Описание решения

Содержание

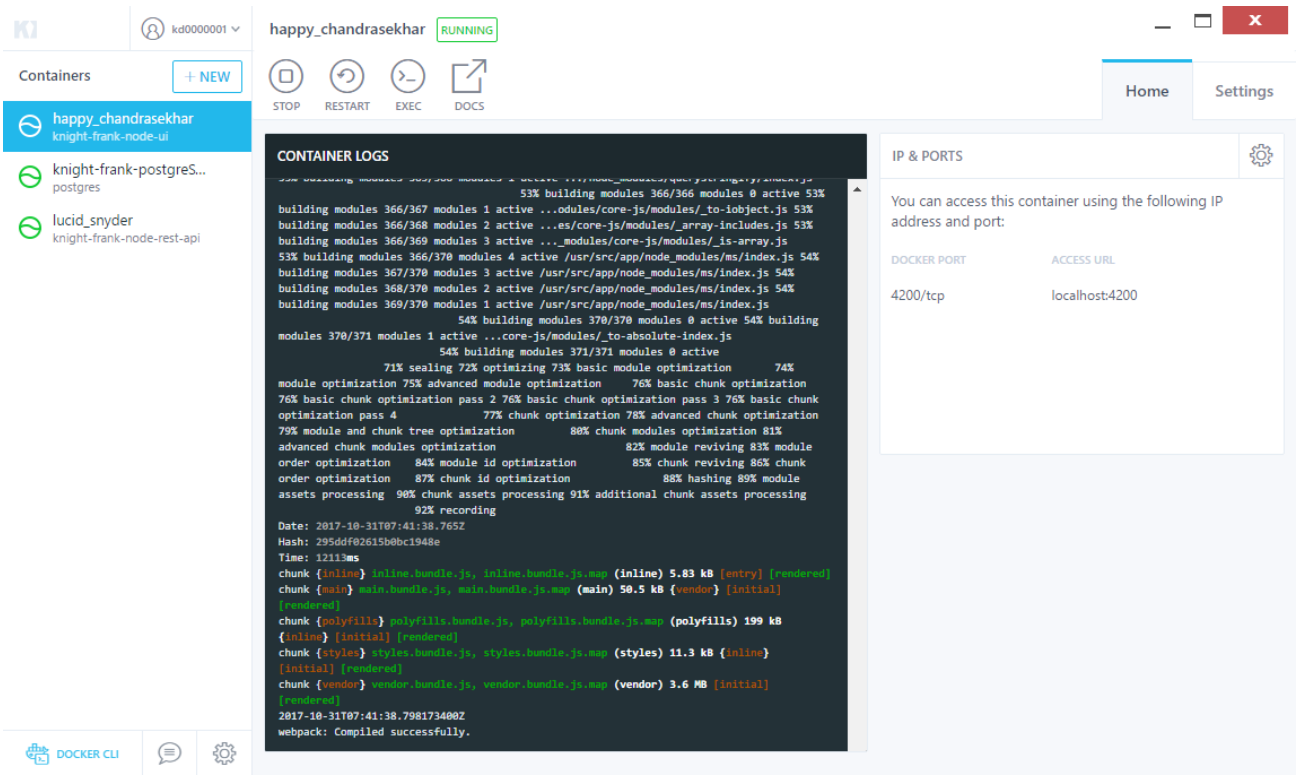
Общее описание

Решение размещено по трём Docker-контейнерам Linux, содержащим каждый уровень приложения.

Для развёртывания достаточно запустить соответствующие Dockerfile (для запуска Dockerfile, в той же папке, имеется соответствующий ему файл DockerStart.cmd, для запуска контейнера):

- \Database\Scripts\Dockerfile - сервер базы данных PostgreSQL
- \Service\Rest\Dockerfile - веб-сервис Node.js
- \UI\ Dockerfile - интерфейс пользователя Angular 4, на веб-сервере Node.js

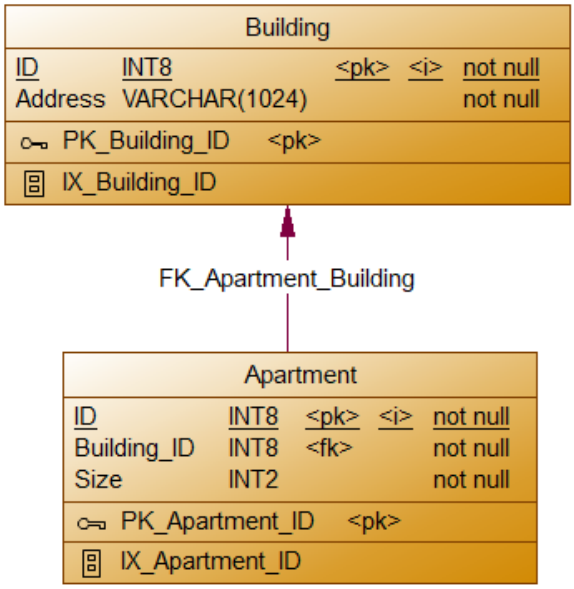
Используя Kitematic, мы можем наблюдать три запущенных контейнера:



База данных

В прилагаемом архиве папка \Database

Согласно требованию, используется PostgreSQL, привожу диаграмму физической модели (Sybase PowerDesigner):



В прилагаемом архиве (папка “Database”) имеется диаграмма и скрипты pdplSQL создания структуры таблиц, а также генерации тестовых данных, со случайным их распределением.

Серверная часть

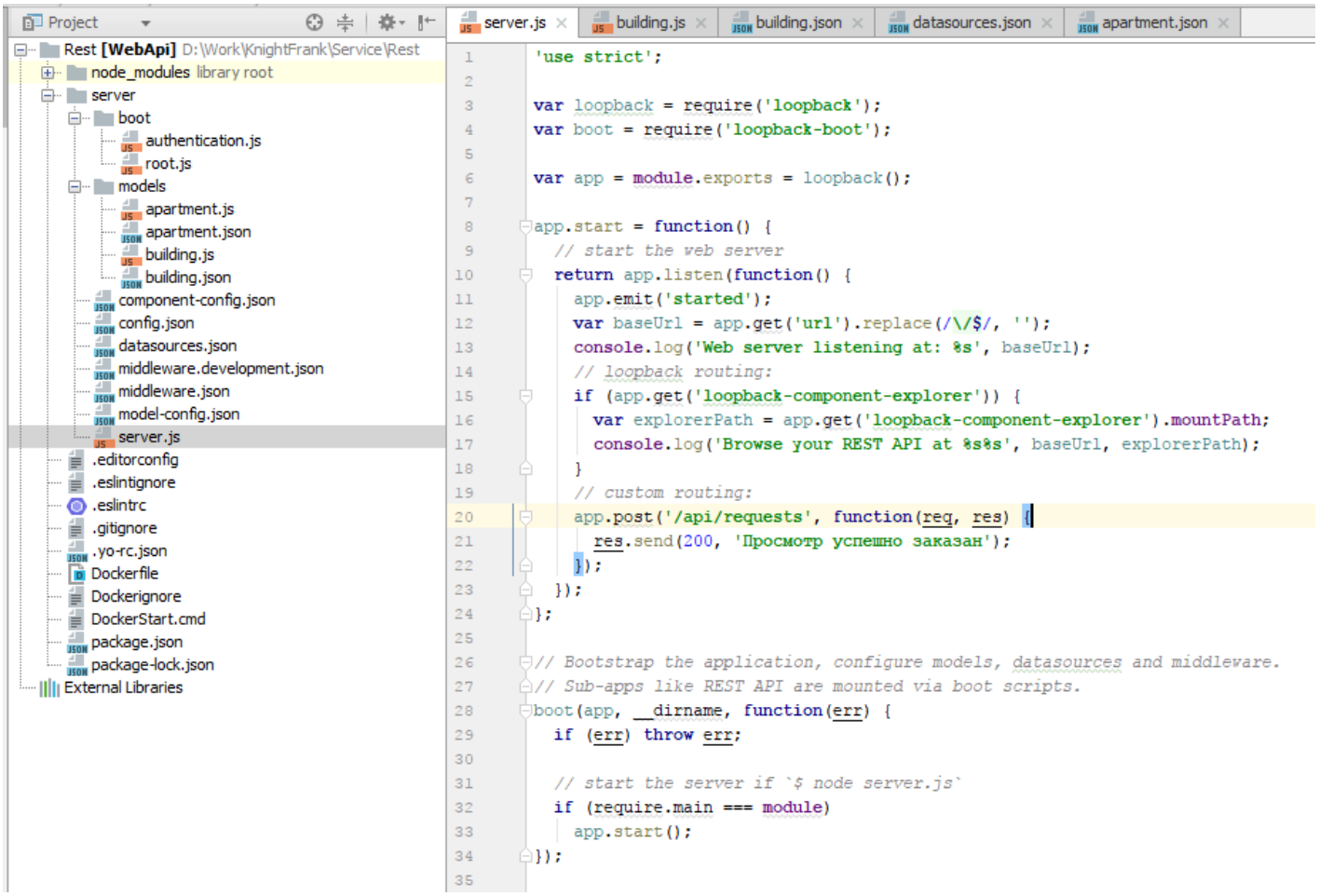
В прилагаемом архиве папка \Service\Rest

Согласно требованию, используется Node.js в качестве платформы для REST-сервиса. Используются библиотеки Express.js и основанная на ней Loopback.io, добавляющая возможности ORM-системы для реляционных данных, а также возможность генерации VIEW-модели.

Loopback позволяет как читать данные из реляционных таблиц, так и выполнять хранимые функции и SQL-запросы с параметрами, при необходимости выборок на стороне сервера.

Для данного решения это оказалось необходимым, т.к. для выборок понадобилась агрегация и соединение таблиц, что оптимально делать в реляционной СУБД, а не на стороне сервиса.

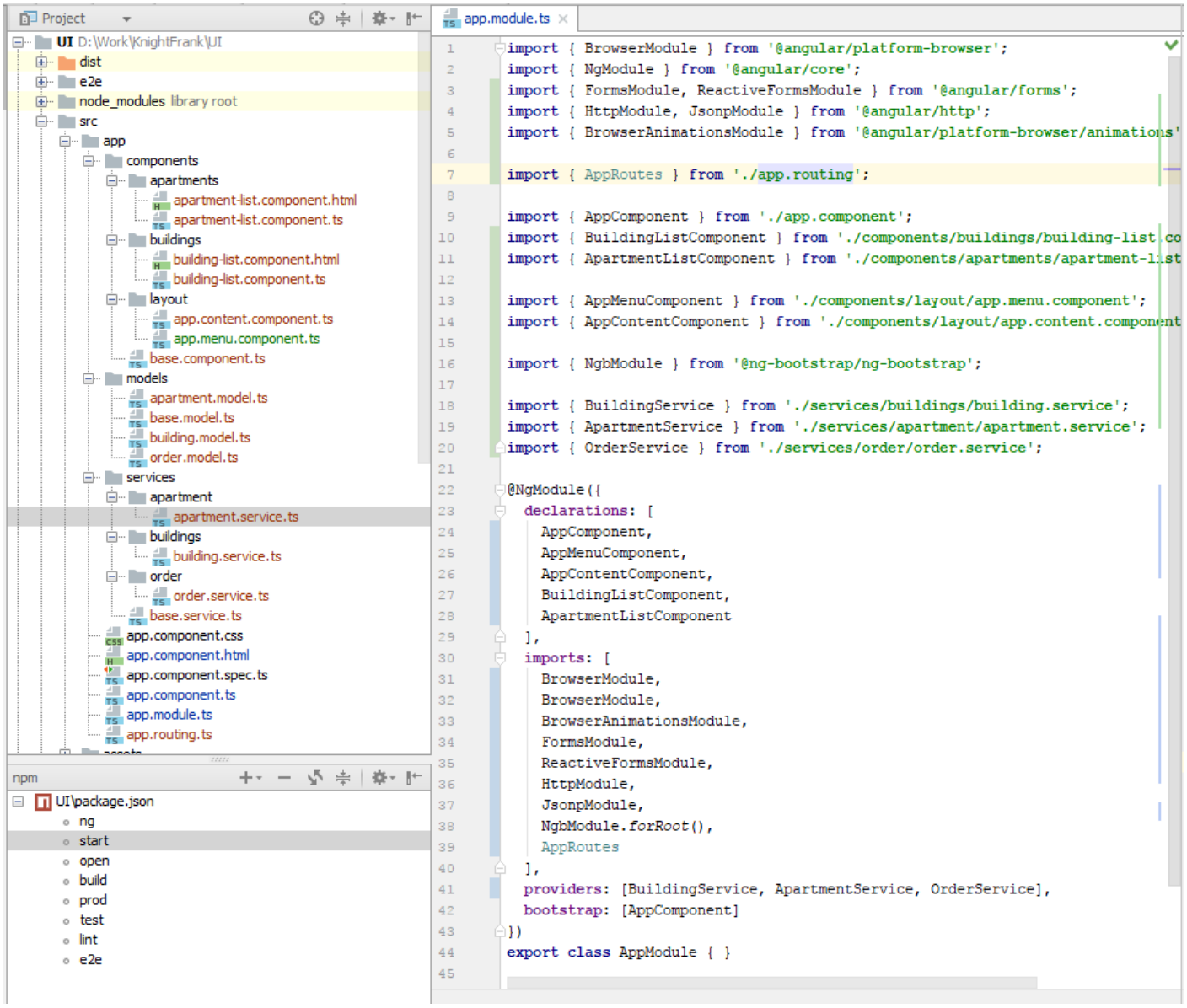
Структура серверной части проекта:


Project
Rest [WebApi] D:\Work\KnightFrank\Service\Rest
node_modules library root
server
boot
authentication.js
root.js
models
apartment.js
apartment.json
building.js
building.json
component-config.json
config.json
datasources.json
middleware.development.json
middleware.json
model-config.json
server.js
.editorconfig
.eslintignore
.eslintrc
.gitignore
.yo-rc.json
Dockerfile
Dockerignore
DockerStart.cmd
package.json
package-lock.json
External Libraries
server.js
building.js
building.json
datasources.json
apartment.json
'use strict';
var loopback = require('loopback');
var boot = require('loopback-boot');
var app = module.exports = loopback();
app.start = function() {
// start the web server
return app.listen(function() {
app.emit('started');
var baseUrl = app.get('url').replace(/\/$/, '');
console.log('Web server listening at: %s', baseUrl);
// loopback routing:
if (app.get('loopback-component-explorer')) {
var explorerPath = app.get('loopback-component-explorer').mountPath;
console.log('Browse your REST API at %s%s', baseUrl, explorerPath);
}
// custom routing:
app.post('/api/requests', function(req, res) {
res.send(200, 'Просмотр успешно заказан');
});
});
};
// Bootstrap the application, configure models, datasources and middleware.
// Sub-apps like REST API are mounted via boot scripts.
boot(app, __dirname, function(err) {
if (err) throw err;
// start the server if `$ node server.js`
if (require.main === module)
app.start();
});

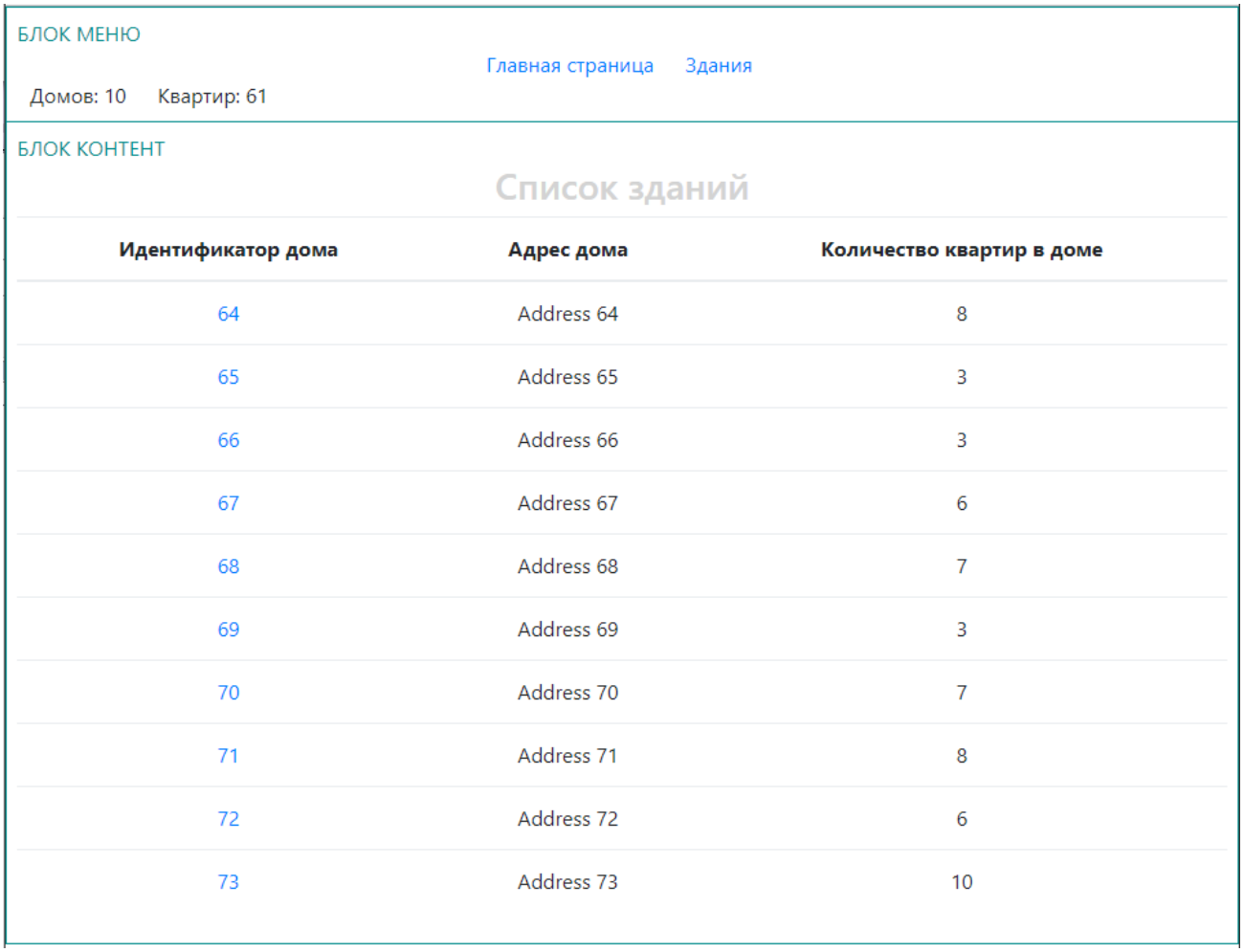
Клиентская часть

В прилагаемом архиве папка \UI

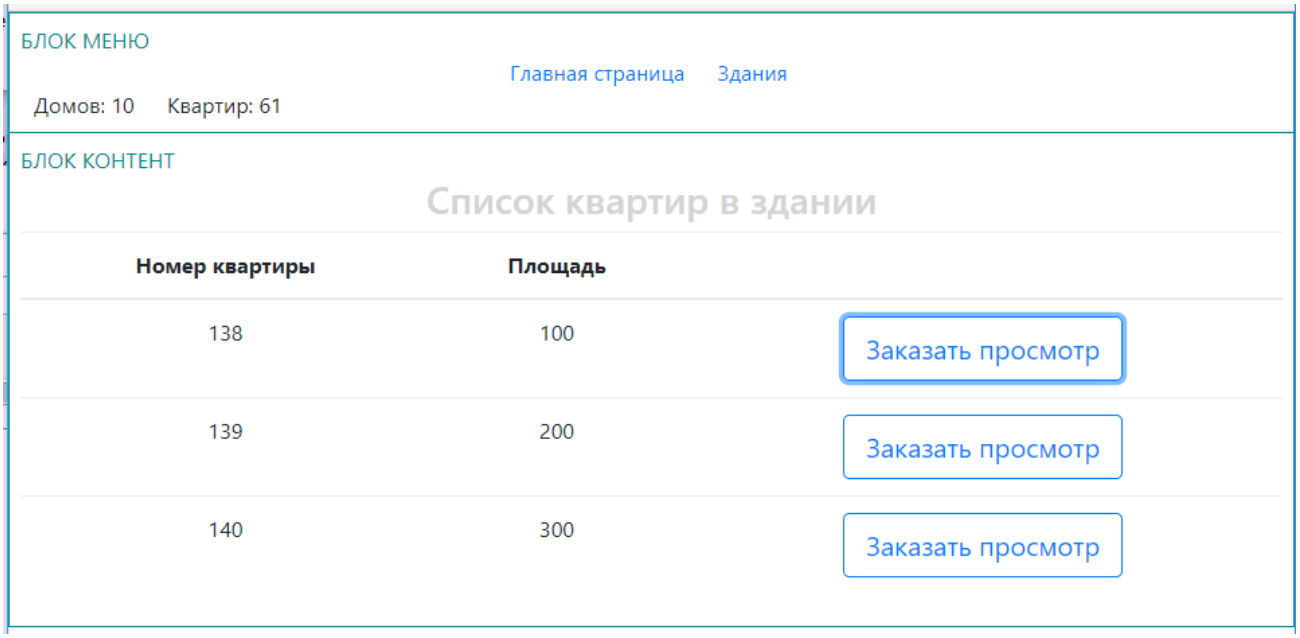
Структура клиентской части проекта:



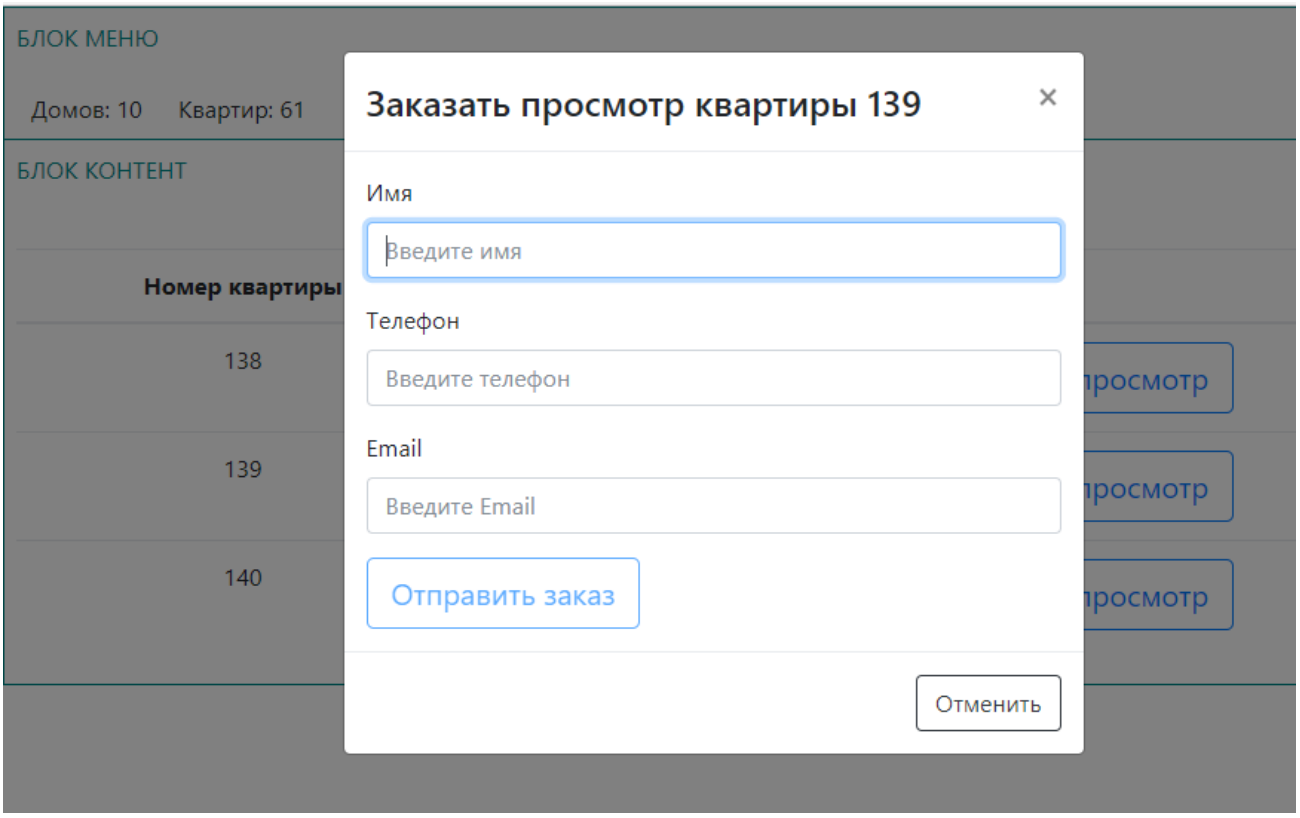
Ниже скриншот формы “Список зданий, в html-разметке специально показаны границы областей – “БЛОК МЕНЮ” и “БЛОК КОНТЕНТ” (в контенте располагается router-outlet, в который подгружаются все прочие компоненты, вызываемые из меню):



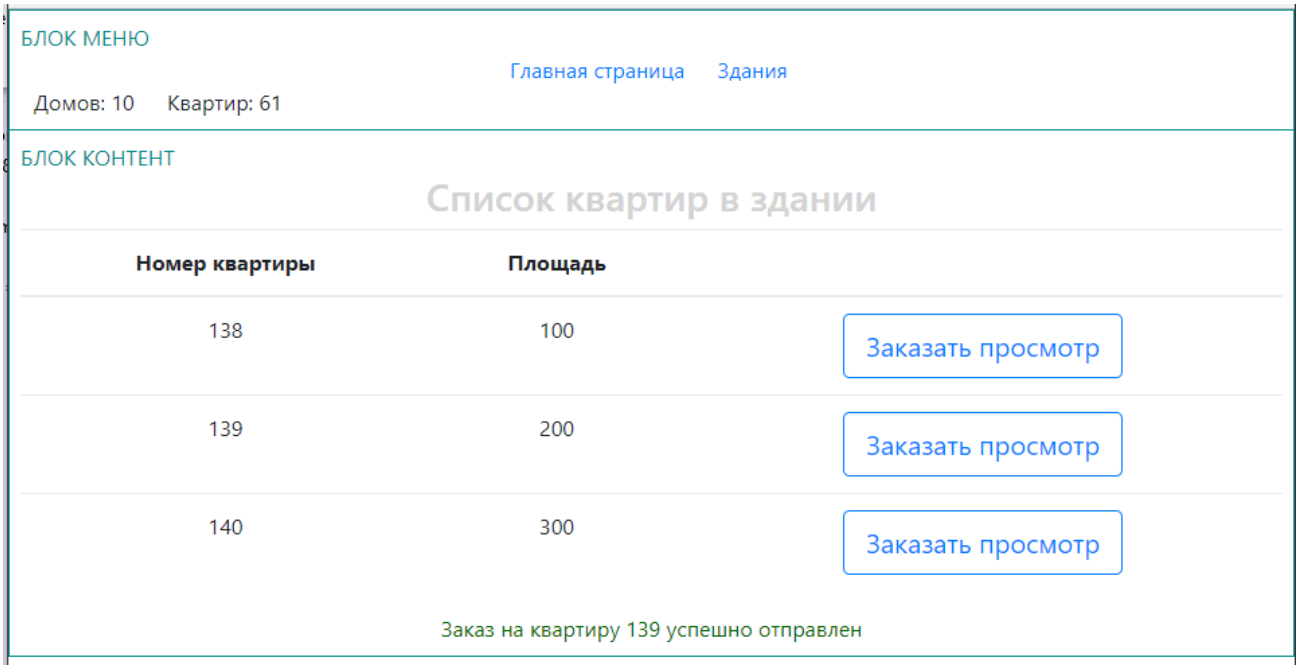
Ниже скриншот формы со списком квартир, вызываемой по клику на идентификаторе дома:



По клику на кнопке “Заказать просмотр” открывается модальное окно ng-bootstrap с формой, позволяющей оправить привязанную к форме модель Order методом Post на Rest-сервис.



В случае успешной отправки, (после отработки Promise), на форме будет показано сообщение об этом.



Тестирование