 11. Condition\_Script

Script\_id -> condition

## 12. List\_Script\_User

Id -> user\_id

Id -> script\_id

## 13. List\_User\_House

Id -> user\_id

Id -> house\_id

## 14. List\_Action\_script

Id -> script\_id

Id -> action\_id

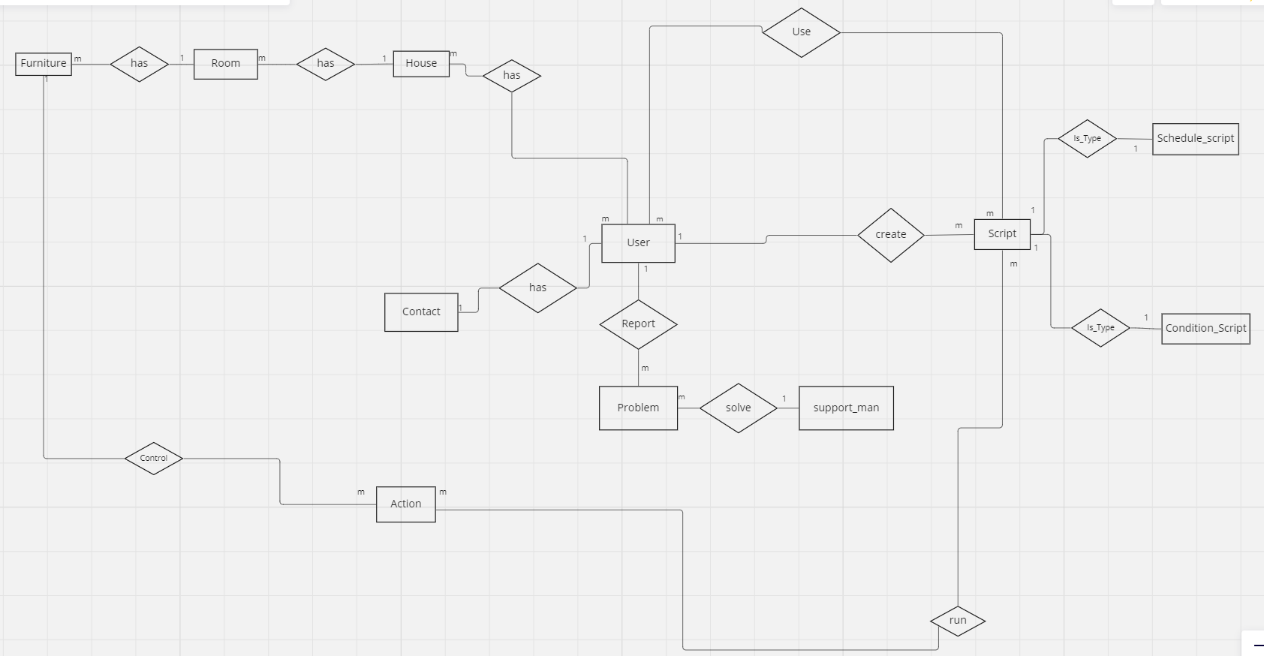
## Address

Id->contry

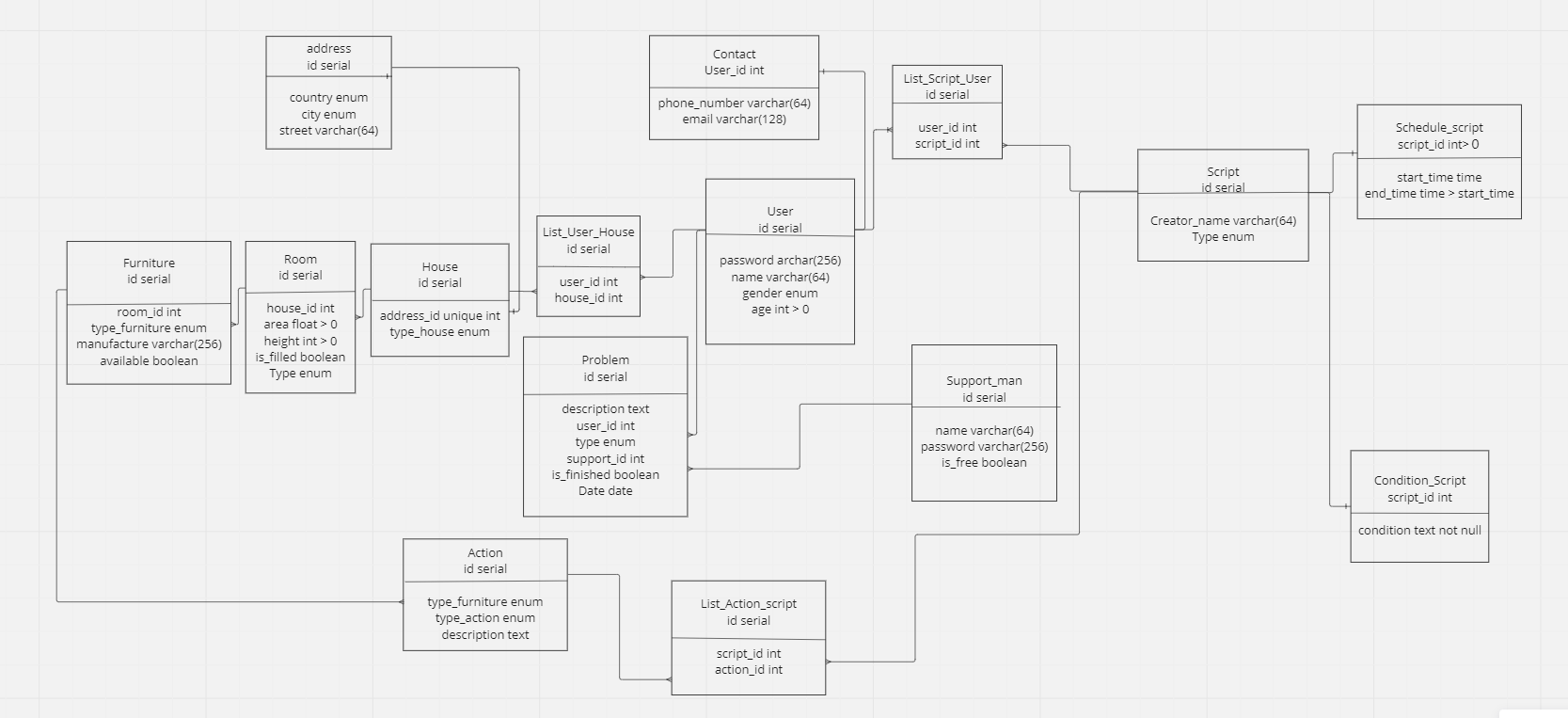
Id->city

Id->street

# Инфологическая модель



# Даталогическая модель



Все таблицы нормальзованы до 3 НФ. У остальных таблиц все атомарные атрибуты, нет частичных зависимостей или транзитивных зависимостей от первичного ключа.

**Этап 3**

DDL:

CREATE TYPE PROBLEM\_TYPE AS ENUM ('UI', 'BUGS', 'SCRIPT');

CREATE TYPE HOUSE\_TYPE AS ENUM ('APARTMENTS', 'VILLAS', 'HIGH-END','ORDINARY');

CREATE TYPE FURNITURE\_TYPE AS ENUM ('AIR\_CONDITION','LIGHT', 'HUMIDIFIER', 'BATHTUB', 'OUTLET','CURTAINS', 'FAN', 'CAMERA', 'WATER\_HEATER');

CREATE TYPE ACTION\_TYPE AS ENUM ('CLOSE','OPEN','SWITCH\_OFF','SWITCH\_ON','ADJUST\_VALUE','TURN\_ON','TURN\_OFF');

CREATE TYPE SCRIPT\_TYPE AS ENUM ('CONDITIONAL','SCHEDULE');

CREATE TYPE COUNTRY AS ENUM('US','UK','RUSSIAN','CHINA','FRANCE');

create type city as enum('Shanghai', 'Beijing', 'Shenzhen', 'Guangzhou', 'Chengdu','Paris', 'Marseille', 'Lyon', 'Toulouse','Cambridge', 'Edinburgh', 'London', 'Liverpool','New York', 'Los Angeles', 'Chicago', 'Boston');

create type room\_type as enum('KITCHEN','BEDROOM','BATHROOM','LIVING');

create type gender as enum ('MAN','WOMAN');

CREATE TABLE if not exists "user"(

id serial primary key not null ,

password varchar(256) not null ,

gender gender not null ,

name varchar(64) not null,

age int not null check ( age > 0 )

);

CREATE TABLE if not exists Support\_man(

id serial primary key not null ,

password varchar(256) not null,

name varchar(64) not null ,

is\_free boolean not null DEFAULT True

);

create table if not exists Problem(

id serial primary key not null ,

user\_id int not null REFERENCES "user"(id),

support\_man\_id int not null REFERENCES Support\_man(id),

is\_finished boolean not null DEFAULT false,

description text not null,

type problem\_type not null,

Data date default current\_date

);

create table if not exists address(

id serial not null primary key ,

country country not null ,

city city not null ,

street varchar(128) not null

);

create table if not exists House(

id serial primary key not null ,

address\_id int references address(id) not null ,

type house\_type not null

);

create table if not exists Room(

id serial primary key not null ,

house\_id int not null references House(id),

square float not null,

height float not null,

type room\_type not null,

is\_filled boolean not null default false

);

create table if not exists furniture(

id serial primary key not null ,

room\_id int references Room(id) not null ,

manufacture varchar(258) not null ,

Available boolean not null default True,

type furniture\_type not null

);

create table if not exists action(

id serial primary key not null ,

type\_furniture furniture\_type not null ,

type action\_type not null ,

description text

);

create table if not exists script(

id serial primary key not null ,

creator\_name varchar(256) not null,

type script\_type not null

);

create table if not exists schedule\_script(

script\_id int references script(id) not null ,

start\_time time not null ,

end\_time time not null

);

create table if not exists Condition\_script(

script\_id int references script(id) not null ,

condition text not null

);

create table if not exists contact(

user\_id int references "user"(id) not null unique,

email varchar(128) not null unique ,

phone varchar(64) not null unique

);

create table if not exists List\_Action\_Script(

id serial not null primary key ,

script\_id int references script(id),

action\_id int references action(id)

);

create table if not exists list\_script\_user(

id serial not null primary key ,

script\_id int references script(id),

user\_id int references "user"(id),

unique (script\_id,user\_id)

);

create table if not exists list\_user\_house(

id serial not null primary key ,

user\_id int references "user"(id),

house\_id int references House(id),

unique (user\_id,house\_id)

);

DML:

-- Inserting data into "user" table

INSERT INTO "user" (id, password, gender, name, age) VALUES

(1, 'password123', 'MAN', 'John Doe', 25),

(2, 'securepass', 'WOMAN', 'Jane Smith', 30),

(3, 'mypassword', 'MAN', 'Bob Johnson', 22);

-- Inserting data into Support\_man table

INSERT INTO Support\_man (id, password, name, is\_free) VALUES

(1, 'supportpass1', 'Support Person 1', true),

(2, 'supportpass2', 'Support Person 2', true),

(3, 'supportpass3', 'Support Person 3', false);

-- Inserting data into Problem table

INSERT INTO Problem (id, user\_id, support\_man\_id, description, type) VALUES

(1, 1, 1, 'UI issue', 'UI'),

(2, 2, 2, 'Bug in application', 'BUGS'),

(3, 3, 3, 'Script not working', 'SCRIPT');

-- Inserting data into address table

INSERT INTO address (id, country, city, street) VALUES

(1, 'US', 'New York', '123 Main St'),

(2, 'UK', 'London', '456 High St'),

(3, 'CHINA', 'Beijing', '789 East St');

-- Inserting data into House table

INSERT INTO House (id, address\_id, type) VALUES

(1, 1, 'APARTMENTS'),

(2, 2, 'VILLAS'),

(3, 3, 'HIGH-END');

-- Inserting data into Room table

INSERT INTO Room (id, house\_id, square, height, type, is\_filled) VALUES

(1, 1, 100.0, 3.0, 'KITCHEN', true),

(2, 2, 200.0, 4.0, 'BEDROOM', false),

(3, 3, 150.0, 3.5, 'BATHROOM', true);

-- Inserting data into furniture table

INSERT INTO furniture (room\_id, manufacture, Available, type) VALUES

(1, 'ABC Furniture', true, 'AIR\_CONDITION'),

(2, 'XYZ Furnishings', false, 'LIGHT'),

(3, 'Home Appliances Co.', true, 'BATHTUB');

-- Inserting data into action table

INSERT INTO action (id, type\_furniture, type, description) VALUES

(1, 'AIR\_CONDITION', 'SWITCH\_ON', 'Turn on air conditioner'),

(2, 'LIGHT', 'SWITCH\_OFF', 'Turn off lights'),

(3, 'BATHTUB', 'OPEN', 'Open bathtub water flow');

-- Inserting data into script table

INSERT INTO script (id, creator\_name, type) VALUES

(1, 'John Doe', 'CONDITIONAL'),

(2, 'Jane Smith', 'SCHEDULE'),

(3, 'Bob Johnson', 'CONDITIONAL');

-- Inserting data into schedule\_script table

INSERT INTO schedule\_script (script\_id, start\_time, end\_time) VALUES

(2, '08:00:00', '10:00:00'),

(3, '12:00:00', '14:00:00');

-- Inserting data into Condition\_script table

INSERT INTO Condition\_script (script\_id, condition) VALUES

(1, 'Temperature > 25'),

(3, 'Humidity < 30');

-- Inserting data into contact table

INSERT INTO contact (user\_id, email, phone) VALUES

(1, 'john.doe@example.com', '123-456-7890'),

(2, 'jane.smith@example.com', '987-654-3210'),

(3, 'bob.johnson@example.com', '555-123-4567');

-- Inserting data into List\_Action\_Script table

INSERT INTO List\_Action\_Script (script\_id, action\_id) VALUES

(1, 1),

(2, 2),

(3, 3);

-- Inserting data into list\_script\_user table

INSERT INTO list\_script\_user (script\_id, user\_id) VALUES

(1, 1),

(2, 2),

(3, 3);

-- Inserting data into list\_user\_house table

INSERT INTO list\_user\_house (user\_id, house\_id) VALUES

(1, 1),

(2, 2),

(3, 3);

Drop table:

drop table list\_script\_user cascade ;

drop table list\_user\_house cascade ;

drop table List\_Action\_Script cascade ;

drop table contact cascade ;

drop table schedule\_script cascade ;

drop table Condition\_script cascade ;

drop table script cascade ;

drop table action cascade ;

drop table furniture cascade;

drop table Room cascade ;

drop table House cascade ;

drop table address cascade ;

drop table Problem cascade ;

drop table Support\_man cascade ;

drop table "user" cascade ;

drop type room\_type;

DROP TYPE action\_type;

DROP TYPE script\_type;

DROP TYPE problem\_type;

DROP TYPE house\_type;

DROP TYPE furniture\_type;

Drop Type city;

drop type COUNTRY;

Remove content

truncate table list\_script\_user cascade ;

truncate table list\_user\_house cascade ;

truncate table List\_Action\_Script cascade ;

truncate table contact cascade ;

truncate table schedule\_script cascade ;

truncate table Condition\_script cascade ;

truncate table script cascade ;

truncate table action cascade ;

truncate table furniture cascade;

truncate table Room cascade ;

truncate table House cascade ;

truncate table address cascade ;

truncate table Problem cascade ;

truncate table Support\_man cascade ;

truncate table "user" cascade ;

Trigger:

CREATE OR REPLACE FUNCTION new\_problem() RETURNS TRIGGER AS $before\_insert\_problem\_trigger$

BEGIN

if ((select count(\*) from problem where support\_man\_id = NEW.support\_man\_id and Problem.is\_finished = false) >= 5) THEN

raise exception 'You cannot dispatch this problem to a busy support man';

ELSIF ((select count(\*) from problem where support\_man\_id = NEW.support\_man\_id and Problem.is\_finished = false) >= 4) THEN

update Support\_man set is\_free = false where Support\_man.id = NEW.support\_man\_id;

end if;

Return NEW;

end;

$before\_insert\_problem\_trigger$ LANGUAGE plpgsql;

CREATE TRIGGER before\_insert\_problem\_tri BEFORE INSERT ON Problem FOR EACH ROW EXECUTE PROCEDURE new\_problem();

CREATE OR REPLACE FUNCTION after\_update\_problem() RETURNS TRIGGER AS $after\_update\_problem\_trigger$

BEGIN

if((select count(\*) from problem where support\_man\_id = new.support\_man\_id and Problem.is\_finished = false)>=5) THEN

update Support\_man set is\_free = false where Support\_man.id = NEW.support\_man\_id;

ELSIF((select count(\*) from problem where support\_man\_id = new.support\_man\_id and Problem.is\_finished = false)<5) then

update Support\_man set is\_free = TRUE where Support\_man.id = NEW.support\_man\_id;

end if;

if((select count(\*) from problem where support\_man\_id = old.support\_man\_id and Problem.is\_finished = false)>=5) THEN

update Support\_man set is\_free = false where Support\_man.id = old.support\_man\_id;

ELSIF((select count(\*) from problem where support\_man\_id = old.support\_man\_id and Problem.is\_finished = false)<5) then

update Support\_man set is\_free = TRUE where Support\_man.id = old.support\_man\_id;

end if;

return new;

end;

$after\_update\_problem\_trigger$ LANGUAGE plpgsql;

CREATE TRIGGER after\_update\_problem\_tri AFTER UPDATE ON Problem FOR EACH ROW EXECUTE PROCEDURE after\_update\_problem();

CREATE OR REPLACE FUNCTION before\_update\_problem() RETURNS TRIGGER AS $before\_update\_problem\_trigger$

BEGIN

if((select count(\*) from problem where support\_man\_id = new.support\_man\_id and Problem.is\_finished = false)>=5 and new.is\_finished = false) THEN

raise exception 'Dispatch a problem to a busy support man is forbidden';

end if;

return new;

end;

$before\_update\_problem\_trigger$ LANGUAGE plpgsql;

CREATE TRIGGER before\_update\_problem\_tri BEFORE UPDATE ON Problem FOR EACH ROW EXECUTE PROCEDURE before\_update\_problem();

CREATE OR REPLACE FUNCTION new\_furniture() returnS TRIGGER AS $insert\_furniture\_trigger$

BEGIN

IF ((select count(\*) from furniture where room\_id = new.room\_id) >= 10) THEN

raise exception 'Room is filled.';

ELSIF ((select count(\*) from furniture where room\_id = new.room\_id) >= 9) THEN

update room set is\_filled = true where room.id = new.room\_id;

end if;

Return new;

end ;

$insert\_furniture\_trigger$ LANGUAGE plpgsql;

CREATE TRIGGER insert\_furniture\_tri BEFORE INSERT ON furniture FOR EACH ROW EXECUTE PROCEDURE new\_furniture();

Remove trigger

drop trigger before\_insert\_problem\_tri on problem;

drop trigger after\_update\_problem\_tri on problem;

drop trigger before\_update\_problem\_tri on problem;

drop trigger insert\_furniture\_tri on furniture;

# Создание идексов

CREATE INDEX problem\_index ON problem (support\_man\_id);

Create Index furniture\_index on furniture(room\_id);

create index schedule\_script\_index on schedule\_script(script\_id);

create index condition\_script\_index on condition\_script(script\_id);

create index list\_action\_script\_index on list\_action\_script(script\_id);

drop trigger before\_insert\_problem\_tri on problem;

drop trigger after\_update\_problem\_tri on problem;

drop trigger before\_update\_problem\_tri on problem;

drop trigger insert\_furniture\_tri on furniture;

# Бизнес-Процессы

1. Регистрация пользователя.

create or replace function *register*(name\_u varchar(64),password\_u varchar(256), age\_u int, gender\_u gender,phone\_u varchar(64),email\_u varchar(128)) returns setof "user" as $$  
 DECLARE  
 new\_id int;  
 BEGIN  
 insert into "user"(password, gender, name, age) values (password\_u, gender\_u, name\_u, age\_u);  
 select id from "user" order by id desc limit 1 into new\_id;  
 insert into contact(user\_id, email, phone) values (new\_id, email\_u, phone\_u);  
 return query select \* from "user" order by id desc limit 1;  
end;  
$$ language plpgsql;

1. Авторизация пользователя.

create or replace function *login\_by\_phone*(phone\_u varchar(64),password\_u varchar(256)) returns bool as $$

BEGIN

if (select *count*(\*) from "user" inner join contact c on "user".id = c.user\_id where "user".password = password\_u and phone = phone\_u) = 1 then

return true;

else

return false;

end if;

end;

$$ language plpgsql;

create or replace function *login\_by\_email*(email\_u varchar(128),password\_u varchar(256)) returns bool as $$

BEGIN

if (select *count*(\*) from "user" inner join contact c on "user".id = c.user\_id where "user".password = password\_u and email = email\_u) = 1 then

return true;

else

return false;

end if;

end;

$$ language plpgsql;

select *login\_by\_email*(' 0@gmail.com', '481.6490150315751');

1. Пользователь добавляет себя в дом в приложение. Пользователь просто присоединяется к другому дому.

create or replace function *add\_user\_to\_home*(user\_id\_u int, home\_id int) returns setof list\_user\_house as $$

BEGIN

insert into list\_user\_house(user\_id, house\_id) VALUES (user\_id\_u, home\_id);

return query select \* from list\_user\_house order by id desc limit 1;

end;

$$ language plpgsql;

1. Добавление комнат в дом. Пользователь добавляет комнаты в квартиру, и данные о них.

create or replace function *add\_room\_to\_home*(house\_id\_u int, square\_u float,height\_u float, t room\_type) returns setof room as $$

BEGIN

insert into room(house\_id, square, height, type) VALUES (house\_id\_u, square\_u, height\_u, t);

return query select \* from room order by id desc limit 1;

end;

$$ language plpgsql;

1. Добавление умных вещей в помещение. Пользователь добавляет умные вещи в комнату.

create or replace function *add\_furniture\_to\_room*(room int, manu varchar(64), avai bool,ft furniture\_type) returns setof furniture as $$

BEGIN

insert into furniture(room\_id, manufacture, Available, type) VALUES (room, manu, avai, ft);

return query select \* from furniture order by id desc limit 1;

end;

$$ language plpgsql;

1. Создание сценария по условию - умная вещь начинает работать, если выполняется какое-то условие (например, увлажнитель начинает работать, если влажность воздуха ниже нормы). Сценарий создан пользователем или программистом.

create or replace function *add\_condition\_script*(con text, name varchar(256)) returns setof script as $$

DECLARE

new\_id int = 0;

BEGIN

insert into script(creator\_name, type) VALUES (name, 'CONDITIONAL');

select script.id from script order by id desc limit 1 into new\_id;

insert into condition\_script(script\_id, condition) VALUES (new\_id, con);

return query select \* from script order by id desc limit 1;

end;

$$ language plpgsql;

1. Создание сценария по расписанию - работа умных вещей по заданному расписанию. Сценарий создан пользователем или программистом.

create or replace function *add\_schedule\_script*(st time,et time, name varchar(256)) returns setof script as $$

DECLARE

new\_id int = 0;

BEGIN

insert into script(creator\_name, type) VALUES (name, 'SCHEDULE');

select script.id from script order by id desc limit 1 into new\_id;

insert into schedule\_script(script\_id, end\_time,start\_time) VALUES (new\_id, et,st);

return query select \* from script order by id desc limit 1;

end;

$$ language plpgsql;

1. Пользователь может поделиться сценариями с другими пользователями.

create or replace function *add\_user\_script*(ui int , si int) returns setof list\_script\_user as $$

BEGIN

insert into list\_script\_user(script\_id, user\_id) VALUES (si,ui);

return query select \* from list\_script\_user order by id desc limit 1;

end;

$$ language plpgsql;

**Этап 4**

Тема: Это система управления умным домом, помогающая нам лучше использовать устройства вокруг нас.

## Технологии

### Backend

* Spring Boot：Создавайте и развертывайте приложения быстрее
* PostgreSQL：Серверные приложения взаимодействуют с базой данных

### Frontend

·React：Популярные библиотеки JavaScript

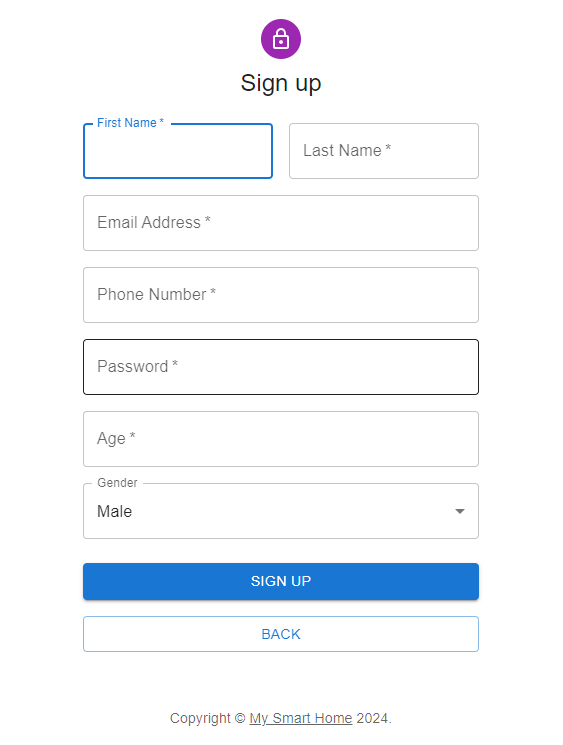
·useState ：Храните и обновляйте данные о состоянии компонентов.

·JQuery：Использовал функцию ajax JQuery для отправки запросов на серверную часть.

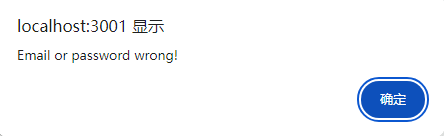
**Пользовательские сценарии**

**Незалогиненный пользователь:**

·Можно зарегистрировать аккаунт

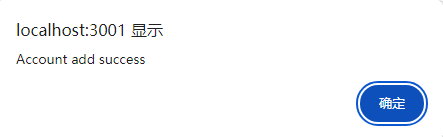


·Невозможно использовать программное обеспечение

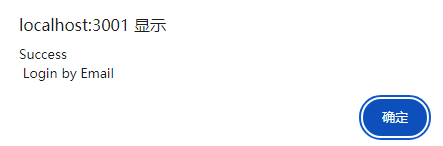


**Залогиненный пользователь:**

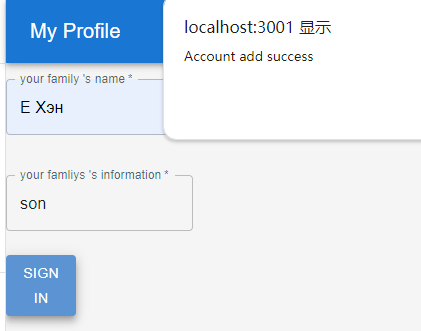
Успешная регистрация



Успешно авторизован

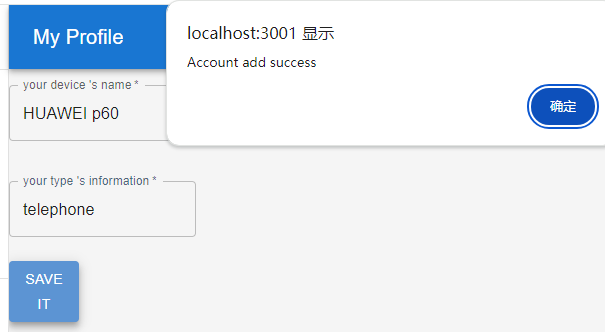


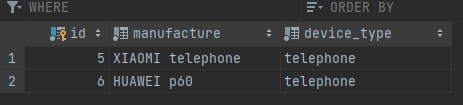
Возможность добавлять членов своей семьи в систему умного дома.





Возможность добавления собственных устройств в систему умного дома.





**Менеджер:**

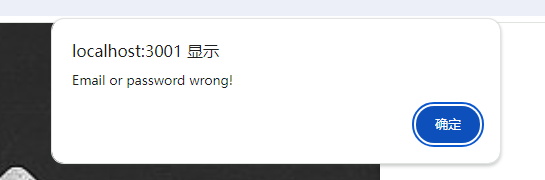
Может просматривать систему каждого

Изменение статуса каждого устройства или учетной записи

Добавить новое устройство или запись

**Некоторые другие особенности:**

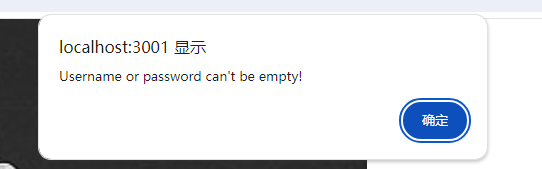
При вводе неправильного пароля будет сообщено об ошибке.



При многократной регистрации одного и того же номера мобильного телефона или адреса электронной почты



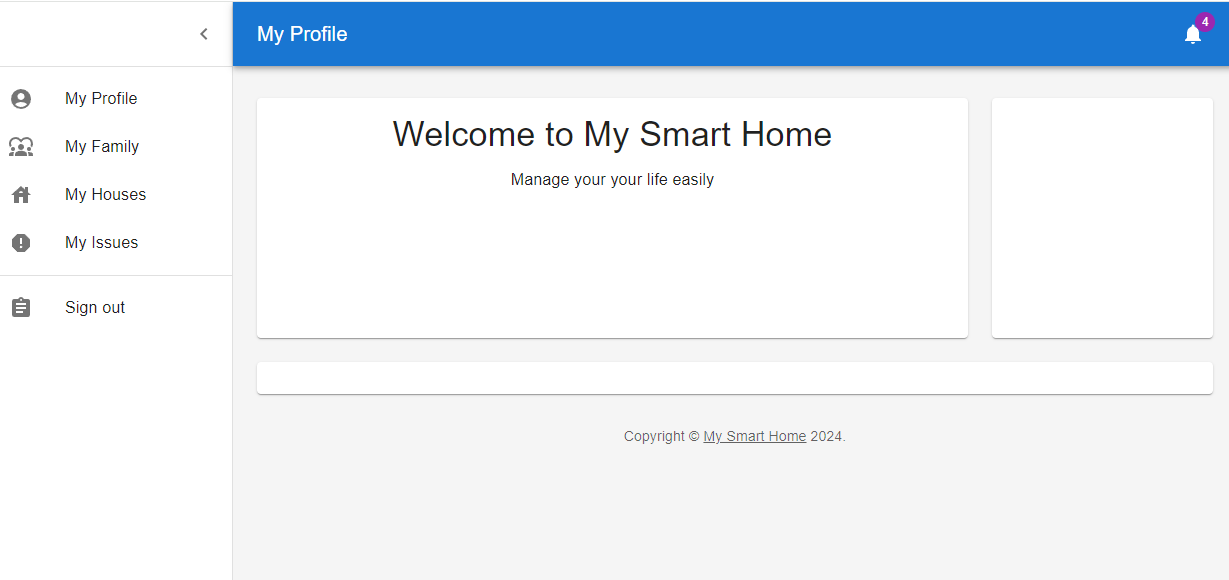
Когда мы ничего не вводим



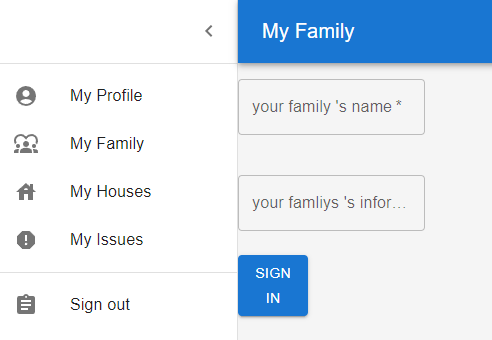
При вводе недопустимых символов



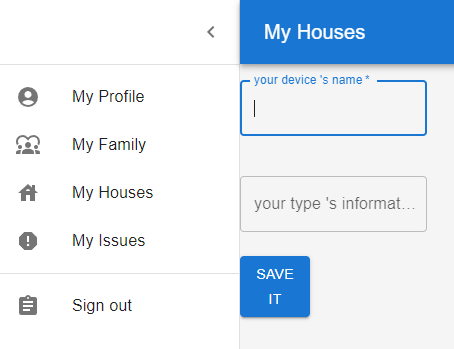
интерфейс пользовательского интерфейса（Взаимодействуйте с помощью кнопок слева）



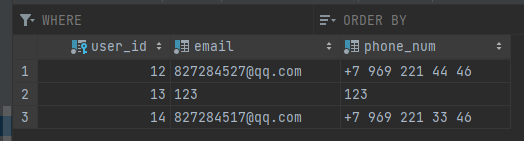
Добавить интерфейс члена семьи



Добавить интерфейс устройства



Вы можете связаться с другими людьми через зарегистрированный номер мобильного телефона.



Кнопка выхода, чтобы вернуться в основной интерфейс