Nolix

Nolix web applications

2025-07-04

Table of contents

1 Introduction

1.1 What Nolix web applications are

The Nolix web applications are a framework to create web applications in pure Java. The Nolix web applications are completely object-oriented.

1.2 Why to use Nolix web applications

- Nolix web applications can be accessed with **all web browsers**.
- Nolix web applications can be **styled dynamically** like CSS styles HTML pages.
- Since Nolix web applications are in pure Java, there is required not any knowledge of HTM, CSS or JavaScript. Furthermore, Nolix web applications can be written in pure Java and do not require resource files or configuration files. Nolix web applications **can be learned in dramatical short time** compared to other web frameworks. This is a big step forward to sustainable software and low maintenance costs.

1.3 Where the Nolix web applications are

The Nolix web applications are in the Nolix library. To use Nolix web applications, import the Nolix library into your project.

1.4 Structure of this document

This document leads straight forward through the main mechanisms of the Nolix web applications. The Nolix web applications provide many features which all cannot be covered by this document.

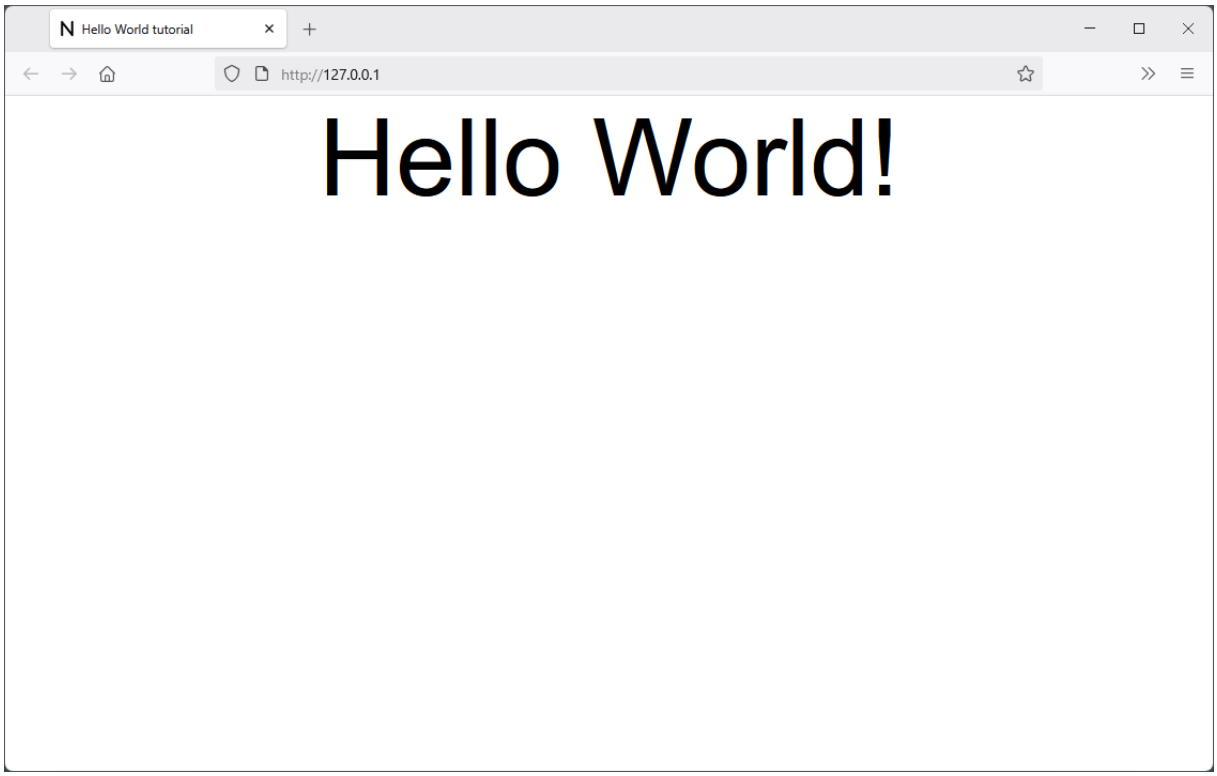
The chapters of this document are ordered that the reader does not need to jump back to a previous chapter. Each step is visualized by a practical example.

1.5 Hello-world-program

Code

```
import ch.nolix.system.application.main.Server;
import ch.nolix.system.application.main.Application;
import ch.nolix.system.application.webapplication.WebClientSession;
import ch.nolix.system.webgui.atomiccontrol.label.Label;
import ch.nolix.systemapi.webguiapi.mainapi.ControlState;
…
//create a Server that will listen to clients on the HTTP port
Server server = Server.forHttpPort();

//add a default Application to the Server
server.addDefaultApplication(new CustomApplication());

//defines a custom Application
public class CustomApplication extends Application<WebClient, Object> {

  @Override
  public String getApplicaitonName() {
    return "Demo";
  }

  @Override
  protected Class<?> getInitialSessionClass() {
    return CustomSession.Class;
  }
}

//define a custom WebClientSession
public class CustomWebClientSession extends WebClientSession<Object>

    @Override
    protected void initialize() {

      //create a Label
      Label label = new Label().setText("Hello World!");

      //configure the style of the Label
      label.getStoredStyle().setTextSizeForState(ControlState.BASE, 100);

      //add the Label to the GUI
      getStoredGui().pushLayerWithRootControl(label);
    }
  }
}
```

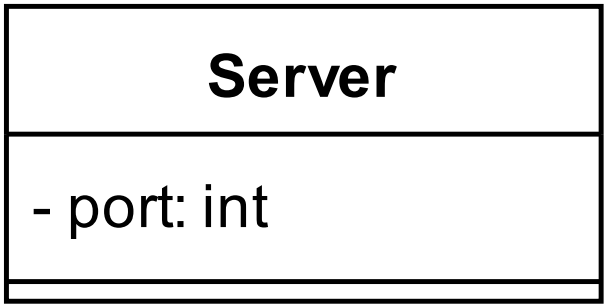
Output



2 Servers

2.1 Root

2.1.1 Server object



A server program is required for any web GUI. Nolix uses a Server object to represent a server program. A Server listens to clients on a specific port.

2.1.2 Create a Server for default port

```
import ch.nolix.system.application.main.Server;
…
Server server = Server.forHttpPort();
```

The forHttpPort static method creates a Server that will listen to clients on the HTTP port. The Server starts automatically when it is created.

The Server class is in the ch.nolix.system.application.main package.

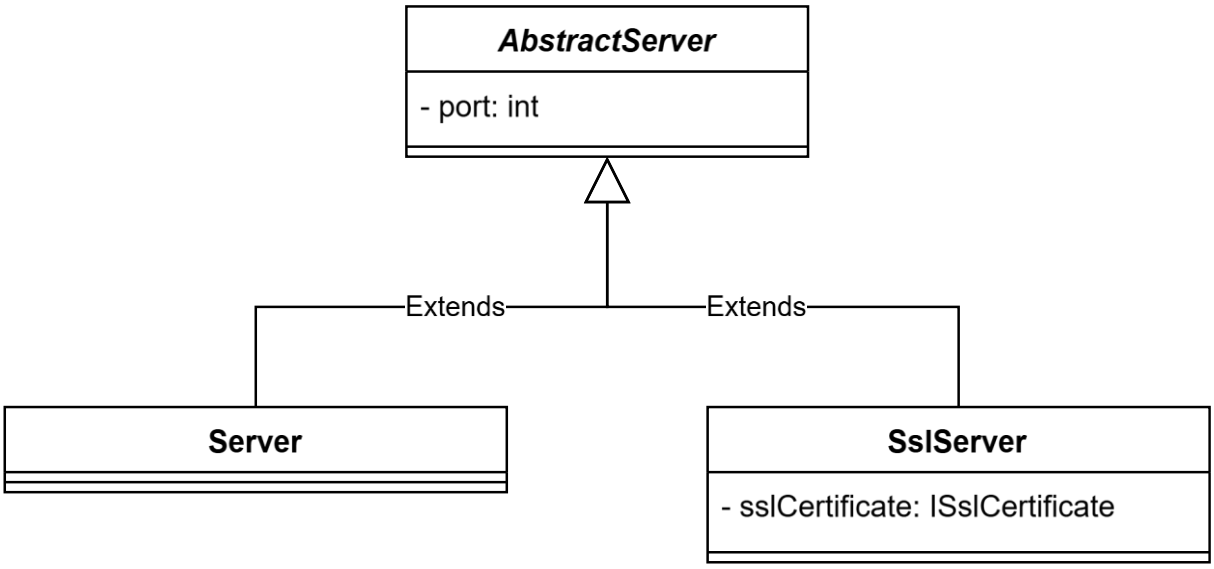
2.1.3 Create a Server for specific port

```
Server server = Server.forPort(50000);
```

The static forPort method creates a Server that will listen to clients on the given port. The Server starts automatically when it is created.

2.2 SslServers

2.2.1 SslServer object



A SslServer is like a Server that can handle SSL certificates. A SslServer listens to clients on a specific port.

2.2.2 Purpose

Server	SslServer
Can not handle SSL certificates.	Can handle SSL certificates.
Designed for development on local computers without a domain.	Designed for production systems that provide SSL certificates
Does not need any configurations.	Needs either the paths of PEM files or a Nolix configuration that tells where the PEM files of the certificates are stored.

2.2.3 Create an SslServer for default port

```
import ch.nolix.system.application.main.SslServer;
…
Server server =
Server.forHttpsPortAndDomainAndSSLCertificateFromNolixConfiguration(
  "nolix.tech"
);
```

The forHttpsPortAndDomainAndSSLCertificateFromNolixConfiguration static method creates a SslServer that:

- will listen to clients on the **HTTPS port**
- will use an SSL certificate for the **given domain**
- will find the **default SSL certificate** from the Nolix configuration

The Server starts automatically when it is created. The Nolix configuration is in the app data folder of the operating system.

The SslServer class is in the ch.nolix.system.application.main package.

2.2.4 Provide a default SSL certificate in the Nolix configuration

```
NolixConfiguration(
  DefaultSSLCertificate(
    Domain(nolix.tech),
    PublicKeyPEMFile(C:/certificates/nolix_tech/public_key.pem),
    PrivateKeyPEMFile(C:/certificates/nolix_tech/private_key.pem)
  )
)
```

A Nolix configuration can provide one default SSL certificate.

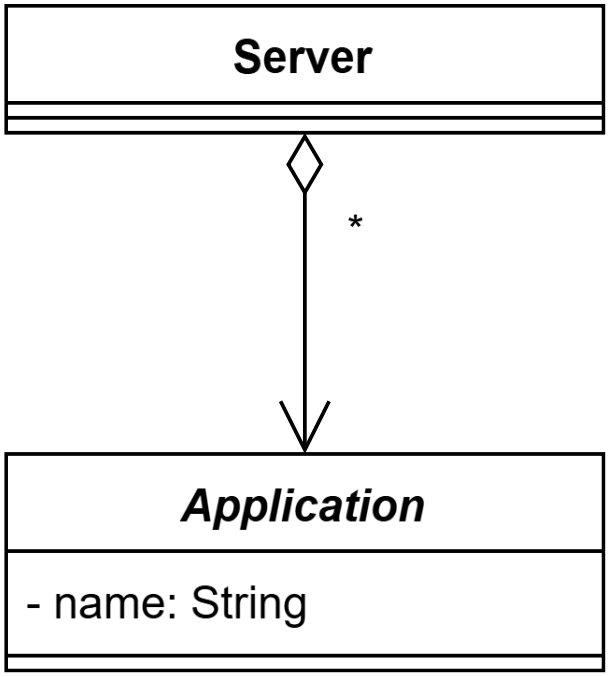
Data of a default SSl certificate:

- the domain of the SSL certificate
- the path of the PEM file with the public key of the SSL certificate
- the path of the PEM file with the private key of the SSL certificate

The Nolix configuration is in the Nolix folder in the app data folder of the operating system.

2.3 Applications

2.3.1 Application object



A server can contain several Applications.

The Application type is abstract. To create an Application, a concrete Application must be defined.

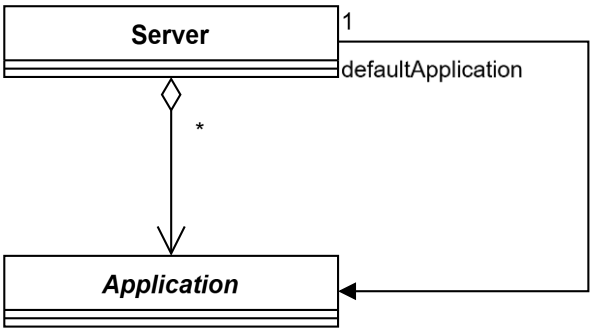
An Application has a name. The Applications of a Server must have distinct names. When a web browser connects to a Server it must provide the name of the target Application in an URL parameter.

2.3.2 Add an Application to a Server

```
Server server;
…
server.addApplication(new CustomApplication());
```

The addApplication method will add the given Application to the Server.

2.3.3 Default Application



A Server can provide one of its Applications as default Application. When a web browser connects to the Server and does not specify a target Application, the web browser will be forwarded to the default Application of the Server.

2.3.4 Add a default Application to a Server

```
Server server;
…
server.addDefaultApplication(new CustomApplication());
```

The addDefaultApplication will add the given Application as default Application to the Server.

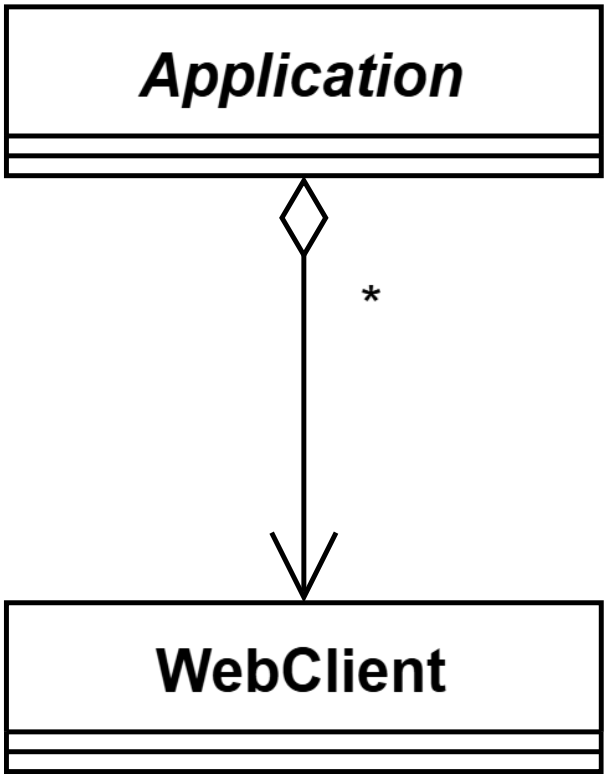
2.3.5 Define a custom Application

```
import ch.nolix.system.application.main.Application;
import ch.nolix.system.application.webapplication.WebClient;

public class CustomApplication extends Application<
  WebClient<Object>,
  Object
> {

  public CustomApplication() {
    super(new VoidObject());
  }

  @Override
  public getApplicationName() {
    return "Demo";
  }

  @Override
  protected Class<?> getInitialSessionClass() {
    return CustomSession.class;
  }
}
```

The CustomApplication is an Application for WebClients and has an application service of the type Object. The constructor of the CustomApplication must call the super constructor of the Application and hand over its application service. The application service must be of the declared application service type.

The Application class is in the ch.nolix.system.application.main package. The WebClient class is in the ch.nolix.system.application.webapplication package.

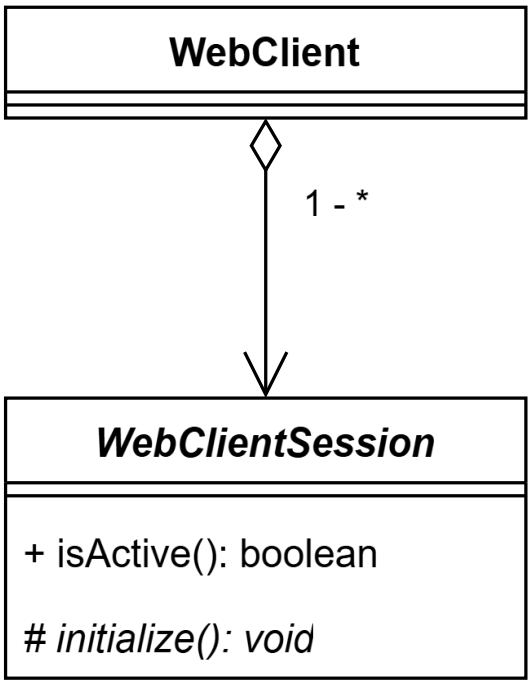
2.4 WebClients



When a web browser connects to an Application on a Server, a WebClient represents the backend of the connection. An Application knows all its WebClients.

2.5 WebClientSessions

2.5.1 WebClientSession object



A WebClient contains one or several WebClientSessions. A WebClientSession has exactly one active WebClientSession. The active WebClientSession of a WebClient defines the actions the WebClient currently can do.

The WebClientSession type is abstract. To create a WebClientSession, a concrete WebClientSession must be defined.

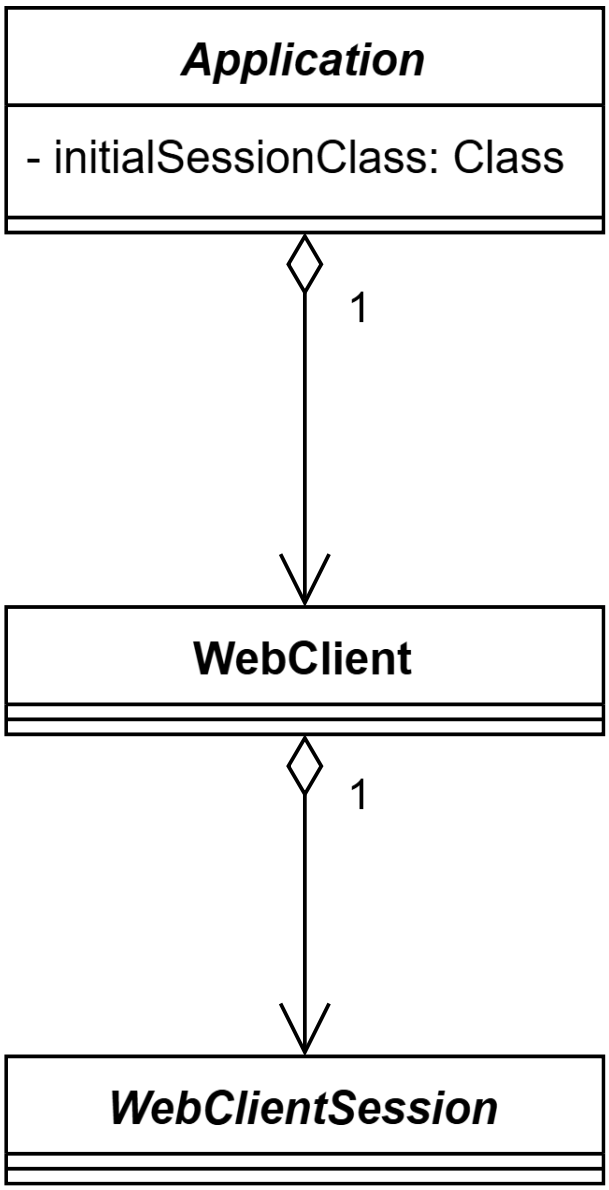
A WebClientSession has an initialize method. The initialize method is called when a WebClient is assigned to a new WebClientSession.

2.5.2 Initial session

When a web browser connects to an Application on a Server, the following steps happen:

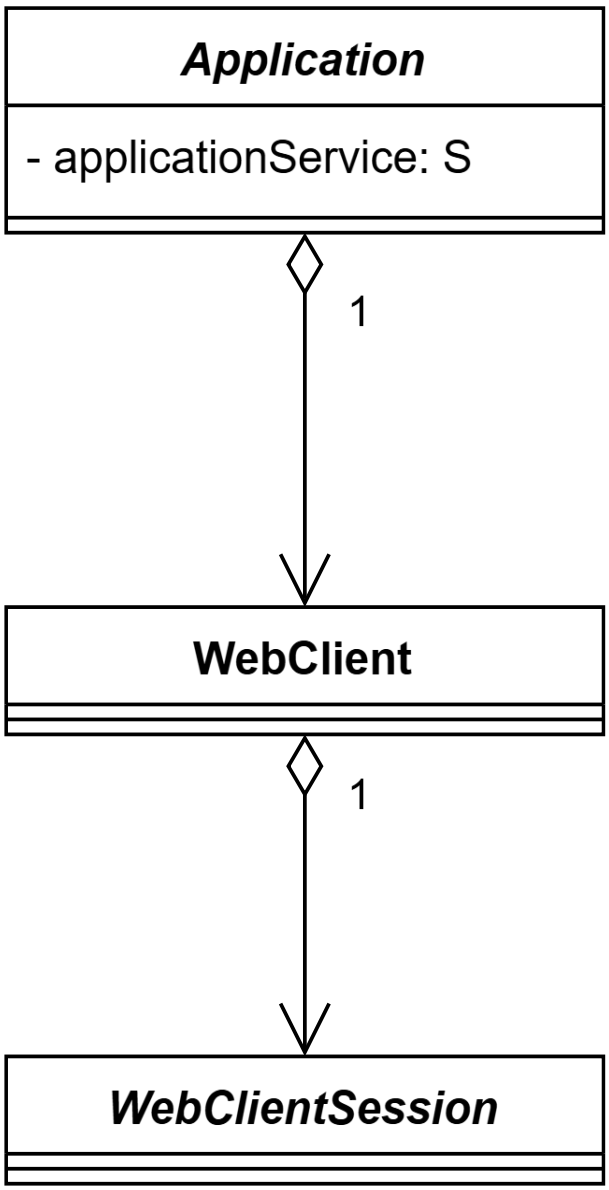
1. On the Application a new WebClient is created.
2. A new WebClientSession is assigned to the new WebClient. This WebClientSession is the initial session of the WebClient.
3. The initialize method of the new WebClientSession is invoked.

2.5.3 Initial session class



For that a WebClient can receive an initial session, the Application of the WebClient has an initial session class.

2.5.4 Application service



An Application has an application service of a specific type. The application service has the same lifetime as its Application. A WebClientSession can access the application service of the Application of its WebClient.

2.5.5 Define a custom WebClientSession

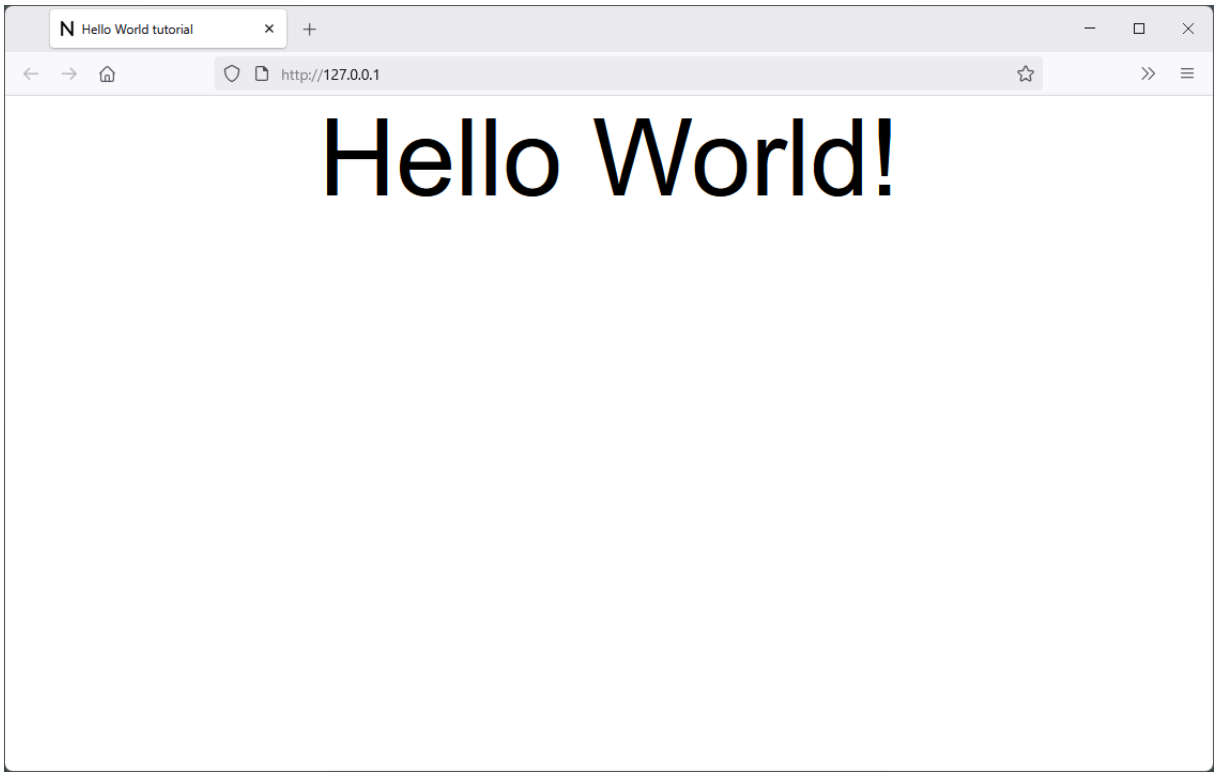
Code

```
import ch.nolix.system.application.webapplication.WebClientSession;

public class CustomWebClientSession extends WebClientSession<Object> {

  @Override
  protected void initialize() {

      //create a Label
      Label label = new Label().setText("Hello World!");

      //configure the style of the Label
      label.getStoredStyle().setTextSizeForState(ControlState.BASE, 100);

      //add the Label to the GUI of the current CustomWebClientSession
      getStoredGui().pushLayerWithRootControl(label);
  }
}
```

The CustomWebClientSession is a WebClientSession for an Application whose application service can be any Object. The CustomWebClientSession overrides the initialize method so that the web browser shows a ‘Hello World!’ text.

The WebClientSession class is in the ch.nolix.system.application.webapplication package.

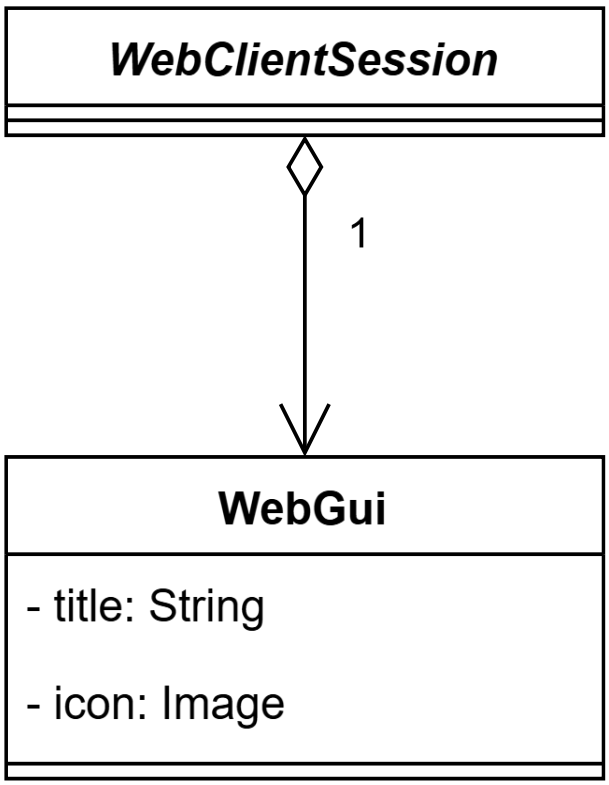
Output



3 WebGuis

3.1 Root

3.1.1 WebGui object



A WebClientSession has one WebGui. A WebGui represents the GUI of a web page. A WebClientSession can access its WebGui.

A WebGui includes attributes like title, icon, background color and many more.

3.1.2 Set the title of a WebGui

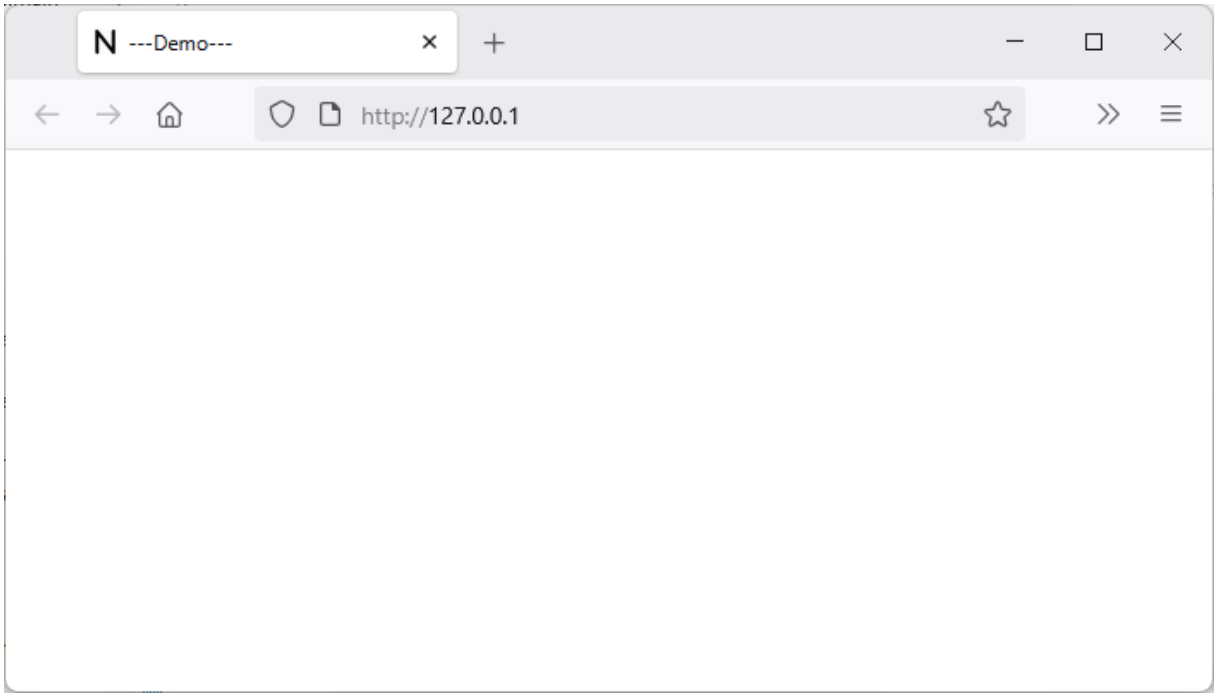
Code

```
public class CustomWebClientSession extends WebClientSession<Object> {

  @Override
  protected void initialize() {
    getStoredGui().setTitle("---Demo---");
  }
}
```

The setTitle method of a WebGui sets the given title to the tab of the web browser.

For default, the title of a WebGui is set from the name of the Application of the WebClientSession of the WebGui.

Output



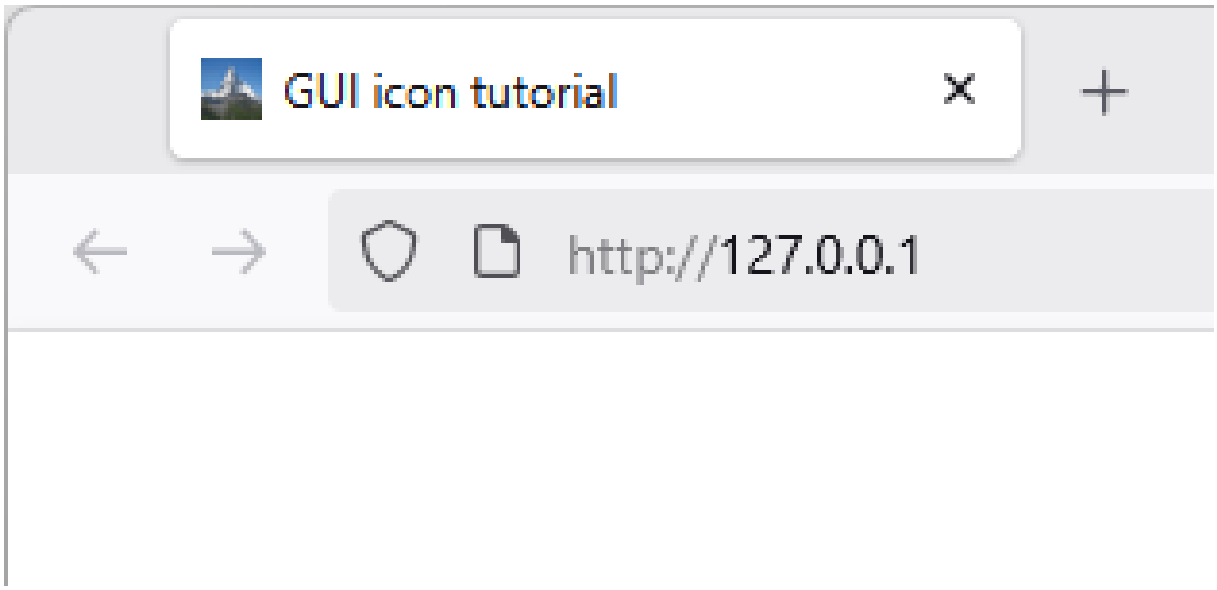
3.1.3 Set the icon of a WebGui

Code

```
public class CustomWebClientSession extends WebClientSession<Object> {

  @Override
  protected void initialize() {

    //load an Image
    Image image = Image.fromResource("image/matterhorn.jpg");

    //set the Image as icon to the WebGui
    getStoredGui().setIcon(image);
  }
}
```

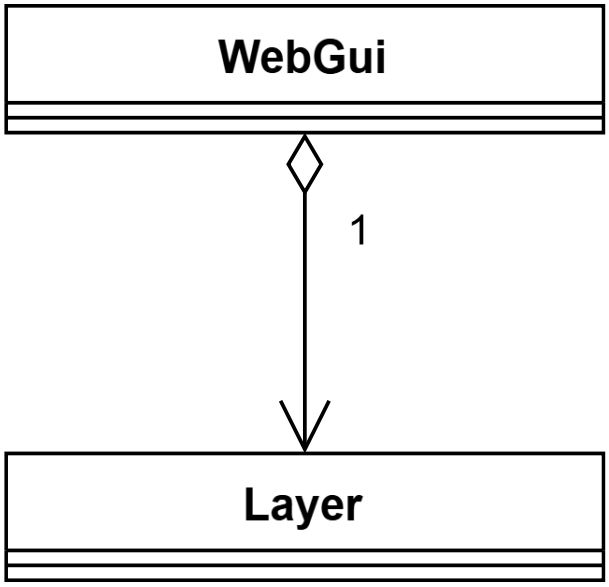
The setIcon method of a WebGui sets the given Image as icon of the tab of the web browser.

Output



3.2 Layers

3.2.1 Layer object



A WebGui can contain several Layers. The Layers of a WebGui are stacked on each other. A Layer can be transparent what makes the Layer behind visible.

3.2.2 Push Layers to a WebGui

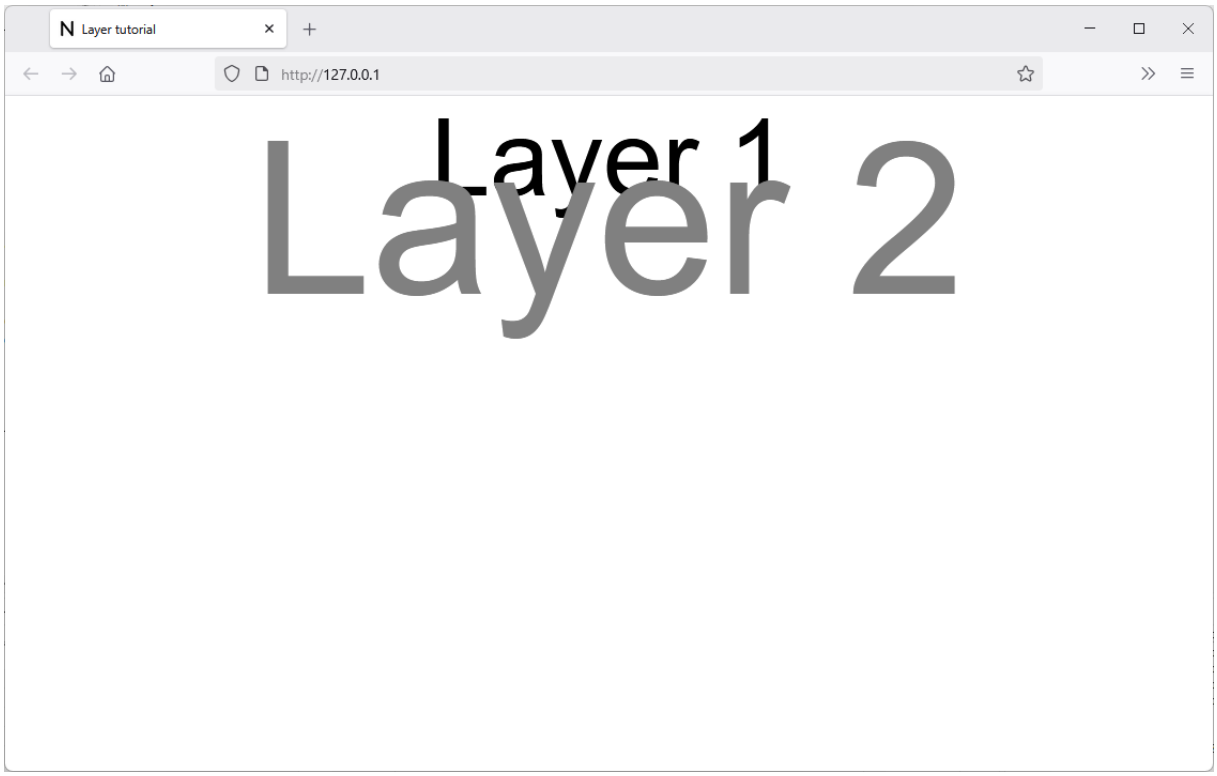
Code

```
public class CustomWebClientSession extends WebClientSession<Object> {

  @Override
  protected void initialize() {

    //create layer1Label
    final Label layer1Label = new Label().setText("Layer 1");

    //create layer2Label
    final Label layer2Label = new Label().setText("Layer 2");

    //configure the style of layer1Label
    layer1Label
      .getStoredStyle()
      .setTextSizeForState(ControlState.BASE, 100)
      .setTextColorForState(ControlState.BASE, X11ColorCatalog.BLACK);

    //configure the style of layer2Label
    layer2Label
      .getStoredStyle()
      .setTextSizeForState(ControlState.BASE, 200)
      .setTextColorForState(ControlState.BASE, X11ColorCatalog.GREY);

    //add a new Layer with the layer1Label to the WebGui
    getStoredGui().pushLayerWithRootControl(layer1Label);

    //add a new layer with the layer2Label to the WebGui
    getStoredGui().pushLayerWithRootControl(layer2Label);
  }
}
```

The pushLayerWithRootControl method of a WebGui pushes a new Layer with the given root Control on top of the WebGui.

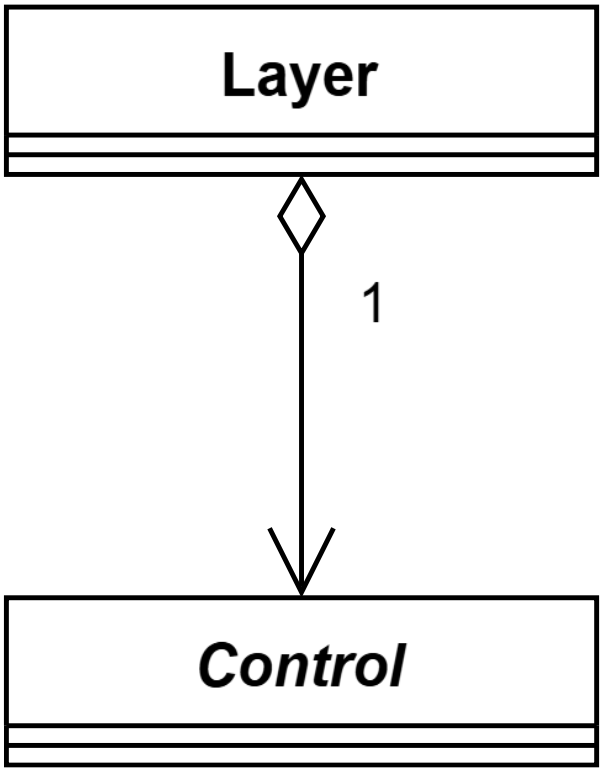
> Controls are explained in a next chapter.

Output



3.3 Controls

3.3.1 Control object



In general, a GUI control is an element on a GUI. In Nolix web GUIs, a Control is an element on a Layer. A Layer can contain one Control.

The Control type is abstract. To create a Control, a Control subtype must be chosen.

- To display not only one Control on a Layer, there exist Controls that can contain multiple other Controls.
- A set of specific Control subtypes is introduced in a next chapter.

3.3.2 Set root Control of a Layer

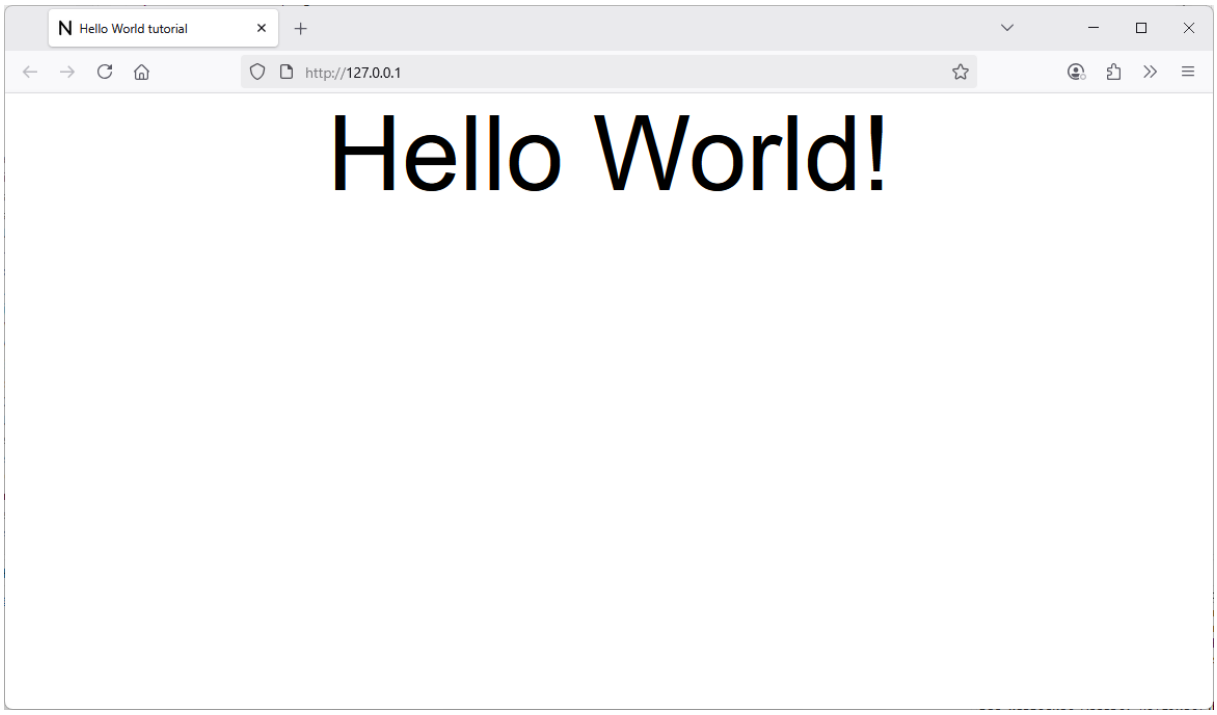
Code

```
import ch.nolix.system.webgui.atomiccontrol.Label;
import ch.nolix.system.webgui.main.Layer;


public class CustomWebClientSession extends WebClientSession<Object> {

  @Override
  protected void initialize() {

    //create a Layer
    Layer layer = new Layer();

    //create a Label
    Label label = new Label().setText("Hello World!");

    //configure the style of the Label
    label.getStoredStyle().setTextSizeForState(ControlState.BASE, 100);

    //set the Label as root Control to the Layer
    layer.setRootControl(label);

    //push the Layer on top of the Gui
    getStoredGui().pushLayer(layer);
  }
}
```

The setRootControl method of a Layer sets the given Control as root Control to the Layer. The Label type is a subtype of the Control type.

The Layer class is in the ch.nolix.system.webgui.main package. The Label class is in the ch.nolix.system.webgui.atomiccontrol package.

Output



3.4 Atomic Controls

Atomic Controls are Controls that do not contain other Controls.

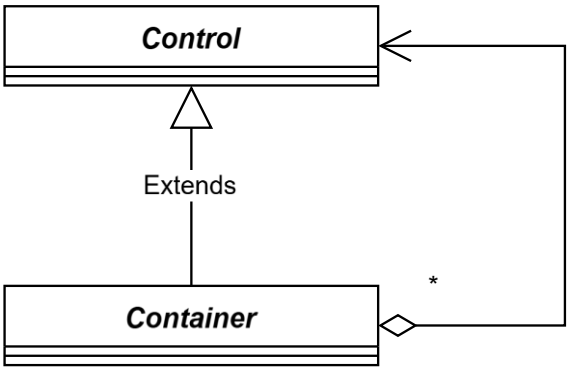
Atomic Controls:

- Button
- ImageControl
- Label
- Link
- Textbox
- Uploader
- ValidationLabel

The atomic Controls are in the ch.nolix.system.webgui.atomiccontrol package.

3.5 Containers

3.5.1 Container object



A Container is a Control that can contain other Controls.

The Container type is abstract. To create a Container, a Container subtype must be chosen.

3.5.2 Specific Container subtypes

Containers:

- Grid
- FloatContainer
- HorizontalStack
- VerticalStack

3.5.3 Create and fill up a HorizontalStack

Code

```
import ch.nolix.system.webgui.linearcontainer.HorizontalStack;

public class CustomWebClientSession extends WebClientSession<Object> {

  @Override
  protected void initialize() {

    //create a HorizontalStack
    HorizontalStack horizontalStack = new HorizontalStack();

    //create and add 4 Labels to the HorizontalStack
    horizontalStack.addControl(
      new Label().setText("A"),
      new Label().setText("B"),
      new Label().setText("C"),
      new Label().setText("D")
    );

    //configure the style of the HorizontalStack
    horizontalStack
    .getStoredStyle()
    .setChildControlMarginForState(ControlState.BASE, 50)
    .setTextSizeForState(ControlState.BASE, 100);

    //add the HorizontalStack to the GUI
    getStoredGui().pushLayerWithRootControl(horizontalStack);
  }
}
```

The addControl method of a VerticalStack adds the given Controls to the VerticalStack.

The VerticalStack is in the ch.nolix.system.webgui.linearcontainer package.

Output