

Описание решения

Содержание

Общее описание.....	1
База данных	2
Серверная часть.....	2
Клиентская часть	4
Тестирование	6

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

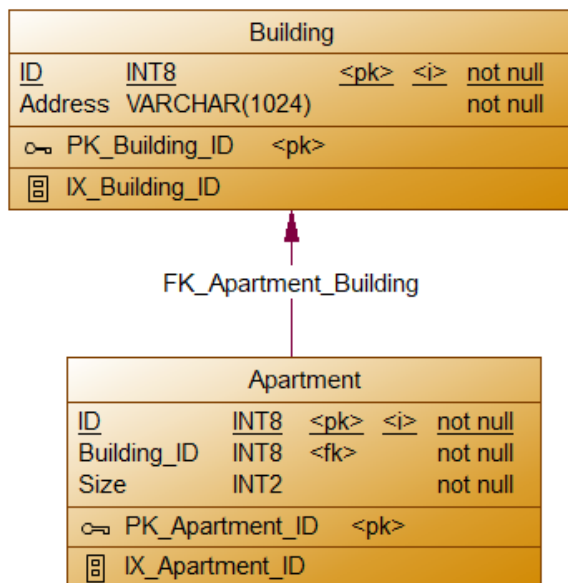
Используя Kitematic, мы можем наблюдать три запущенных контейнера:

The screenshot displays the Kitematic application interface. On the left, a sidebar lists three containers: 'happy_chandrasekhar' (knight-frank-node-ui), 'knight-frank-postgres', and 'lucid_snyder' (knight-frank-node-rest-api). The 'happy_chandrasekhar' container is selected and shown in the main area. It is in a 'RUNNING' state. Below the container name, there are controls for STOP, RESTART, EXEC, and DOCS. The 'CONTAINER LOGS' section shows the output of the container, which includes the build process of the application. The logs show the build of modules, optimization, and recording. The 'IP & PORTS' section indicates that the container can be accessed via localhost:4200.

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 выборки на стороне сервера.

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

Структура серверной части проекта:

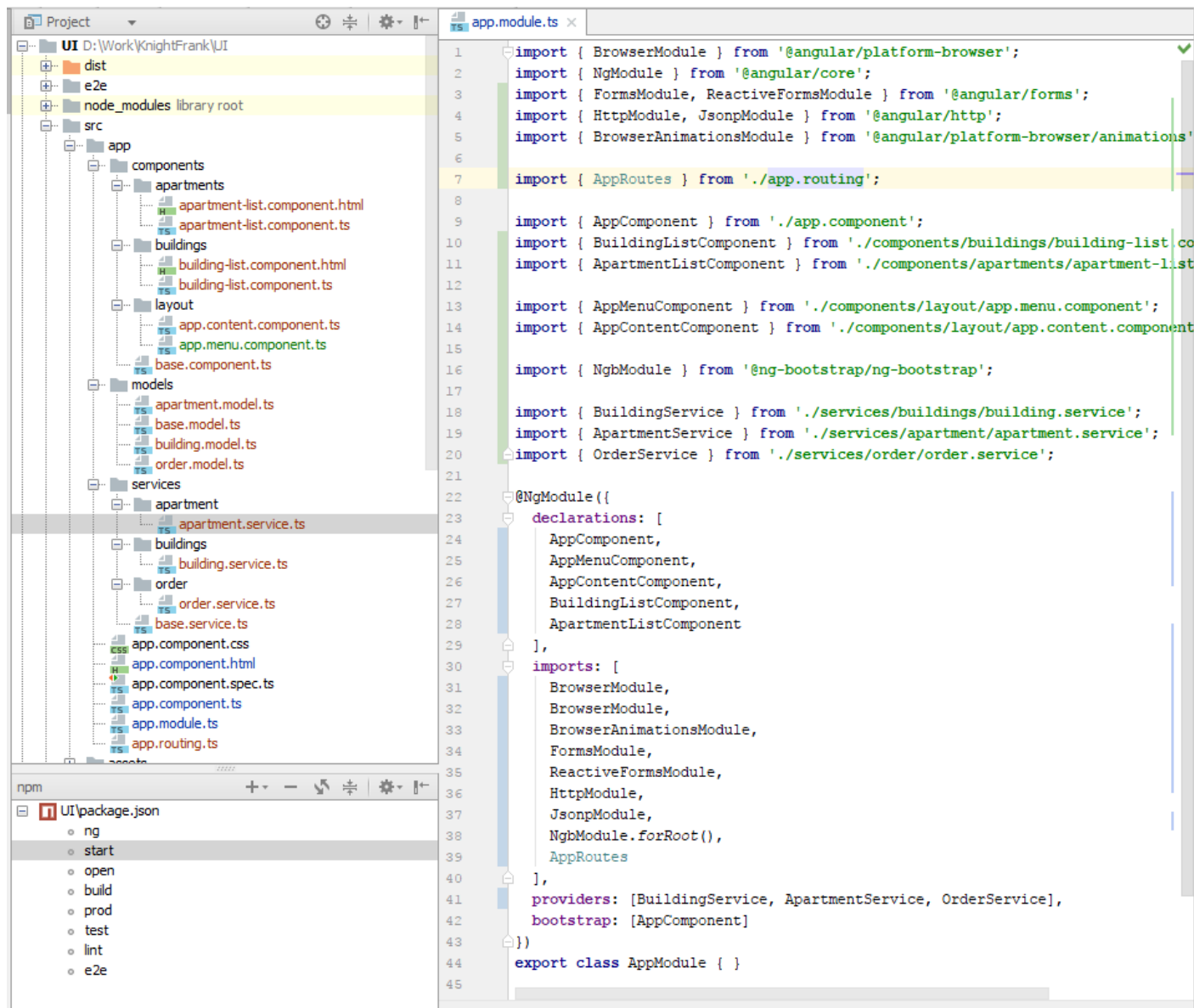
The screenshot shows an IDE with a project named 'Rest [WebApi]' located at 'D:\Work\KnightFrank\Service\Rest'. The project structure in the left sidebar includes a 'node_modules' library root and a 'server' directory. The 'server' directory contains a 'boot' subdirectory with 'authentication.js' and 'root.js', and a 'models' subdirectory with 'apartment.js', 'apartment.json', 'building.js', and 'building.json'. Other files in the 'server' directory include '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', and 'package-lock.json'. The 'server.js' file is selected and its content is displayed in the main editor. The code is as follows:

```
1 'use strict';
2
3 var loopback = require('loopback');
4 var boot = require('loopback-boot');
5
6 var app = module.exports = loopback();
7
8 app.start = function() {
9   // start the web server
10  return app.listen(function() {
11    app.emit('started');
12    var baseUrl = app.get('url').replace(/\/$/, '');
13    console.log('Web server listening at: %s', baseUrl);
14    // loopback routing:
15    if (app.get('loopback-component-explorer')) {
16      var explorerPath = app.get('loopback-component-explorer').mountPath;
17      console.log('Browse your REST API at %s%s', baseUrl, explorerPath);
18    }
19    // custom routing:
20    app.post('/api/requests', function(req, res) {
21      res.send(200, 'Просмотр успешно заказан');
22    });
23  });
24 };
25
26 // Bootstrap the application, configure models, datasources and middleware.
27 // Sub-apps like REST API are mounted via boot scripts.
28 boot(app, __dirname, function(err) {
29   if (err) throw err;
30
31   // start the server if `$ node server.js`
32   if (require.main === module)
33     app.start();
34 });
35
```

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):

БЛОК МЕНЮ		
Домов: 10 Квартир: 61		
Главная страница Здания		
БЛОК КОНТЕНТ		
Список зданий		
Идентификатор дома	Адрес дома	Количество квартир в доме
64	Address 64	8
65	Address 65	3
66	Address 66	3
67	Address 67	6
68	Address 68	7
69	Address 69	3
70	Address 70	7
71	Address 71	8
72	Address 72	6
73	Address 73	10

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

БЛОК МЕНЮ		
Домов: 10 Квартир: 61		
Главная страница Здания		
БЛОК КОНТЕНТ		
Список квартир в здании		
Номер квартиры	Площадь	
138	100	Заказать просмотр
139	200	Заказать просмотр
140	300	Заказать просмотр

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

БЛОК МЕНЮ

Домов: 10 Квартир: 61

БЛОК КОНТЕНТ

Номер квартиры

138

139

140

Заказать просмотр квартиры 139

Имя

Введите имя

Телефон

Введите телефон

Email

Введите Email

Отправить заказ

Отменить

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

БЛОК МЕНЮ	Главная страница Здания	
Домов: 10 Квартир: 61		
БЛОК КОНТЕНТ	Список квартир в здании	
Номер квартиры	Площадь	
138	100	Заказать просмотр
139	200	Заказать просмотр
140	300	Заказать просмотр
Заказ на квартиру 139 успешно отправлен		

Тестирование