Лабораторная работа

(Гибридные приложения)

Цель работы.

Ознакомиться с основами разработки Гибридных приложений под ОС Android.

В данной лабораторной работе вы познакомитесь со способами создания приложений под ОС Android в среде Android Studio. Вам предстоит создать простое приложение по типу списка покупок(контактов, задач итп). Можно просматривать весь список, добавлять новый элемент списка, можно удалять и, если получится, редактировать. Приблизительный план работы будет в самом конце.

Создание сервера Express для API при помощи Node.js

В этой лабораторной требуется создать сервер, на котором будет располагаться наш ***API***. При создании ***API*** будем использовать платформу ***Node.js***, которая является реализацией **JavaScript** для стороны сервера, и ***Express*** – популярный фреймворк для ***Node.js*** с самым необходимым функционалом.

Установка

Для начала убедимся, что ***Node.js*** и ***npm*** установлены глобально на нашем компьютере. Это можно проверить при помощи выполнения команды с флажком `-v`, в результате чего будут показаны установленные версии этих инструментов. Откройте консоль и введите туда следующую команду:

```
1  node -v && npm -v
v10.8.0
```

```
6.2.0
```

Если оба инструмента имеются, то можем приступить.

Создадим папку проекта под названием ***express-api*** и перейдем в нее.

```
1mkdir express-api && cd express-api
```

Далее можно инициализировать наш проект, выполнив команду ***init***.

```
1    npm init
```

В результате вам будет предложен ряд вопросов о проекте, на которые вы можете как отвечать, так и нет. После настройки у вас будет файл ***package.json***, который выглядит следующим образом:

```
01  {
02    "name": "express-api",
03    "version": "1.0.0",
04    "description": "Node.js and Express REST API",
05    "main": "index.js",
06    "scripts": {
07      "test": "echo \"Error: no test specified\" &&
08  exit 1"
09    },
10    "author": "Tania Rascia",
11    "license": "MIT"
  }
```

Теперь, когда у нас есть ***package.json,*** мы можем установить зависимости, необходимые для нашего проекта. Нам будут нужны:

- ***body-parser*** – промежуточное ПО для разбора тела запросов;
- ***express*** – веб-фреймворк с самым необходимым функционалом, который мы будем использовать для создания нашего сервера;
- ***mysql***: драйвер MySQL;
- ***request*** (необязателен) – легкий способ выполнения запросов ***HTTP***;

Мы воспользуемся командой ***install***, за которой будут следовать все зависимости, для завершения настройки нашего проекта.

```
1  npm install body-parser express mysql request
```

В результате будут созданы файл **package-lock.json**, папка **node_modules**, и **package.json** теперь будет выглядеть подобно следующему:

```
{
  "name": "express-api",
  "version": "1.0.0",
  "description": "Node.js and Express REST API",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" &&
exit 1"
  },
  "author": "Tania Rascia",
  "license": "MIT",
  "dependencies": {
  "dependencies": {
    "body-parser": "^1.18.3",
    "express": "^4.16.3",
    "mysql": "^2.16.0",
    "request": "^2.88.0"
  }}
```

Создание сервера HTTP

Перед тем как создать сервер ***Express***, мы быстренько создадим сервер ***HTTP*** при помощи встроенного модуля `http` *Node*, чтобы вы получили общее представление о том, как работает простенький сервер.

Создайте файл под названием ***hello-server.js***. Загрузите модуль `http`, установите значение порта (можно выбрать `3001`) и создайте сервер при помощи метода `createServer()`.

```
// Build a server with Node's HTTP module
const http = require('http');
const port = 3001;
const server = http.createServer();
```

Во вступительном руководстве этой серии мы рассмотрели, какую роль выполняют запросы и ответы для сервера ***HTTP***. Мы настроим наш сервер так,

чтобы он мог обрабатывать запрос и отображать его ***URL***-адрес на стороне сервера, а также так, чтобы на стороне клиента отображалось сообщение «**Hello, server!**».

```
1  server**on('request'** (request, response) => {
2      console.log(`URL: ${request.url}`);
3      response.end('Hello, server!')
4  })
```

Наконец, мы укажем серверу, какой порт прослушивать и будем выводить ошибки при их наличии.

```
1  // Start the server
2  server.listen(port, (error) => {
3      if (error) return console.log(`Error: ${error}`);
4       console.log(`Server is listening on port
5  ${port}`)
6  })
```

Теперь мы можем запустить наш сервер при помощи команды **node**, за которой следует имя файла.

```
1  node hello-server.js
```

Вы увидите ответ в консоли.

```
Server is listening on port 3001
```

Для того чтобы проверить, действительно ли запустился сервер, перейдите в вашем браузере по адресу `https://localhost:3001/`. Если все нормально, то вы увидите «**Hello, server!**» на странице. В вашей консоли вы увидите запрошенный ***URL***-адрес.

```
URL: /
URL: /favicon.ico
```

Если бы вы перешли по адресу **http://localhost:3001/hello**, то увидели бы

```
URL: /hello.
```

Также мы можем использовать инструмент с ***URL*** на нашем локальном сервере, за счет чего нам будут показаны точные возвращенные заголовки и тело ответа.

```
curl -i http://localhost:3001
HTTP/1.1 200 OK
Date: Wed, 15 Aug 2018 22:14:23 GMT
Connection: keep-alive
Content-Length: 14

Hello, server!
```

Если вы закроете консоль, то сервер прекратит работу.

Теперь, когда мы получили общее представление о том, как работает сервер, запрос и ответ вместе, мы можем переписать этот код для ***Express***, интерфейс которого даже проще и возможности которого более широки.

Создание сервера Express

Мы создадим новый файл, ***app.js***, который будет выступать в роли точки входа (* файл для запуска приложения) для собственно нашего проекта. Так же, как и в случае с оригинальным сервером ***http***, мы запросим модуль и укажем порт для запуска сервера.

Создайте файл ***app.js*** и добавьте туда следующий код:

```
// Require packages and set the port
const express = require('express');
const port = 3002;
const app = express();
```

Теперь, вместо того чтобы прослушивать все запросы, мы явно укажем серверу, что нам необходимы только запросы, выполненные по методу `GET` к корневой папке сервера (`/`). При получении конечной точкой запроса « `/` » мы отобразим запрошенный ***URL***-адрес и выведем сообщение ***«Hello, Server!».***

```
app.get('/', (request, response) => {
```

```
    console.log(`URL: ${request.url}`);
    response.send('Hello, Server!');
});
```

Наконец, мы запустим сервер, который будет прослушивать запросы, выполненные по `3002` порту, при помощи метода `listen()`.

```
// Start the server
const server = app.listen(port, (error) => {
    if (error) return console.log(`Error: ${error}`);

    console.log(`Server listening on port
${server.address().port}`);
});
```

Мы можем запустить сервер при помощи команды `node app.js`, как и ранее, однако мы можем изменить свойство `scripts` в файле ***package.json*** для автоматического запуска этой конкретной команды.

```
"scripts": {
  "start": "node app.js"
},
```

Теперь мы можем использовать команду `npm start` для запуска сервера, и после запуска мы увидим сообщение в консоли.

```
Server listening on port 3002
```

Если мы выполним команду `curl -i` для обсуждаемого ***URL***-адреса, то увидим, что сервер в этом случае работает на базе **Express** и что имеются некоторые дополнительные заголовки вроде `Content-Type`.

```
curl -i http://localhost:3002
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 14
ETag: W/"e-gaHDsc0MZK+LfDiTM4ruVL4pUqI"
Date: Wed, 15 Aug 2018 22:38:45 GMT
Connection: keep-alive

Hello, Server!
```

Добавление промежуточного ПО для разбора тела запросов

Для того чтобы облегчить работу с запросами по методам `POST` и `PUT` к нашему API, мы добавим промежуточное ПО для разбора тела запроса. Тут нам и пригождается модуль `body-parser`. За счет этого модуля будет извлечено все тело пришедшего запроса, а его данные преобразованы в ***JSON***-объект, с которым мы можем работать.

Мы просто запросим модуль вверху кода нашего файла. Добавьте следующую инструкцию `require` в верхнюю часть вашего файла ***app.js***.

```
1  const bodyParser = require('body-parser');
2  ...
```

Затем мы укажем нашему приложению Express, что необходимо использовать `body-parser` и преобразовывать данные в формат JSON.

```
1  // Use Node.js body parsing middleware
2  app.use(bodyParser.json());
3  app.use(bodyParser.urlencoded({
4      extended: true,
5  }));
```

Также давайте изменим наше сообщение таким образом, чтобы вместо простого текста в качестве ответа отправлялся ***JSON***-объект.

```
1  response.send({message: 'Node.js and Express REST
   API'});
```

Далее приводится код нашего файла ***app.js***, который имеем на данный момент:

```
01 // Require packages and set the port
02 const express = require('express');
03 const port = 3002;
04 const bodyParser = require('body-parser');
05 const app = express();
06
07 // Use Node.js body parsing middleware
08 app.use(bodyParser.json());
09 app.use(bodyParser.urlencoded({
10     extended: true,
11 }));
12
```

```
13 app.get('/', (request, response) => {
14     response.send({
15         message: 'Node.js and Express REST API'}
16     );
17 });
18
19 // Start the server
20 const server = app.listen(port, (error) => {
21     if (error) return console.log(`Error: ${error}`);
22
23     console.log(`Server listening on port
24 ${server.address().port}`);
   });
```

Если вы отправите запрос при помощи `curl -i` на сервер, то увидите, что в заголовке ***Content-Type*** теперь указано значение `application/json; charset=utf-8`.

Настройка маршрутов

Пока что у нас имеется только маршрут для обработки запросов по методу `GET` к корню приложения « `/` »), однако наш ***API*** также должен быть способен обрабатывать запросы ***HTTP*** по всем главным методам к различным ***URL***. Мы настроим маршрутизатор (* предоставляет функциональные возможности для обработки ответов) и добавим некоторые выдуманные данные для. отправления пользователю

Создадим новую папку под названием **routes** и файл под названием **routes.js**. Мы подключим его вверху **app.js**.

```
1    const routes = require('./routes/routes');
```

Обратите внимание на то, что расширение `.js` в require указывать необязательно. Теперь мы переместим маршрут для обработки запросов `GET` в ***routes.js***. Добавьте следующий код в ***routes.js***:

```
1  const router = app => {
2      app.get('/', (request, response) => {
3          response.send({
```

```
4               message: 'Node.js and Express REST API'
5           });
6       });
7   }
```

Наконец, экспортируйте маршрутизатор, чтобы мы могли им воспользоваться в нашем файле ***app.js***

```
1  // Export the router
2  module.exports = router;
```

В ***app.js*** замените имеющийся код `app.get()` вызовом `routes()`:

```
1  routes(app);
```

Теперь вы могли бы перейти по `http://localhost:3002` и увидеть то же, что и ранее. (Не забудьте перезапустить сервер!)

После удачной настройки вышеуказанного мы предоставим некоторые данные в формате **JSON** при помощи другого маршрута. Пока что мы воспользуемся просто выдуманными данными, поскольку наша база данных еще не создана.

Давайте создадим переменную `users` в ***routes.js*** с некоторыми выдуманными пользовательскими данными в формате **JSON**.

```
01  const users = [{
02          id: 1,
03          name: "Richard Hendricks",
04          email: "richard@piedpiper.com",
05      },
06      {
07          id: 2,
08          name: "Bertram Gilfoyle",
09          email: "gilfoyle@piedpiper.com",
10      },
11  ];
```

Мы добавим еще один маршрут для обработки запросов по адресу `/users` и методу `GET` в наш маршрутизатор и будем отправлять с его помощью пользовательские данные.

```
app.get('/users', (request, response) => {
    response.send(users);
});
```

После перезапуска сервера теперь вы можете перейти по `http://localhost:3002/users` и увидеть все наши данные.

Обратите внимание: если у вас не установлено в браузере расширения для просмотра файлов в формат ***JSON***, то необходимо скачать его, например ***JSONView*** для ***Chrome***.