Nolix web applications

2023-12-05

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 can configure HTML pages.
- Nolix web applications are quite **simple**. The learning curve of the Nolix web applications is dramatically short compared to other web frameworks. This is a 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 concepts of the Nolix web applications. The Nolix web applications provide many more features which cannot all 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.

2 Quick overview (hello-world-program)

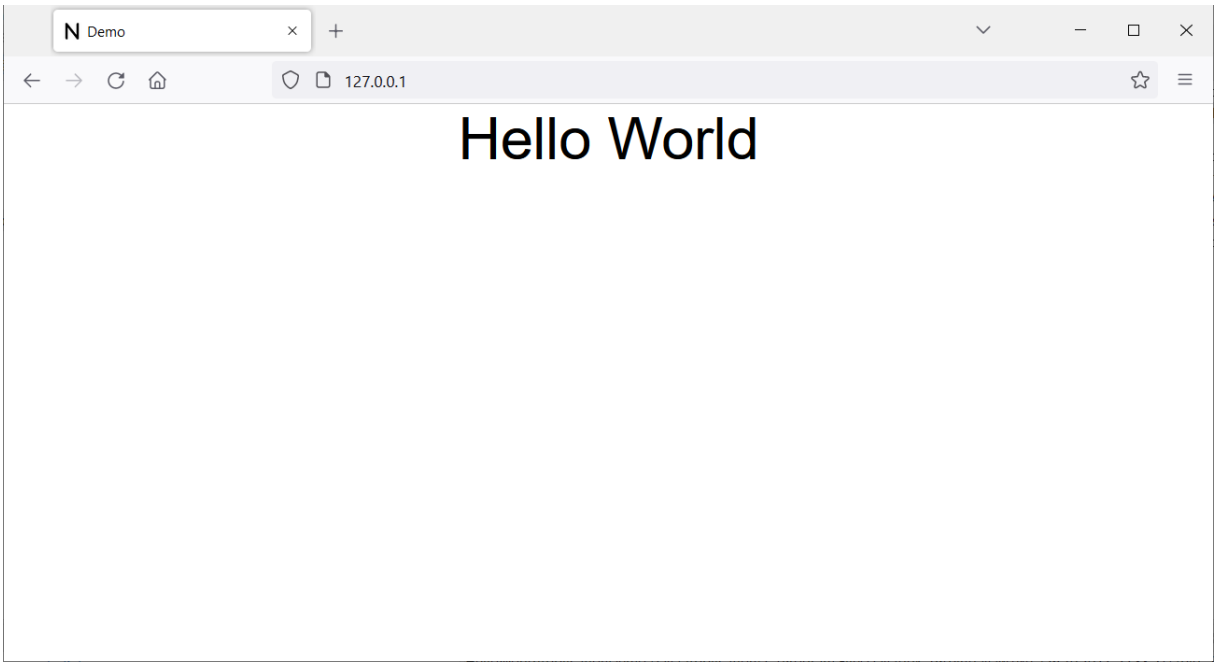
To show where the journey is going to, there is presented a complete sample program. The details of each feature will be surveyed in the next chapters.

The sample program has just to be started. The computer where the program runs on is the server. Any web browser that calls the address of the server will show up a ‘Hello World’ text.

Code

```
import ch.nolix.core.programatom.voidobject.VoidObject;
import ch.nolix.system.application.main.Application;
import ch.nolix.system.application.main.Server;
import ch.nolix.system.application.webapplication.WebClient;
import ch.nolix.system.application.webapplication.WebClientSession;
import ch.nolix.system.webgui.atomiccontrol.Label;
import ch.nolix.systemapi.WebGuiapi.mainapi;

//Create a Server.
var server = Server.forDefaultPort();

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

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

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

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

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

//Define a custom WebClientSession
public class CustomSession extends WebClientSession<Object> {

  @Override
  protected void initialize() {
    getStoredGui().pushLayerWithRootControl(
      new Label()
      .setText("Hello World")
      .editStyle(s -> s. setTextSizeForState(ControlState.BASE, 50))
    );
  }
}
```

Web browser

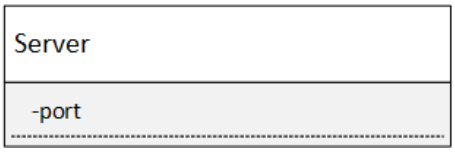


3 Servers and Applications

3.1 Server

3.1.1 Structure

For running any Web GUI, there is required to start a server program first. Nolix web applications run on a server program from Nolix. Such a server program is represented by a Server object.



A Server listens to clients on a specific port.

3.1.2 Create a Server for default port

```
import ch.nolix.system.application.main.Server;

…

var server = Server.forDefaultPort();
```

The forDefault static method creates a Server that will listen to clients on the default port. The default port is port 80. The Server will start automatically. The Server class is in the ch.nolix.system.application.main package.

3.1.3 Create a Server for specific port

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

The forPort static method creates a Server that will listen to clients on the given port. The Server will start automatically.

3.2 SecureServer

3.2.1 Purpose

A SecureServer is a Server that can handle Ssl certficates. Both, Servers and SecureServers, do the same. A Server is used for development, since a test run on a developer's local computer does not work with a Ssl certificate, because the developer's local computer does not have a web domain. A SecureServer is used in a production system where Ssl certificate have to be provided.

Server	SecureServer
Cannot handle Ssl certificates.	Can handleSsl certificates.
Is required for test runs on local computers without a domain.	Is required for production systems that have to provide Ssl certificates
Does not need any further configurations.	Needs a Nolix configuration that tells where pem files of the certificates are stored.

3.2.2 Create a SecureServer for default port

```
import ch.nolix.system.application.main.SecureServer;

…

var server =
Server.forDefaultPortAndDomainAndSSLCertificateFromNolixConfiguration(
  "nolix.tech"
);
```

The forDefaultPortAndDomainAndSSLCertificateFromNolixConfiguration static method creates a SecureServer that:

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

The default port is port 443. The Server will start automatically. The Nolix configuration is in the app data folder of the operating system. The SecureServer class is in the ch.nolix.system.application.main package.

3.2.3 Define 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 contain a DefaultSSLCertificate configuration. A DefaultSSLCertificate configuration provides the data of a SSL certificate.

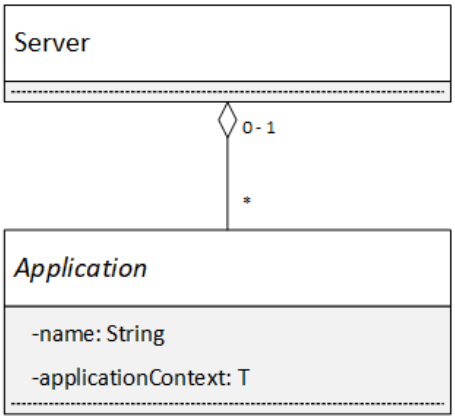
The data of a SSL certificate are:

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

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

3.3 Applications

3.3.1 Structure



Almost all web GUIs belong to an application. The Nolix web GUIs are bundled in an Application. A server can contain several Applications. An Application has a name. All Applications of a Server must have distinct names. When a web client connects to a Server, the web client must append to the URL a slash and then the name of the target Application on the Server.

3.3.2 Application context

An Application has an application context. The application context is an object of a specific type. An Application has exactly one application context and keeps for the whole lifetime.

3.3.3 Default Application

A Server can have one of its Applications as default Application. When a web client that connects to the Server does not specify a target Application, the web client will be forwared to the default Application.

3.3.4 Add Application to Server

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

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

3.3.5 Add default Application to Server

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

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

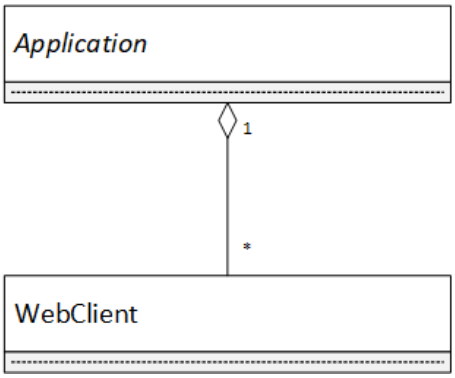
3.3.6 Define a custom Application

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

public class CustomApplication extends Application<
  WebClient,
  Object
> {

  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 context of type Object. The Application class is in the ch.nolix.system.application.main package. The WebClient class is in the ch.nolix.system.application.webapplication package.

3.4 WebClients

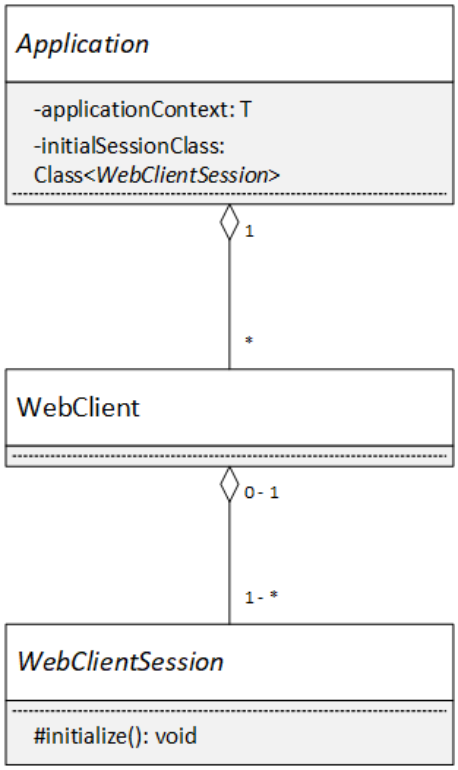
When a web client is connected to an Application on a Server, the backend of the web client is represented by a WebClient object on the Server. An Application on a Server knows all its active WebClients. A WebClient always belongs to an Application.



3.5 WebClientSessions

3.5.1 Structure

A WebClient has its own WebClientSessions. A WebClientSession is dedicated for one WebClient. A WebClientSession defines the actions the WebClient currently can do.



A WebClient has at least one WebClientSession.

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

3.5.2 Initial session

An Application has an initial session class. When a client connects to an Application on a Server, an instance of that initial session class will be created for the client. This will be the first session a client has after connecting. A WebClient can, but does not need to, be forwarded or switched to another WebClientSession.

3.5.3 Application context

A WebClientSessions can access the application context of the Application of its WebClient. With this possibility, a WebClientSession can reach all information that are important for the whole Application.

3.5.4 Define a custom WebClientSession

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

public class CustomSession extends WebClientSession<Object> {

  @Override
  protected void initialize() {
    getStoredGui().pushLayerWithRootControl(
      new Label()
      .setText("Hello World")
      .editStyle(s -> s. setTextSizeForState(ControlState.BASE, 50))
    );
  }
}
```

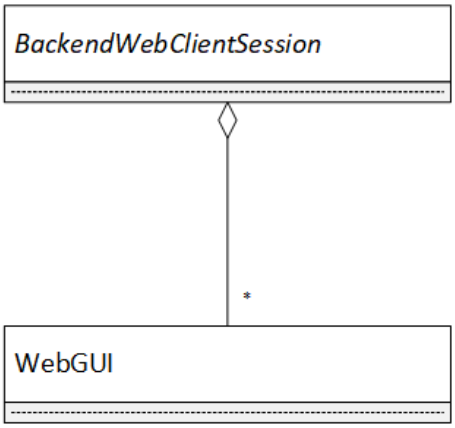
The CustomSession is a WebClientSession that is for an Application whose application context can be any Object. The WebClientSession class is in the ch.nolix.system.application.webapplication package.

4 WebGuis and Controls

4.1 WebGuis

4.1.1 Structure

A WebGui represents a GUI. There can also be said that a WebGui represents a single web page. A WebClientSession has a WebGui. The WebGui of a WebClientSession will not be exchanged and is statefull.



A WebGui has several attributes like a title, icon or GUI elements. The several abilities of a WebGui are decovered in the next sections.

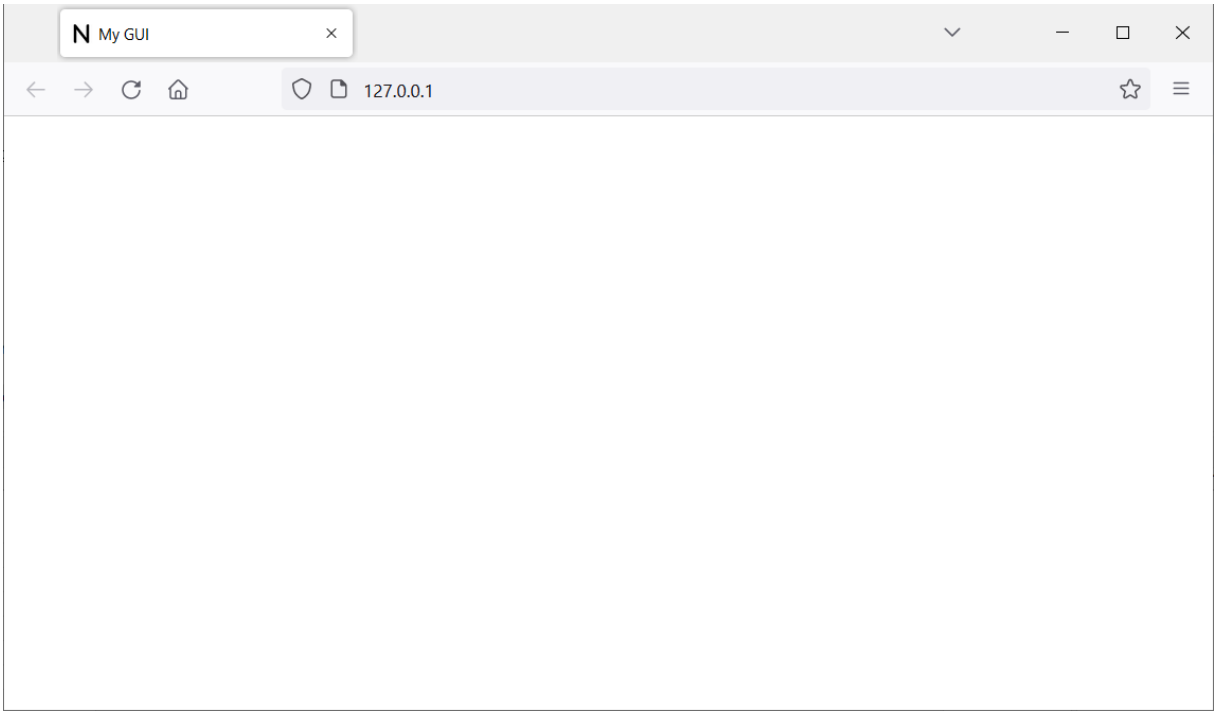
4.1.2 Set title of WebGui

Code

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

  @Override
  protected void initialize() {
    getStoredWebGui().setTitle("My GUI");
  }
}
```

The method setTitle sets the given title to the Frame. For default, the name of the parent Application is set as title to the WebGui.

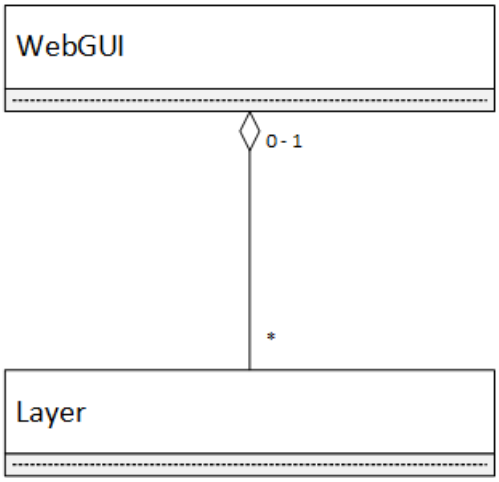
Output



4.2 Layers

4.2.1 Structure

A WebGui can contain several Layers. The Layers of a WebGui are stacked on each other. A Layer can have transparent parts where the Layer behind is visible.



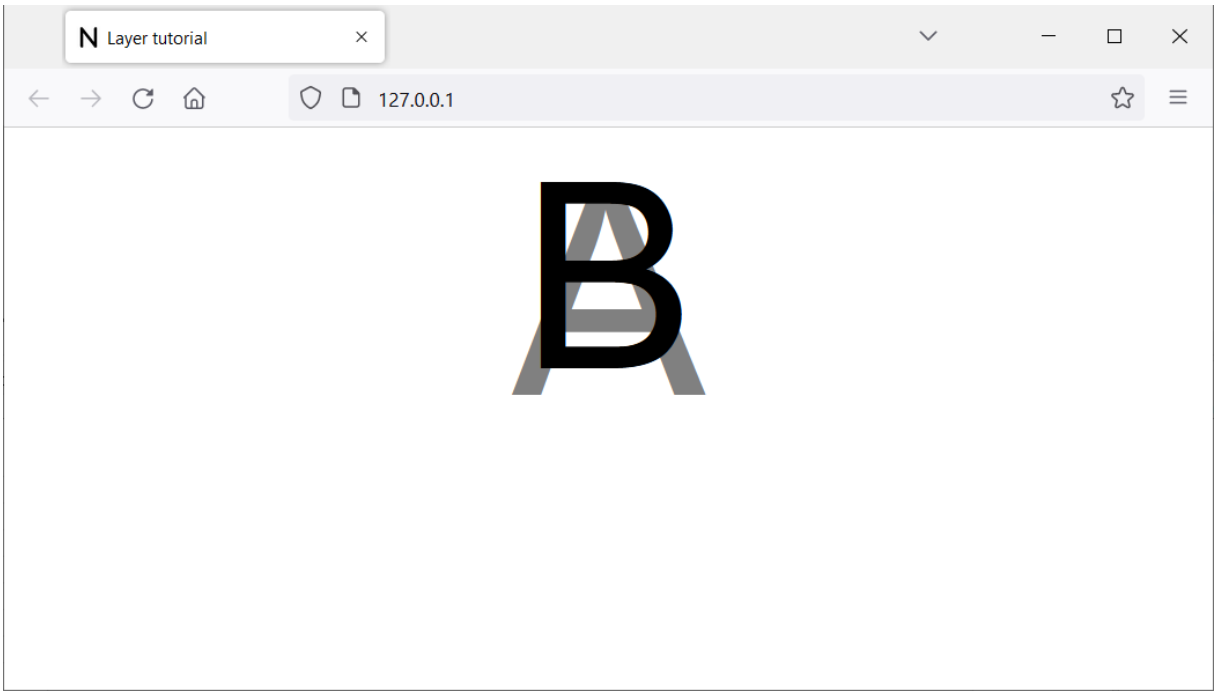
4.2.2 Push Layers to WebGui

Code

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

  @Override
  protected void initialize() {

    //Create labelA and labelB.
    var labelA = new Label().setText("A");

    //Creates labelB.
    var labelB = new Label().setText("B");

    //Configure the style of labelA.
    labelA.getStoredStyle()
    .setTextSizeForState(ControlState.BASE, 200)
    .setTextColorForState(ControlState.BASE, Color.GREY);

    //Configure the look of labelB.
    labelB.getStoredStyle()
    .setTextSizeForState(ControlState.BASE, 180)
    .setTextColorForState(ControlState.BASE, Color.BLACK);

    //Add a new Layer with labelA to the WebGui.
    getStoredGui().pushLayerWithRootControl(labelA);

    //Add a new Layer with labelB to the WebGui.
    getStoredGui().pushLayerWithRootControl(labelB);
  }
}
```

The method pushLayerWithRootControl pushes a new Layer with the given root control on the top of the WebGui. What Controls are, is described in the next chapter.

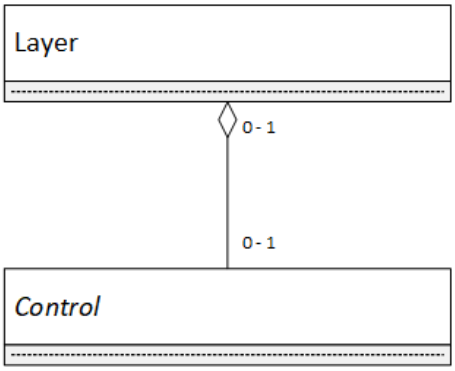
Output



4.3 Controls

4.3.1 Structure

A Control is a GUI element. A Layer contains 0 or 1 Control.



Control is an abstract type. Specific Controls are for example Buttons, Textboxes or DropdownMenues.

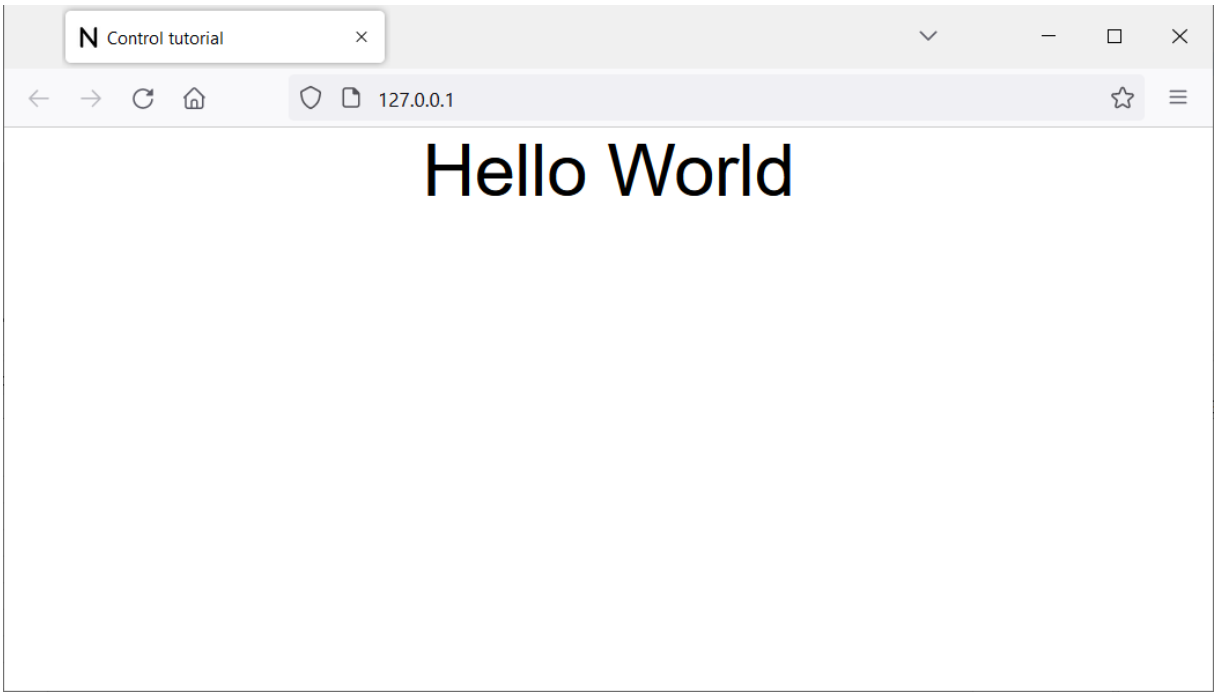
4.3.2 Add Control to Layer

Code

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


public class CustomSession extends WebClientSession<Object> {

  @Override
  protected void initialize() {

    //Create a Layer.
    var layer = new Layer();

    //Create a Label.
    var label = new Label().setText("Hello World")

    //Configure the style of the Label.
    label.editStyle(s -> setTextSizeForState(ControlState.BASE, 50));

    //Set the Label as root Control to the Layer.
    layer.setRootControl(text);

    //Push the Layer on the top of the WebGui.
    getStoredGui().pushLayer(layer);
  }
}
```

The setRootControl method sets the given Control as root Control to the Layer. The Label class is in the ch.nolix.system.webgui.atomiccontrol package.

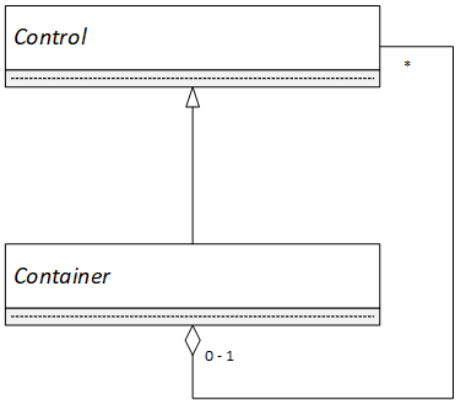
Output



4.4 Containers

4.4.1 Structure

A Layer of a WebGui can contain only one root Control. The question is how several Controls can be added to a Layer? – This can be achieved by using Containers. A Container is a Control that itself can contain several other Controls.



Container is an abstract type. Specific Containers are for example HorizontalStacks, VerticalStacks or FloatContainers.

4.4.2 Use Container

Code

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

public class CustomSession extends WebClientSession<Object> {

  @Override
  protected void initialize() {

    //Creates a VerticalStack.
    var verticalStack = new VerticalStack();

    //Create 4 Labels.
    var label1 = new Label().setText("A");
    var label2 = new Label().setText("B");
    var label3 = new Label().setText("C");
    var label4 = new Label().setText("D");

    //Add the Labels to the VerticalStack.
    verticalStack.addControl(label1, label2, label3, label4);

    //Configure the style of the VerticalStack.
    VerticalStack
    .getStoredStyle()
    .setChildControlMarginForState(ControlState.BASE, 20);

    //Configure the style of the Labels.
    label1.getStoredStyle().setTextSizeForState(ControlState.BASE, 50);
    label2.getStoredStyle().setTextSizeForState(ControlState.BASE, 50);
    label3.getStoredStyle().setTextSizeForState(ControlState.BASE, 50);
    label4.getStoredStyle().setTextSizeForState(ControlState.BASE, 50);

    //Add the VerticalStack to the WebGui.
    getStoredGui().pushLayerWithRootControl(verticalStack);
  }
}
```

The addControl method adds the given Controls as child Controls to the VerticalStack. The VerticalStack is in the ch.nolix.system.webgui.linearcontainer package.

Output