

Installing the GWT

First of all make sure you have installed a [Java SDK version 1.6 or later](#) and also you need to install [Ant](#).

For downloading the GWT you have 2 possibilities.

1. You download [the GWT SDK zip](#) and then unzip it in a directory of your choice.
2. Or you install a plugin in Eclipse. The how-to you can find [here](#).

Starting GWT for the first time

Now we can start to make our first GWT web application. There are also 2 possibilities to make the first sample application, one with the command line and the other one is with Eclipse.

Command line

For Windows:

```
cd myDirectory/gwt-2.7.0

webAppCreator -out <nameOfYourApp>
               -junit "pathToEclipse/plugins/org.junit_4.x/junit.jar"
               com.example.myapp.<nameOfYourApp>
```

The line `-junit` is not necessary, but if you want to create JUnit tests you need it.

For Mac or Linux

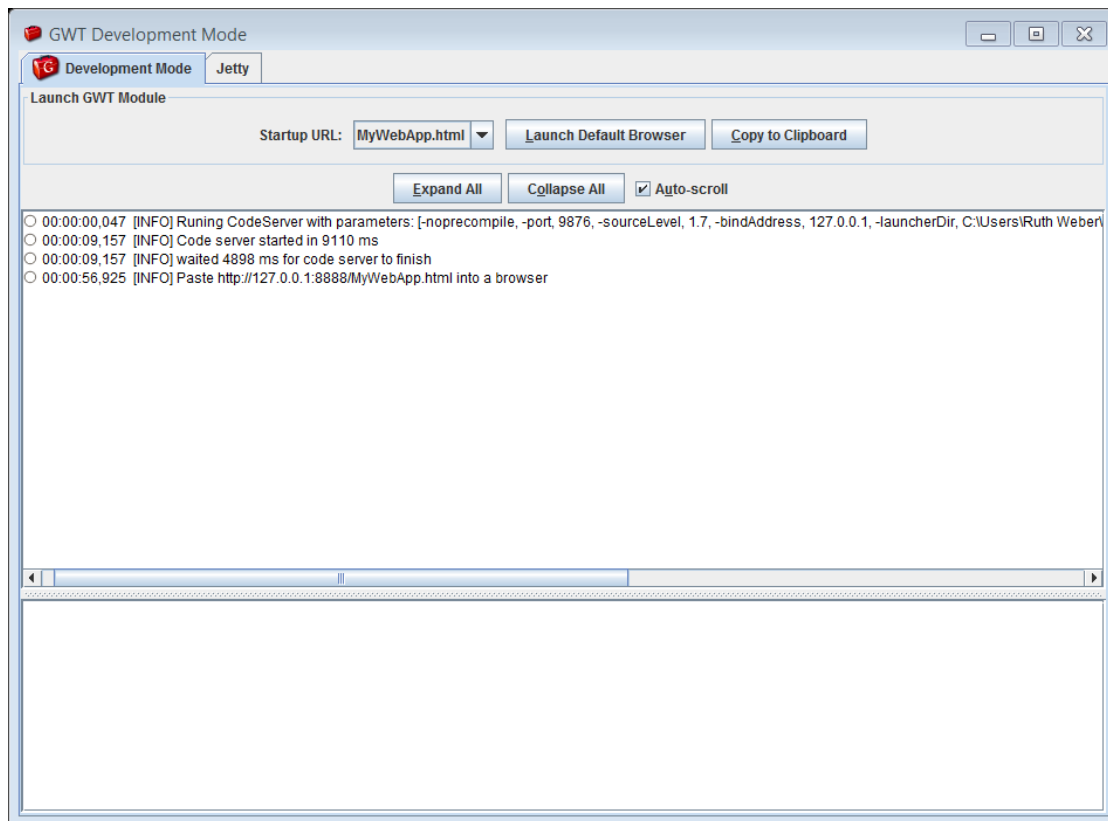
```
cd myDirectory/gwt-2.7.0

chmod u+x webAppCreator
./webAppCreator -out <nameOfYourApp>
                -junit "pathToEclipse/plugins/org.junit_4.x/junit.jar"
                com.example.myapp.<nameOfYourApp>
```

Through these commands there are automatically created some files. These are also executable, so we can run it for the first time.

```
cd <nameOfYourApp>
ant devmode
```

The command line starts the GWT Development Mode.



Now you can launch it in a Browser. It looks like this


Web Application Starter Project

Please enter your name:

You can enter your name and there will be a pop up with the answer from the server.

Eclipse

Go to File -> New -> Other -> Google -> Web Application Project

 New Web Application Project

Create a Web Application Project

Create a Web Application project in the workspace or in an external location

Project name:

Package: (e.g. com.example.myproject)

Location

☒ Create new project in workspace

☐ Create new project in:

Directory:

Google SDKs

☒ Use Google Web Toolkit

☒ Use default SDK (GWT - 2.6.0) [Configure SDKs...](#)

☐ Use specific SDK:

☐ Use Google App Engine

☒ Use default SDK (App Engine - 1.9.25) [Configure SDKs...](#)

☐ Use specific SDK:

The project will use App Engine's [High Replication Datastore \(HRD\)](#) by default.

Identifiers for Google App Engine

☒ Leave App Id field blank


☐ Use App Id

Your app will be deployed at:

- <http://yourappid.appspot.com> for regular applications
- <http://yourappid.yourdomain.com> for domain applications

Sample Code

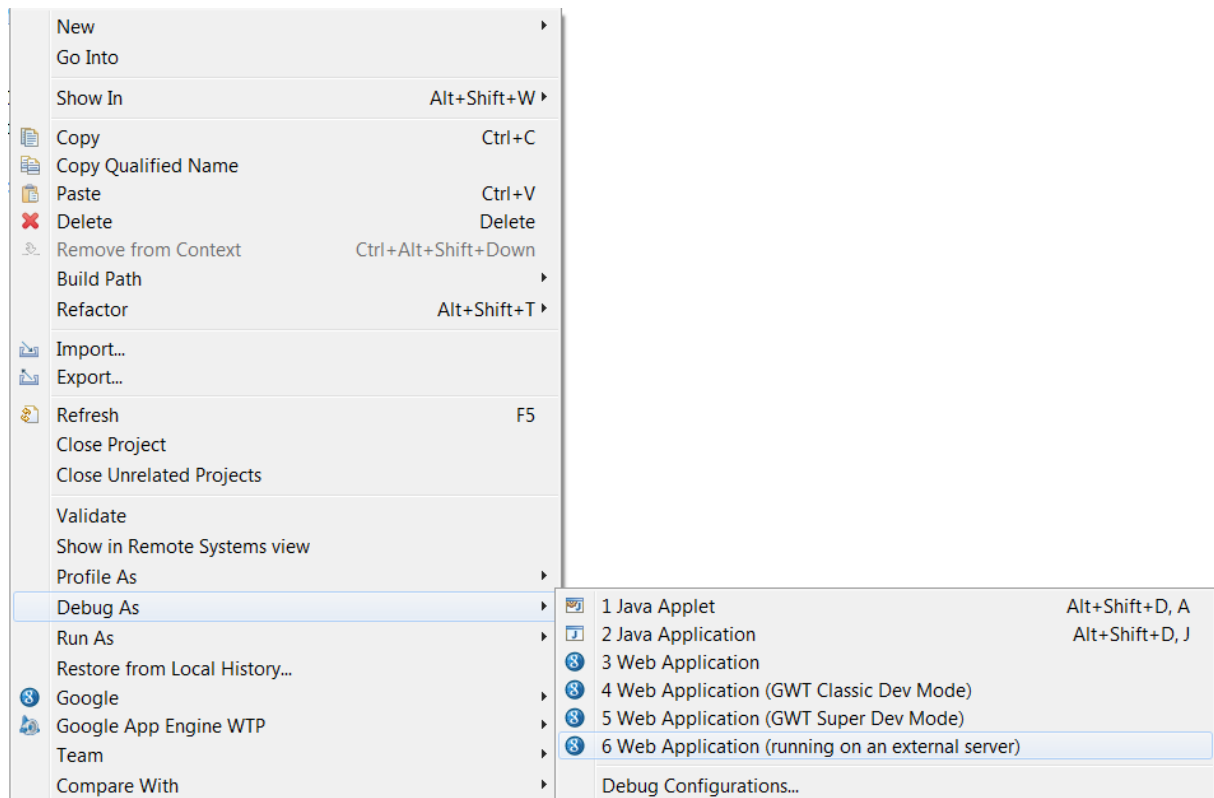
☒ Generate project sample code



Fill in the information above and click finish.

Now you have also created the sample app in Eclipse.

Now right click on the project -> Debug As -> Web Application (GWT Super Dev Mode)



Perhaps you need to install the GWT Developer Plugin, but it will show up if you need it.

Now you can see the same site as shown while using the command line.

You have now created your first example.

You do not need to cancel the development mode and restart it. When you started it ones you can see in the right bottom corner a red refresh button, when you click it, the application is compiling.

Tutorial

So now you can make your own application.

First of all make a new web application, like shown above e.g. a Stockwatcher. The following tutorial shows how to implement it in Eclipse. But it works nearly the same in IntelliJ, but for launching the GWT app you need to use the command line.

For a stock watcher we need some widgets to make it happen. We need an input field and a button also we need a table which contains the data with the columns 'price', 'change' and a button to delete the row. Also we need a label which shows when it last was updated.

For the layout we need also a horizontal panel, a vertical panel and a root panel which we use as a container.

So now let's start.

In the /WEBINF/Stockwatcher.html we change the following things.

```
<h1>Stockwatcher 2016</h1>
```

```
<div id="stockList"></div>
```

The "<div>" part is a placeholder for the table of the stock data.

Next we go in `.../stockwatcher.client` and open `Stockwatcher.java`. There we need to instantiate the needed widgets.

We replace the code from the starter app, with the following.

```
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.FlexTable;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.VerticalPanel;

/**
 * Entry point classes define onModuleLoad().
 */
public class Stockwatcher implements EntryPoint {

    // needed panels
    // contains the stockTable
    private VerticalPanel vertPanel = new VerticalPanel();
    //contains the input widgets
    private HorizontalPanel horzPanel = new HorizontalPanel();

    // needed widgets
    private FlexTable stockTable = new FlexTable();
    private Button addStock = new Button("Add stock");
    private TextBox stockInput = new TextBox();
    private Label update = new Label();
}
```

In the table you need to set up the header row, which is made in `onModuleLoad()`. And also add the widgets to the panels.

```
public void onModuleLoad() {
    // create Table
    stockTable.setText(0, 0, "Stock symbol");
    stockTable.setText(0, 1, "Current price");
    stockTable.setText(0, 2, "Change (in percent)");
    stockTable.setText(0, 3, "Remove");

    // Horizontal panel textbox and button on panel
    horzPanel.add(stockInput);
    horzPanel.add(addStock);

    // Vertical panel for table
    vertPanel.add(stockTable);
    vertPanel.add(horzPanel);
    vertPanel.add(update);
}
```

After that you need to associate the panel, which has all the widgets, with the HTML host page. Here you insert the id you assigned to the <div> in the 'Stockwatcher.html' file. Also it would be nice if the input field has the focus.

```
// connect the HTML host page with the vertical panel
RootPanel.get("stockList").add(vertPanel);

// Set the focus in the input field
stockInput.setFocus(true);
```

Restart the development mode and after starting it new and copying it in the browser you should see this.



Now the button for add a stock need to be handled. Also by clicking the enter key the stock should be added.

```
// If the button is clicked or enter is pressed
// stock should be added
addStock.addClickHandler(new ClickHandler() {

    @Override
    public void onClick(ClickEvent event) {
        addStock();
    }

});

stockInput.addKeyDownHandler(new KeyDownHandler() {

    @Override
    public void onKeyDown(KeyDownEvent event) {
        if (event.getNativeKeyCode() == KeyCodes.KEY_ENTER) {
            addStock();
        }
    }

});

private void addStock() {
}
```

Now it is time to implement the method addStock.

For this we want only symbols which start with numbers from 0 to 9 or A to Z and then there can come anything, but it should not be longer than 10. (^[0-9A-Z\\.|]{1,10}\$) We want to add just uppercase symbols. After it is assigned to the variable the input field can be cleared.

```

private void addStock() {
    // Complete text from input field all uppercase and blanks are deleted
    final String stockSymbol = stockInput.getText().toUpperCase().trim();

    if (!stockSymbol.matches("[0-9a-zA-Z\\.]{1,10}$")) {
        Window.alert(stockSymbol + "is not a valid input.");
        stockInput.selectAll();
        return;
    }
    stockInput.setText("");
}

```

After refreshing your browser you should get the alert message and the input field should be cleared.



Now you need to create the structure.

```

// Data structure
private ArrayList<String> stocks = new ArrayList<String>();

```

And add the stocks to the table if they not already there. And also create the remove button in every row.

```

if (stocks.contains(stockSymbol))
    return;

int row = stockTable.getRowCount();
stocks.add(stockSymbol);
stockTable.setText(row, 0, stockSymbol);

Button removeStock = new Button("x");
removeStock.addClickHandler(new ClickHandler() {

    @Override
    public void onClick(ClickEvent event) {
        int removeStockIndex = stocks.indexOf(stockSymbol);
        stocks.remove(removeStockIndex);
        stockTable.removeRow(removeStockIndex + 1);
    }
});
stockTable.setWidget(row, 3, removeStock);

```

By now you can add stock symbols and remove them.

Now we need to refresh the price. For this there is a new method which is called every time the constant is accomplished.

```
private void refreshStockTable() {
    // TODO Auto-generated method stub

}
```

This is also called when a stock is added.

```
//update of the price
refreshStockTable();
```

The table will be updated every 10 seconds.

```
public class Stockwatcher implements EntryPoint {

    //Time in which the table is refreshed
    private static final int REFRESH_TIME = 10000;
```

Also there is a new instance of timer created in the on ModuleLoad().

```
Timer refresh = new Timer() {

    @Override
    public void run() {
        refreshStockTable();
    }

};

refresh.scheduleRepeating(REFRESH_TIME);
```

Now it is time to make a new class for the stock price.

```
private String stockSymbol;
private double stockPrice;
private double stockPriceChange;

public StockPrice() {

}

public StockPrice(String stockSymbol, double stockPrice, double stockPriceChange) {
    super();
    this.stockSymbol = stockSymbol;
    this.stockPrice = stockPrice;
    this.stockPriceChange = stockPriceChange;
}
```

Additionally you also have to create the getters and setters. Also you need to implement a method which calculates the change in percent.

```
public double getStockPriceChangePercent() {
    return 100.0 * this.stockPriceChange / this.stockPrice;
}
```


So now that in the table is shown the stock price you need to implement the refreshStockTable method.

```
private void refreshStockTable() {
    final double MAX_PRICE = 100.0;
    final double MAX_CHANGE = 0.02;

    StockPrice[] prices = new StockPrice[stocks.size()];
    for (int i = 0; i < stocks.size(); i++) {
        double price = Random.nextDouble() * MAX_PRICE;
        double change = price * MAX_CHANGE * (Random.nextDouble() * 2.0 - 1.0);

        prices[i] = new StockPrice(stocks.get(i), price, change);
    }

    updateTable(prices);
}
```

To show the randomly generated prices in the table we create a method to update the table.

```
private void updateTable(StockPrice[] prices) {
    for (int i = 0; i < prices.length; i++) {
        updateTable(prices[i]);
    }
}

private void updateTable(StockPrice stockPrice) {
    // if the stock doesn't exist do nothing
    if (!stocks.contains(stockPrice.getStockSymbol())) {
        return;
    }

    // if it exists, show it in the format
    int row = stocks.indexOf(stockPrice.getStockSymbol()) + 1;

    String priceText = NumberFormat.getFormat("#,##00.0").format(stockPrice.getStockPrice());
    NumberFormat changePriceFormat = NumberFormat.getFormat("+#,##0.00; -#,##0.00");
    String changePriceText = changePriceFormat.format(stockPrice.getStockPriceChange());
    String changePercentText = changePriceFormat.format(stockPrice.getStockPriceChangePercent());

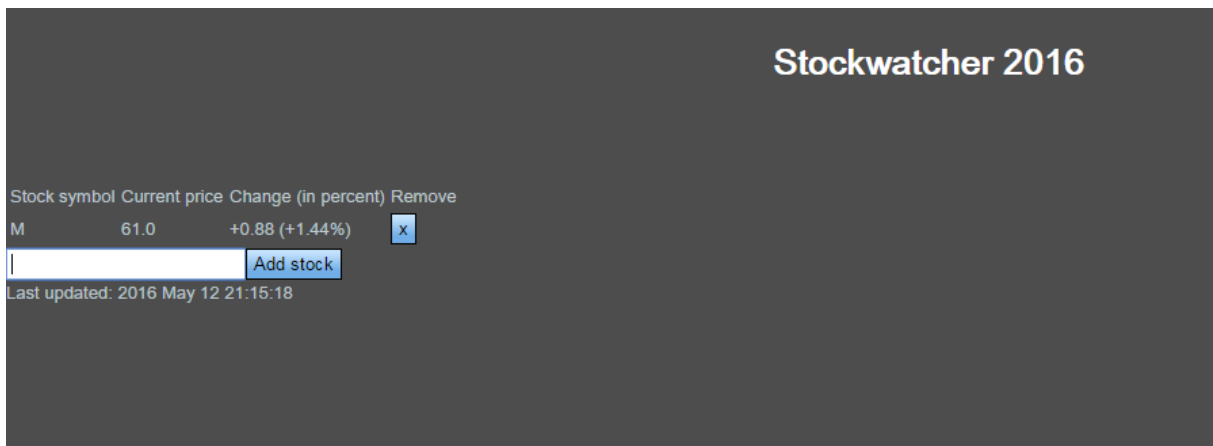
    stockTable.setText(row, 1, priceText);
    stockTable.setText(row, 2, changePriceText + " (" + changePercentText + "%)");
}
```

So now like said at the beginning there should be a label which shows when the table is last updated. For this the method updateTable is changed.

```
private void updateTable(StockPrice[] prices) {
    for (int i = 0; i < prices.length; i++) {
        updateTable(prices[i]);
    }

    // display timestamp for last updated
    DateTimeFormat dateTimeFormat = DateTimeFormat.getFormat(
        DateTimeFormat.PredefinedFormat.DATE_TIME_MEDIUM);
    update.setText("Last updated: " + dateTimeFormat.format(new Date()));
}
```

It should look something like that.



Extended

When you want to test your created application you can easily write Junit tests for it. This works with Eclipse and IntelliJ the same.

Create a class like this.

```
package com.example.stockwatcher.client;

import com.google.gwt.junit.client.GWTTestCase;

public class StockwatcherTest extends GWTTestCase{

    @Override
    public String getModuleName() {
        // TODO Auto-generated method stub
        return null;
    }

}
```

The auto-generated method should return the main class of your project.

```
@Override
public String getModuleName() {
    // TODO Auto-generated method stub
    return "com.example.stockwatcher.Stockwatcher";
}
```

To test if it works, write a method which should always be true.

```
@Test
public void testSimple() {
    assertTrue(true);
}
```

Now you can run the tests.

My Solution

https://github.com/RuWe/e_Portfolio_GWT