

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

Кафедра вычислительной техники

**Отчет по лабораторной работе №7**  
**по дисциплине «Web-программирование»**

**Тема: разработка web-приложений с использованием GWT**

Студент гр. 2310

Альсакма О.С.М

Преподаватель

Павловский М.Г.

Санкт-Петербург

2024

## Цель работы

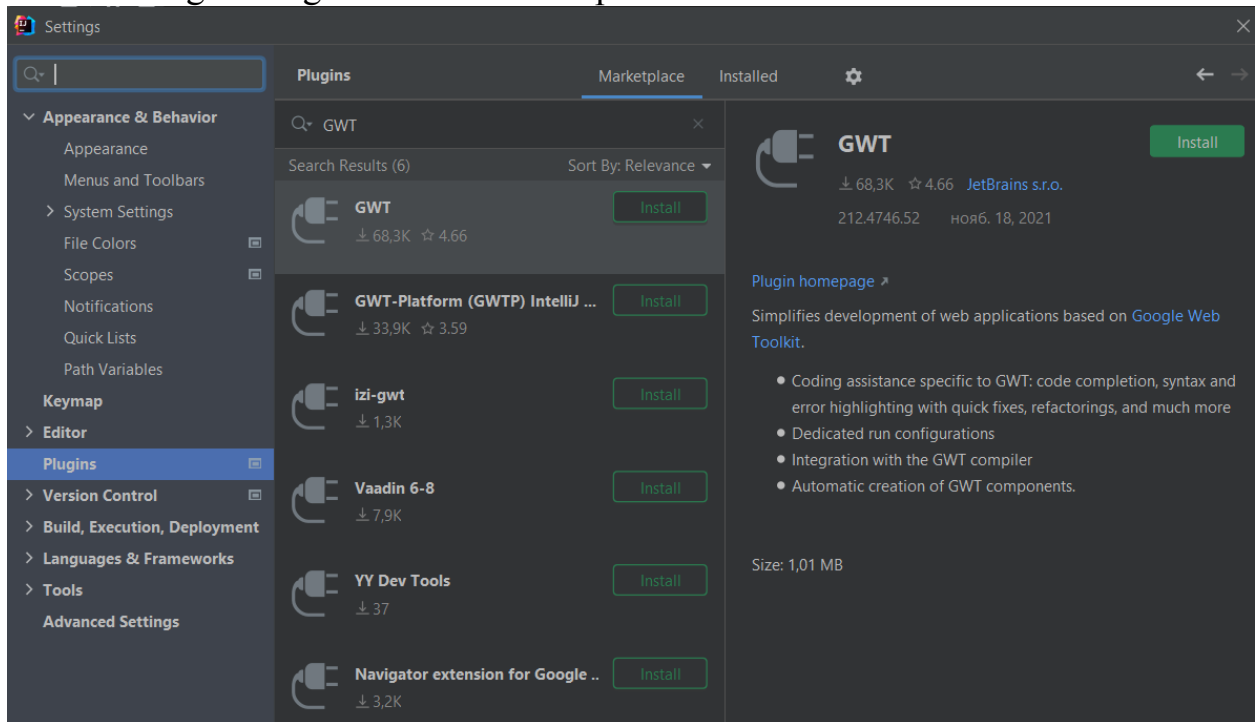
Знакомство с процессом создания GWT-приложения в среде IntelliJ Idea

GWT-приложение должно реализовывать работу второй лабораторной.

## Создание GWT-проекта

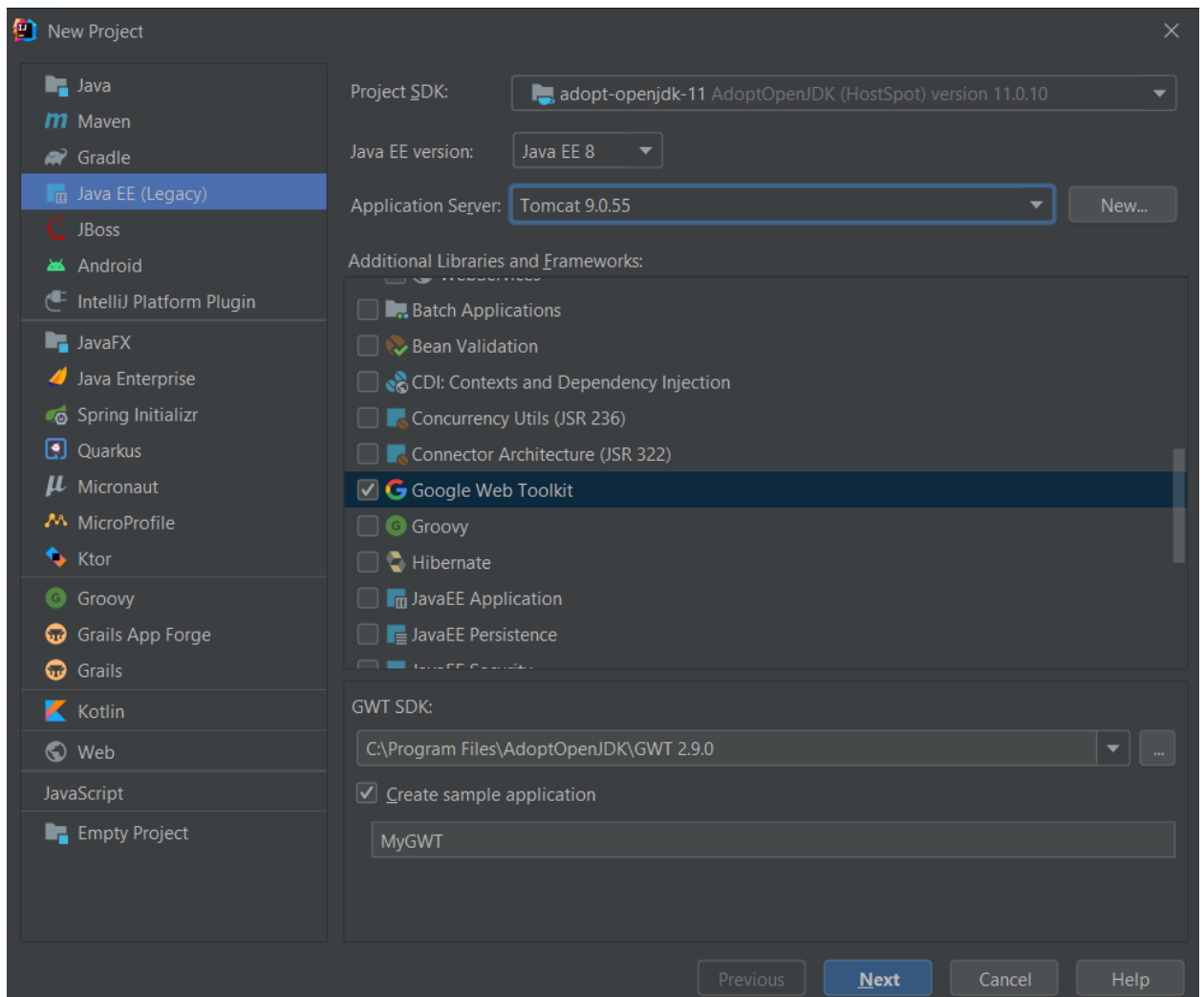
Необходимо установить GWT-плагин для удобной разработки приложения

File->Settings->Plugins-> “GWT” в строке поиска



После этого следует создать приложение

New project -> Java EE -> Google Web Toolkit

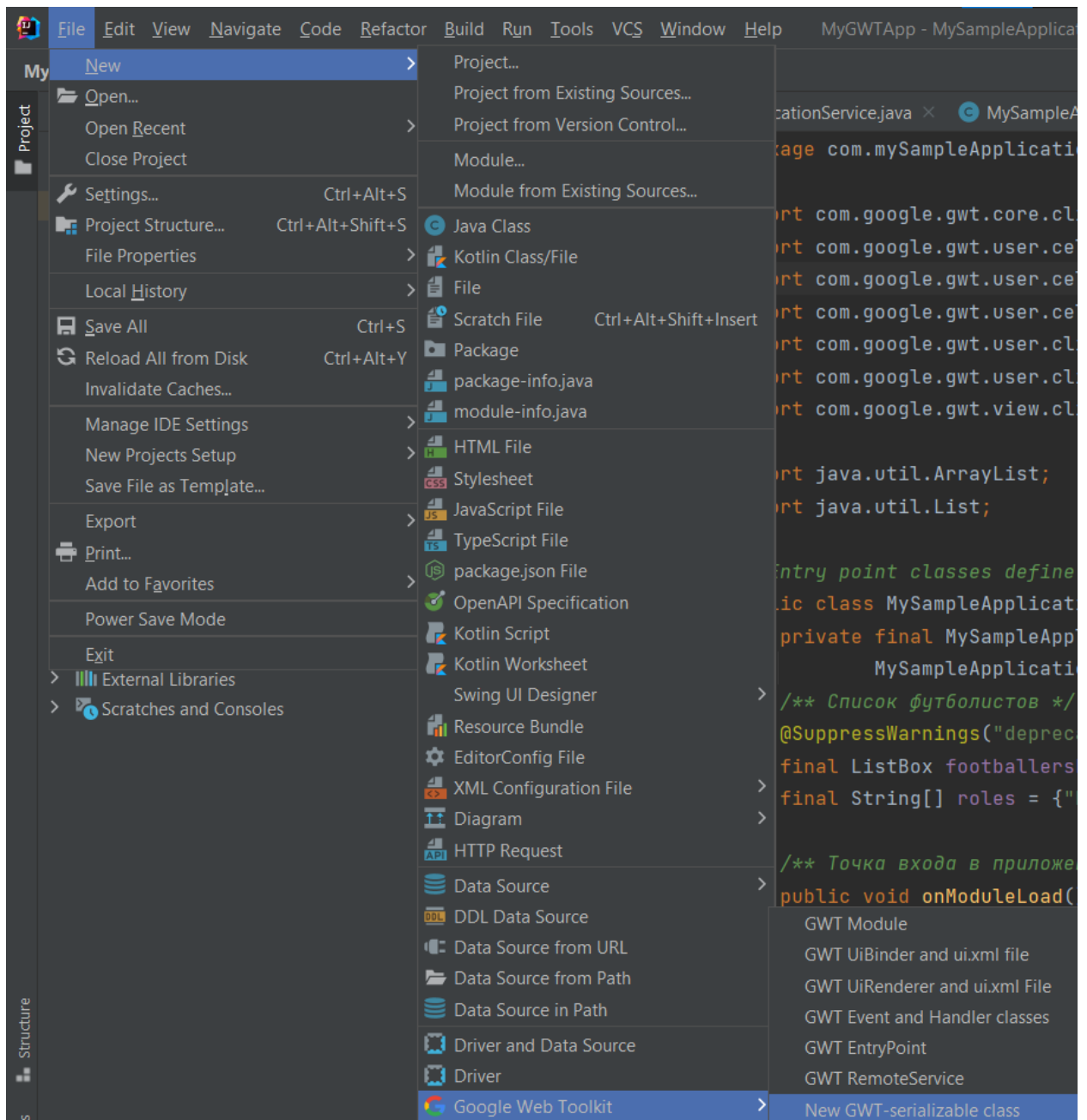


Следует уточнить, что для работы приложения нужно указать путь к скачанной GWT-папке и нажать галочку у “Create sample application”

После инициализации создается базовый проект, его можно запустить с помощью Tomcat

## Разработка GWT-приложения

Для работы со списком нам потребуется класс Footballer, аналогичный тому, что я проектировал во 2 лабе, однако еще требуется реализовать интерфейс Serializable для работы с сущностями из GWT. Создание класса:



## MySampleApplicationService.java

Далее требуется определить интерфейс для работы с приложением.

```
package com.mySampleApplication.client;

import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

import java.util.List;

@RemoteServiceRelativePath("MySampleApplicationService")
public interface MySampleApplicationService extends RemoteService {
    /**
     * Получение списка футболистов
     */
}
```

```

    * @return список футболистов
    */
    List<Footballer> getFootballerList();

    /**
     * Utility/Convenience class.
     * Use MySampleApplicationService.App.getInstance() to access static instance of
     * MySampleApplicationServiceAsync
     */
    public static class App {
        private static MySampleApplicationServiceAsync ourInstance =
        GWT.create(MySampleApplicationService.class);

        public static synchronized MySampleApplicationServiceAsync getInstance() {
            return ourInstance;
        }
    }
}

```

## MySampleApplicationServiceAsync.java

Затем требуется определить интерфейс для асинхронного вызова запросов пользователей. Методы должны дублироваться из MySampleApplicationService.java:

```

package com.mySampleApplication.client;

import com.google.gwt.user.client.rpc.AsyncCallback;

import java.util.List;
/** интерфейс для асинхронного обращения для каждого футболиста */
public interface MySampleApplicationServiceAsync {
    void getFootballerList(AsyncCallback<List<Footballer>> callback);
}

```

## MySampleApplicationServiceImpl.java

```

package com.mySampleApplication.server;

import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import com.mySampleApplication.client.Footballer;
import com.mySampleApplication.client.MySampleApplicationService;

import java.util.ArrayList;
import java.util.List;

/** Реализация логики сервера */
public class MySampleApplicationServiceImpl extends RemoteServiceServlet implements
MySampleApplicationService {

    private static List<Footballer> footballers = null;
    static {
        footballers = new ArrayList<>();
        footballers.add(new Footballer("Билли Херрингтон", 0, "Калуга", 16000));
        footballers.add(new Footballer("Антон Чехов", 1, "Санкт-Петербург", 30000));
        footballers.add(new Footballer("Илья Антонов", 2, "Екатеринбург", 25000));
        footballers.add(new Footballer("Андрей Сачков", 3, "Вологда", 19000));
    }
}

```

```

@Override
public List<Footballer> getFootballerList() {
    return footballers;
}
}

```

## MySampleApplication.html

```

<!DOCTYPE html>
<html lang="en">
<head>

    <title>Главная страница</title>

    <link type="text/css" rel="stylesheet" href="MySampleApplication.css">

    <script type="text/javascript" src="MySampleApplication/MySampleApplication.nocache.js"></script>
    <meta charset="UTF-8">
</head>
<body>

<h1>Команда футболистов</h1>
<div id = "salaryForm"></div> <br>
<div id = "PanelContainer"></div>
<br>Informant, 2021

</body>
</html>

```

## MySampleApplication.java

### Основная часть приложения

```

package com.mySampleApplication.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.cellview.client.CellTable;
import com.google.gwt.user.cellview.client.HasKeyboardSelectionPolicy;
import com.google.gwt.user.cellview.client.TextColumn;
import com.google.gwt.user.client.ui.*;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.view.client.ListDataProvider;

import java.util.ArrayList;
import java.util.List;

/**Entry point classes define <code>onModuleLoad()</code> */
public class MySampleApplication implements EntryPoint {
    private final MySampleApplicationServiceAsync myService =
        MySampleApplicationService.App.getInstance();
    /** Список футболистов */
    @SuppressWarnings("deprecation")
    final ListBox footballersListBox = new ListBox(false);
    final String[] roles = {"Вратарь", "Нападающий", "Полузащитник", "Защитник"};

    /** Точка входа в приложение - аналог main */

```

```

public void onModuleLoad() {
    footballersListBox.setFocus(true);
    refreshFootballersList();

    //создание и заполнение таблицы
    final CellTable<Footballer> mainTable = createCellTable();
    final ListDataProvider<Footballer> mainDataProvider = new ListDataProvider<>();
    mainDataProvider.addDataDisplay(mainTable);
    RootPanel.get("PanelContainer").add(mainTable);
    myService.getFootballerList(
        new AsyncCallback<List<Footballer>>() {
            @Override
            public void onFailure(Throwable caught) {

            }

            @Override
            public void onSuccess(List<Footballer> result) {
                mainDataProvider.setList(result);
            }
        }
    );

    final VerticalPanel salaryPanel = new VerticalPanel();
    salaryPanel.setHorizontalAlignment(VerticalPanel.ALIGN_CENTER);
    salaryPanel.setVisible(true);
    final Label salaryLabel = new Label("Введите минимальную зарплату");
    final Label errorLabel = new Label("Неверно введенная зарплата");
    final Button button = new Button("Получить список");
    final TextBox salaryField = new TextBox();
    salaryField.getElement().setPropertyString("placeholder", "Зарплата");
    errorLabel.setVisible(false);
    salaryPanel.add(salaryLabel);
    salaryPanel.add(errorLabel);
    salaryPanel.add(salaryField);
    salaryPanel.add(button);

    button.addClickHandler(event -> {
        int salary;
        try {
            salary = Integer.parseInt(salaryField.getText());
            List<Footballer> tempList = new ArrayList<>(mainDataProvider.getList());
            tempList.removeIf(boy -> boy.getSalary() < salary);
            mainDataProvider.setList(tempList);
            mainDataProvider.refresh();
            refreshFootballersList();
            salaryField.setText("");
            errorLabel.setVisible(false);
        }
        catch (Exception e) {
            errorLabel.setVisible(true);
        }
    });

    RootPanel.get("salaryForm").add(salaryPanel);
}

private CellTable<Footballer> createCellTable(){

```

```

        final CellTable<Footballer> table = new CellTable<>();
        //без этой строчки ничего не будет видно

table.setKeyboardSelectionPolicy(HasKeyboardSelectionPolicy.KeyboardSelectionPolicy.ENABLED);

        TextColumn<Footballer> nameColumn = new TextColumn<Footballer>() {
            @Override
            public String getValue(Footballer object) {
                return object.getName();
            }
        };
        table.addColumn(nameColumn, "Имя");//колонка, ее название

        TextColumn<Footballer> specColumn = new TextColumn<Footballer>() {
            @Override
            public String getValue(Footballer object) {
                return roles[object.getSpec()];
            }
        };
        table.addColumn(specColumn, "Специализация");

        TextColumn<Footballer> cityColumn = new TextColumn<Footballer>() {
            @Override
            public String getValue(Footballer object) {
                return object.getCity();
            }
        };
        table.addColumn(cityColumn, "Город");

        TextColumn<Footballer> salaryColumn = new TextColumn<Footballer>() {
            @Override
            public String getValue(Footballer object) {
                return String.valueOf(object.getSalary());
            }
        };
        table.addColumn(salaryColumn, "Зарплата");

        return table;
    }

    private void refreshFootballersList(){
        myService.getFootballerList(new AsyncCallback<List<Footballer>>() {
            @Override
            public void onFailure(Throwable caught) {

            }

            @Override
            public void onSuccess(List<Footballer> result) {
                footballersListBox.clear();
                for (Footballer boy : result)
                    footballersListBox.addItem(boy.getName());
            }
        });
    }
}

```

**web.xml**



```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-
app_4_0.xsd"
  version="4.0">
  <servlet>
    <servlet-name>com.mySampleApplication.MySampleApplication MySampleApplicationService</
servlet-name>
    <servlet-class>com.mySampleApplication.server.MySampleApplicationServiceImpl</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>com.mySampleApplication.MySampleApplication MySampleApplicationService</
servlet-name>
    <url-pattern>/MySampleApplication/MySampleApplicationService</url-pattern>
  </servlet-mapping>
</web-app>
```

## Демонстрация работы

Если ввести в поле минимальную зарплату, то в таблице выведутся те футболисты, чья зарплата больше заявленной:

### Команда футболистов

Введите минимальную зарплату

Получить список

Имя	Специализация	Город	Зарплата
Билли Херрингтон	Вратарь	Калуга	16000
Антон Чехов	Нападающий	Санкт-Петербург	30000
Илья Антонов	Полузащитник	Екатеринбург	25000
Андрей Сачков	Защитник	Вологда	19000

Однако если в поле ввести некорректное значение, то высветится уведомляющее об ошибке поле

Введите минимальную зарплату

Неверно введенная зарплата

Получить список

Имя	Специализация	Город	Зарплата
Антон Чехов	Нападающий	Санкт-Петербург	30000
Илья Антонов	Полузащитник	Екатеринбург	25000
Андрей Сачков	Защитник	Вологда	19000

## Вывод по лабораторной работе №7

В ходе выполнения лабораторной работы были изучены и освоены основы разработки Web-приложений с использованием GWT. В частности, были выполнены следующие шаги:

### 1. Создание GWT-проекта:

- Установлен плагин GWT для IntelliJ Idea.
- Создан новый проект GWT с использованием шаблона приложения.

### 2. Разработка GWT-приложения:

- Создан класс `Footballer`, который представляет сущность футболиста и реализует интерфейс `Serializable`.
- Разработаны интерфейсы `MySampleApplicationService` и `MySampleApplicationServiceAsync` для работы с сервером.
- Реализована логика сервера в классе `MySampleApplicationServiceImpl`, который возвращает список футболистов.
- Создана основная страница `MySampleApplication.html` и основной класс `MySampleApplication.java`, который отвечает за отображение данных и взаимодействие с пользователем.

### 3. Тестирование приложения:

- Приложение успешно фильтрует список футболистов по зарплате, введенной пользователем.
- При вводе некорректного значения выводится сообщение об ошибке.
- 

Таким образом, в результате выполнения лабораторной работы было успешно создано GWT-приложение, которое реализует функциональность второй лабораторной работы и работает с использованием RPC для взаимодействия с сервером.