Министерство науки и образования РФ
Федеральное государственное автономное образовательное
учреждение высшего профессионального образования
«Санкт-Петербургский государственный электротехнический
университет «ЛЭТИ» им. В. И. Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)

Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

Отчёт по лабораторной работе № 7
на тему: “РАЗРАБОТКА Web-ПРИЛОЖЕНИЯ С ИСПЛЬЗОВАНИЕМ GWT”
по дисциплине
“Web-программирование”

Выполнили студенты гр.9308: Дементьев Д.П.

Проверил: Павловский М.Г.

Санкт-Петербург, 2021 г.

Оглавление

Введение

Целью работы является знакомство с процессом создания GWT-приложения в среде IntelliJ IDEA.

Создание проекта GWT-приложения

Для начала установим плагин GWT в среде разработки для удобной разработки GWT-приложения File->Settings->Plugins->Поиск «GWT».

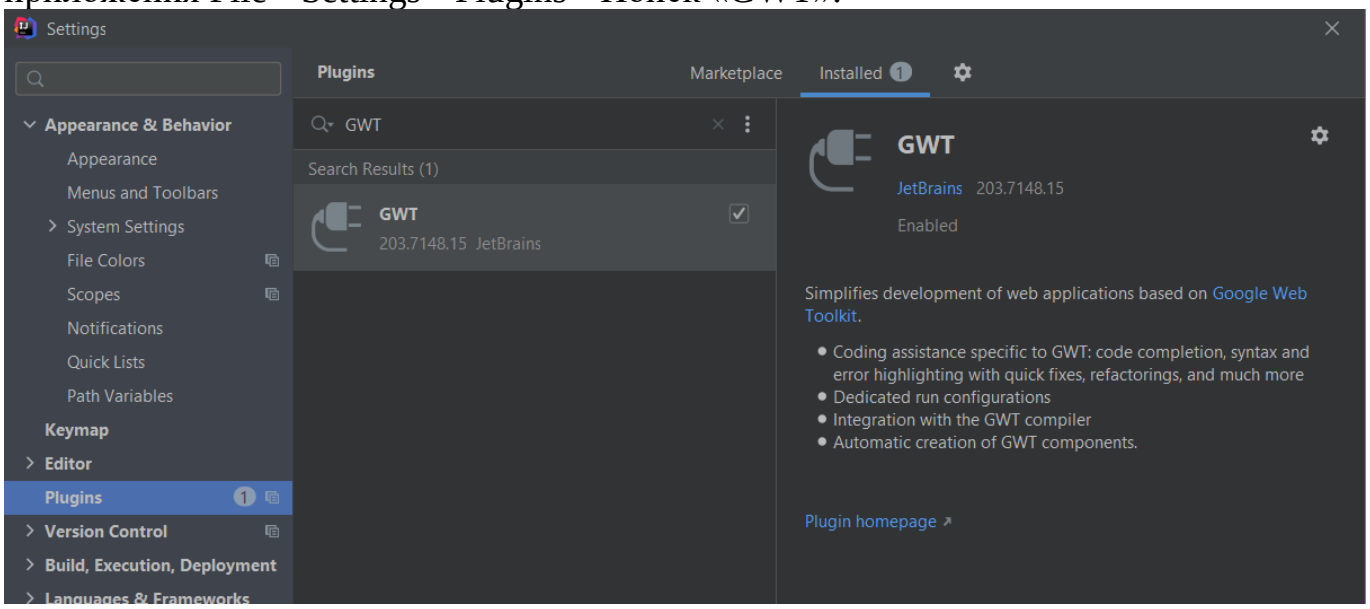

Рис.1. Добавление плагина GWT в IntelliJ IDEA

IntelliJ IDEA предоставляет нам возможность создания стандартного GWT приложения: New project -> Java EE -> Выбор Google Web Toolkit:

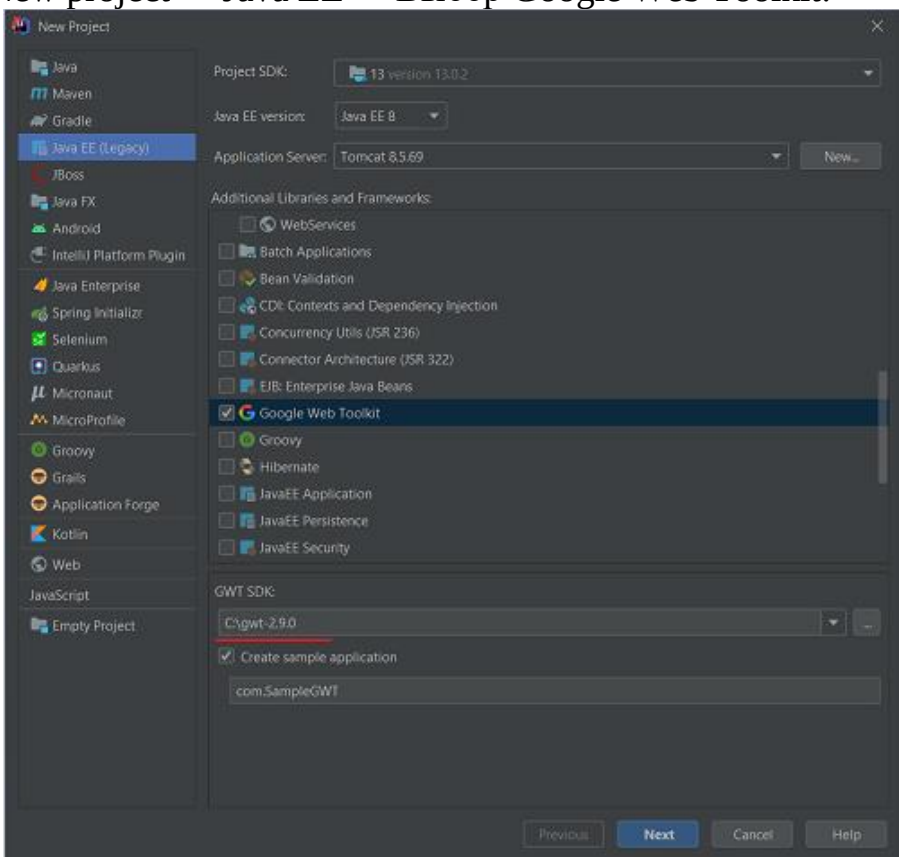

Рис.2. Окно создание проекта GWT-приложения

Также потребуется предварительно скачать GWT SDK и указать путь до полученной папки (подчеркнуто красным) и сразу создадим пример по образцу «Create sample application».

IDE создаст проект со следующей структурой:

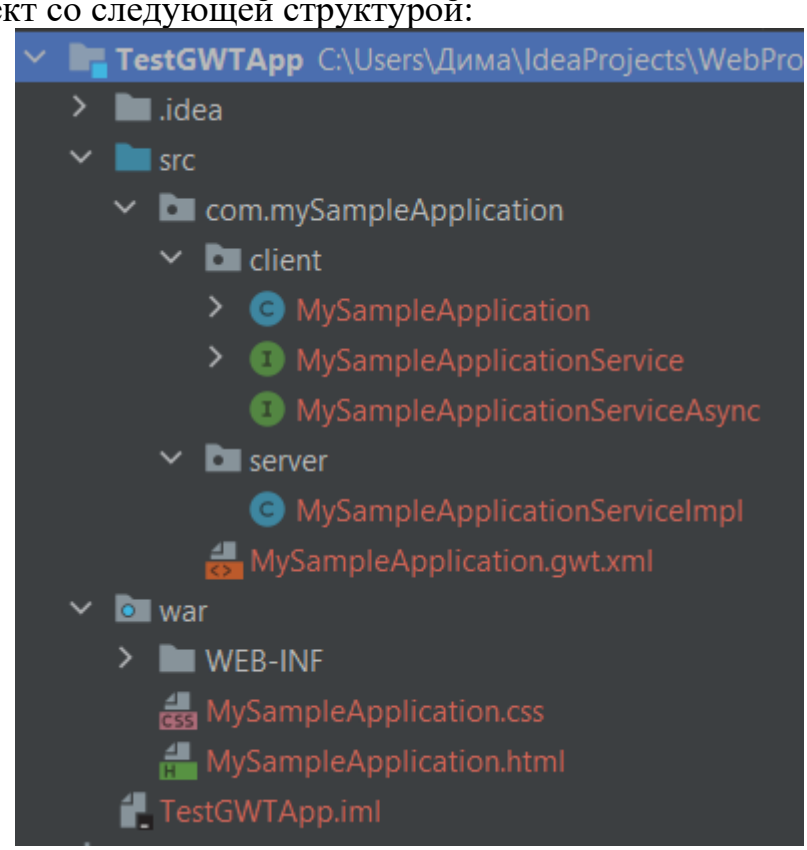

Рис.3. Структура созданного проекта

Разберемся, что к чему по пунктам:

- Пакет (client), в котором находится исходный клиентский Java-код. Этот код после компиляции его в JavaScript будет выполняться браузером.
    - MySampleApplication будет отвечать за создание пользовательского интерфейса в методе onModeluLoad()
    - MySampleApplicationService определяет методы удаленного интерфейса
    - MySampleApplicationServiceAsync: поскольку передавать данные серверу мы будем через механизм HTTP, вообще-то не предназначенный для таких дел, Google вводит еще одно передаточное звено - асинхронный интерфейс для осуществления запросов от множества пользователей.
- Пакет (server), в котором находится исходный серверный Java-код. Этот код обрабатывает асинхронные HTTP-запросы, поступающие от клиента.
    - MySampleApplicationServiceImpl будет содержать непосредственно реализацию кода на сервере для каждого из асинхронных запросов
- Файл конфигурации GWT-модуля – XML-файл, имеющий название вида MySampleApplication.gwt.xml, где MySampleApplication – название GWT-приложения.
- MySampleApplication.html – страница, на которой и будут использоваться скомпилированные GWT JS-скрипты, для удобства переименуем в index.html (не забудем в конфигурации изменить адрес начальной страницы на index.html)
- MySampleApplication.css – оформление страницы, сразу переименуем в style.css

Web-приложение мы также будем запускать на сервере Apache Tomcat, поэтому из множества созданных средой разработки конфигураций выберем следующую:

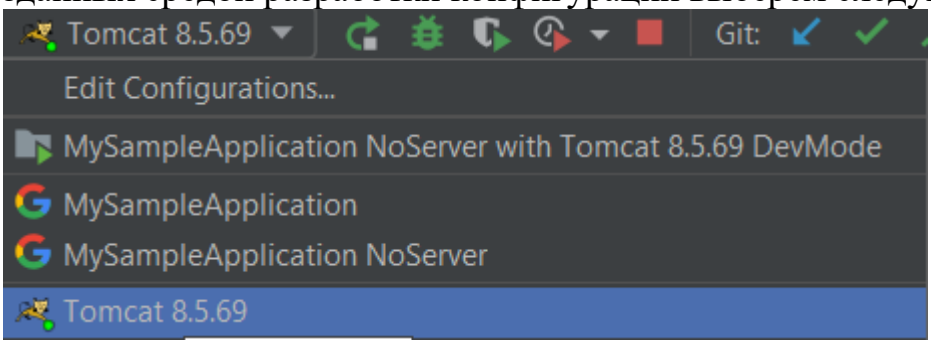

Рис.4. Запуск приложения через сервер Apache Tomcat

Post.java

Для работы со списком всех комментариев нам понадобится сущность Post, которая должна реализовывать интерфейс IsSerializable или Serializable для корректной передачи объекта между сервером и клиентом. Создадим такой класс:

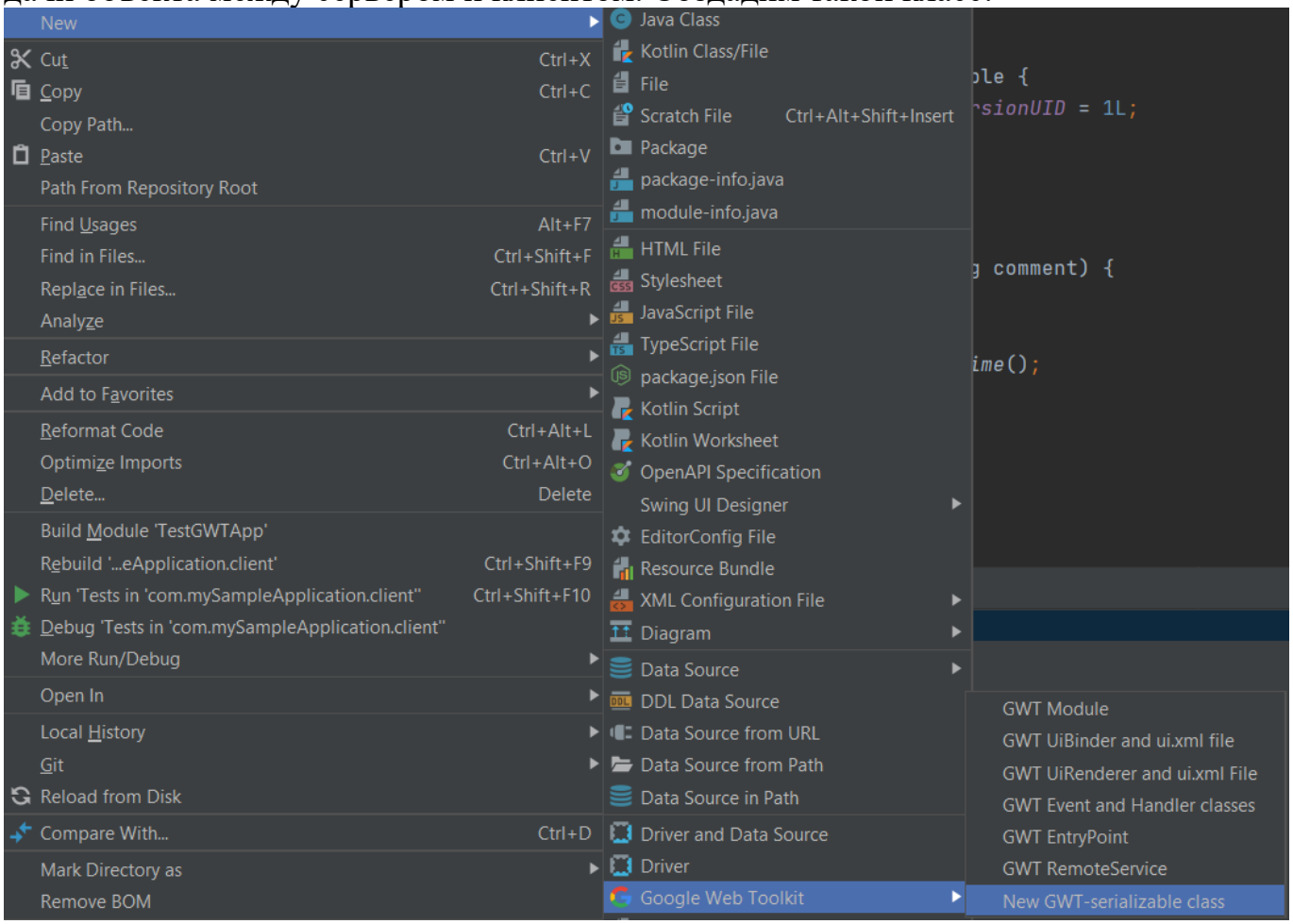

Рис.5. Создание сериализуемого класса Post

Сущность будет иметь следующее содержание:

```
package com.mySampleApplication.client;

import com.mySampleApplication.utils.LocaleManager;

import java.io.Serializable;

public class Post implements Serializable {
    private static final long serialVersionUID = 1L;
    private String username;
    private String comment;
    private String date;

    public Post(String username, String comment, String date) {
        this.username = username;
        this.comment = comment;
        this.date = date;
    }

    public Post() {
    }

    public String getUsername() {
        return username;
```

```
    }

    public String getComment() {
        return comment;
    }

    public String getDate() {
        return date;
    }
}
```

MySampleApplicationService.java

Далее нам потребуется определить интерфейс сервиса для работы с приложением. Мы должны иметь возможность получить все комментарии, список уникальных имён пользователей и возможность добавлять новую запись (комментарий):

```
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")
/** интерфейс сервиса возможных действий со страницей */
public interface MySampleApplicationService extends RemoteService {

    /** Получение списка имён пользователей
     * @return список имён пользователей
     */
    List<String> getUserList();

    /** Получение всего списка комментариев
     * @return список записей пользователей
     */
    List<Post> getPostList();

    /** Получение списка сообщений пользователя {@link Post#username}
     * @param username - имя пользователя
     * @return список записей пользователя
     */
    List<Post> getPostList(String username);

    /** Добавление записи
     * @param post - новая запись
     */
    void addPost(Post post);

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

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

MySampleApplicationServiceAsync.java

Также потребуется определить интерфейс для асинхронного вызова запросов пользователей, ведь для каждого уникального пользователя должен вызываться своё действие. Методы должны дублировать возможности MySampleApplicationService.java:

```
package com.mySampleApplication.client;

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

import java.util.List;
/** интерфейс для асинхронного обращения для каждого из пользователей */
public interface MySampleApplicationServiceAsync {
    void getUserList(AsyncCallback<List<String>> callback);
    void getPostList(AsyncCallback<List<Post>> callback);
    void getPostList(String username, AsyncCallback<List<Post>> callback);
    void addPost(Post post, AsyncCallback<Void> async);
}
```

MySampleApplicationServiceImpl.java

Сервер должен обрабатывать запросы пользователей, определенных в интерфейсах выше. Для этого мы изучали сервлеты в Java, Google Web Toolkit предлагает нам расширение этой технологии:

```
package com.mySampleApplication.server;

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

import java.util.*;
/** Реализация логики сервера - некий сервлет */
public class MySampleApplicationServiceImpl extends RemoteServiceServlet implements
MySampleApplicationService {

    private static List<Post> db = null;
    static {
        db = new ArrayList<>();
        db.add(new Post("ДимонЛимон", "всем приветв этом чатике", new
Date().toString()));
        db.add(new Post("Чипибарум", "ждУ оБНовлЕний", new Date().toString()));
        db.add(new Post("ДимонЛимон", "проверка 7 лабы", new Date().toString()));
    }
    @Override
    public List<String> getUserList() {
        Set<String> result = new HashSet<>();
        for (Post post: db) {
            result.add(post.getUsername());
        }
        return new ArrayList<>(result);
    }

    @Override
    public List<Post> getPostList() {
        return db;
    }
    @Override
    public List<Post> getPostList(String username) {
        List<Post> result = new ArrayList<>();
        for (Post post: db) {
            if (post.getUsername().equals(username)) {
                result.add(post);
            }
        }
        return result;
```

```
    }

    @Override
    public void addPost(Post post) {
        if (post != null) db.add(post);
    }
}
```

Видим, что помимо реализации методов интерфейсов тут также присутствует инициализации «базы данных», находящейся в оперативной памяти, то есть при завершении приложения данные удаляются.

index.html

Наше приложение будет одностраничным, для этого потребуется задать некий шаблон с «заглушками», которые позже будут динамически заполняться нужными элементами:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

    <title>Главная страница</title>

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

    <script type="text/javascript" src="MySampleApplication/MySampleApplica-
tion.nocache.js"></script>
</head>

<body>

<h1>Все записи</h1>

<table>
    <tr>
        <td id="addButtonContainer"></td>
        <td><pre>        </pre></td>
        <td id="userListBoxContainer"></td>
        <td id="errorLabelContainer"></td>
        <td id="sendButtonContainer"></td>
    </tr>
    <tr id="addPanelContainer"></tr>
</table>
<div id="allCommentsTable"></div>
<br>

<a href="index?locale=ru">ru</a>
<a href="index?locale=en">en</a>
</body>
</html>
```

Наш шаблон имеет следующие «заглушки»:

- Кнопка добавления новой записи
- Выпадающее меню со списком пользователей
- Место для возможного вывода текста ошибки
- Кнопка для генерации всех сообщений выбранного пользователя
- Панель (изначально скрытая) добавления комментария
- Таблица, содержащая все записи сообщества
- Ссылки на переход к поддерживаемым локализациям

MySampleApplication.java

Теперь наш шаблон нужно чем-то наполнить, в этом нам поможет GWT, имеющая в своей библиотеке удобные инструменты для создания интерфейса приложения (чем-то похоже на java.swing):

```
package com.mySampleApplication.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.event.dom.client.*;
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 com.mySampleApplication.client.internationalization.Resources;

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

/**
 * Точка начала приложения <code>onModuleLoad()</code>
 */
public class MySampleApplication implements EntryPoint {
    /** RPC-сервис */
    private final MySampleApplicationServiceAsync myService =
MySampleApplicationService.App.getInstance();

    /** Списокк пользователей */
    final ListBox userListBox = new ListBox(false);
    /** Место для вывода ошибки */
    final Label errorLabel = new Label();

    /** Метод загрузки страницы - аналог main() в Java */
    public void onModuleLoad() {
        // создаем главные элементы страницы
        final Button sendButton = new Button(Resources.TEXT.btnGetComment());
        final Button addCommentButton = new Button(Resources.TEXT.btnAddComment());
        sendButton.addStyleName("sendButton");
        RootPanel.get("userListBoxContainer").add(userListBox);
        RootPanel.get("errorLabelContainer").add(errorLabel);
        RootPanel.get("sendButtonContainer").add(sendButton);
        RootPanel.get("addButtonContainer").add(addCommentButton);

        // заполняем список пользователей, имеющих сообщения
        userListBox.setFocus(true);
        refreshUserList();

        // создание и заполнение таблицы со всеми комментариями
        final CellTable<Post> mainTable = createCellTable();
        final ListDataProvider<Post> mainDataProvider = new ListDataProvider<Post>();
        mainDataProvider.addDataDisplay(mainTable);
        RootPanel.get("allCommentsTable").add(mainTable);
        myService.getPostList(
            new AsyncCallback<List<Post>>() {
                public void onFailure(Throwable caught) {
                    errorLabel.setText(Resources.TEXT.errServer()+Resources.TEXT.err-
Server_commentList());
                }
                public void onSuccess(List<Post> result) {
                    mainDataProvider.setList(result);
                }
            }
        );

        // работа с добавлением комментария
```

```
        final VerticalPanel addPanel = new VerticalPanel();
        addPanel.setHorizontalAlignment(VerticalPanel.ALIGN_RIGHT);
        addPanel.setVisible(false);
        final TextArea nameArea = new TextArea();
        nameArea.getElement().setPropertyString("placeholder", Resources.TEXT.place-
holder_name());
        final TextArea commentArea = new TextArea();
        commentArea.getElement().setPropertyString("placeholder", Re-
sources.TEXT.placeholder_comment());
        final Button sendCommentBtn = new Button(Resources.TEXT.btnSend());
        addCommentButton.addClickHandler(new ClickHandler() {
            public void onClick(ClickEvent event) {
                addPanel.setVisible(true);
                sendCommentBtn.setEnabled(true);
                sendCommentBtn.setFocus(true);
                addCommentButton.setEnabled(false);
            }
        });
        addPanel.add(nameArea);
        addPanel.add(commentArea);
        sendCommentBtn.addClickHandler(new ClickHandler() {
            public void onClick(ClickEvent event) {
                String username = nameArea.getText();
                String comment = commentArea.getText();
                Post post = new Post(username, comment, new Date().toString());
                myService.addPost(post,
                        new AsyncCallback<Void>() {
                            @Override
                            public void onFailure(Throwable caught) {

                            }
                            @Override
                            public void onSuccess(Void result) {
                                List<Post> newList = new ArrayList<Post>(mainDataPro-
vider.getList());

                                newList.add(post);
                                mainDataProvider.setList(newList);
                                mainDataProvider.refresh();
                                refreshUserList();
                                addPanel.setVisible(false);
                                nameArea.setText("");
                                commentArea.setText("");
                                addCommentButton.setEnabled(true);
                            }
                        }
                );
            }
        });
        addPanel.add(sendCommentBtn);
        RootPanel.get("addPanelContainer").add(addPanel);

        // окно с результатом поиска комментариев заданного пользователя
        final DialogBox dialogBox = new DialogBox();
        dialogBox.setText(Resources.TEXT.titleResultTable());
        dialogBox.setAnimationEnabled(true);
        final Button closeButton = new Button(Resources.TEXT.btnClose());
        closeButton.getElement().setId("closeButton");
        final HTML serverResponseLabel = new HTML();
        VerticalPanel dialogVPanel = new VerticalPanel();
        dialogVPanel.addStyleName("dialogVPanel");

        final CellTable<Post> userTable = createCellTable();
        dialogVPanel.add(userTable);

        dialogVPanel.setHorizontalAlignment(VerticalPanel.ALIGN_RIGHT);
        dialogVPanel.add(closeButton);
        dialogBox.setWidget(dialogVPanel);
        closeButton.addClickHandler(new ClickHandler() {
```

```
            public void onClick(ClickEvent event) {
                dialogBox.hide();
                sendButton.setEnabled(true);
                sendButton.setFocus(true);
            }
        });
        class RPCClickHandler implements ClickHandler, KeyUpHandler {
            public void onClick(ClickEvent event) {
                sendUserToServer();
            }
            public void onKeyUp(KeyUpEvent event) {
                if (event.getNativeKeyCode() == KeyCodes.KEY_ENTER) {
                    sendUserToServer();
                }
            }
            private void sendUserToServer() {
                errorLabel.setText("");
                final String username = userListBox.getValue(userListBox.getSelecte-
dIndex());
                sendButton.setEnabled(false);
                myService.getPostList(username,

                new AsyncCallback<List<Post>>() {
                    public void onFailure(Throwable caught) {
                        dialogBox.setText(Resources.TEXT.errServer());
                        serverResponseLabel
                                .addStyleName("serverResponseLabelError");
                        serverResponseLabel.setHTML(Resources.TEXT.errServer()+Re-
sources.TEXT.errServer_commentList());
                        dialogBox.center();
                        closeButton.setFocus(true);
                    }
                    public void onSuccess(List<Post> result) {
                        dialogBox.setText(Resources.TEXT.titleResultTable() +
username);
                        userTable.setRowCount(result.size(), true);
                        userTable.setRowData(0, result);
                        dialogBox.center();
                        closeButton.setFocus(true);
                    }
                });
            }
        }
        RPCClickHandler handler = new RPCClickHandler();
        sendButton.addClickHandler(handler);
    }

    /** Создание таблицы для отображения комментариев пользователей
     * @return таблица с заданным форматом для отображения комментариев
     * */
    private CellTable<Post> createCellTable() {
        final CellTable<Post> table = new CellTable<Post>();
        table.setKeyboardSelectionPolicy(HasKeyboardSelectionPolicy.KeyboardSelec-
tionPolicy.ENABLED);

        TextColumn<Post> authorColumn = new TextColumn<Post>() {
            public String getValue(Post object) {
                return object.getUsername();
            }
        };
        table.addColumn(authorColumn, Resources.TEXT.username());
        TextColumn<Post> titleColumn = new TextColumn<Post>() {
            public String getValue(Post object) {
                return object.getComment();
            }
        };
        table.addColumn(titleColumn, Resources.TEXT.comment());
        TextColumn<Post> dateColumn = new TextColumn<Post>() {
```

```
            public String getValue(Post object) {
                return object.getDate();
            }
        };
        table.addColumn(dateColumn, Resources.TEXT.date());

        return table;
    }

    /** Обновление списка пользователей */
    private void refreshUserList() {
        myService.getUserList(new AsyncCallback<List<String>>() {
            public void onFailure(Throwable caught) {
                errorLabel.setText(Resources.TEXT.errServer()+Resources.TEXT.err-
Server_userList());
            }

            public void onSuccess(List<String> result) {
                userListBox.clear();
                for(String r : result){userListBox.addItem(r);
                }
            }
        });
    }
}
```

Интернационализация

В лабораторной работе использовался способ с заданием локализированных значений через интерфейс приложения MySampleApplication.java с использованием констант (com.google.gwt.i18n.client.Constants).

Resouces.java

```
package com.mySampleApplication.client.internationalization;
import com.google.gwt.core.client.GWT;

/** Класс Ресурсов для получения параметров из внешних файлов */
public class Resources {
    /** Ресурс, соответствующий параметрам из Text_XXX.properties */
    public static final Text TEXT = GWT.create(Text.class);
}
```

Text.java

```
package com.mySampleApplication.client.internationalization;
import com.google.gwt.i18n.client.Constants;
/** интерфейс получения ресурсов для интернационализации из Text_XXX.properties */
public interface Text extends Constants
{
    String btnAddComment();

    String errServer();

    String errServer_userList();

    String errServer_commentList();

    String btnGetComment();

    String btnClose();

    String titleResultTable();
```

```
    String username();

    String comment();

    String date();

    String btnSend();

    String placeholder_name();

    String placeholder_comment();

    String title();

    String header();
}
```

Text_XXX.properties

Text.properties

```
errServer = Ошибка сервера!
errServer_userList = Невозможно получить список пользователей.
errServer_commentList = Невозможно получить список комментариев пользователя.
btnGetComment = Получить список комментариев
btnAddComment = Добавить комментарий
btnClose = Закрыть
btnSend = Отправить
titleResultTable = Все комментарии пользователя
username = Пользователь
comment = Комментарий
date = Дата
placeholder_name = Ваше имя
placeholder_comment = Ваш комментарий
```

Text_ru.properties

```
errServer = Ошибка сервера!
errServer_userList = Невозможно получить список пользователей.
errServer_commentList = Невозможно получить список комментариев пользователя.
btnGetComment = Получить список комментариев
btnAddComment = Добавить комментарий
btnClose = Закрыть
btnSend = Отправить
titleResultTable = Все комментарии пользователя
username = Пользователь
comment = Комментарий
date = Дата
placeholder_name = Ваше имя
placeholder_comment = Ваш комментарий
```

Text_en.properties

```
errServer = Server error!
errServer_userList = Unable to get user list.
errServer_commentList = Unable to get a list of user comments.
btnGetComment = Get list of comments
btnAddComment = Add Comment
btnClose = Close
btnSend = Send
titleResultTable = All User Comments:
username = User
comment = Comment
date = Date
placeholder_name = Your name
placeholder_comment = Your comment
```

MySampleApplication.gwt.xml

Пожалуй, самая важная часть GWT-приложения – его конфигурация. В ней мы задаём локализации, точку начала приложения (начальный сервлет) и URL этого сервлета:

```
<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 2.8.0//EN"
        "http://www.gwtproject.org/doctype/2.8.0/gwt-module.dtd">

<module rename-to="MySampleApplication">
    <!-- подключения локализаций для интернационализации -->
    <inherits name="com.google.gwt.i18n.I18N"/>
    <extend-property name="locale" values="ru"/>
    <extend-property name="locale" values="en"/>

    <!-- Inherit the core Web Toolkit stuff.                  -->
    <inherits name='com.google.gwt.user.User'/>

    <!-- Specify the app entry point class.                   -->
    <entry-point class='com.mySampleApplication.client.MySampleApplication'/>


    <!-- Specify the app servlets.                  -->
    <servlet path='/MySampleApplicationService' class='com.mySampleApplica-
tion.server.MySampleApplicationServiceImpl'/>

</module>
```

web.xml

Не обойтись и без дескриптора развертывания на сервере Apache Tomcat, где мы переименуем адрес сервлета и также сделаем его начальной страницей. Также тут будет находится базовая аутентификация по шаблону пятой лабораторной работы:

```
<?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">
    <!-- Задание стартовой страницы -->
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>

    <!--Изменение ссылки URL С "/index.html" НА "/index" -->
    <servlet>
        <servlet-name>Index</servlet-name>
        <jsp-file>/index.html</jsp-file>
    </servlet>
    <servlet-mapping>
        <servlet-name>Index</servlet-name>
        <url-pattern>/index</url-pattern>
    </servlet-mapping>


    <servlet>
        <servlet-name>com.mySampleApplication.MySampleApplication MySampleApplica-
tionService</servlet-name>
        <servlet-class>com.mySampleApplication.server.MySampleApplicationService-
Impl</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>com.mySampleApplication.MySampleApplication MySampleApplica-
tionService</servlet-name>
        <url-pattern>/MySampleApplication/MySampleApplicationService</url-pattern>
    </servlet-mapping>
```

```
    <!-- Добавление авторизации пользователей -->
    <security-role>
        <role-name>admin</role-name>
    </security-role>

    <security-role>
        <role-name>manager</role-name>
    </security-role>

    <security-role>
        <role-name>user</role-name>
    </security-role>

    <security-constraint>
        <web-resource-collection>
            <web-resource-name>/</web-resource-name>
            <url-pattern>/</url-pattern>
            <http-method>GET</http-method>
            <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
            <role-name>admin</role-name>
            <role-name>manager</role-name>
        </auth-constraint>
    </security-constraint>

    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>Write Post List</realm-name>
    </login-config>
</web-app>
```

Демонстрация результатов

Предварительно компилируем проект GWT, после чего можно генерировать war архив и загружать его на сервер Apache Tomcat. Дожидаясь окончания загрузки, переходим по адресу http://localhost:

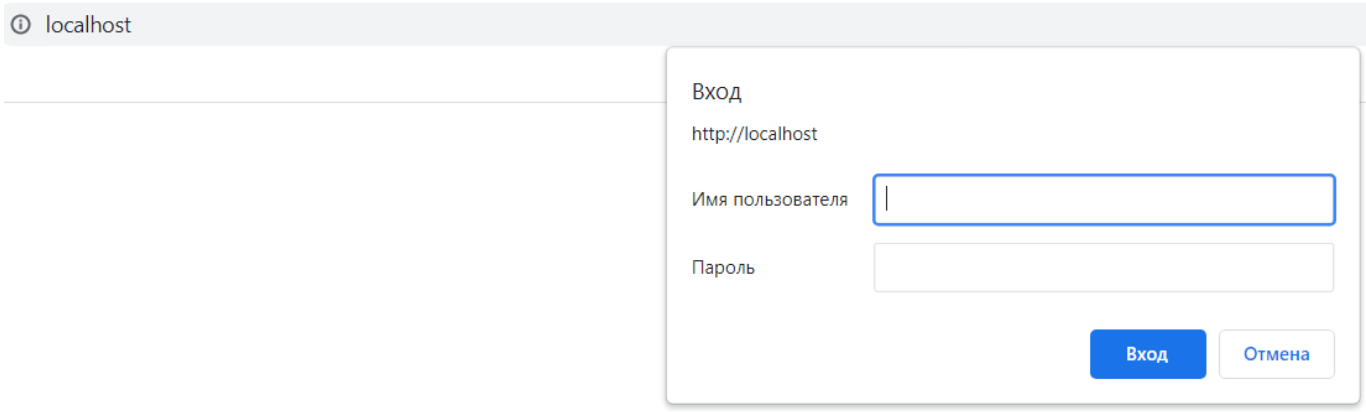

Рис.6. Аутентификация на сайте

Заходим в качестве администратора (admin, admin) и переходим на страницу приложения:

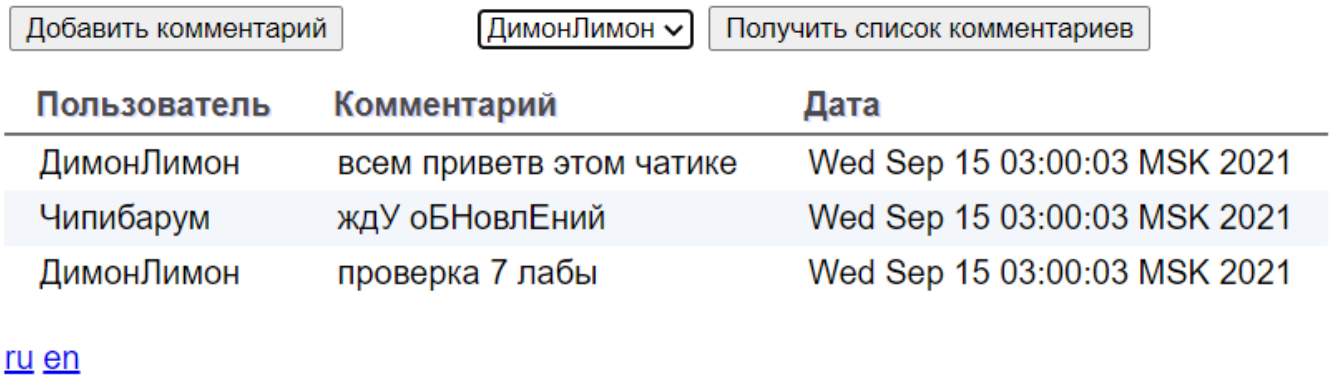

Рис.7. Стартовая страница приложения

Сразу добавим новый комментарий, нажав на соответствующую кнопку. Развернётся следующая форма добавления записи:

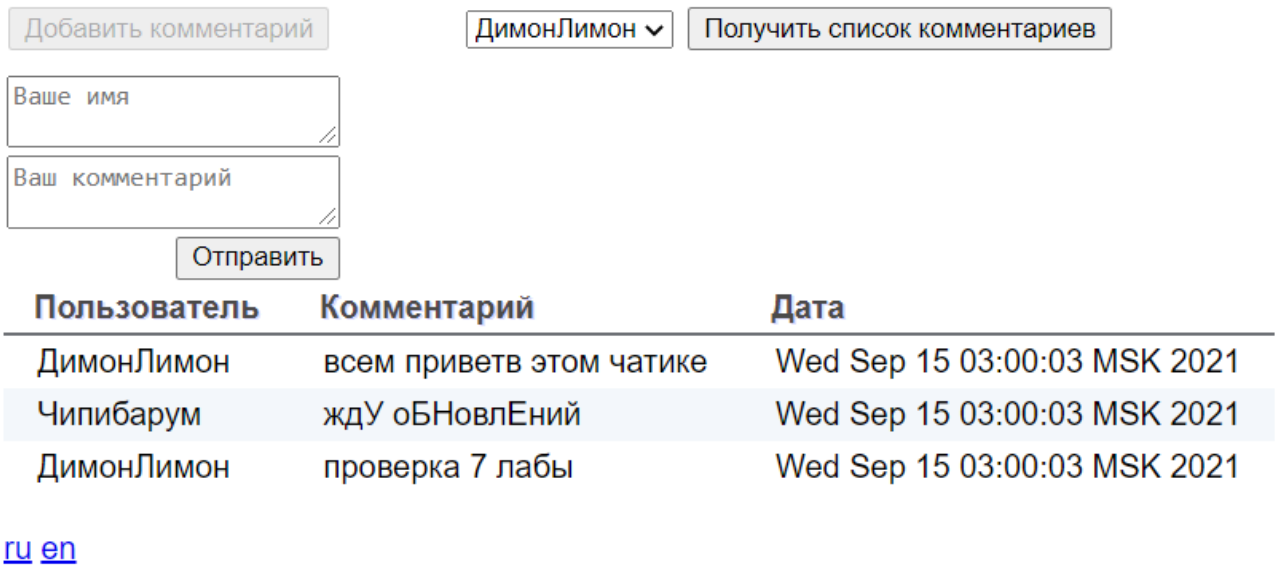

Рис.8. Форма добавления записи

Например, вводим «ДимонЛимон» и «Тест» в поля имени и комментария соответственно и нажимаем кнопку «Отправить»:

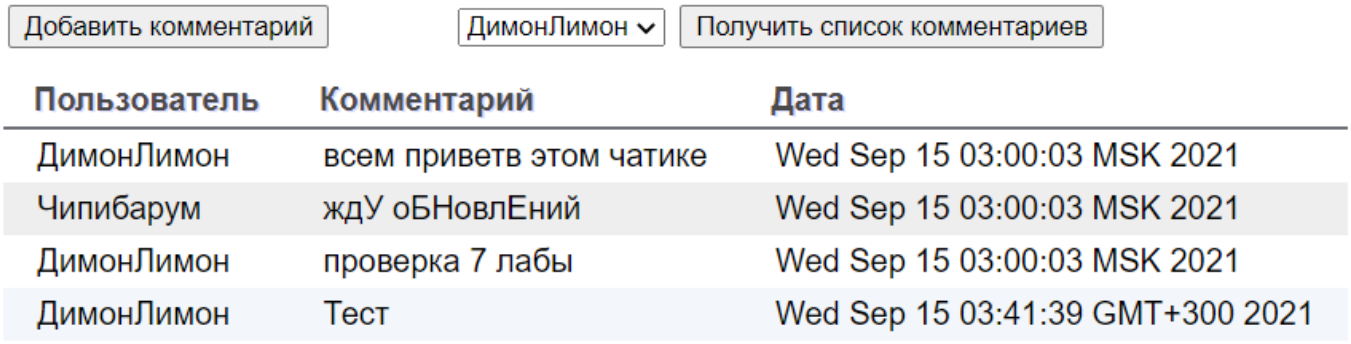

Рис.9. Отображение добавленной записи

Форма добавления вновь скрывается, а кнопку «Добавить» вновь становится активной. Теперь попробуем получить все комментарии пользователя «ДимонЛимон», выбрав в выпадающем меню соответствующее имя пользователя и нажав «Получить список комментариев»:

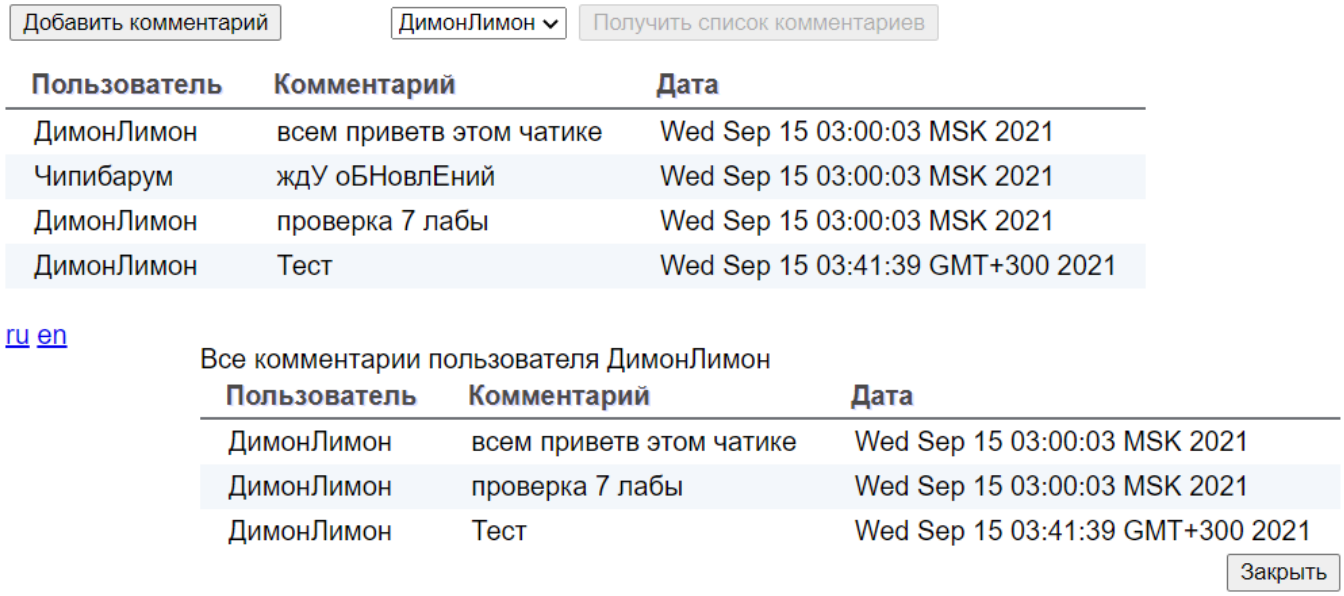

Рис.10. Все записи выбранного пользователя

Диалоговое окно можно перемещать в любое место страницы, единовременно можно просматривать комментарии лишь одного пользователя, поэтому кнопка получения становится неактивной до момента закрытия диалогового окна. Можем нажать «Закрыть».

И последним шагом проверим локализацию приложения, для этого имеются ссылки внизу страницы «ru» и «en» для русской и английской версии приложения соответственно. Перейдем на английскую версию:

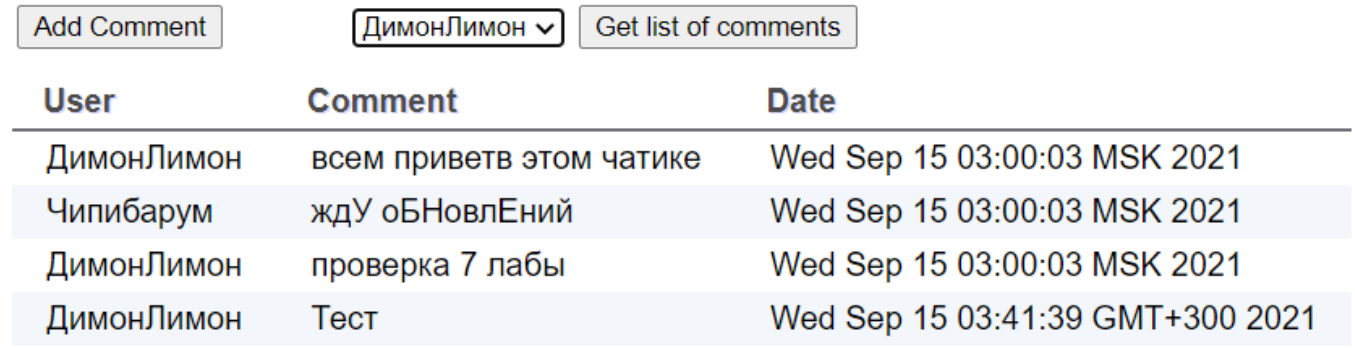

Рис.11. Английская версия главной страницы

Форма добавления записи:

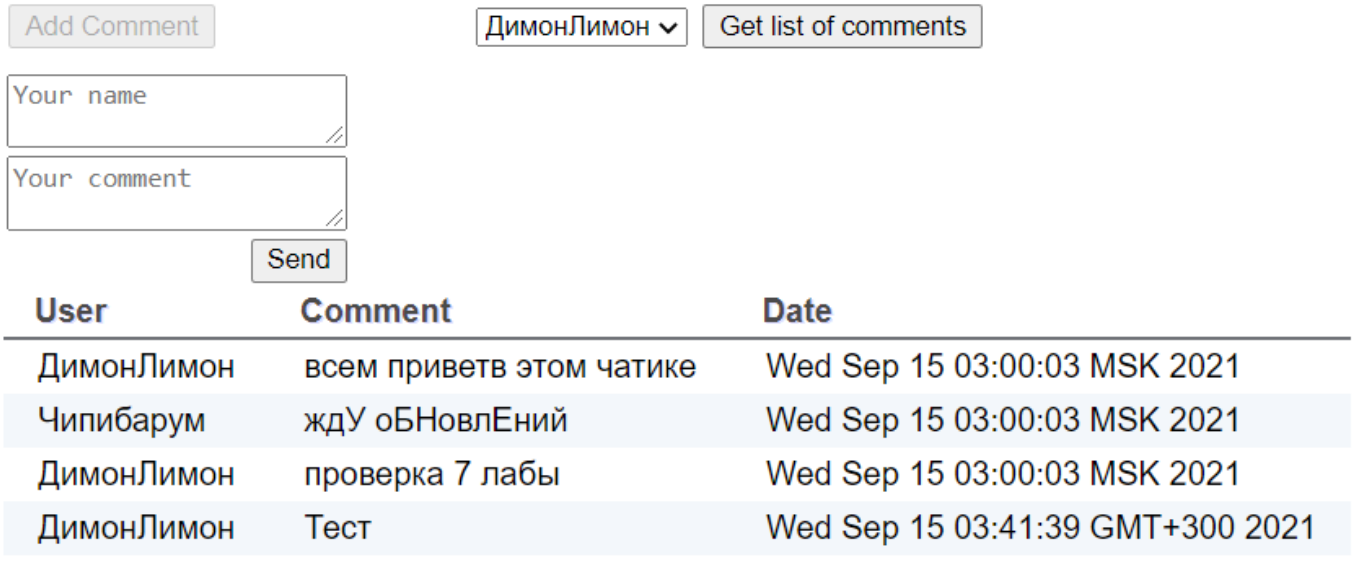

Рис.12. Форма добавления записи

И последнее, получение всех комментариев пользователя:

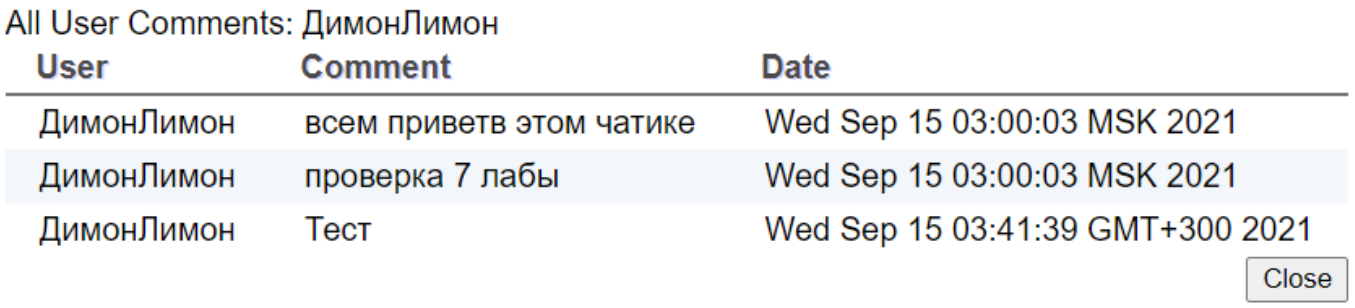

Рис.13. Окно всех комментариев выбранного пользователя

Вывод

В ходе выполнения лабораторной работы был изучен процесс создания GWT-приложения в среде IntelliJ IDEA и последующего развёртывания приложения на локальном сервере Apache Tomcat.