

Vitaliy Koshelenko



50 Shades of Liferay

Part 1 - Liferay Introduction

About Book

This book is recommended for Java-developers, which want to start development with Liferay, and also for web-developers and portal administrators.

Required skills:

- Java (Core, EE)
- Database - MySQL
- Application server - Apache Tomcat
- Build tools - Ant/Maven
- Frontend (HTML/CSS/JS)
- IDE - IntelliJ IDEA
- OS - Windows7/Ubuntu

Although those skills are not required, they're desirable. The mentioned above tools will be used in book as examples.

Roadmap

First chapter of book explains how to download, install and configure Liferay. It also explains how to configure development environment in IntelliJ IDEA IDE.

Main concepts of Liferay portal are covered in the second chapter. This is theoretical chapter, explaining how Liferay works without deeping inside development process. But this information is required to know before you start development with Liferay.

Chapters 3-6 are practical ones, they have instructions how to create your first portlet, theme, layout and hook.

Chapters 7-9 are dedicated to themes development.

Content

About Book	2
Greetings	5
Introduction	7
Chapter 1. Installation and Configuration	9
Download Liferay	9
First Startup	12
Liferay Configuration	14
IDE Configuration	16
Chapter 2. Liferay Basics	21
What is Liferay?	21
Liferay's Page Structure	22
Liferay Pages	23
Liferay Sites	38
Liferay Applications Management	46
Liferay Administration	48
Chapter 3. First Portlet	53
Creating new module in IDEA	54
Structure of the created module	57
Chapter 4. First Theme	58
Creating new module in IDEA	58
Chapter 5. First Layout	61
Chapter 6. First Hook	64
Which hooks do exist?	64
Creating module for hook	65
Formulation of the problem	66
Defining JSP-page, for which hook should be installed	66
Creating hook on JSP	68
Chapter 7. More about Themes	71
Themes structure	71
Creating own theme	76
Changing navigation	79
Embedding the portlets into the theme	81

Using VELOCITY-macros	84
Using Liferay-services in theme	89
Using the custom attributes	91
Chapter 8. Color Schemes	93
What is the color schemes?	93
Color schemes for CLASSIC theme	93
Creating our own COLOR SCHEMES	95

Greetings



Hi!

My name is Vitaliy Koshelenko, I'm a Liferay developer at AimProSoft company.

I have been working with Liferay for almost 5 years already, including portlets development, hooks, themes, layouts development, Liferay customization, migration from previous versions and integration with external services.

When I started working with Liferay, it seemed quite difficult and incomprehensible for me. Even small tasks took too much time, especially those ones, which required modifications in Liferay's internal code. Most of time was spend for searching required information - due to lack of Liferay-specific knowledge and experience in Liferay development.

Although there is official Liferay documentation

(<https://dev.liferay.com>) as well as specifications of portlet technology (<https://jcp.org/en/jsr/detail?id=286>), it's quite hard to find what you need. In most cases you just become drown inside endless Liferay forum threads, Wiki pages, tutorials instead of finding the solution you need.

There is no either good practical guide for beginners with step-by-step instructions for creation first portlet, theme, etc., or theoretical tutorial explaining in a few words how Liferay actually works.

So, I decided to write this book to help new developers quickly understand main concepts of Liferay structure, see how it works with simple examples and begin development with Liferay. I'll try to make it easy and understandable, but at the same time practical and useful.

Hope, my efforts will not be wasted, and this book will help you in understanding Liferay.

It is my **contact** info:

Skype: vitaliy.koshelenko

E-mail: v.koshelenko@aimprosoft.com

Profile at Liferay community: <https://www.liferay.com/web/vet.kosh/profile>

Profile at Liferay JIRA: [https://issues.liferay.com/secure/ViewProfile.jspx?
name=v.koshelenko](https://issues.liferay.com/secure/ViewProfile.jspx?name=v.koshelenko)

You may contact me, if you have some questions, suggestions, recommendations.

Introduction

Nothing better in the world,
Than implementing Liferay code...

Why “50 shades”?

Because when you start Liferay development, it seems perversion and masochism for you, but once you get used to it, things become better.

Why Liferay?

Before you start development of new site, you ask yourself - which tool to use for this? You can either create new site from scratch, or use some ready solutions (CMS).

If you create site from scratch, you spend time for implementing things, which have already been actually implemented: authentication, roles/permissions mechanism, content management, etc. They are already integrated into existing CMS, tested and are being successfully used by many companies. In this case, you “re-invent a bicycle” yourself.

If you’re using CMS, you have less freedom and you’re forced to use some conventions that are being used there, but at the same you have a lot of built-in functionality, which you may use in your project.

Personally I prefer 2nd way, especially for large systems development. But making choice which CMS to use should be also responsible, and project requirements should be taken into account.

After analyzing some existing CMS on Java (Liferay Portal, Magnolia, OpenCMS, Apache Lenya), we decided, that Liferay is the best option for enterprise portal development.

It provides developers the following features:

- **authentication** (with ability to configure LDAP/Facebook sign-in, etc.);

- **content aggregation** (web-page is generated from separate independent parts - so-called 'portlets', which are assembled together into a single page);
- **roles and permissions management**;
- **personalization** - due to content aggregation and roles/permissions management one and the same page can be shown differently for different users;
- **dynamic content management** - portlets can be added to a page on-the-fly, dropped or moved without code changes and server restarts;
- **content separation** - content in Liferay belongs to some site inside a portal, which restricts access to it for site members only;
- **a lot of ready-for-use portlets** - like Web Content, Wiki, Blog, Document Library, Message Boards, Asset Publisher, etc.;
- **powerful administration tools**.

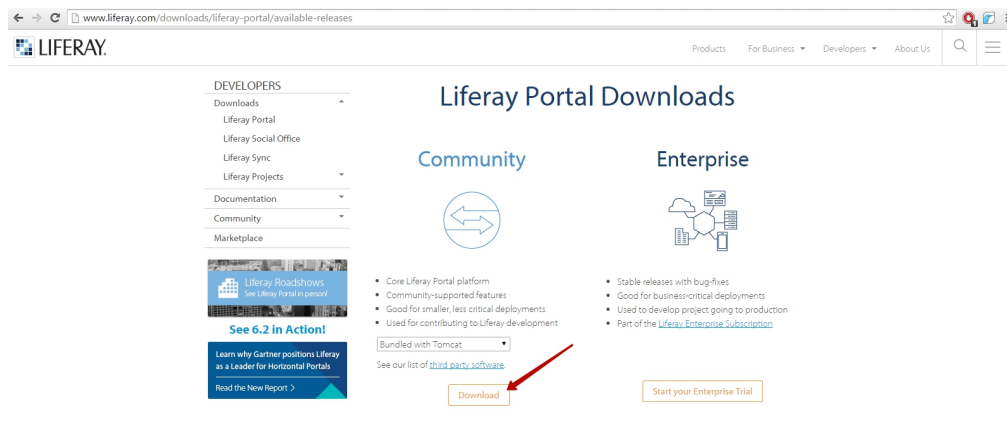
Chapter 1. Installation and Configuration

At the moment of writing this book the latest stable Liferay version is **6.2-ce-ga4**.

Installation and configuration issues are described in this section.

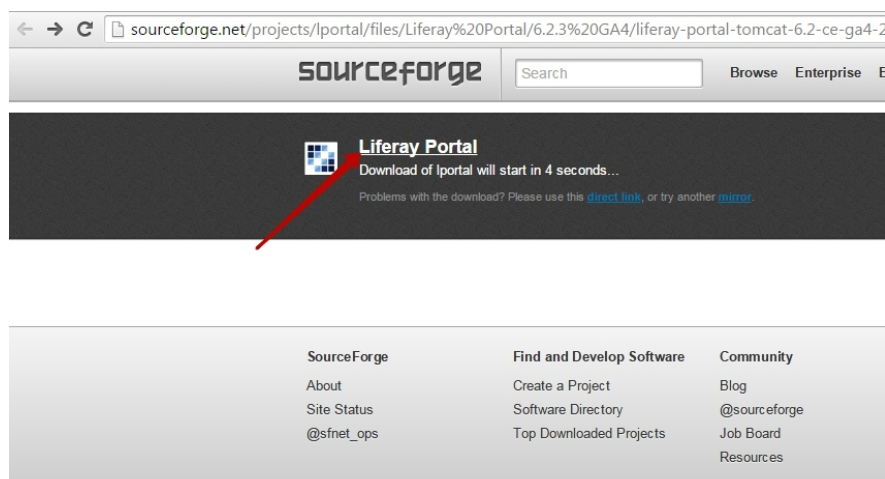
Download Liferay

To download Liferay visit Liferay's official site: <https://www.liferay.com>. Click **Download**, then choose **Bundled with Tomcat**, click **Download** again:

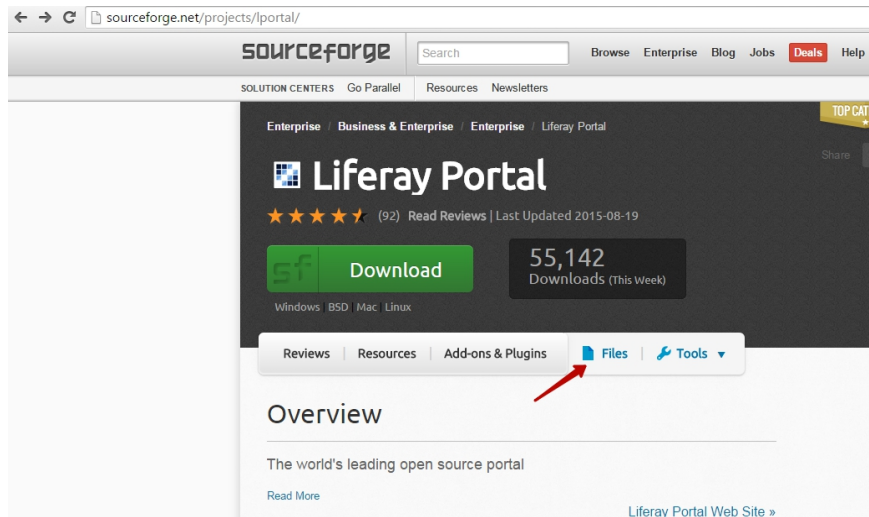


After clicking **Download** the latest stable Liferay version will be downloaded automatically (after 5 seconds).

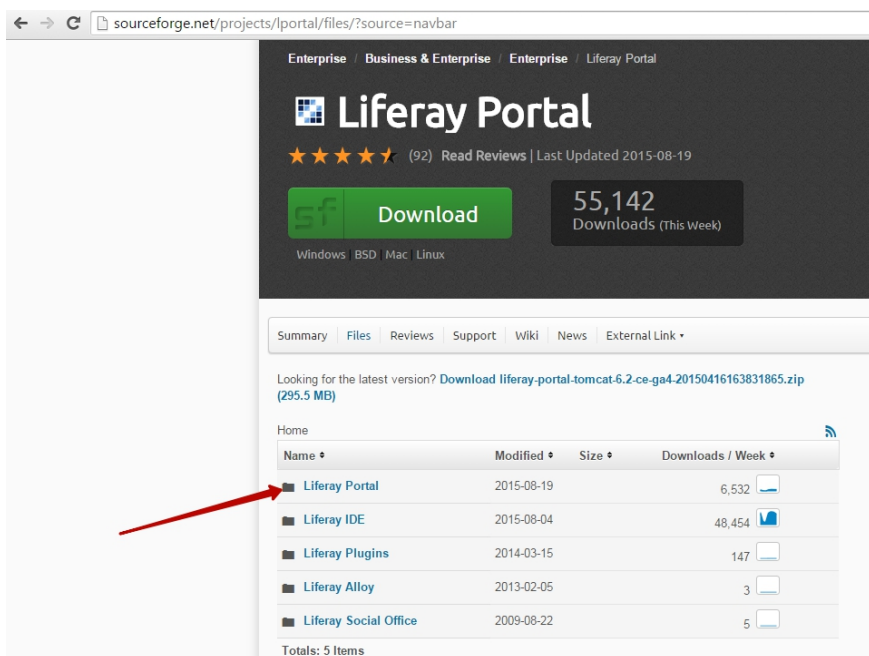
To choose the version you need, and also to download sources and documentation, you may click on **Liferay Portal** link (without waiting for automatic downloading):



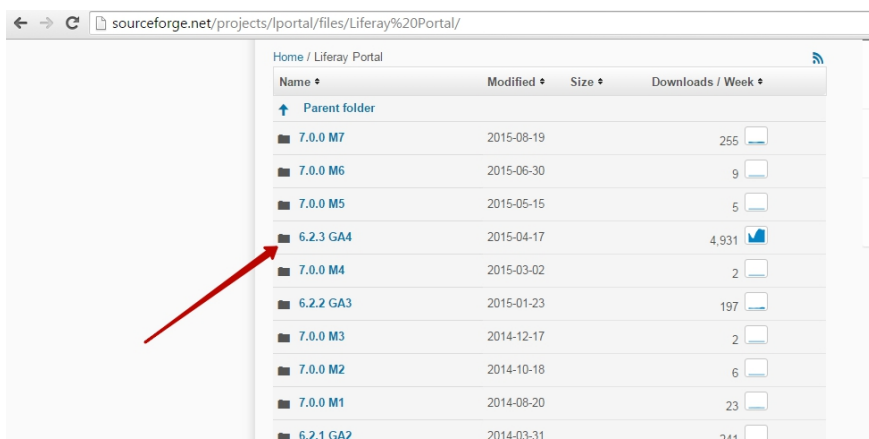
then click **Files** link:



then click Liferay Portal link:

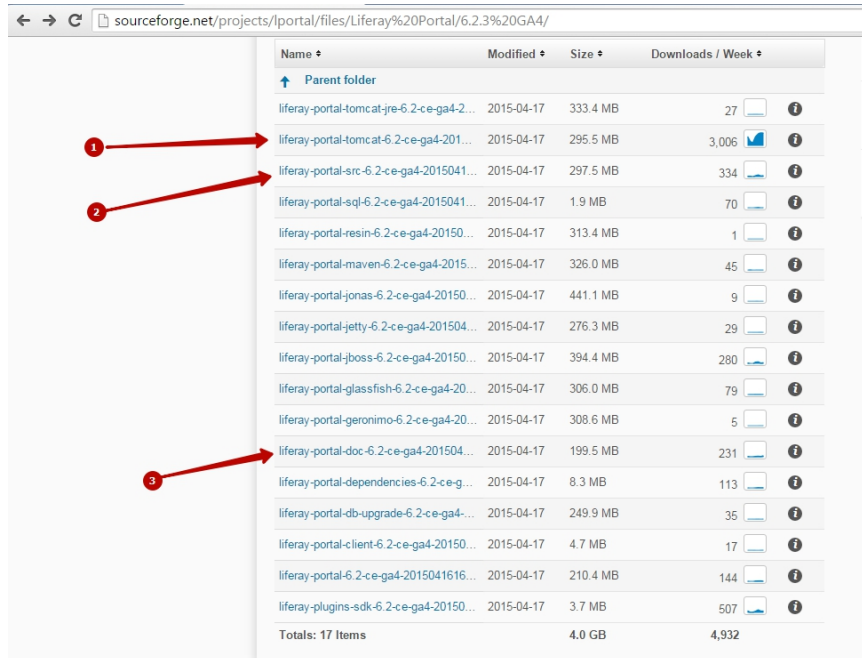


Now choose the Liferay version (the latest stable one is 6.2-ce-ga4 at the moment):



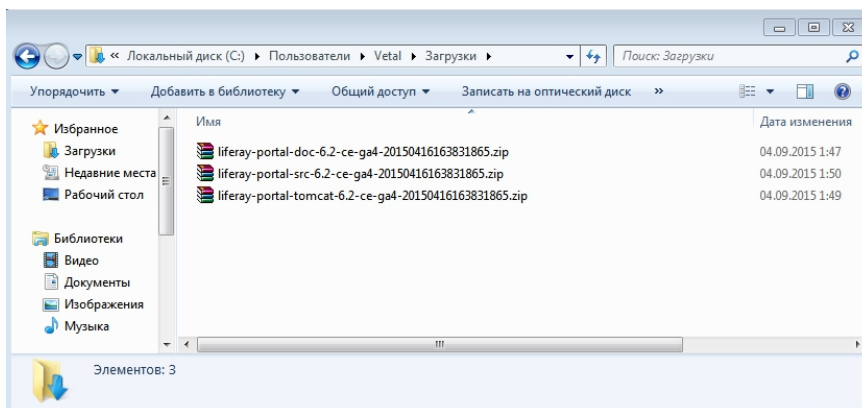
You'll be redirected to this page <http://sourceforge.net/projects/lportal/files/Liferay Portal/6.2.3 GA4/>.

Now you need to download **Liferay** (1), **Liferay sources** (2) and **Liferay documentation** (3) from here:



Name	Modified	Size	Downloads / Week
Parent folder			
liferay-portal-tomcat-jre-6.2-ce-ga4-20150416163831865.zip	2015-04-17	333.4 MB	27
liferay-portal-tomcat-6.2-ce-ga4-20150416163831865.zip	2015-04-17	295.5 MB	3,006
liferay-portal-src-6.2-ce-ga4-20150416163831865.zip	2015-04-17	297.5 MB	334
liferay-portal-sql-6.2-ce-ga4-20150416163831865.zip	2015-04-17	1.9 MB	70
liferay-portal-resin-6.2-ce-ga4-20150416163831865.zip	2015-04-17	313.4 MB	1
liferay-portal-maven-6.2-ce-ga4-20150416163831865.zip	2015-04-17	326.0 MB	45
liferay-portal-jonas-6.2-ce-ga4-20150416163831865.zip	2015-04-17	441.1 MB	9
liferay-portal-jetty-6.2-ce-ga4-20150416163831865.zip	2015-04-17	276.3 MB	29
liferay-portal-boss-6.2-ce-ga4-20150416163831865.zip	2015-04-17	394.4 MB	280
liferay-portal-glassfish-6.2-ce-ga4-20150416163831865.zip	2015-04-17	306.0 MB	79
liferay-portal-geronimo-6.2-ce-ga4-20150416163831865.zip	2015-04-17	308.6 MB	5
liferay-portal-doc-6.2-ce-ga4-20150416163831865.zip	2015-04-17	199.5 MB	231
liferay-portal-dependencies-6.2-ce-ga4-20150416163831865.zip	2015-04-17	8.3 MB	113
liferay-portal-db-upgrade-6.2-ce-ga4-20150416163831865.zip	2015-04-17	249.9 MB	35
liferay-portal-client-6.2-ce-ga4-20150416163831865.zip	2015-04-17	4.7 MB	17
liferay-portal-6.2-ce-ga4-20150416163831865.zip	2015-04-17	210.4 MB	144
liferay-plugins-sdk-6.2-ce-ga4-20150416163831865.zip	2015-04-17	3.7 MB	507
Totals: 17 Items		4.0 GB	4,932

You'll get the following files in **Downloads**:



Copy those files to your Liferay's directory (D:/Work/Liferay/[Project-name] on Windows, /home/{user}/Work/Liferay/[Project-name] on Linux), and unpack Liferay (liferay-portal-tomcat-6.2-ce-ga4-20150416163831865.zip).

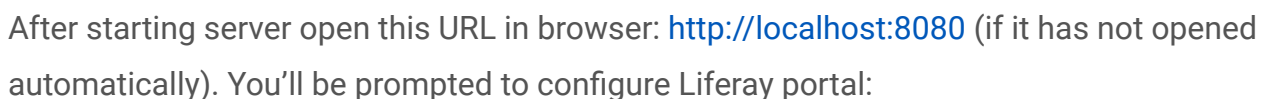
Now you have all required Liferay files and you're ready to start it.

Go to tomcat/bin folder inside unpacked Liferay (liferay-portal-6.2-ce-ga4/tomcat-7.0.42/bin).

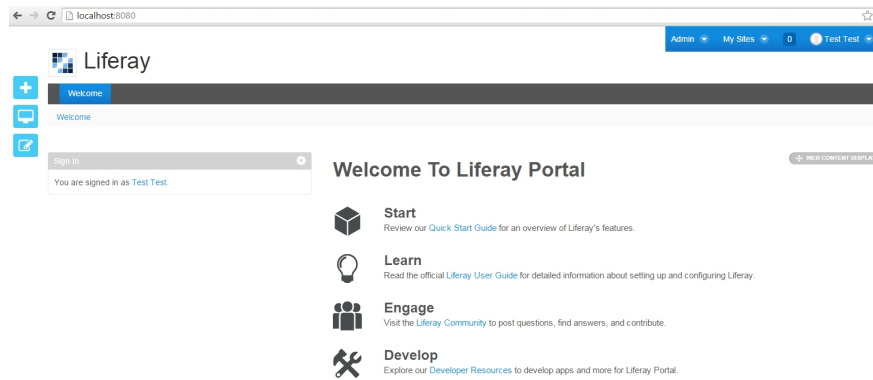
To start Liferay on Linux, run from console:

then go to **tomcat-7.0.42/logs** directory and watch logs:

After normal startup the following message should appear in logs: **INFO: Server startup in [...] ms**



Leave everything as default here, remove **Add Sample Data** flag to speed-up configuration process, press **Finish Configuration**. After this click **Go to My Portal**, agree with **Terms of Use**, create new password for user, answer reminder question, and you'll be redirected to Liferay home page:



Now you're done, Liferay is started up and configured. You may use it.

Liferay Configuration

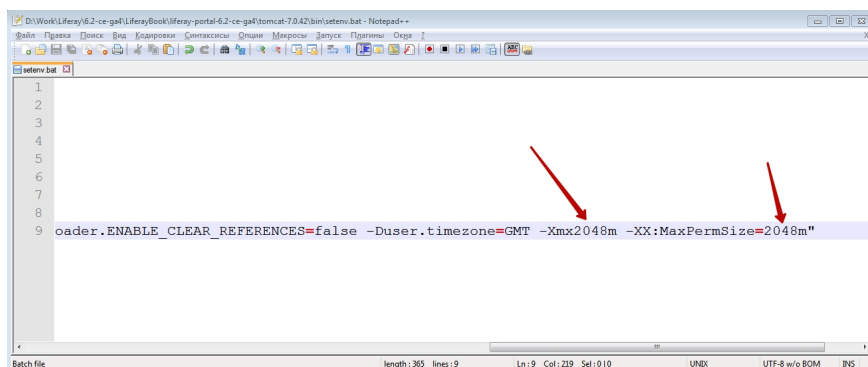
We have already started up Liferay with the default configuration settings. However, this configuration has the following disadvantages for the real projects:

- not enough allocated memory;
- working with built-in database (Hypersonic).

Now we'll fix it.

Adding allocated memory

Liferay is quite resource-consuming portal, that's why we need to add allocated memory size. To do this edit **setenv.bat** file on Windows (or **setenv.sh** on Linux) and change values for **-Xmx** and **-XX:MaxPermSize** parameters:



If you have enough RAM, it's recommended to set those values to **4096m**, otherwise - to **2048m**. Less values are not recommended.

Configuring database

Liferay works with built-in Hypersonic database by default. It's normally to use it for demo projects, but it's not recommended to use it on production. To change database settings - edit Liferay's config file **liferay-portal-6.2-ce-ga4/portal-setup-wizard.properties**, and add the following properties:

```

jdbc.default.driverClassName=com.mysql.jdbc.Driver

jdbc.default.url=jdbc:mysql://localhost:3306/[DB_NAME]

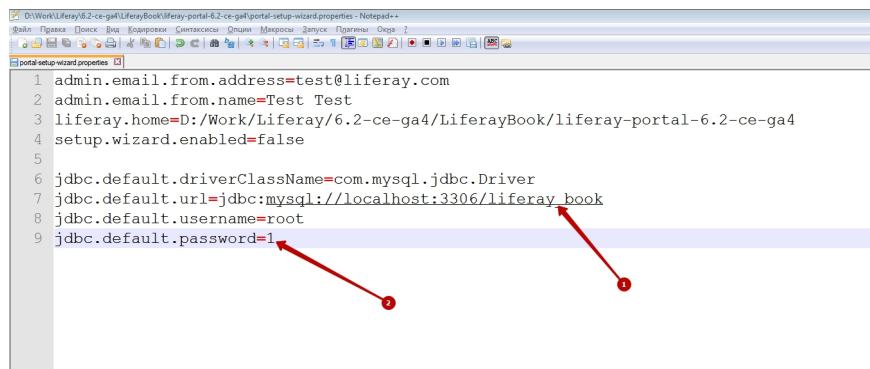
jdbc.default.username=root

jdbc.default.password=[PASSWORD]

```

where [DB_NAME] - database name, [PASSWORD] - MySQL password.

Your config file will be like this:



NOTE:

Make sure, you have specified correct values for **database name** (1) and **MySQL password** (2).

Restart Liferay to apply those changes.

IDE Configuration

We have already started Liferay and made some configuration enhancements for it. But we were starting it from console. And it's not good for the development process, as you actually need to debug your code. So, we'll need to configure our Liferay in **IDE** and run it from inside **IDE** in **Debug mode**. This section explains how to do this.

You may use different IDEs for Liferay development.

Some people prefer Liferay IDE, based on **Eclipse** (<https://www.liferay.com/downloads/liferay-projects/liferay-ide>). It simplifies Liferay development and deployment process, and has special plugins for portlets/themes development, etc. But I don't use it, as I don't like Eclipse products at all.

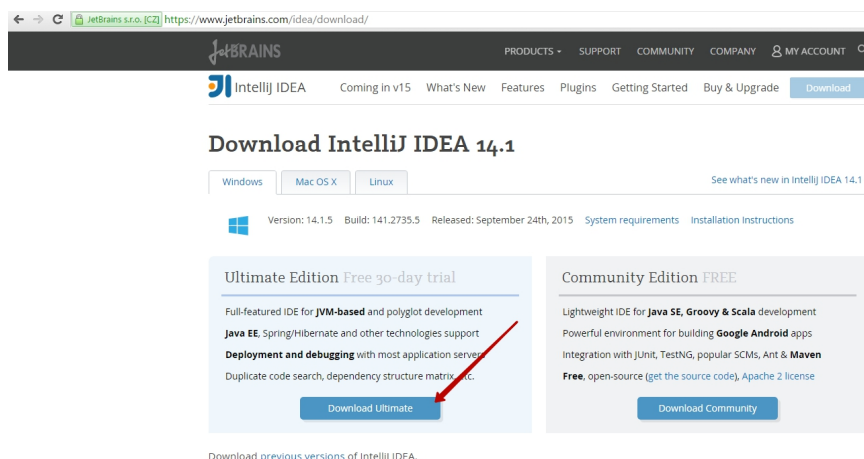
Other people use **NetBeans** or some other IDEs.

Personally I prefer using **IntelliJ IDEA** (<https://www.jetbrains.com/idea/>). Although it's not Liferay-specific, development process is pretty clean and simple inside it.

Now I'll explain how to configure Liferay in IntelliJ IDEA IDE.

Download IDEA

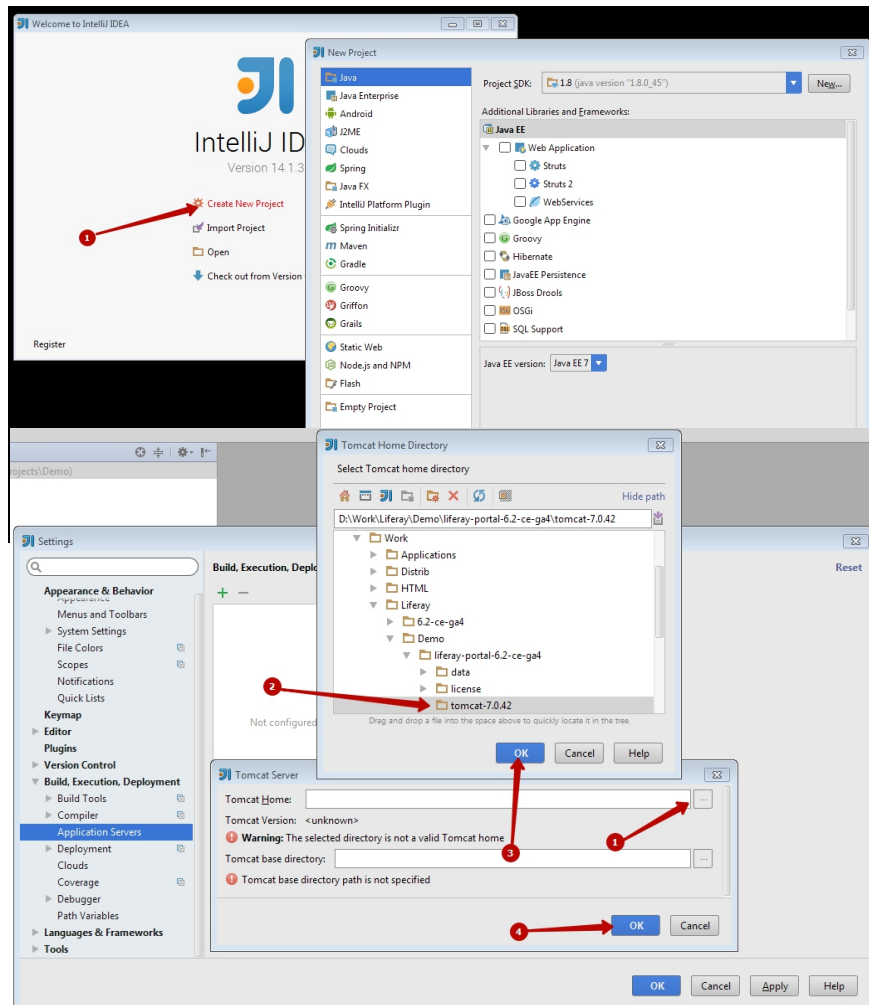
If you still don't have IDEA installed, download it from here: <https://www.jetbrains.com/idea/download/> (the latest version is **14.1** at this moment). Download **Ultimate Edition** (as **Community Edition** doesn't have enough features for development):



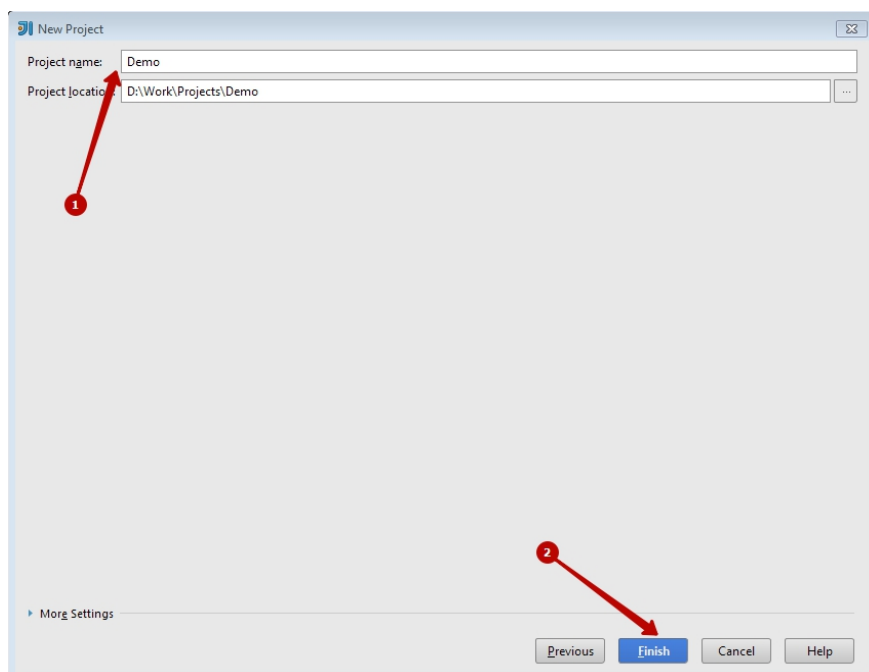
Keep in mind, that it's paid product.

Create new project

Start IntelliJ IDEA and create new project in it:

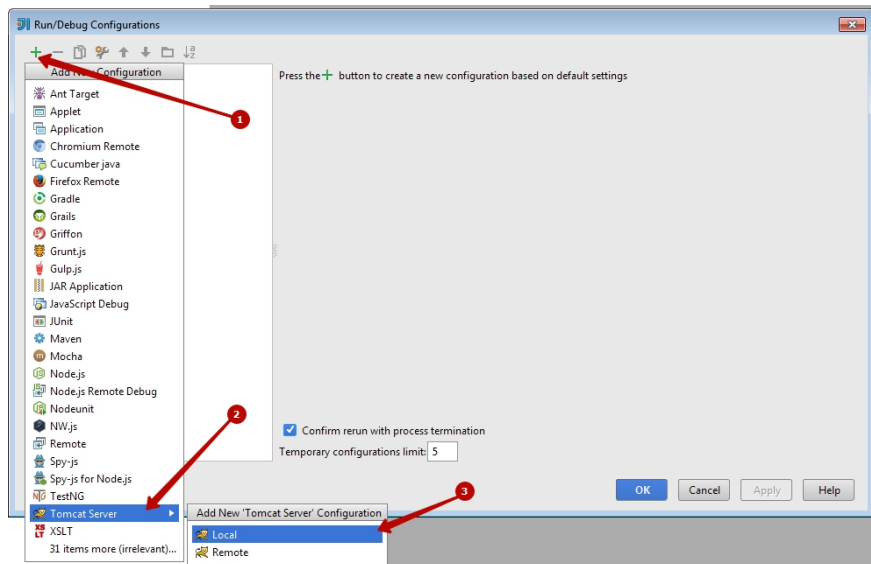


Click **Next**, specify project name, and click **Finish**:

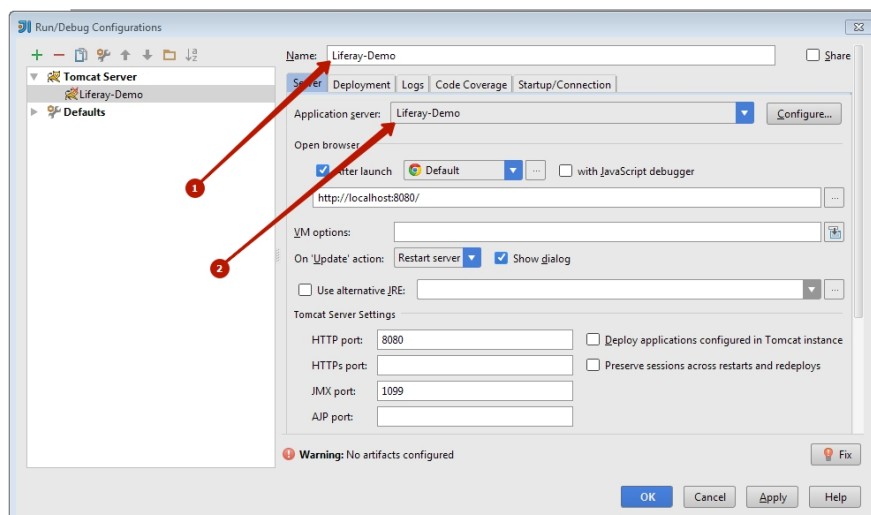


You're done, new project has been created.

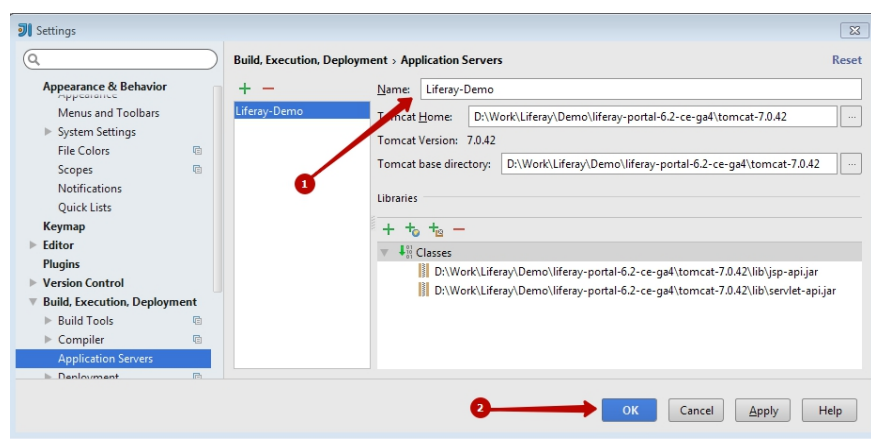
Create new Application Server in 'Settings' menu

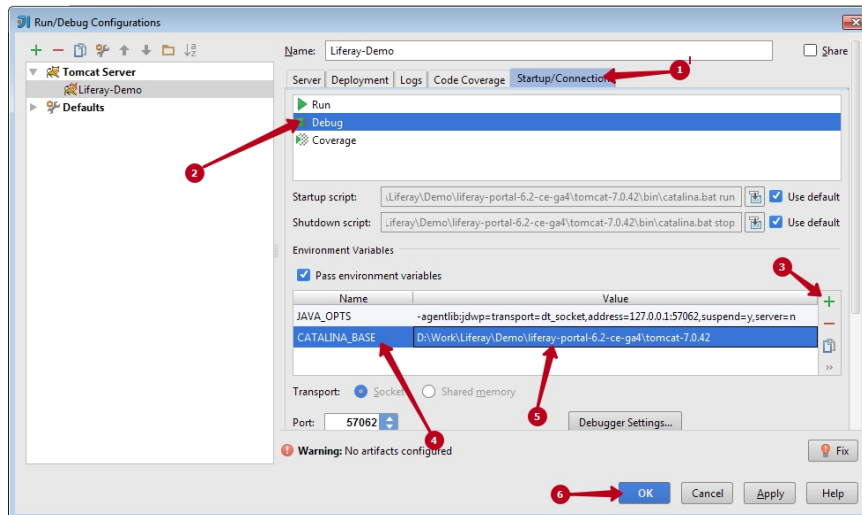


Now we'll create new application server for our Liferay in IDEA. Press **Settings** button (1) in Toolbar, and create new Tomcat server, as illustrated here:



Now choose tomcat folder inside Downloaded Liferay:





Specify name for newly created application server (based on project name), and press **Ok**:

Application server should have been created after these steps.

Configure Server in 'Run/Debug Configurations' menu

Now we'll configure our Liferay server in IDEA. Go to **Run -> Edit Configuration...** menu (or press appropriate icon in Toolbar), and add new Tomcat server:

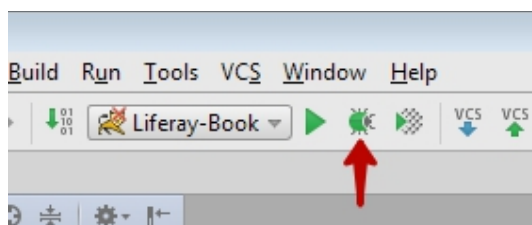
Specify server name (based on project name), and choose **Application server** (created on previous step):

On **Startup/Connection** tab (1) add (3) **CATALINA_BASE** (4) Environment variable with path to Liferay's tomcat folder (5) for **Run and Debug**(2). Then press **Ok** (6):

Basic Liferay configuration in IDEA is finished at this point, and you're ready to start it.

Startup Liferay from IDEA

Start Liferay in Debug mode from IDEA. Press **Debug** button for this (or press **Shift+F9** hotkey):



Watch logs in **Output** window in IDEA, make sure everything is Ok, and start development with Liferay.

Chapter 2. Liferay Basics

We have already reviewed Liferay installation/configuration issues and starting it from inside IntelliJ IDEA. Now we'll go through the main concepts of Liferay structure, without delving into internal implementation details.

What is Liferay?

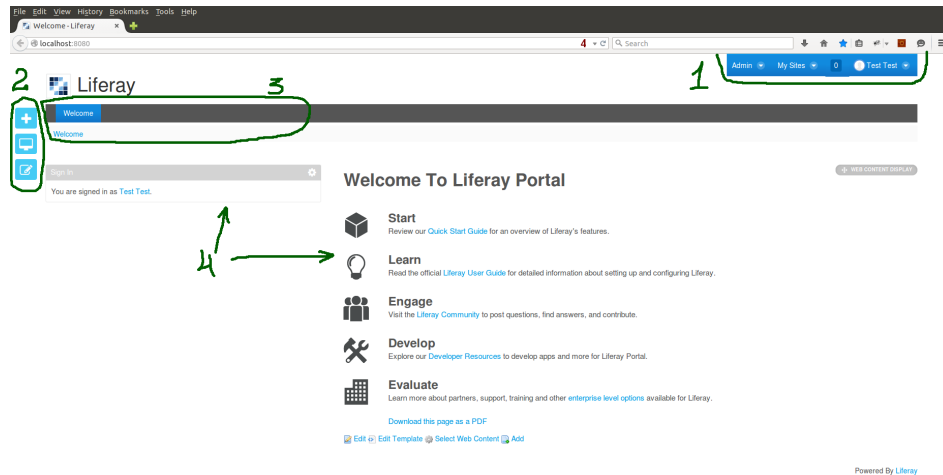
Liferay is a platform written on Java, intended for web-development, and which provides a lot of ready-for-use solutions for sites creation. It is a web-portal with capabilities for users management, roles and permissions management, and also pages/content, applications management.

Liferay has a lot of built-in applications (portlets) - such as, **Blogs, Message Boards, Wiki, Calendar**, etc. In most case Liferay development is development of separate applications (**portlets, themes, layouts**) and their composition into a single site (portal). In some cases (when you need to modify/extend Liferay's default behavior) **Liferay-hooks** are developed.

All these questions are described in more details later in this book.

Liferay's Page Structure

After starting Liferay server and going to it's home page, you see, that it has the following page structure:



As you see from here, Liferay page consists from the following parts:

- 1) **DockBar** - contains menu for Liferay administration, site navigation menu, notifications menu, profile management menu.
- 2) **MenuBar** - menu for content management. Provides capabilities for adding/editing pages, and also applying different themes/layouts for pages, adding portlets to pages, page permissions management, etc.
- 3) **NavigationBar** - navigation menu. It contains pages and sub-pages (only 1st level by default) for the current site. Only those pages are displayed in this menu, which are not marked as 'hidden', and on which current user has sufficient permissions.
- 4) **Portlets** - separate functional modules in Liferay, which may be added to a page, and which represent a part of portal's page view. Each portlet may have different permissions settings, so users with different roles may see different portlets on the same portal pages.

Liferay Pages

We have just reviewed Liferay's page structure. Now we'll discuss Liferay pages in more details.

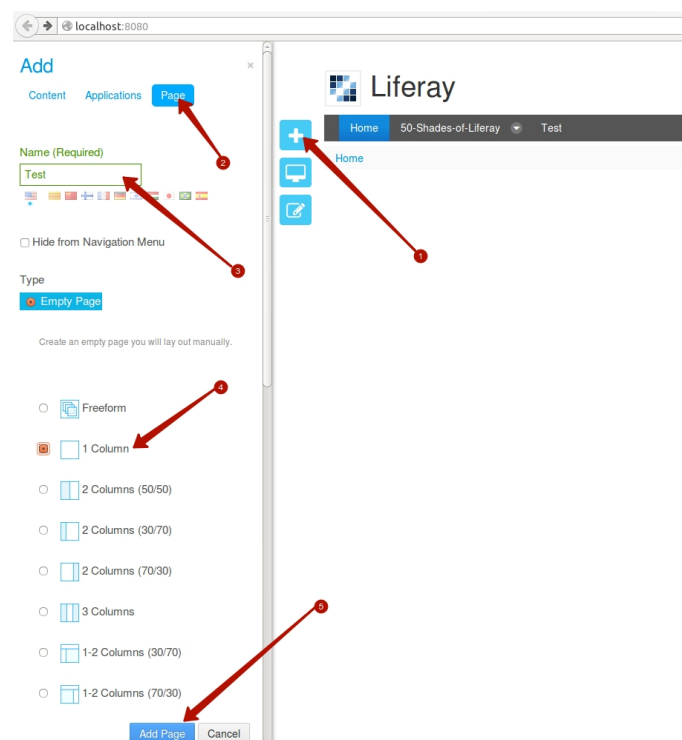
What is Liferay Page?

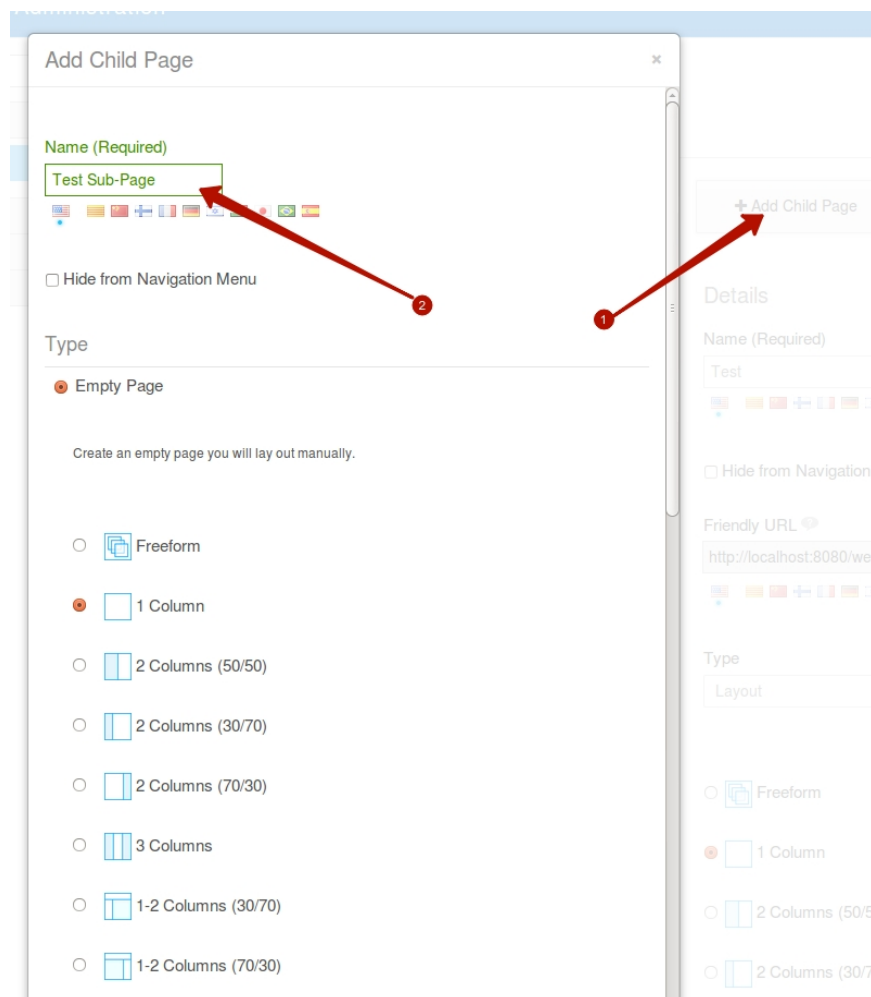
Liferay page (also called '**layout**') is a separate portal page, which has it's own URL, set of portlets on it, theme, layout, permissions settings, etc. Each page in Liferay belongs to some site, has it's owner, permissions set, and displays information rendered by portlets on this page.

Page Management

To create a new Liferay page, you need to either use **MenuBar** for this, or to do it from inside **Site Pages** menu ('**Admin**' -> '**Site Administration**' -> '**Pages**' from **Dockbar**).

Here is example of adding page from **MenuBar**:





Click on **+** icon (1), **Page** tab (2), specify page name (3), choose layout for page (4), and press **Add Page** (5) button. New page will be created.

For advanced page management go to Admin -> Site Administration -> Pages from Dockbar. You'll see the following:

Here you can perform page management. You can do the following:

- change page order using drag-and-drop
- add new pages
- add child pages for current page
- delete page (to delete some page you need to be currently on some other page)
- modify permissions on page
- specify page name and URL for a page
- make page hidden to hide it from navigation menu

Site Pages

Public Pages Private Pages

Public Pages

+ 50-Shades-of-Liferay

Home

Test

+ Add Child Page Permissions Delete Copy Applications

Details

Name (Required)

Test

Hide from Navigation Menu

Friendly URL

http://localhost:8080/web/guest/test

Type

Layout

Freeform

1 Column

Details

SEO

Look and Feel

JavaScript

Custom Fields

Advanced

Mobile Device Rules

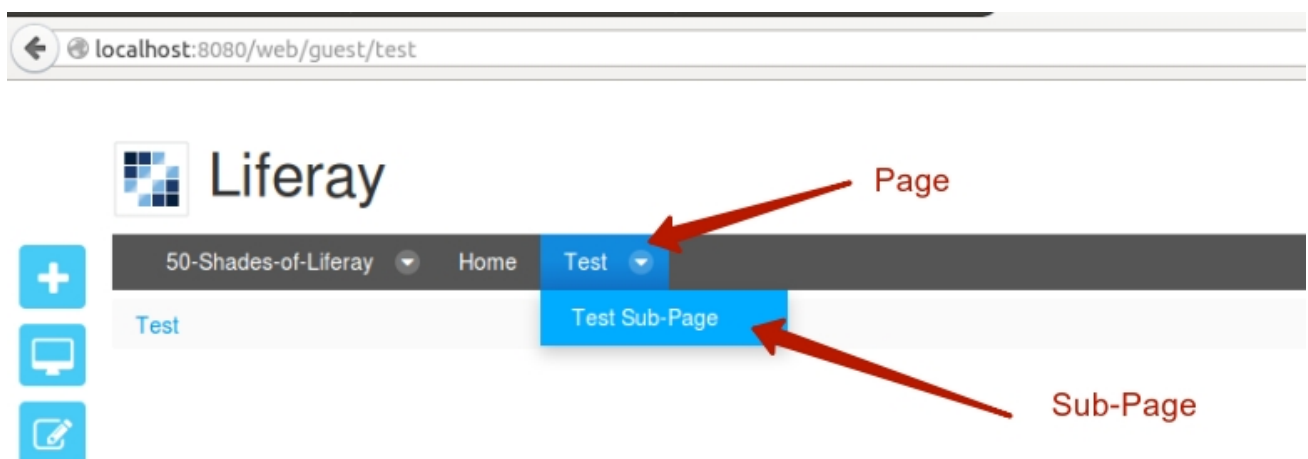
Customization Settings

Save Cancel

- change page type and layout
- specify custom settings for a page (custom fields, SEO settings, etc.)

For example, to add child page for some already existing page - select this page in the pages list at the left, click **Add Child Page** button (1), specify page name (2), and click **Add Page**:

If you go back to site home page, you'll see that subpage is displayed in the navigation menu:



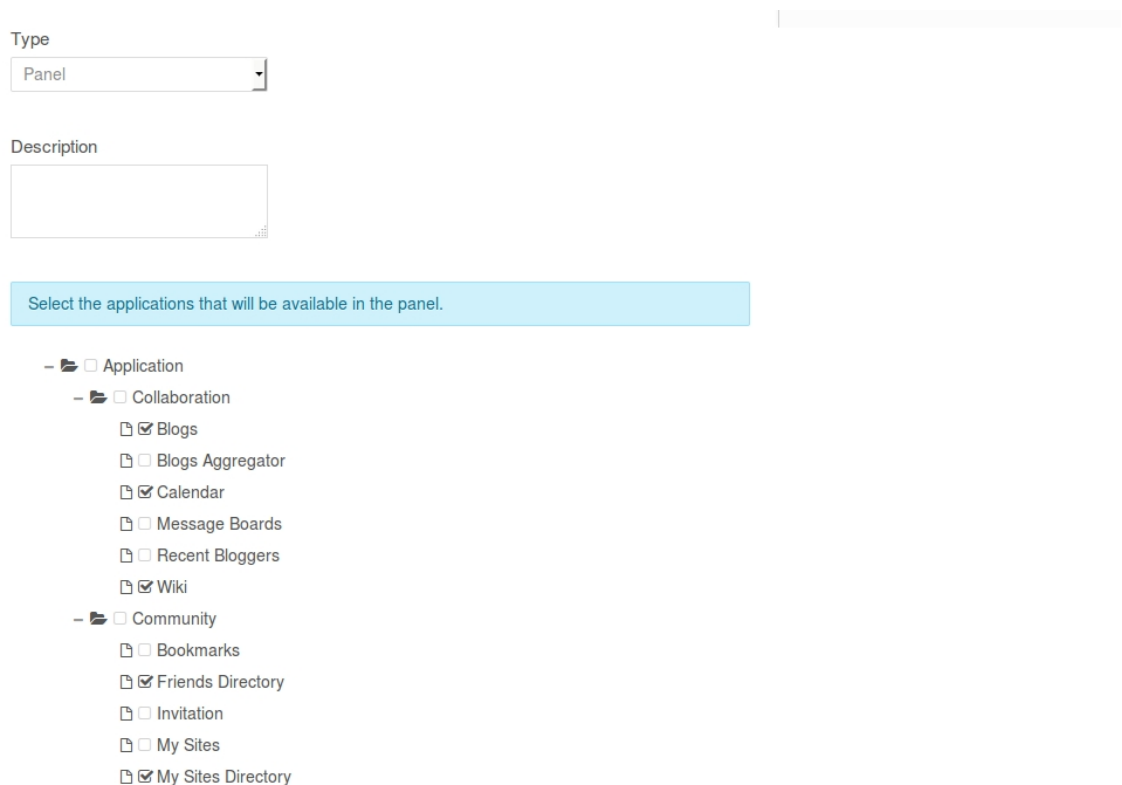
If you edit this subpage, and mark **Hide** from **Navigation Menu** flag, you'll see, that page will no longer be displayed in the navigation.

Page Types

Liferay has different types of pages. The default one is **Layout** - it's standard, empty by default page, which is displayed in navigation menu. Portlets may be added to such page, themes and layouts may be also applied to this type of page. In most cases this page type is used.

But there are different page types, here are they:

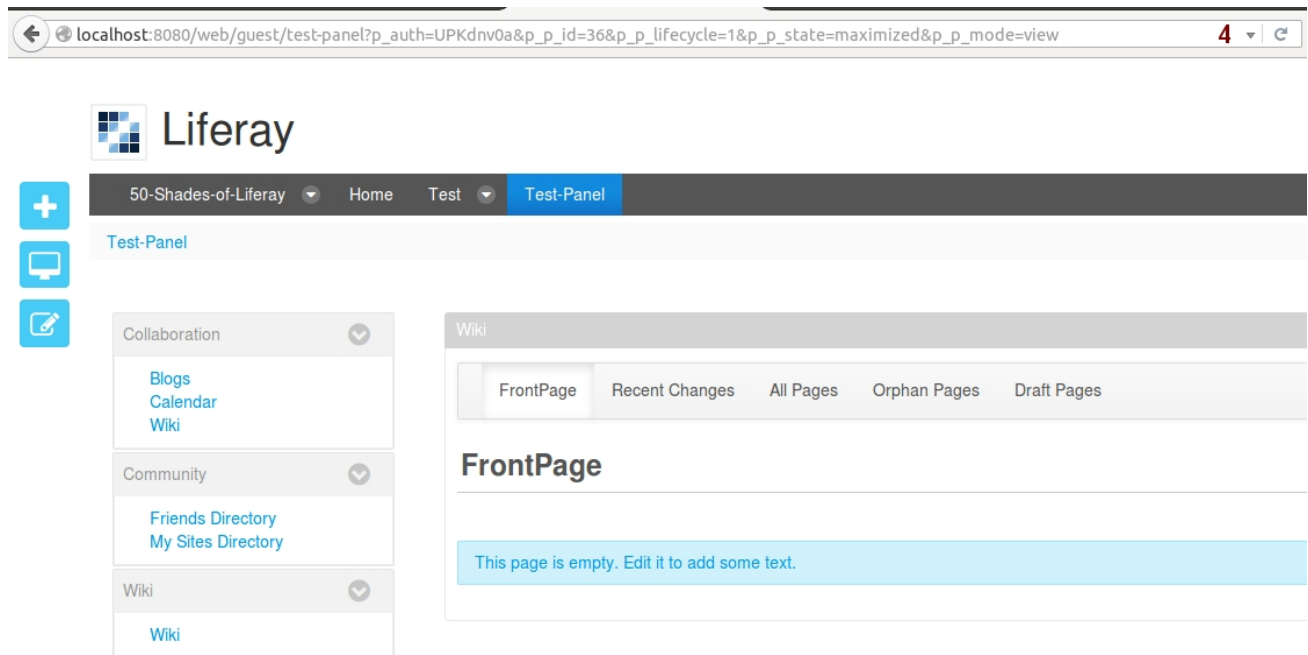
- **Link to a Page of This Site** - as the name says, this is a link to some page within site. Page of this type doesn't have it's own content, it's used to redirect to some other page within the same site by clicking on this page in navigation menu;
- **Link to URL** - this is similar to previous type, but may refer to a page of some other site within portal, or even to some external URL;
- **Panel** - this type of page is used to work with different portlets on one and the same page. When you set page type to **Panel** in **Site Pages** menu, you can specify which applications (portlets) will be available on this panel page:



The screenshot shows the configuration interface for a Liferay page. At the top, there is a 'Type' dropdown menu set to 'Panel'. Below it is a 'Description' text area. A light blue instruction bar states: 'Select the applications that will be available in the panel.' Below this, a tree view of applications is shown with checkboxes:

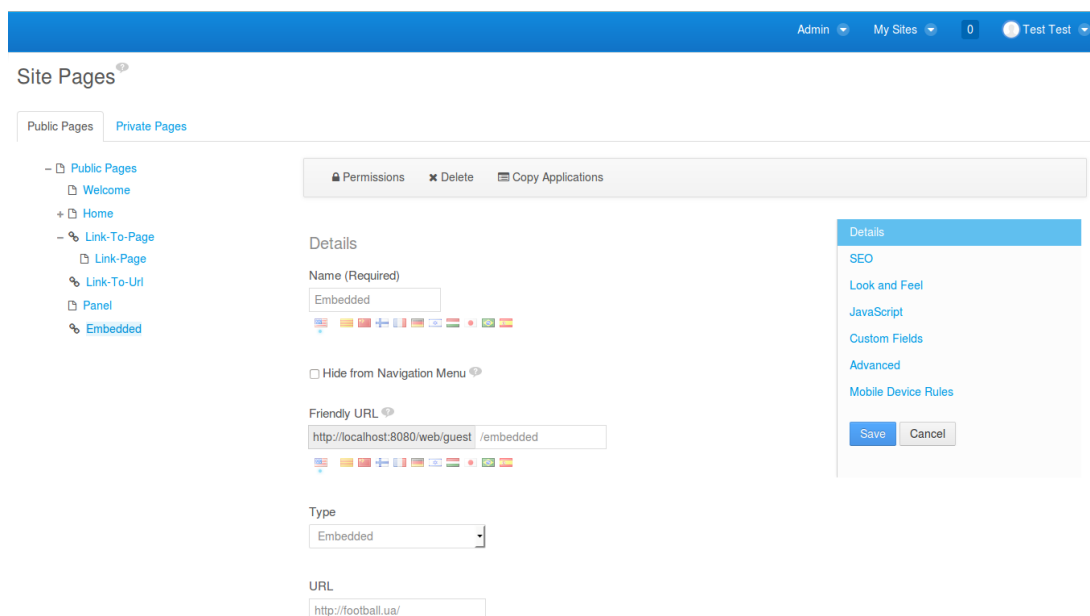
- ☐ Application
 - ☐ Collaboration
 - ☒ Blogs
 - ☐ Blogs Aggregator
 - ☒ Calendar
 - ☐ Message Boards
 - ☐ Recent Bloggers
 - ☒ Wiki
 - ☐ Community
 - ☐ Bookmarks
 - ☒ Friends Directory
 - ☐ Invitation
 - ☐ My Sites
 - ☒ My Sites Directory

When you go to panel page, you'll see, that it contains two parts: list of available applications in the left section, and selected application in the right one:

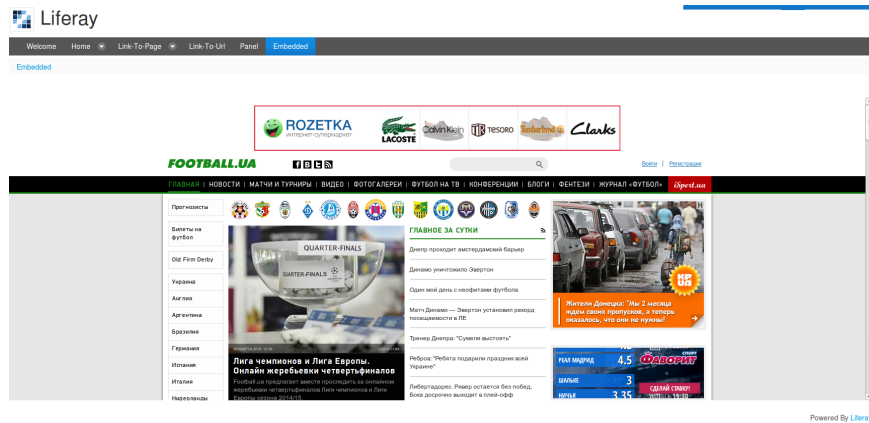


This type of page may be used, if you need to work with different portlets. In this case you don't need to create pages and add portlets you need to those pages - all the portlet you need are already on panel page. Panel page type doesn't allow to select page layout and to add applications to a page.

- **Embedded** - it's a Liferay page, which contains **IFrame**, which displays content from specified URL. When you select *Embedded* page type, you may specify URL to display:



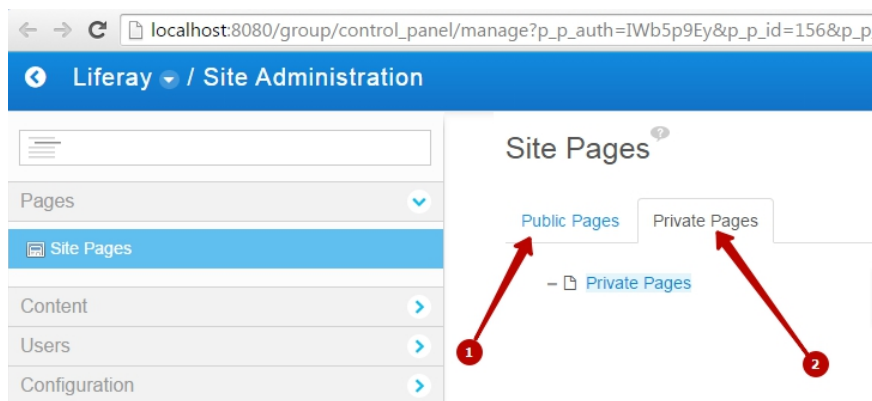
When you go to embedded page, you'll see that this URL is displayed withing **IFrame** on this page:



Adding portlets to embedded page and changing it's layout is also not allowed (as with panel pages).

Public and Private pages

In **Site Pages** menu we see, that there are **Public Pages** and **Private Pages**:



Here is the difference between them:

Public Pages	Private Pages
Accessible by all users by default (regardless of their membership in site)	Accessible only by site members

Have the following URL structure:

`http://{host}:{port}/web/{site-name}/
{page-friendly-url}`

Have the following URL structure:

`http://{host}:{port}/group/{site-name}/
{page-friendly-url}`

Page management for private pages is the same as for public pages (which was covered earlier). If you create private page, you'll see, that it's **Friendly URL** contains **/group/** in it's URL instead of **/web/** (as it was for public pages):

The screenshot shows the Liferay page management interface. The 'Private Pages' tab is active. A page named 'Test' is selected. The 'Friendly URL' field is highlighted with a red box and an arrow, showing the URL 'http://localhost:8080/group/guest/test'. The 'Name (Required)' field contains 'Test'. There are also options for 'Add Child Page', 'Permissions', 'Delete', and 'Copy Applications'.

NOTE:

Those private/public friendly URLs settings are configured in **portal.properties**:

`layout.friendly.url.private.group.servlet.mapping=/group`

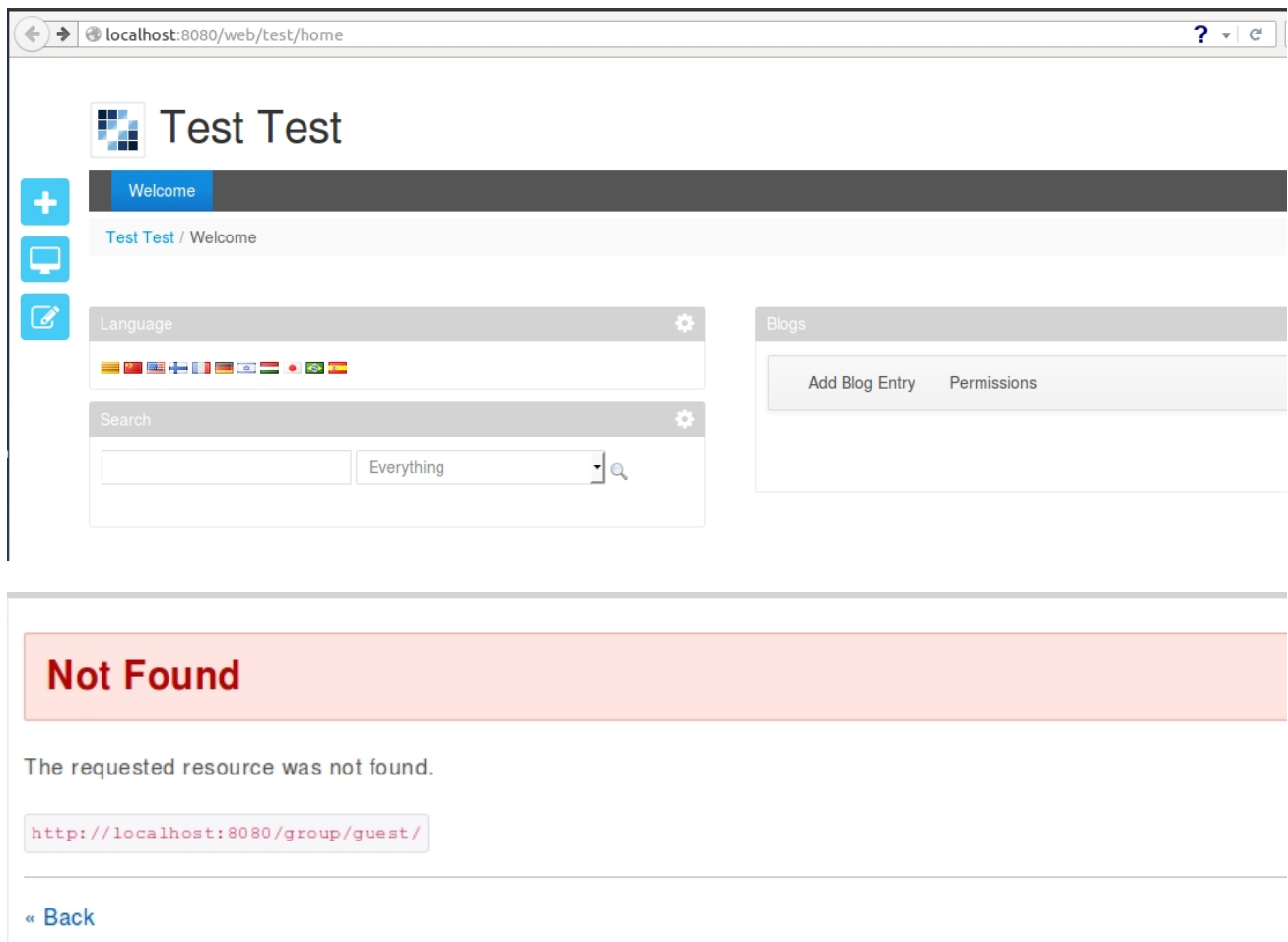
`layout.friendly.url.public.servlet.mapping=/web`

and may be changed in portal-setup-wizard.properties file.

If you try to access some private page being not logged in (for example, **/group/guest/**), you'll be redirected to the default public page in the same site (**/web/guest/** for the default **Liferay** site) with full-screened (in maximized mode) **Sign In** portlet on it (1). Once you sign in, you'll be redirected back to the private page which you originally requested (it's URL is stored in **redirect** parameter (2) in the URL):

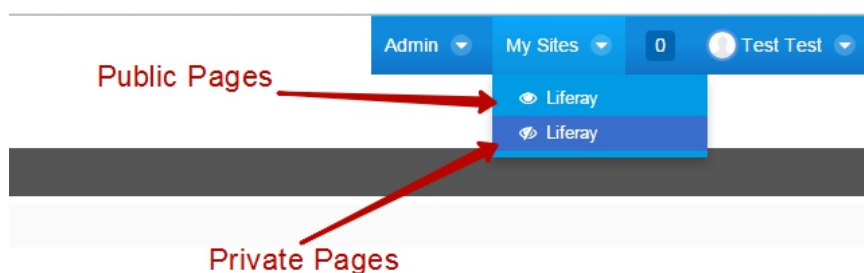
The screenshot shows the Liferay Sign In portlet. The URL bar shows a redirect parameter: 's8_redirect=%2Fgroup%2Fguest%2F'. The Sign In portlet is highlighted with a red box and an arrow. The portlet contains fields for 'Email Address' (test@liferay.com) and 'Password' (masked with dots). There is a 'Remember Me' checkbox and a 'Sign In' button. Below the portlet are links for 'OpenID', 'Create Account', and 'Forgot Password'.

but only in case when you're a member of the site and have sufficient permissions to view this page. Otherwise, if you can not see this page - **'Not Found'** exception will be shown to you:



(the same exception is shown if page really doesn't exist).

You can quickly navigate to public/private pages of your sites from the Dockbar menu:



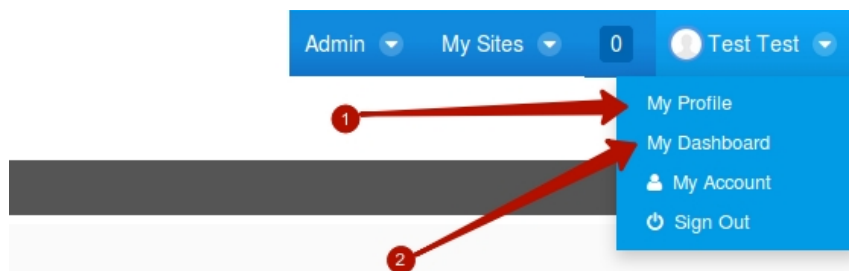
User's pages

Each user in Liferay can have his own personal site - a set of public pages (called '**My Profile**') and set of private pages (called '**My Dashboard**').

My Profile	My Dashboard
Accessible by all users	Accessible only by owner (and portal administrator)
Have the following URL structure:	Have the following URL structure:
<code>http://{host}:{port}/web/{screen-name}/{page-friendly-url}</code>	<code>http://{host}:{port}/user/{screen-name}/{page-friendly-url}</code>

It's similar to site public/private pages, but here user's screen name is used in URL instead of site group friendly URL, and `/user/` prefix is used for private pages (instead of `/group/`).

My Profile (1) and **My Dashboard** (2) are accessible from **Dockbar**:

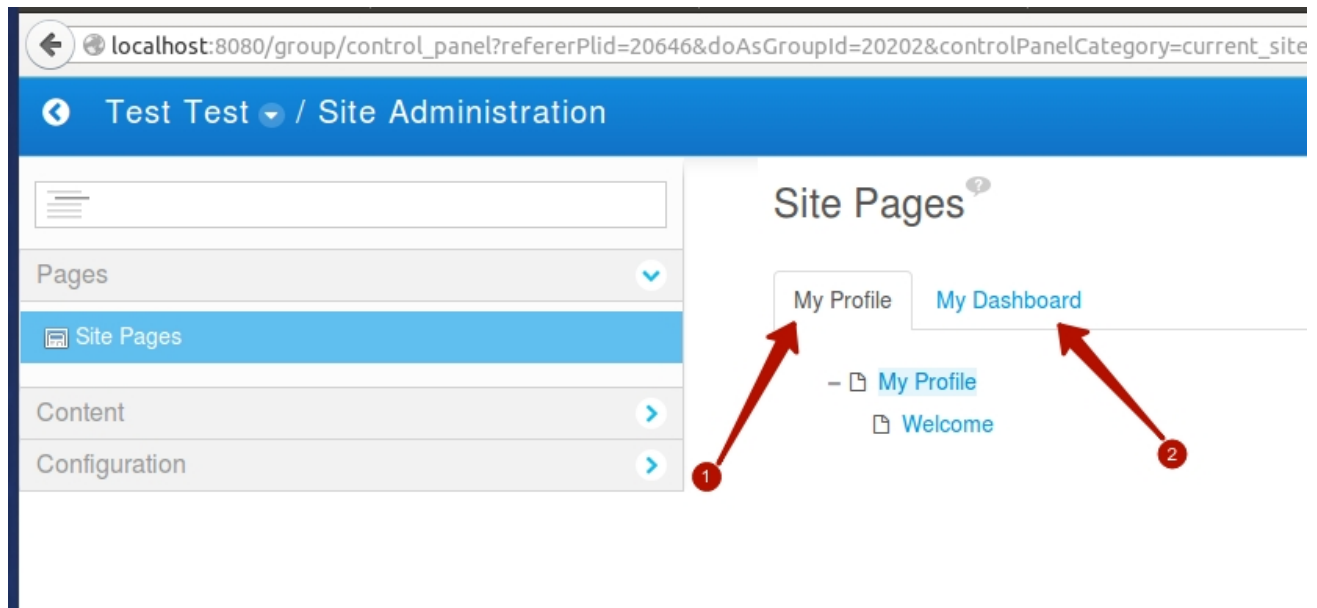


My Profile pages:

My Dashboard pages:

A screenshot of the Liferay user interface. The browser address bar shows 'localhost:8080/user/test/home'. The page header includes the 'Test Test' logo and a 'Welcome' message. On the left, there is a sidebar with icons for adding content, viewing the site, and editing. The main content area is divided into sections: 'Language' with a dropdown menu showing various flags, 'Dictionary' with a search bar and a 'Find' button, and 'Portal Directory' with tabs for 'Users', 'Organizations', and 'User Groups'. On the right, there is a 'My Sites' section with a table showing the user's sites. The table has columns for 'Name' and 'Members'. The first row shows 'Liferay' with 1 member.

As with site pages, page management for **My Profile** and **My Dashboard** is available from 'Admin' -> 'Site Administration' -> 'Pages' menu in **Dockbar**



Here you can manage pages for **My Profile** (1) and **My Dashboard** (2) in the same way, as you do this for site public/private pages.

When you create new user - both public (profile) and private (dashboard) pages are created for him automatically in the default Liferay configuration. To disable automatic creation of public/private pages for user, the following properties may be used in '**portal-setup-wizard.properties**':

```
layout.user.public.layouts.auto.create=false
```

```
layout.user.private.layouts.auto.create=false
```

To disable public/private pages at all, you may use:

```
layout.user.public.layouts.enabled=false
```

```
layout.user.private.layouts.enabled=false
```

You may also specify your custom structure of created public/private page for user by overwriting those properties:

```
#
# Set the name of the private layout.
#
default.user.private.layout.name=Welcome

#
# Set the layout template id of the private layout.
#
default.user.private.layout.template.id=2_columns_ii

#
# Set the portlet ids for the columns specified in the layout template.
#
default.user.private.layout.column-1=82,23,11
default.user.private.layout.column-2=29
default.user.private.layout.column-3=
default.user.private.layout.column-4=

#
# Set the friendly url of the private layout.
#
default.user.private.layout.friendly.url=/home
```

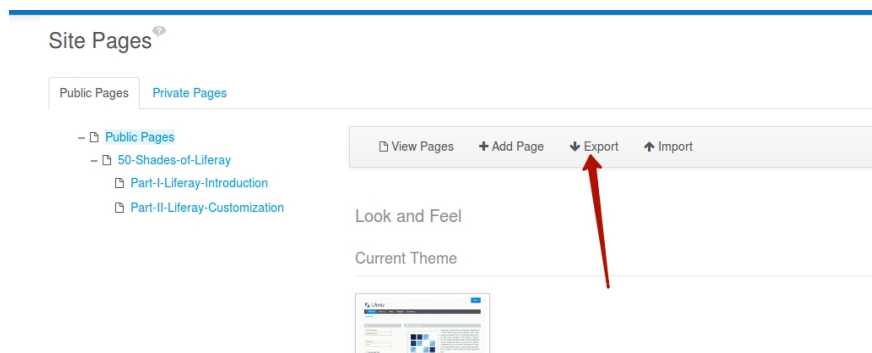
If you need to create more than one page for user - you may also specify which LAR to use:

```
default.user.private.layouts.lar=${liferay.home}/{path-to-lar-file}/{lar-file}.lar
```

Export/Import pages

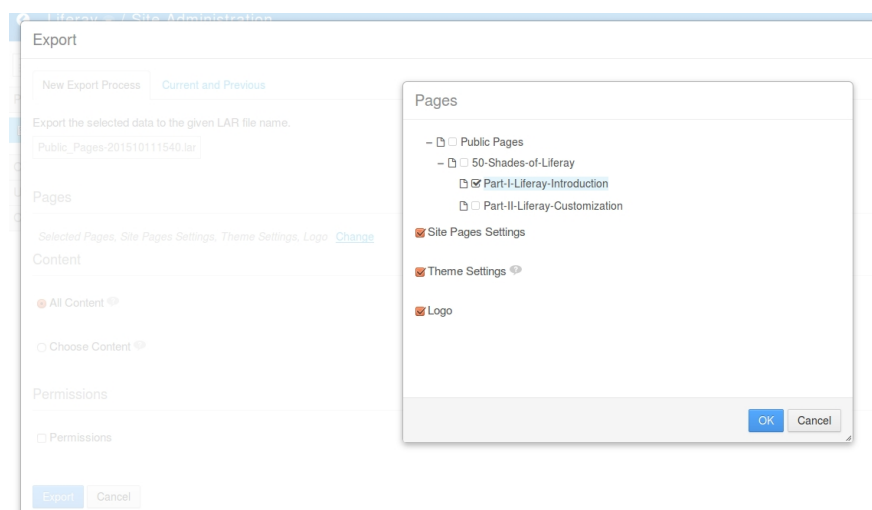
Liferay provides capability to import/export site pages with a help of **LAR files (Layout ARchives)**. If you need to copy your site pages to some other Liferay instance - you may create **.LAR** file by exporting pages from your site, and then import this file to target Liferay instance.

To create **LAR** file, you need to press **Export** button from inside **Site Pages** (if you need to export private pages - click on **Private Pages** tab before):



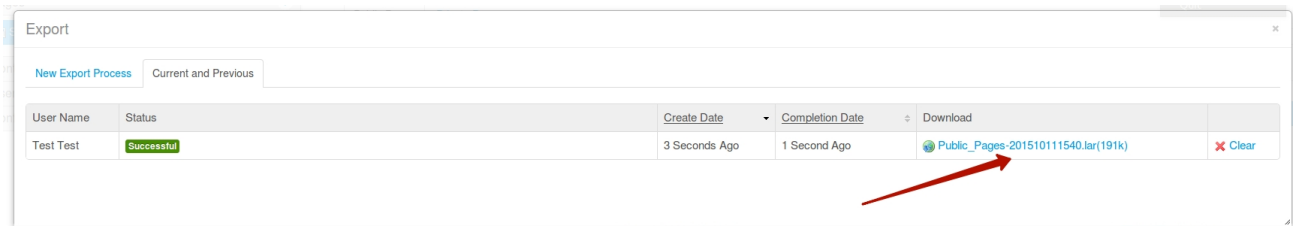
Export popup will be opened, where you can specify export settings - name of exported **LAR** file, choose which pages and which content to export.

Here I specify which pages to include into the **LAR** file being created:

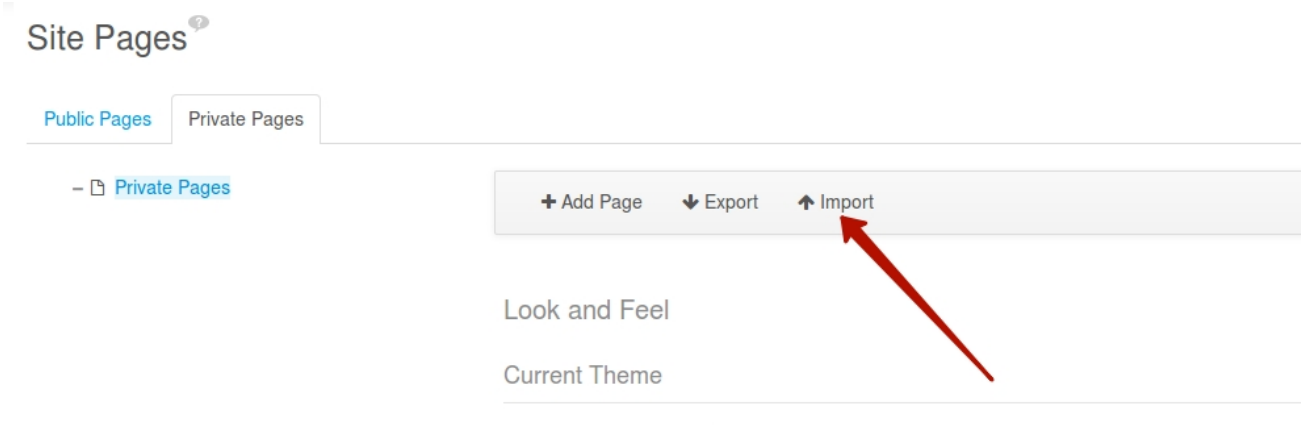


(this feature will appeared in **6.2** only)

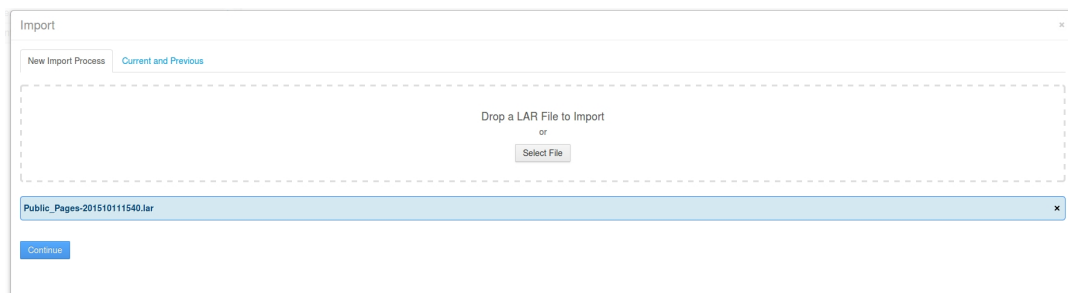
After **LAR** has been created, you may download it:



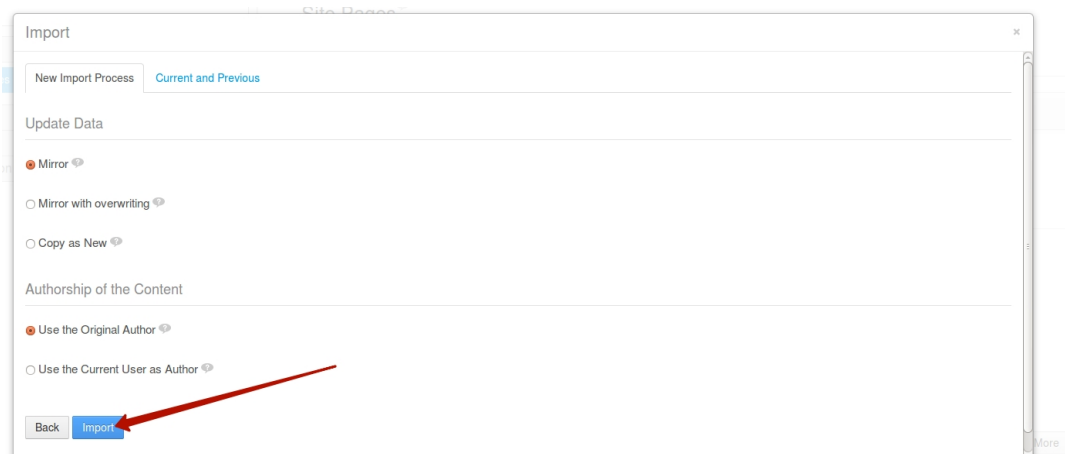
To import **LAR** file, click on **Import** button (assuming, you're already inside **Site Pages** menu for the site you need):



Upload **LAR** file in the opened **Import** popup:



Click on **Continue**, select import options, click on **Import**:



Make sure, that import process has finished successfully:



The screenshot shows the 'Import' window in Liferay. It has two tabs: 'New Import Process' and 'Current and Previous'. Below the tabs is a table with the following data:

User Name	Status	Create Date	Completion Date	
Test Test	Successful	3 Seconds Ago	1 Second Ago	Clear

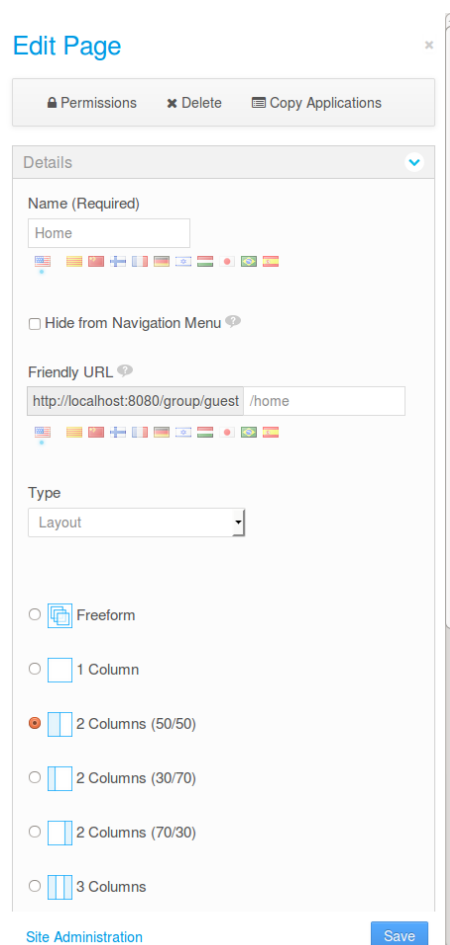
A red arrow points to the 'Successful' status of the 'Test Test' entry.

After this all pages and content from the **LAR** file should be available in the site where you uploaded this **LAR** file.

Layout Templates

Layout Templates (layouts) define the displaying of the portlets on the page. By default Liferay contains ten layouts.

To change the page layout we should click on **Edit** in the **MenuBar**, select the desired layout and click on **Save**.



The screenshot shows the 'Edit Page' window in Liferay. It has a top bar with 'Permissions', 'Delete', and 'Copy Applications' buttons. Below the top bar is a 'Details' section with the following fields:

- Name (Required)**: A text input field containing 'Home'.
- Hide from Navigation Menu**: A checkbox that is unchecked.
- Friendly URL**: A text input field containing 'http://localhost:8080/group/guest /home'.
- Type**: A dropdown menu with 'Layout' selected.

Below the 'Type' dropdown are several radio button options for layout types:

- ☐ Freeform
- ☐ 1 Column
- ☒ 2 Columns (50/50)
- ☐ 2 Columns (30/70)
- ☐ 2 Columns (70/30)
- ☐ 3 Columns

At the bottom of the window are 'Site Administration' and 'Save' buttons.

After applying the layout to the page the portlets can be moved in it according to the structure of the selected layout.

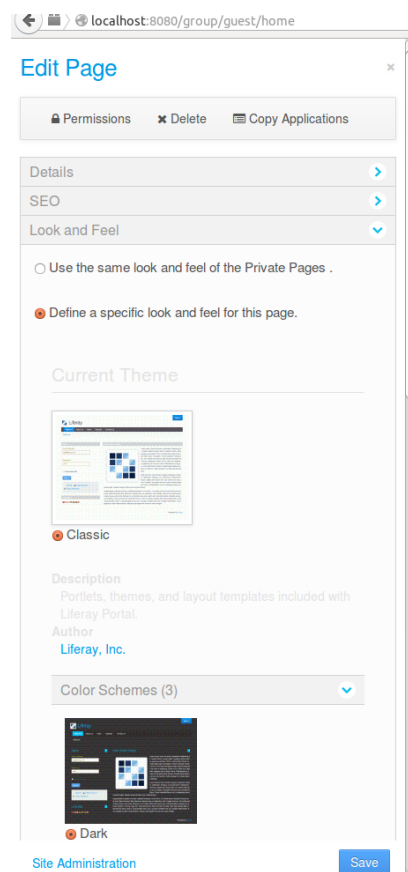
In most cases it is enough for developing the layouts, that are provided by default. But in some cases the portlets should be placed in certain layout, but the acceptable layout is hasn't included in the set. To do this, you can write a custom layout (or download the ready from [MarketPlace](#)).

Sometimes, instead of a custom layout we can use the portlet **Nested portlets**.

More about the **Layout Templates** read in this book later.

Liferay Themes

Liferay Themes (themes) - are liferay applications, that allow us to define the structure and look-and-feel for Liferay pages. If layouts define only the structure for portlet layout, then the themes allow us to determine the structure of the entire portal page (including dockbar, header, content, footer, etc.), as well as css-rules for it. By default, there are two themes in Liferay (**Welcome** and **Classic**). You can change the theme for a page through the **MenuBar (Edit → Look and Feel)**



In the menu **Site Pages** we can specify the theme as for the entire site (for all **private** or all **public** pages), as well for specific pages.

More about the **Liferay Themes** development read in this book later.

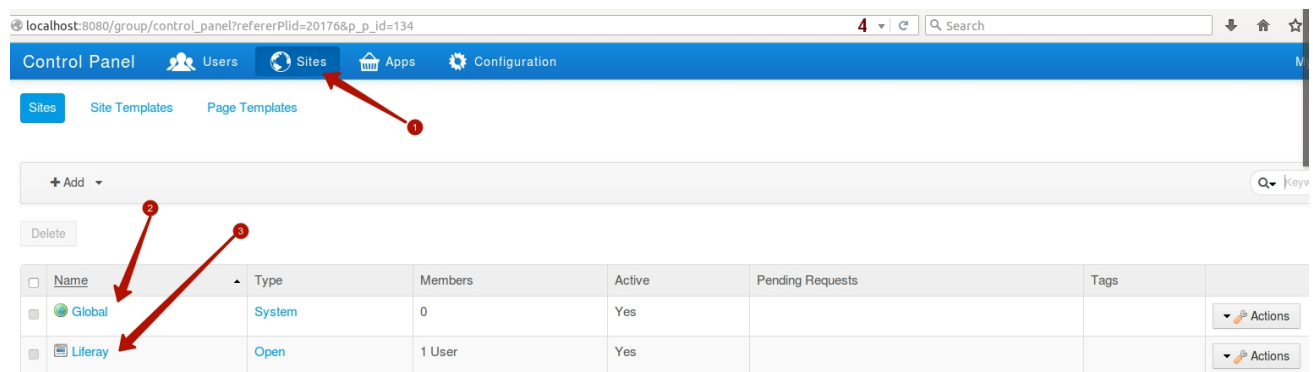
Liferay Sites

What is Liferay site?

Liferay Site (in earlier Liferay versions called **Community**) is one of the fundamental term in Liferay. From the page management point of view it's a set of public and private pages (see previous section). Except pages, site has also proper content (Web Content articles, Document and Media documents, Wiki pages, etc.), and own members (user, user groups or organizations).

Default sites in Liferay

There are two pre-defined sites in Liferay by default. When we were working with page management in previous section - we were working with Liferay's default site - '**Liferay**'. There is also another site '**Global**'. When you go to '**Sites**' menu inside **Control Panel**, you'll see them:

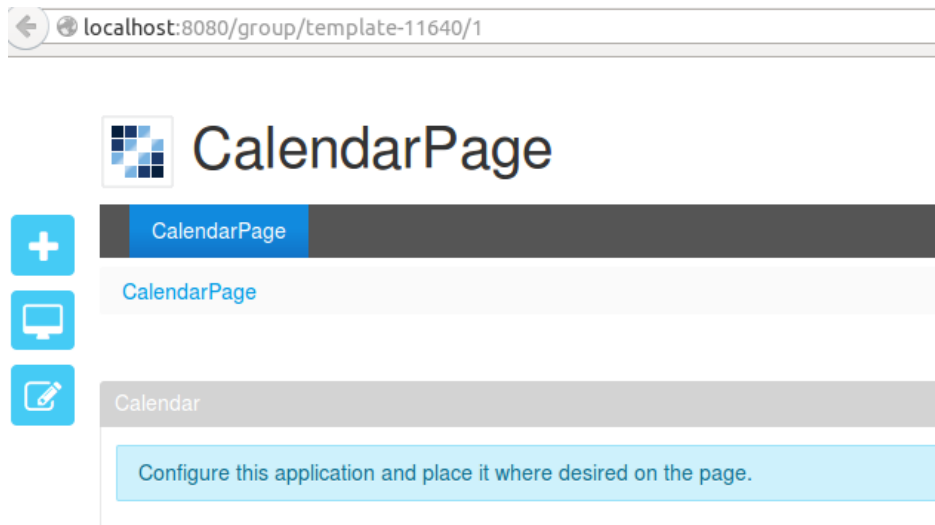


Here are differences between them:

Liferay site	Global Site
It has own pages and members of the site	It doesn't have pages and members
Site content is accessible only within this site	Site content is global, i.e. accessible from any other site

Default **Liferay** site may be used in portals, which don't have content/pages separation

between different parts of portal (sites). So, if your portal will have only one site - you may use the pre-defined **Liferay** site.



Global site is used to store shared content. Content in Liferay (web content, documents, blogs, etc.) belongs to some site inside portal. By default this content is accessible only inside this site. If you want to make some content shared between different sites - you may upload it to **Global** site, and then refer to it from other sites.

Sites management





Sites can be created: + for **organization** - herewith a site can have a set of public/private pages and content + for **user** or **user group** - in this case it is a set of public/private pages only (without content and members) + **separately** - in this case a site can have a set of public/private pages, content and members of a site (users/organizations/user groups)

The site can be created as blank one (**Add** → **Blank Site**) or via **Site Template**.

Site Template consists of Page Templates.

We will create a page template: **Page Template** → **Add**. We will point a name as **CalendarPage**. Then go **Action** → **Edit** for created page template and select **Open Page Template**:

We will add a portlet to the **Calendar** page and select **1-column** layout, as a result we will get:


Control Panel  Users  Sites  Apps  Configuration

Sites Site Templates **Page Templates**

[+ Add](#)

[←](#) CalendarPage

Name (Required)







Description

☒ Active


Configuration
[Open Page Template](#)


[Save](#) [Cancel](#)


After that we will create **SiteTemplate** (**Site Templates** → **Add**). We will assign a name as **CalendarSite**, we will get as a result:


[←](#) Control Panel  Users  Sites  Apps  Configuration


Sites **Site Templates** Page Templates


[←](#) CleandarSite 
 Visit: [Site Pages](#)





Pages 

Content 

Configuration 


 **Site Template Settings**

 [Application Display Templates](#)

 [Mobile Device Families](#)


Site Template Settings

Name (Required)



Description

☒ Active

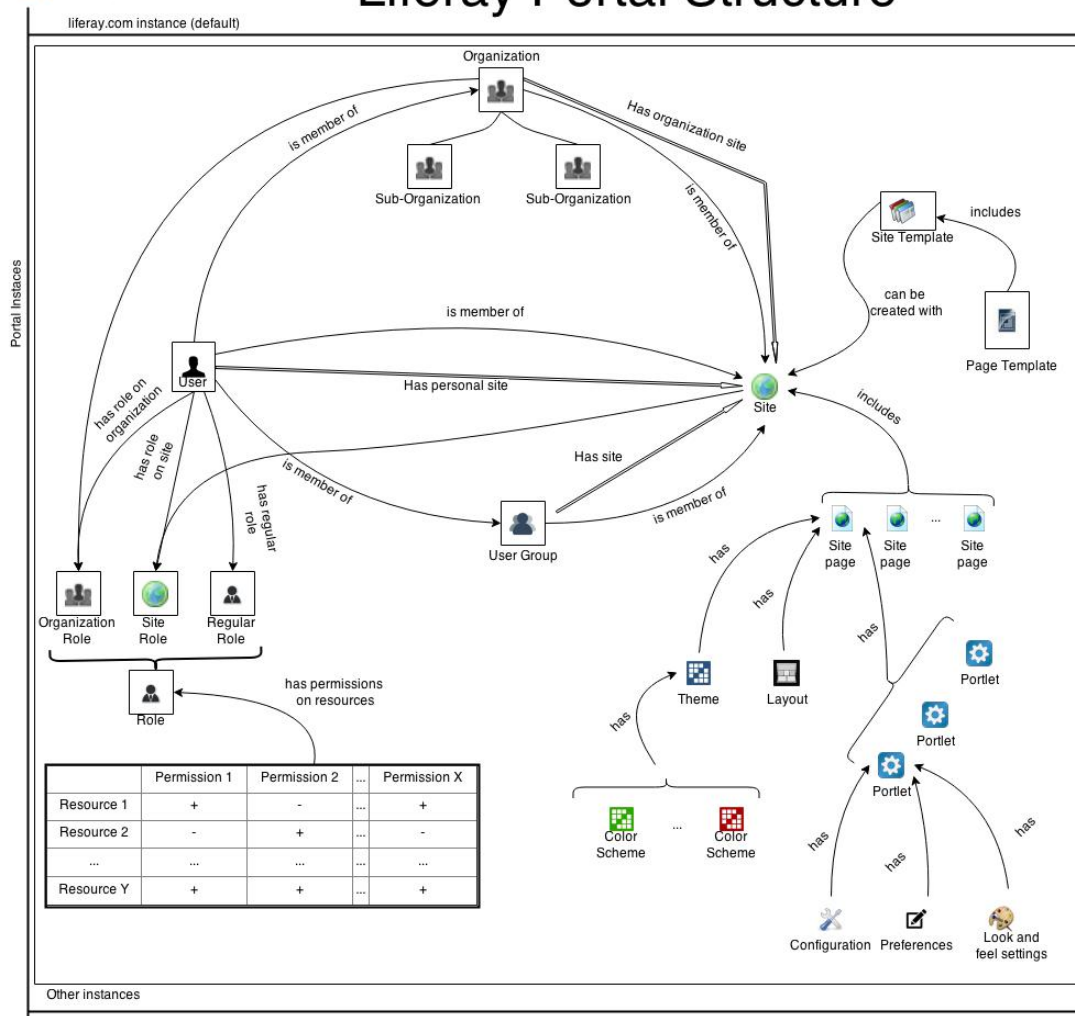
☒ Allow Site Administrators to Modify the Pages Associated with This Site Template 

[Save](#) [Cancel](#)

We will add a new page to the section **Pages**, selecting herewith the created earlier **Page Template**:



Liferay Portal Structure



Add Page

☐ MyPageTemplate

MyPageTemplate

☒ CalendarPage

CalendarPage

☒ Automatically apply changes done to the page template.

☐ Panel

Create a page with predefined applications and n...

☐ Embedded

Show content from another website.

☐ Link to URL

Link to another website.

☐ Link to a Page of This Site

Link to another page in the current site.

☐ Copy of a Page of This Site

Copy an existing page from this site.

Add Page

Cancel

Then we will create a new layout, using created **Site Template: Sites → Add → Calendar Site**.

The configuration of pages/portlets, that is specified in **Site Template** will be applied to the created site.

When creating a site template you can specify the pages (**public** or **private**), to which the template will be applied, and whether the template changes will change automatically the created site.

To add the site members it is necessary to select **Site Membership** in the **Users** section, then select **Users**, click on **Assign Users**,

Name	Screen Name	Site Roles and Teams	Actions
Test Test	test	Site Owner	Assign Users

then select the users you need, click on **Save**:

The screenshot shows the Liferay Control Panel interface. The top navigation bar includes 'Control Panel', 'Users', 'Sites', 'Apps', and 'Configuration'. Below this, there are tabs for 'Sites', 'Site Templates', and 'Page Templates'. The 'Sites' tab is active, showing a list of sites with 'CleandarSite' selected. The left sidebar contains a menu with options like 'Pages', 'Content', 'Users', 'Site Memberships', 'Site Teams', and 'Configuration'. The main content area is titled 'Site Memberships' and includes a section for 'Add Members: Users'. Below this is a search bar with a 'Keywords' input and a 'Search' button. A table lists the current members:

<input checked="" type="checkbox"/>	Name	Screen Name
<input checked="" type="checkbox"/>	Vitaliy Koshelenko	vetal
<input checked="" type="checkbox"/>	Test Test	test

At the bottom of the main content area is a 'Save' button.

Similarly, the organization and user groups can be specified as site members.

LIFERAY PORTAL structure

LIFERAY configuration

The facilities for configuration / administration Liferay are located on the '**Configuration**' tab in the **Control Panel**:

The screenshot shows the Liferay Control Panel interface with the 'Configuration' tab selected. The top navigation bar includes 'Control Panel', 'Users', 'Sites', 'Apps', and 'Configuration'. Below this, there are tabs for 'Portal Settings', 'Custom Fields', 'Server Administration', 'Portal Instances', and 'Workflow'. The 'Portal Settings' tab is active, showing a form for 'Main Configuration'. The form is divided into several sections:

- Main Configuration:** Includes fields for 'Name (Required)' (Liferay), 'Mail Domain (Required)' (liferay.com), 'Virtual Host (Required)' (localhost), 'CDN Host HTTP', 'CDN Host HTTPS', and a checkbox for 'CDN Dynamic Resources Enabled'.
- Navigation:** Includes a field for 'Home URL' and fields for 'Default Landing Page' and 'Default Logout Page'.
- Additional Information:** Includes fields for 'Legal Name', 'Legal ID', 'SIC Code', and 'Ticker Symbol'.

On the right side of the configuration form is a sidebar with a list of configuration categories: 'General', 'Authentication', 'Users', 'Mail Host Names', 'Email Notifications', 'Content Sharing', 'IDENTIFICATION', 'Addresses', 'Phone Numbers', 'Additional Email Addresses', 'Websites', and 'MISCELLANEOUS'. The 'General' category is selected. At the bottom of the sidebar are 'Save' and 'Cancel' buttons.

Liferay Applications Management

By default Liferay contains a large number of applications (portlets) for different purposes (for example, **Wiki**, **Blog**, **Message Boards**, **WebContent**, etc.).

To add a portlet to the page you need select **Applications** from the **MenuBar**, and then select the appropriate portlet. After clicking **Add** the portlet will be embedded to the page.

Portlets can be **instanceable** (can be added to a page several times) and **not-instanceable** (can be added only once). Each portlet has its own unique **ID**.

For instanceable portlets for providing their uniqueness, **portlet namespace** is written additionally to **ID** (for each portlet on the page the unique **portlet namespace** is generated).

After adding a portlet to a page, you can assign it the configuration settings and right of the access (**permissions**). Here, for example, the form for specifying permissions for **Calendar** portlet.

Role	Access in Control Panel	Add to Page	Configuration	View	Permissions
Owner	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Portal Content Reviewer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Power User	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Site Content Reviewer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Site Member	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
User	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

To display/hide this portlet for certain role it is necessary to check/uncheck the flag **View** (opposite the role). The visual settings for portlet displaying it is possible to specify in **'Look and Feel'** menu. You can specify a custom title for the portlet, display/hide the borders, prescribe the custom **css** rules for portlet:

Look and Feel

Portlet Configuration | Text Styles | Background Styles | Border Styles | Margin and Padding | Advanced Styling

☒ Use Custom Title

My Calendar | English (United States)

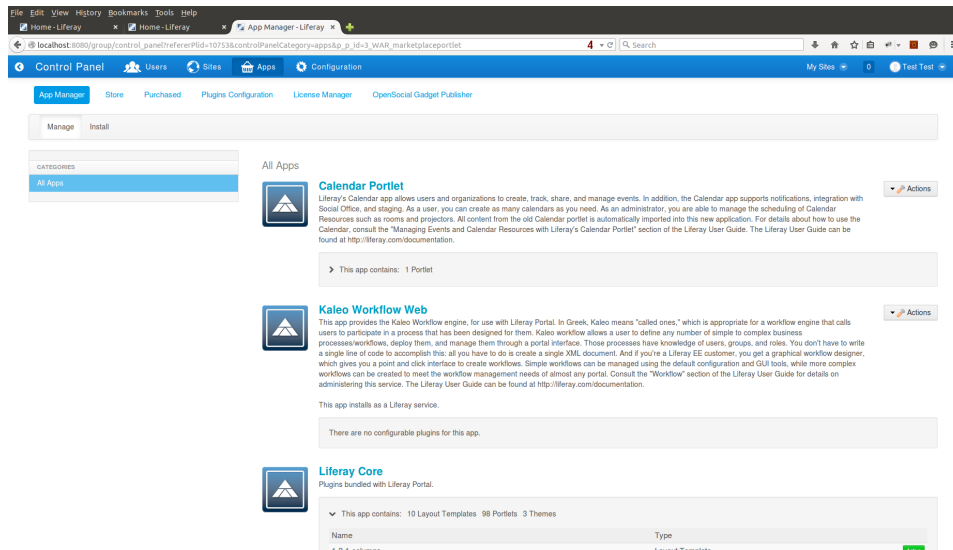
Link Portlet URL to Page: Current Page

Show Borders: No

Save | Reset

The portlets on a page can be moved by drag-and-drop (according to the selected layout).

The list of all installed applications can be viewed in **Control Panel -> Apps**

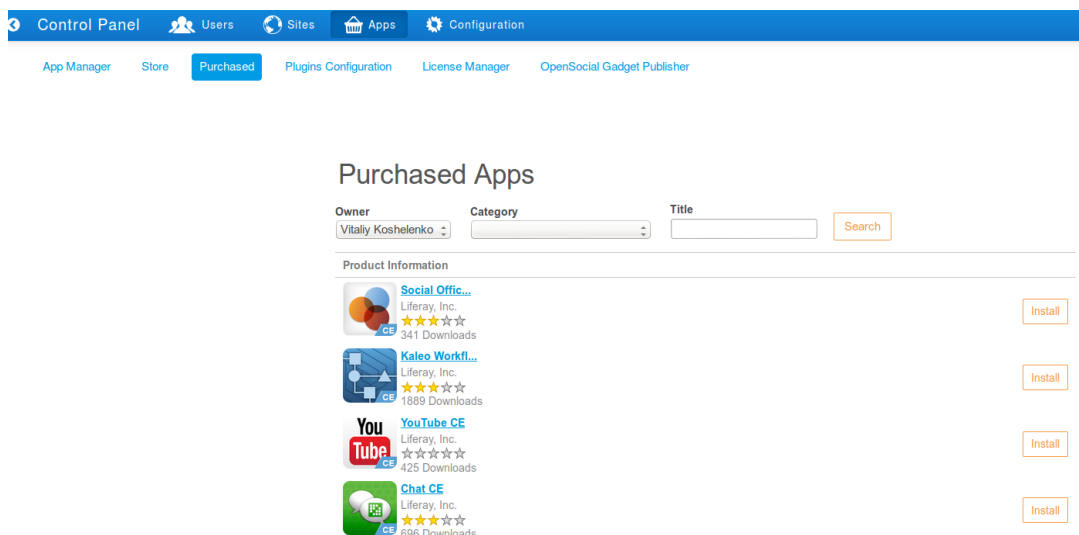


Here you can activate/deactivate or remove the desired application.

On the **Install** tab, you can set the desired application (via **upload war/lpkg** file or via **URL**).

To install an application from the [Liferay Marketplace](#) is necessary to go to **Store** on the **Apps** tab and log in (if you don't have an account - register on [liferay.com](#) before).

After selecting the desired application click on **Purchase**. After that it should be displayed on the tab '**Purchased**':



Click on **Install**, and the application will be installed.

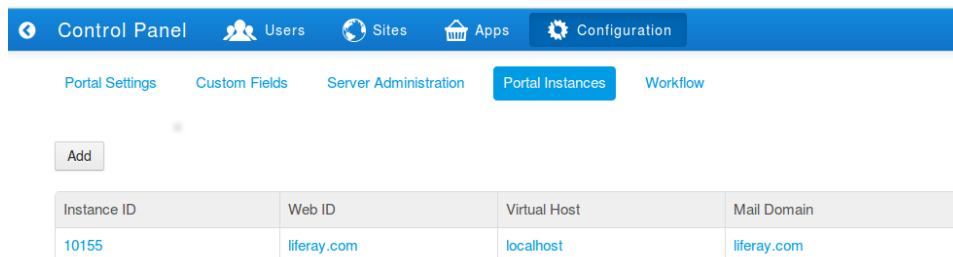
Developing own portlets is discussed in this book later.

Liferay Administration

For **Liferay** administering we could go **Admin** → **Control Panel**.

Portal Instances

Portal instance (or company) - it is a separate instantiation of the portal. By default, one

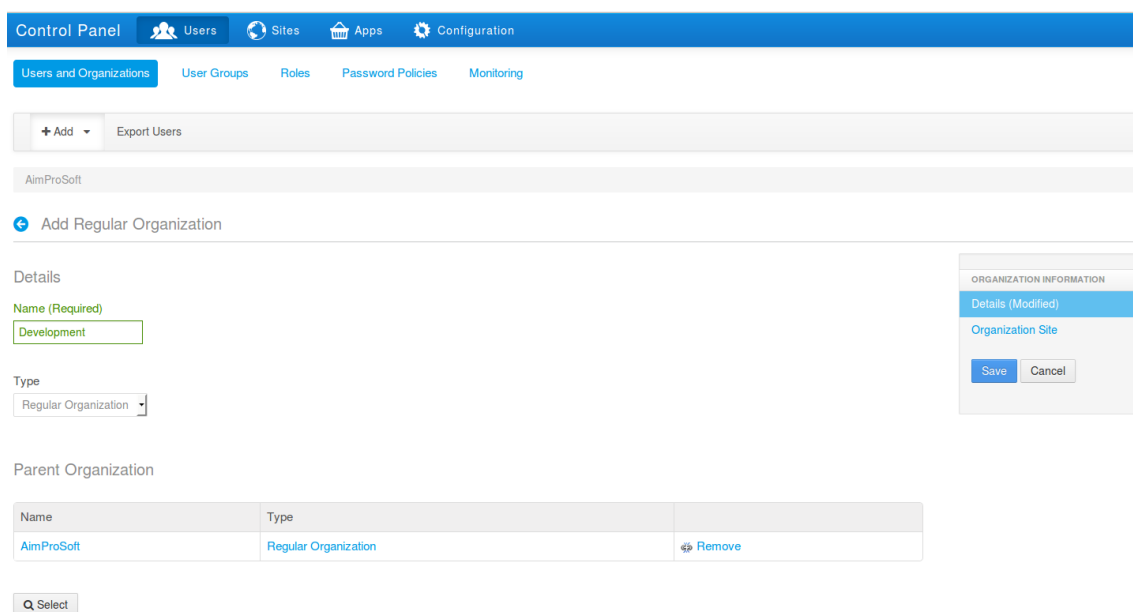


Instance ID	Web ID	Virtual Host	Mail Domain
10155	liferay.com	localhost	liferay.com

company **liferay.com** is created in Liferay:

They are used for differentiation of data (users, organizations, content, etc.) between different companies.

For this purpose, almost in the every liferay-table the field **companyId** exists, that determines the belonging to the company.



Control Panel Users Sites Apps Configuration

Users and Organizations User Groups Roles Password Policies Monitoring

+ Add Export Users

AimProSoft

+ Add Regular Organization

Details

Name (Required)
Development

Type
Regular Organization

Parent Organization

Name	Type	
AimProSoft	Regular Organization	Remove

Q Select

ORGANIZATION INFORMATION

Details (Modified)

Organization Site

Save Cancel

Mainly, single company is used (that is created by default).

Users and Organizations

In Liferay users are created in **Users** and **Organizations** → **Add** → **User**:

Users have the opportunities: + to login Liferay + to have their own website (set of **public** and/or **private** pages) + to be members of the organizations/websites/user groups + to have certain roles, which give them certain permissions (access rights)

Organization – is Liferay structural unit, uniting the users (users can be organization members).

The organization can have its own website (a set of **public** and/or **private** pages).

Organizations have a hierarchical structure (some organizations can be sub-organizations of the others).

Organizations can be of two types: **Regular Organization** (it is usual organization) and **Location** (it differs from the usual in that has the fields for entering a names of a country and a region). Organizations can be of two types: Regular Organization (it is usual organization) and Location (it differs from the usual in that has the fields for entering a names of a country and a region).

Example organization:

Control Panel
Users
Sites
Apps
Configuration

Users and Organizations
User Groups
Roles
Password Policies
Monitoring

+ Add
Export Users

AimProSoft

Edit AimProSoft


Details

Name (Required)
AimProSoft

Type
Regular Organization

Site ID
11145

Parent Organization
Select


Change
Delete

AimProSoft

ORGANIZATION INFORMATION

Details

Organization Site

Categorization

IDENTIFICATION

Addresses

Phone Numbers

Additional Email Addresses

Websites

Services

MISCELLANEOUS

Comments

Reminder Queries

Custom Fields

Save
Cancel

To add a user to the organization, we should go to **Actions** → **Assign Users**, then go to the tab **Available** and select the required users and click on **Update Association**:

Control Panel
Users
Sites
Apps
Configuration

Users and Organizations
User Groups
Roles
Password Policies
Monitoring

AimProSoft

Current
Available

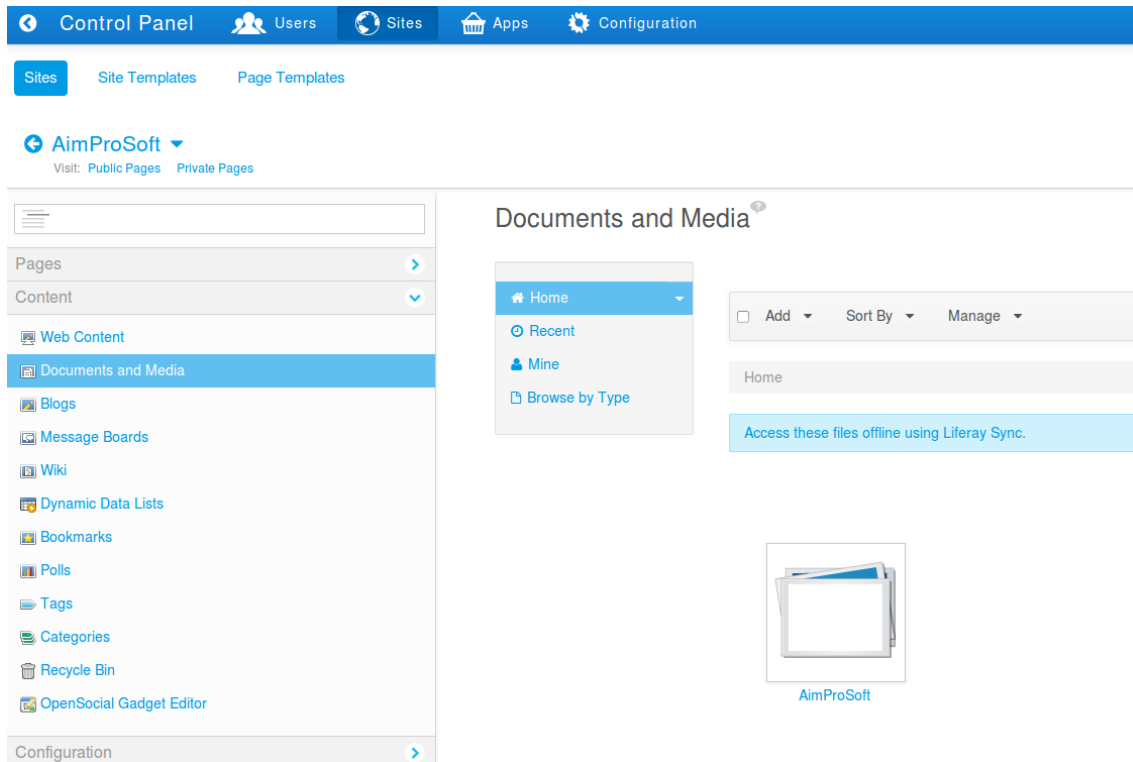
Keywords
Search

Update Associations

<input type="checkbox"/>	Name	Screen Name
<input checked="" type="checkbox"/>	Vitaliy Koshelenko	vetal
<input type="checkbox"/>	Test Test	test

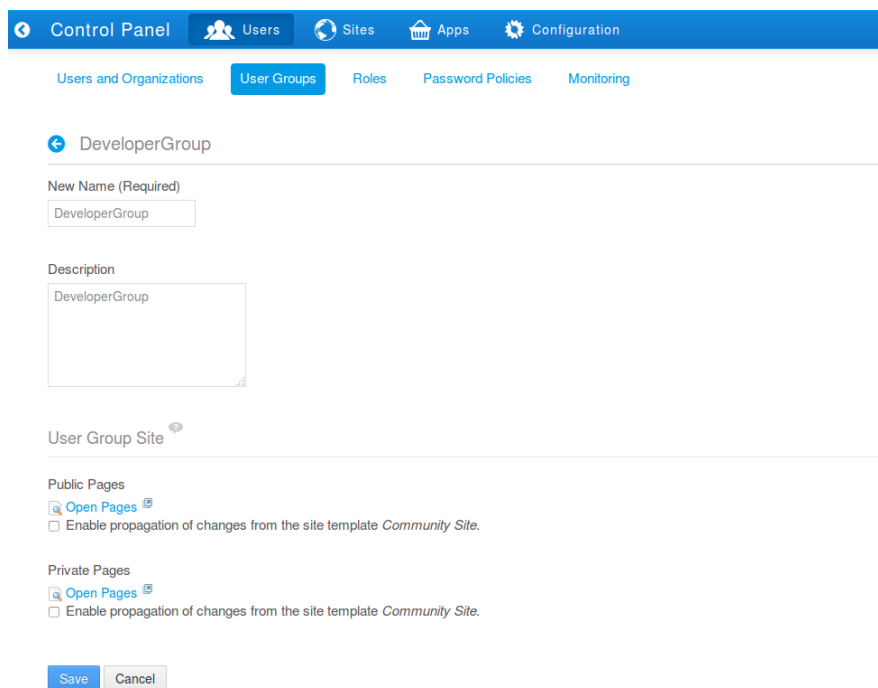
To add a subsidiary, we should press **Add Regular Organization** (or **Add Location**) of the current organization:

Website of the organization can have its own content (documents, web content, wiki pages, etc.):



The User Groups

The **user group** – is also Liferay structural unit, uniting the users. Users can be user group members. **User group** can have a set of **public** and **private** pages.

















But, unlike the organization, **user group** site can not have the content (documents, web-content, etc.).

User groups don't have a hierarchical structure.

Roles and Permissions

Roles in Liferay intended for determination of users permissions (access rights) to certain objects (content, pages, portlets, etc.). These roles can be assigned to users to provide them with required access rights.

In Liferay 14 roles have already been created by default:

Control Panel			
Users Sites Apps Configuration			
Users and Organizations User Groups Roles Password Policies Monitoring			
Add ▾			
20 Items per Page ▾ Page 1 of 1 ▾ Showing 14 results.			
Title	Type	Description	
 Administrator	Regular	Administrators are super users who can do anything.	
 Guest	Regular	Unauthenticated users always have this role.	
 Organization Administrator	Organization	Organization Administrators are super users of their organization but cannot make other users into Organization Administrators.	
 Organization Content Reviewer	Organization	Autogenerated role from workflow definition	
 Organization Owner	Organization	Organization Owners are super users of their organization and can assign organization roles to users.	
 Organization User	Organization	All users who belong to an organization have this role within that organization.	
 Owner	Regular	This is an implied role with respect to the objects users create.	
 Portal Content Reviewer	Regular	Autogenerated role from workflow definition	
 Power User	Regular	Power Users have their own personal site.	
 Site Administrator	Site	Site Administrators are super users of their site but cannot make other users into Site Administrators.	
 Site Content Reviewer	Site	Autogenerated role from workflow definition	
 Site Member	Site	All users who belong to a site have this role within that site.	
 Site Owner	Site	Site Owners are super users of their site and can assign site roles to users.	
 User	Regular	Authenticated users should be assigned this role.	

If necessary, we can create our own custom roles.

The roles can be: + **Regular Role** (general role) – granted to user for entire portal + **Site Role** (role for site) – granted to user for the site + **Organization Role** (organizational role) – granted user for the entire organization

The same user can have different roles in different sites/organization. For example, the user can be admin in one organization, but at the same time he can be usual user in another one.

To add a role to a user, we should go to **Edit User** and select the required roles:

AimProSoft / Vitaliy Koshelenko

Edit User *Vitaliy Koshelenko*

Regular Roles

Title	
Power User	Remove
CustomRegularRole	Remove

[Select](#)

Inherited Regular Roles

This user does not have any inherited regular roles.

Organization Roles

Title	Organization	
CustomOrgRole	AimProSoft	Remove

[Select](#)

Site Roles

Title	Site	
CustomSiteRole	BlankSite	Remove

[Select](#)

Inherited Site Roles

This user does not have any inherited site roles.

Vitaliy Koshelenko

USER INFORMATION

[Details](#)

[Password](#)

[Organizations](#)

[Sites](#)

[User Groups](#)

[Roles \(Modified\)](#)

[Personal site](#)

[Categorization](#)

IDENTIFICATION

[Addresses](#)

[Phone Numbers](#)

[Additional Email Addresses](#)

[Websites](#)

[Instant Messenger](#)

[Social Network](#)

[SMS](#)

[OpenID](#)

MISCELLANEOUS

[Announcements](#)

[Display Settings](#)

[Comments](#)

[Custom Fields](#)

To assign the permissions for the role, we should select **Actions** → **Define Permissions** for this role, then select the required permissions and click on **Save**.

For example, for created role **Organization Publisher** we select the permissions, required for content management.

Control Panel [Users](#) [Sites](#) [Apps](#) [Configuration](#) My Sites 0 Test Test

[Users and Organizations](#) [User Groups](#) **[Roles](#)** [Password Policies](#) [Monitoring](#)

OrganizationPublisher

[Edit](#) [Define Permissions](#)

Summary

Users and Organizations

▼ Site Administration

▼ Pages

Site Pages

▼ Content

Recent Content

Web Content

Documents and Media

Blogs

Message Boards

Wiki

Dynamic Data Lists

Bookmarks

Polls

Software Catalog

Tags

Categories

Recycle Bin

OpenSocial Gadget Editor

Web Content

General Permissions

Action	Permissions
<input type="checkbox"/> Action	
<input type="checkbox"/> Access in Site Administration	
<input type="checkbox"/> Configuration	
<input type="checkbox"/> Permissions	
<input checked="" type="checkbox"/> View	

Resource Permissions

Web Content

Action	Permissions
<input type="checkbox"/> Action	
<input type="checkbox"/> Add Feed	
<input checked="" type="checkbox"/> Add Folder	
<input checked="" type="checkbox"/> Add Structure	
<input checked="" type="checkbox"/> Add Template	
<input checked="" type="checkbox"/> Add Web Content	
<input type="checkbox"/> Permissions	

Users with this role will have the appropriate access rights.

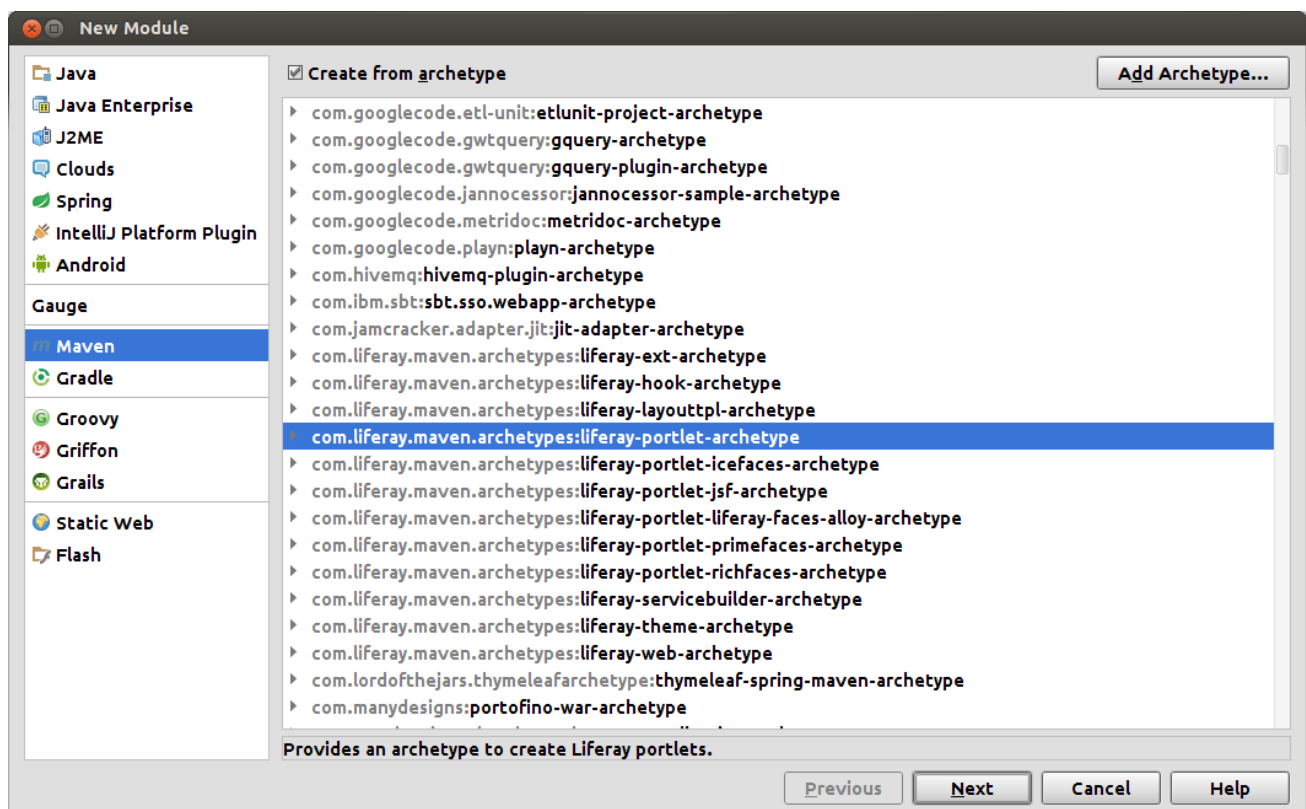
Chapter 3. First Portlet

We got to know installation/startup Liferay in the previous chapters, and also basic principles of work with it. Now we write our first Liferay portlet.

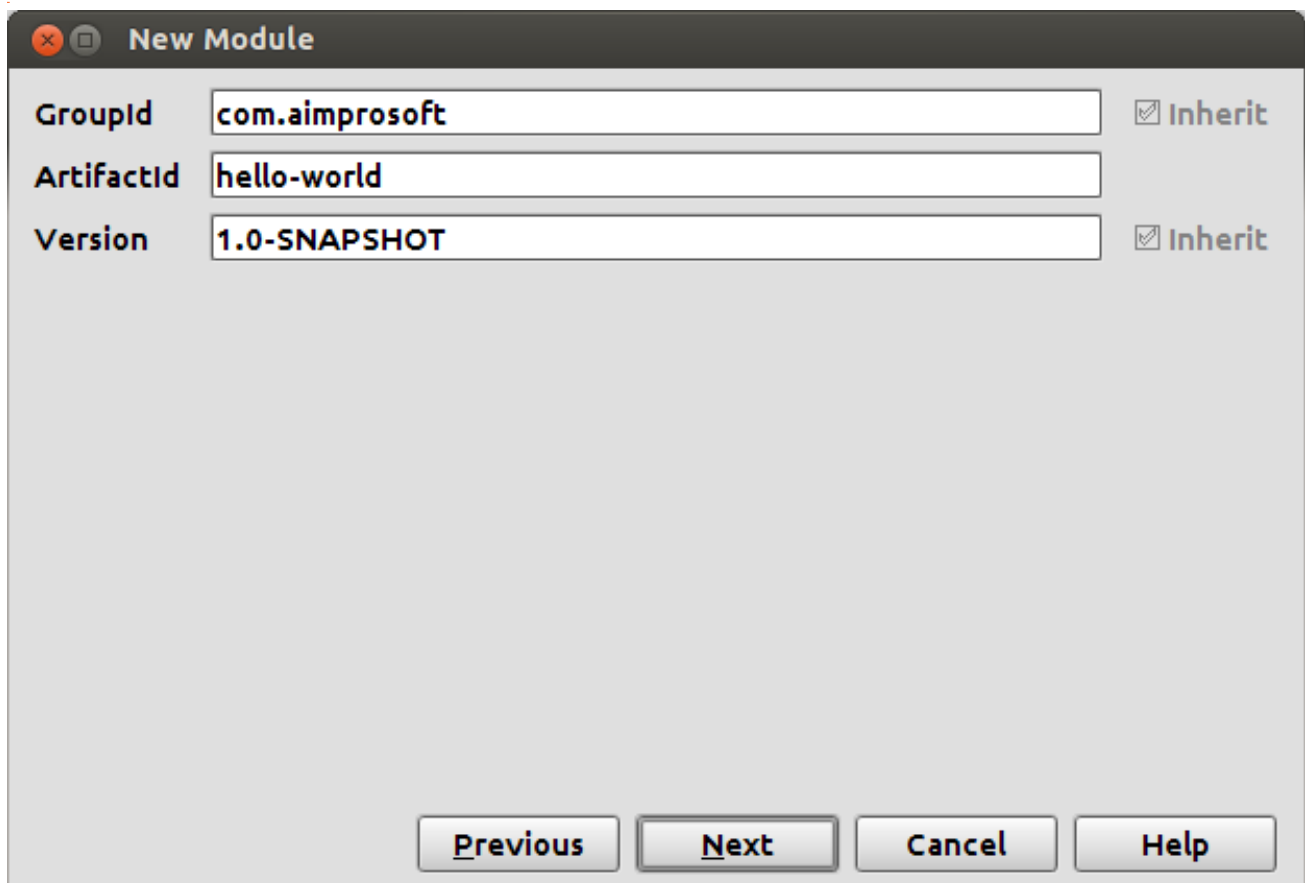
Creating new module in IDEA

Create a new module *'hello-world'* by using of *Maven archetype*.

To do it, go File → New Module..., select Maven, check 'Create from archetype' and select 'liferay-portlet-archetype':



Click on **Next**, specify *groupId* and *artifactId*



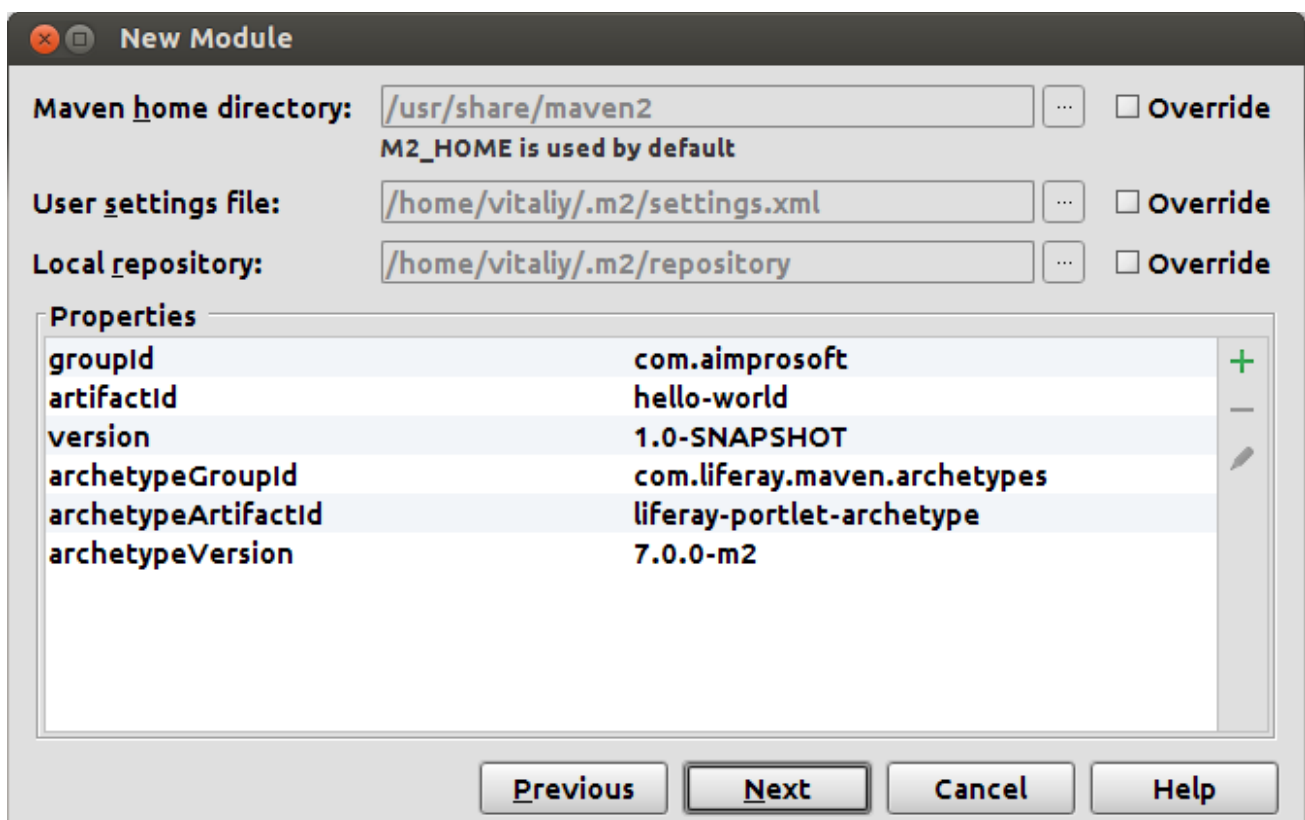
New Module

GroupId: ☒ Inherit

ArtifactId:

Version: ☒ Inherit

Click on **Next**, setup *Maven* (if it was not setup formerly):



New Module

Maven home directory: ... ☐ Override
M2_HOME is used by default

User settings file: ... ☐ Override

Local repository: ... ☐ Override

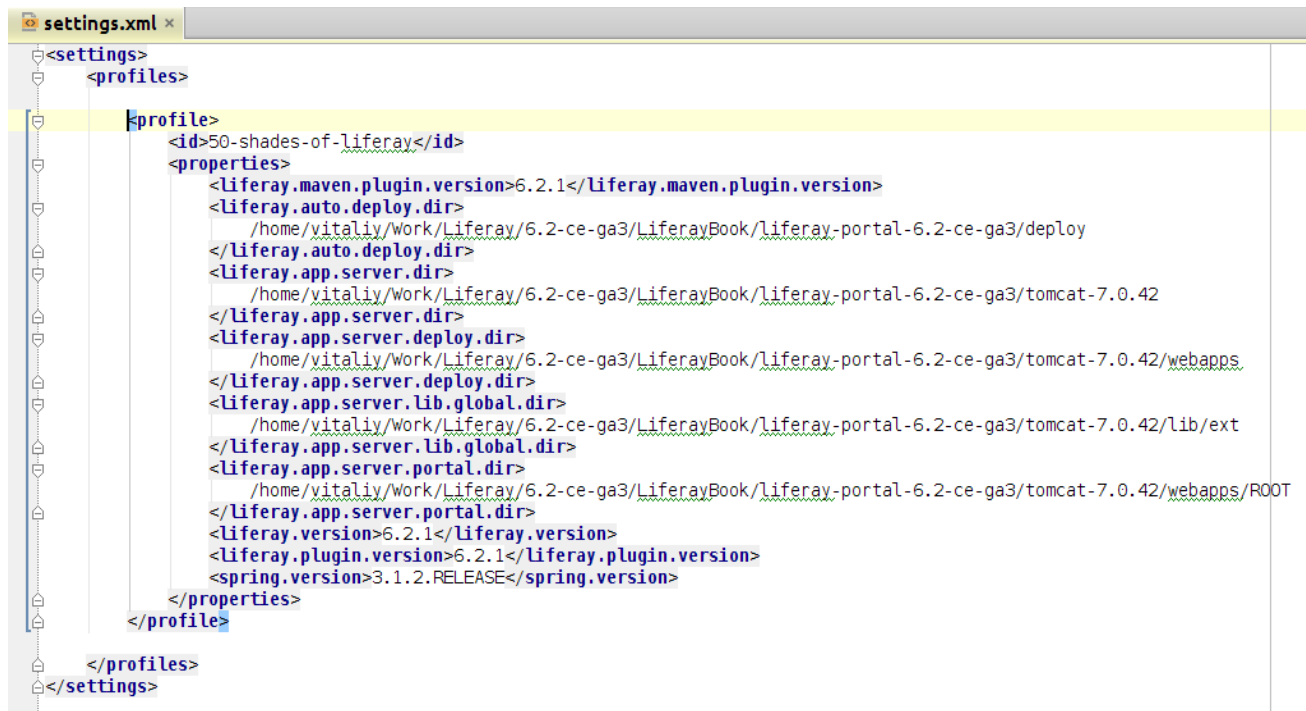
Properties

groupId	com.aimprosoft	+ - ✎
artifactId	hello-world	
version	1.0-SNAPSHOT	
archetypeGroupId	com.liferay.maven.archetypes	
archetypeArtifactId	liferay-portlet-archetype	
archetypeVersion	7.0.0-m2	

Click on **Next**, specify the way to the module and click on **Finish**.

Maven will create the structure of the project automatically.

After that, add Maven profile. To do it, create the file **settings.xml** in the folder **\$HOME\$/.m2**:



```

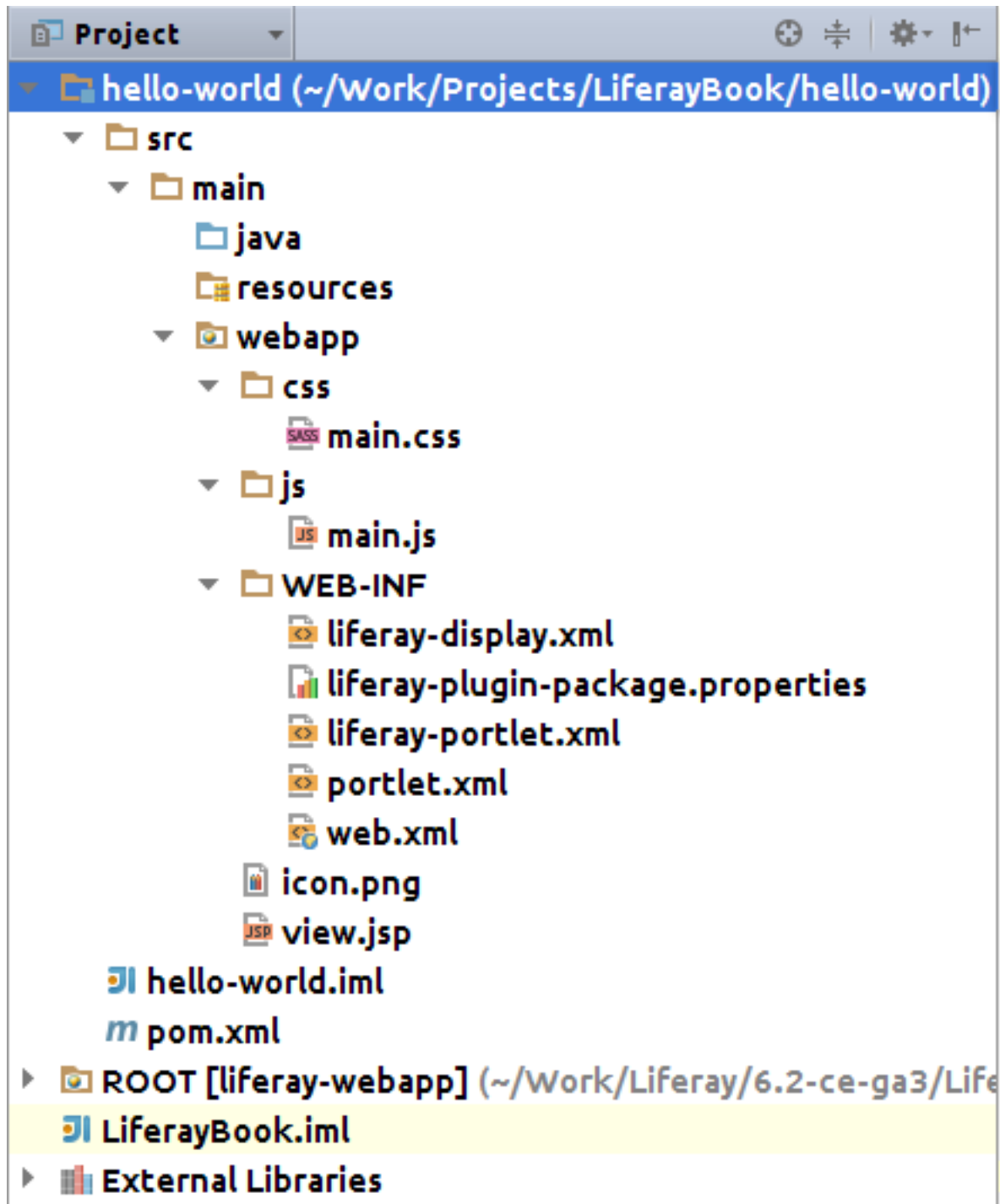
<settings>
  <profiles>
    <profile>
      <id>50-shades-of-liferay</id>
      <properties>
        <Liferay.maven.plugin.version>6.2.1</Liferay.maven.plugin.version>
        <Liferay.auto.deploy.dir>
          /home/vitaliy/Work/Liferay/6.2-ce-ga3/LiferayBook/liferay-portal-6.2-ce-ga3/deploy
        </Liferay.auto.deploy.dir>
        <Liferay.app.server.dir>
          /home/vitaliy/Work/Liferay/6.2-ce-ga3/LiferayBook/liferay-portal-6.2-ce-ga3/tomcat-7.0.42
        </Liferay.app.server.dir>
        <Liferay.app.server.deploy.dir>
          /home/vitaliy/Work/Liferay/6.2-ce-ga3/LiferayBook/liferay-portal-6.2-ce-ga3/tomcat-7.0.42/webapps
        </Liferay.app.server.deploy.dir>
        <Liferay.app.server.lib.global.dir>
          /home/vitaliy/Work/Liferay/6.2-ce-ga3/LiferayBook/liferay-portal-6.2-ce-ga3/tomcat-7.0.42/lib/ext
        </Liferay.app.server.lib.global.dir>
        <Liferay.app.server.portal.dir>
          /home/vitaliy/Work/Liferay/6.2-ce-ga3/LiferayBook/liferay-portal-6.2-ce-ga3/tomcat-7.0.42/webapps/ROOT
        </Liferay.app.server.portal.dir>
        <Liferay.version>6.2.1</Liferay.version>
        <Liferay.plugin.version>6.2.1</Liferay.plugin.version>
        <spring.version>3.1.2.RELEASE</spring.version>
      </properties>
    </profile>
  </profiles>
</settings>

```

Select this created profile on the tab Maven in IDEA. After that Maven will pull up the necessary dependencies.

Structure of the created module

After creating the module with using *Maven archetype*, we will have as a result the module 'hello-world' with such structure:



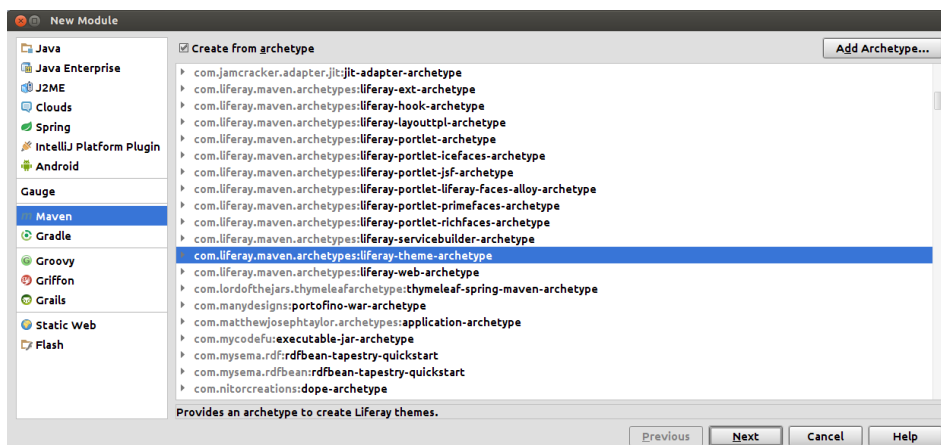
Chapter 4. First Theme

Perform all nicely..

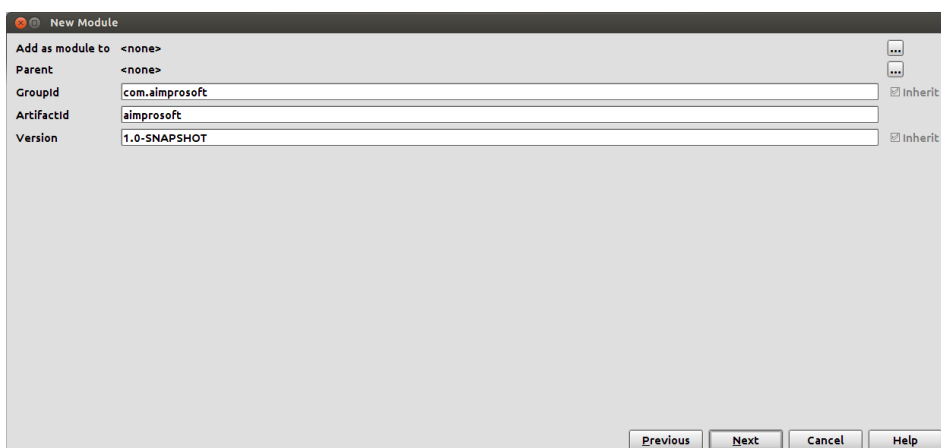
Now we will write our first Liferay theme.

Creating new module in IDEA

We will create the new module 'aimprosoft' with using of 'Maven archetype'. To do it, go *File* → *New Module...*, select *Maven*, check 'Create from archetype' and select 'liferay-theme-archetype':



Specify groupId and artifactId:

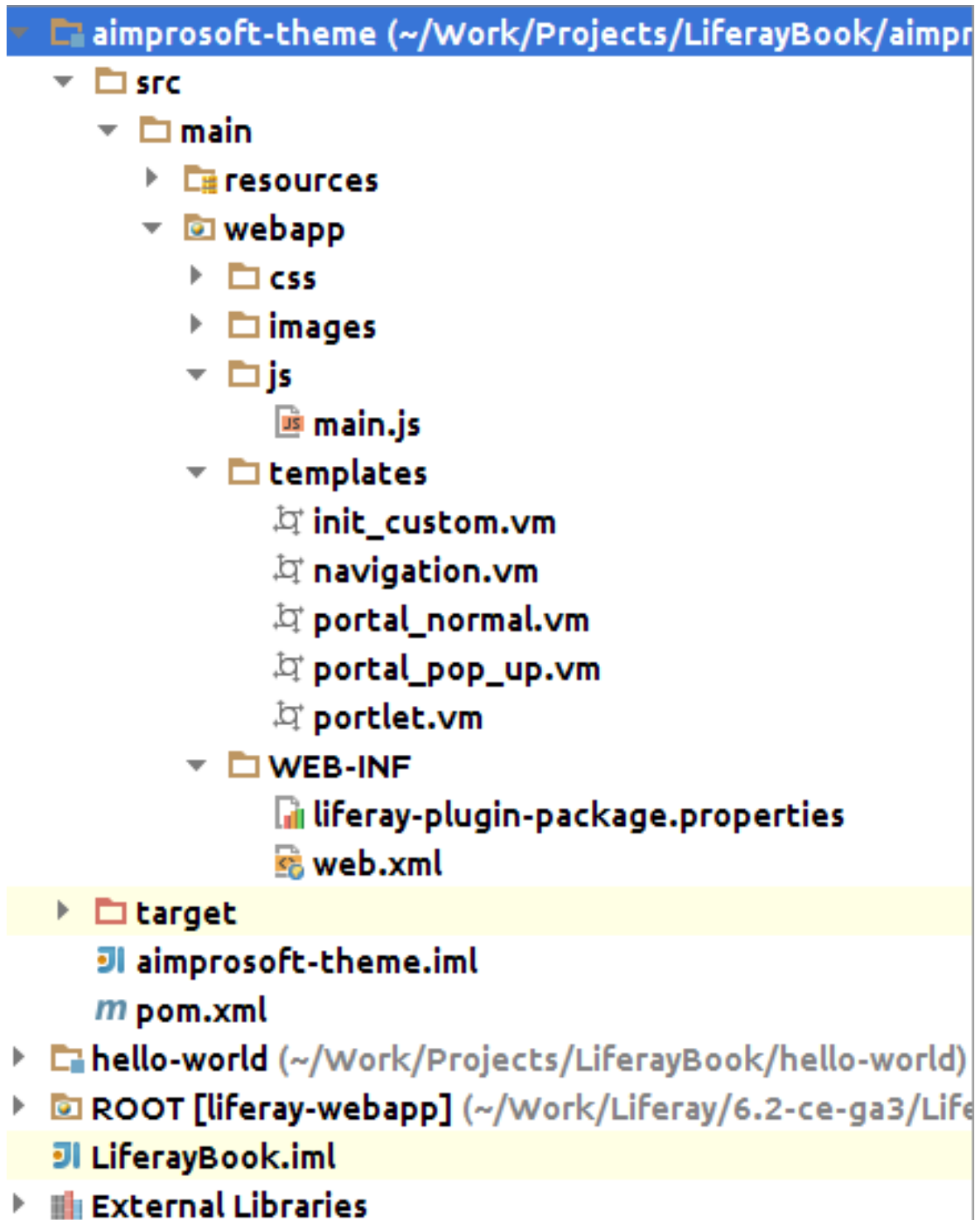


Check *Maven settings*, specify location of module with the theme.

Structure of the theme will be created automatically.

We will create a theme on the basis of standard *Liferay classic* theme. To do it, copy all files **.vm* from *ROOT/html/themes/classic/templates*, and also copy *js*, *css*, *images* folders.

We will get such structure:



After compiling we get the ready theme, the same as the *classic* one.

We will change:

- layout in *portal_normal.vm*
- styles in *custom.css*
- images *favicon.png* and *thumbnail.png*
- of necessity *navigation.vm* (for changing of navigation)
- *portlet.vm* (for changing of portlets)

In *init_custom.vm* we can specify our own variables, which we will be able to use later in *portal_normal.vm*. List of all predefined variables is located in the file */ROOT/html/themes/_unstyled/templates/init.vm*.

After necessary changes we should recompile the theme and apply it to necessary pages.

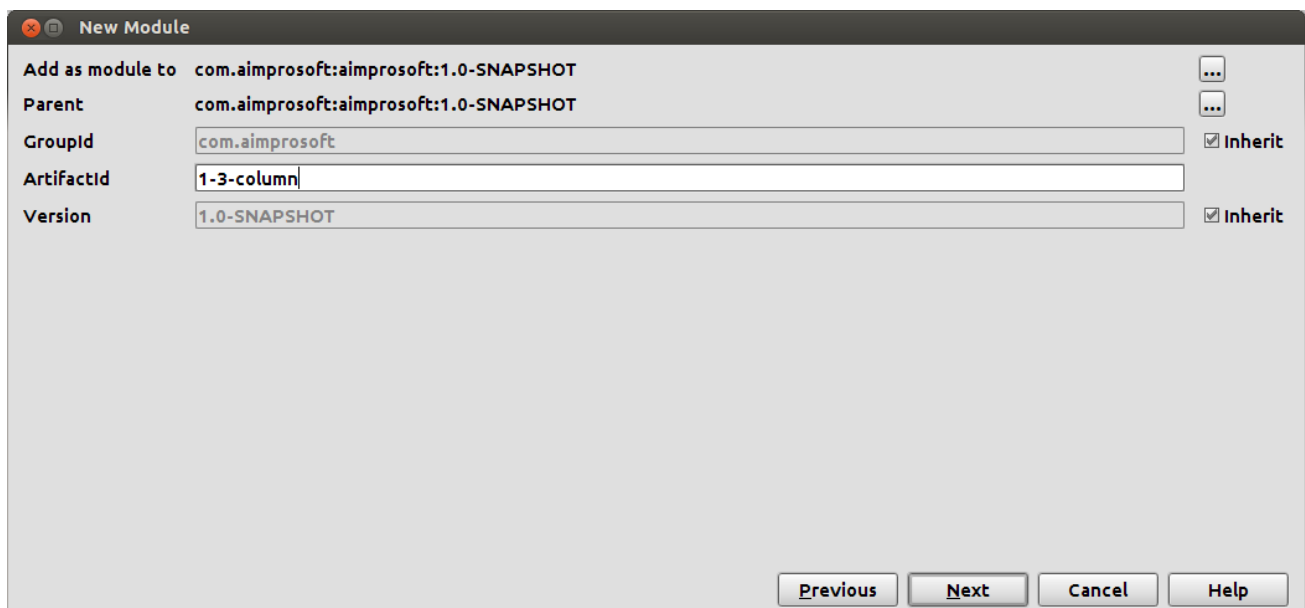
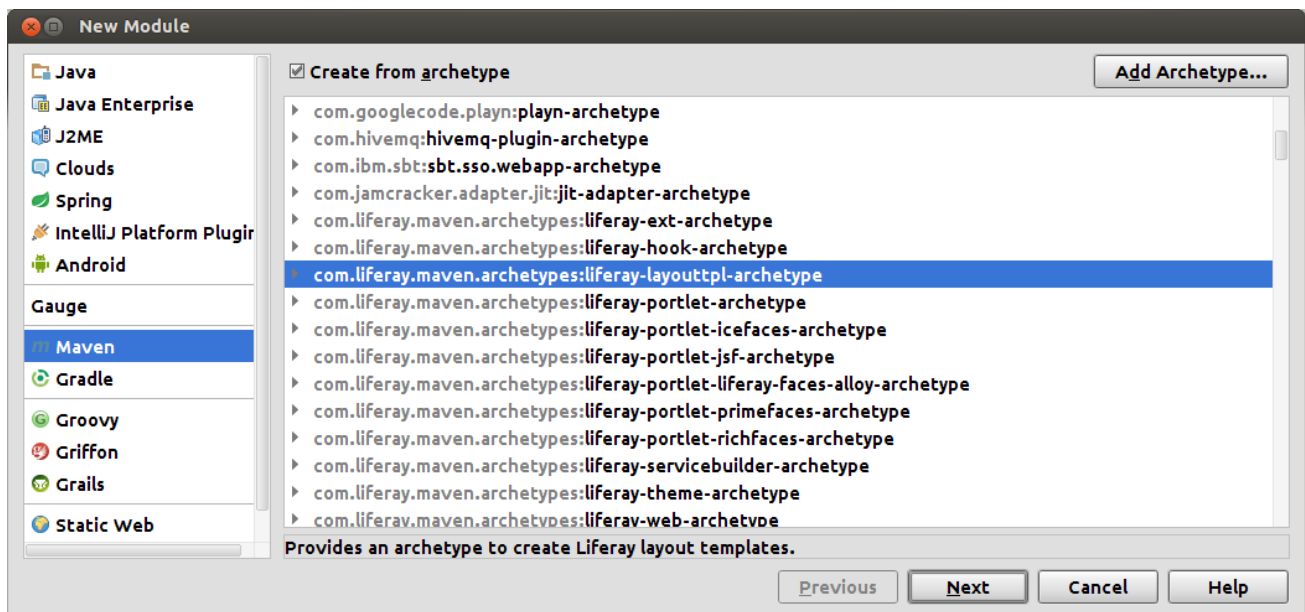


Chapter 5. First Layout

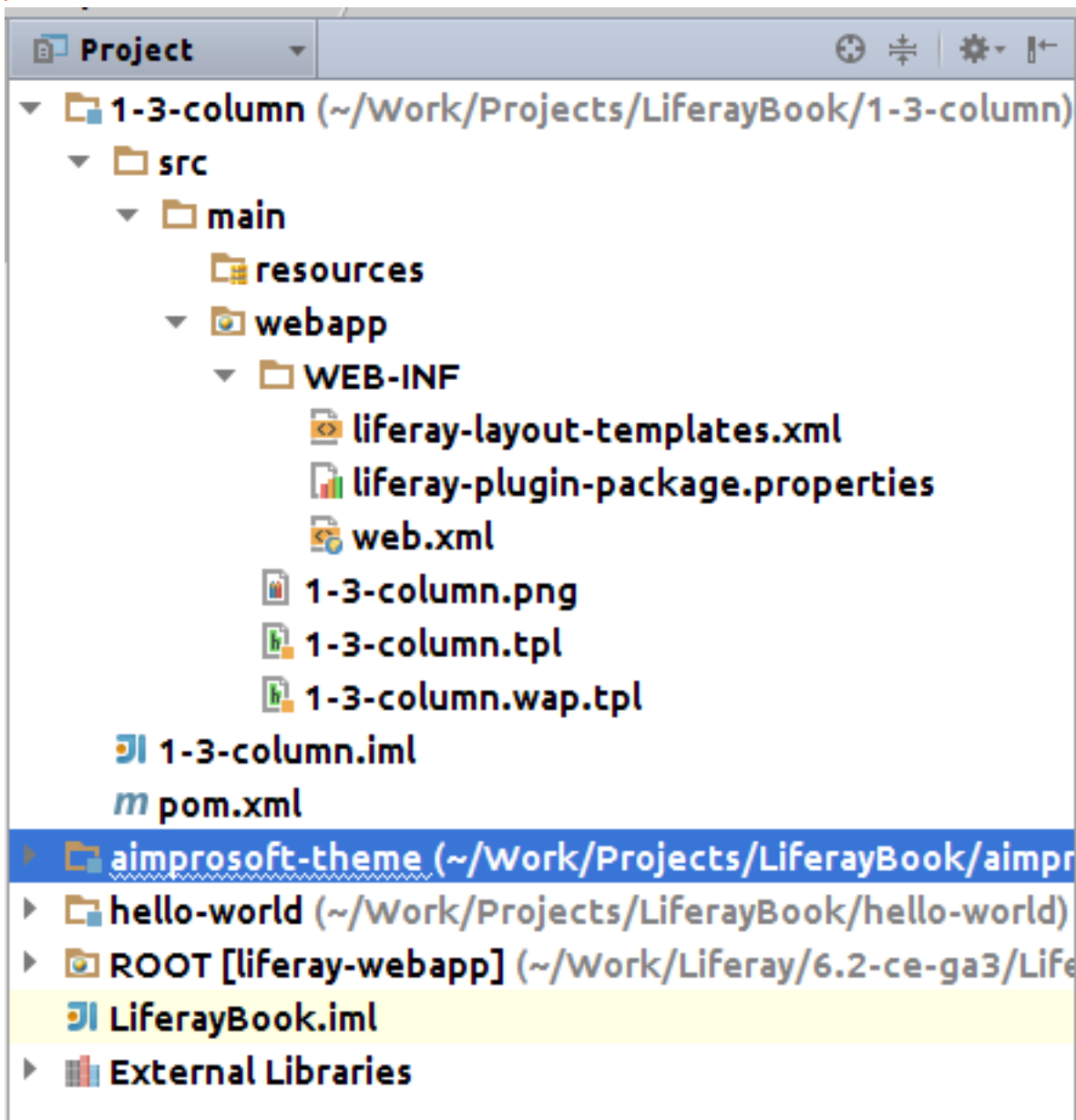
Perform all straightly..

Now we will write our first Liferay theme.

Similar to portlet and theme, we will create a new module for layout:



We get such module structure:



We will create 1-3-column layout, that has such structure:

We will do it on the basis of existing *1_2_columns_i* layout (*file/ROOT/layouttpl/custom/1_2_columns_i.tpl*). Copy content of this file in 1-3-column.tpl. Change layout in order to there in the bottom were three columns instead of two ones. Change layout so that in the bottom were three columns instead of two:

```

<div class="columns-1-3" id="main-content" role="main">
  <div class="portlet-layout row-fluid">
    <div class="portlet-column portlet-column-only span12" id="column-1">
      $processor.processColumn("column-1", "portlet-column-content portlet-column-content-only")
    </div>
  </div>

  <div class="portlet-layout row-fluid">
    <div class="portlet-column portlet-column-first span4" id="column-2">
      $processor.processColumn("column-2", "portlet-column-content portlet-column-content-first")
    </div>

    <div class="portlet-column portlet-column-first span4" id="column-3">
      $processor.processColumn("column-3", "portlet-column-content")
    </div>

    <div class="portlet-column portlet-column-last span4" id="column-4">
      $processor.processColumn("column-4", "portlet-column-content portlet-column-content-last")
    </div>
  </div>
</div>

```

The same is done for the file *1-3-column.wap.tpl*.

Recompile layout and apply to the required page:



Chapter 6. First Hook

My hands are doing hooks...

Hook is needed for changing the standart Liferay functionality.

Which hooks do exist?

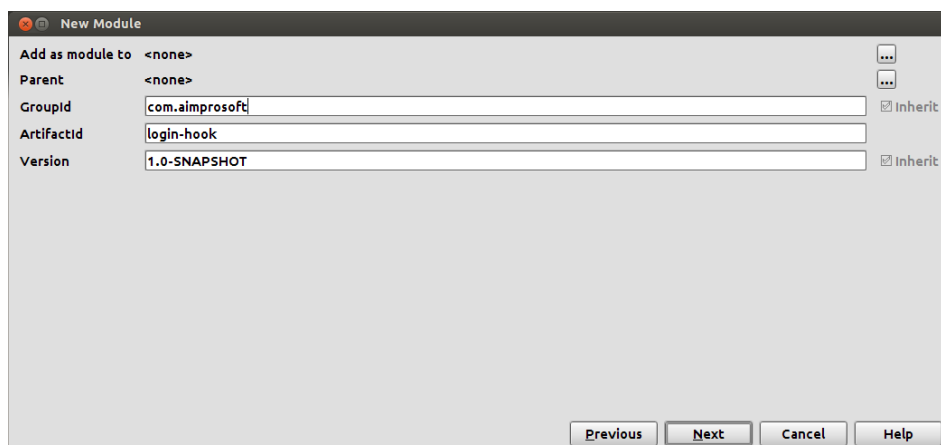
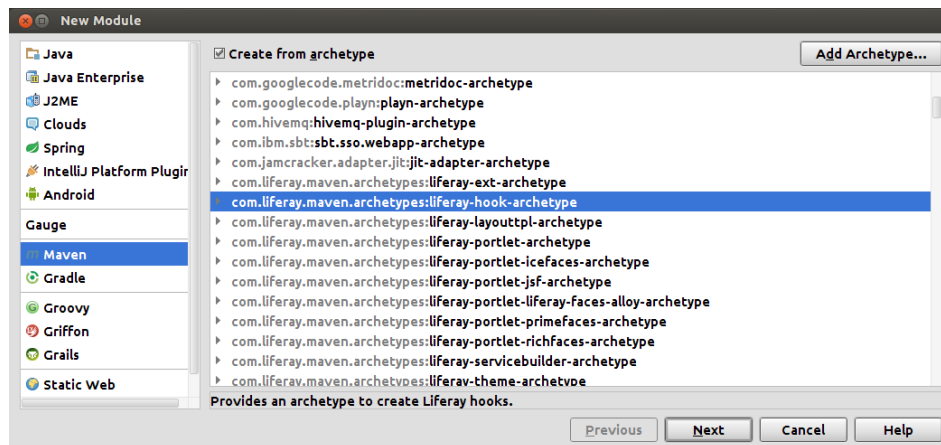
Types of hooks:

- on portal.properties - for redefinition properties of the portal (which are specified in /ROOT/WEB-INF/lib/portal-impl.jar!/portal.properties). Not all portal properties can be re-specified via hook, it is possible only for described in /ROOT/dtd/liferay-hook_6_2_0.dtd properties;
- on language-properties - for redefinition properties in Resource Bundle (/ROOT/WEB-INF/lib/portal-impl.jar!/content/Language.properties);
- on indexer (indexer-post-processor);
- on service;
- on struts action;
- on servlet-filter;
- on jsp page (jsp-hook) - it allows us to change Liferay jsp-page.

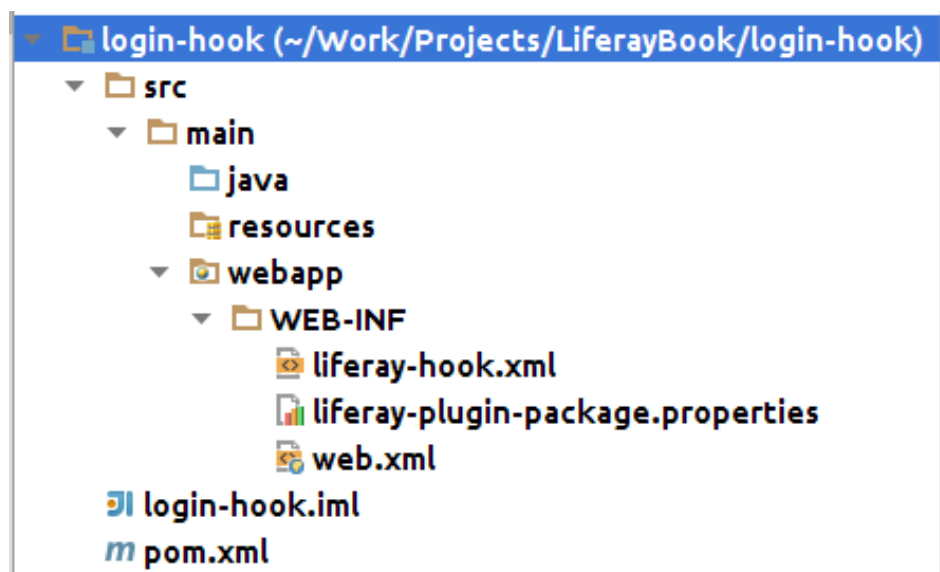
In this section we will create hook on jsp-page (out of login portlet).

Creating module for hook

As previously, we create a new module via *Maven archetype*:



We will get such structure:



Formulation of the problem

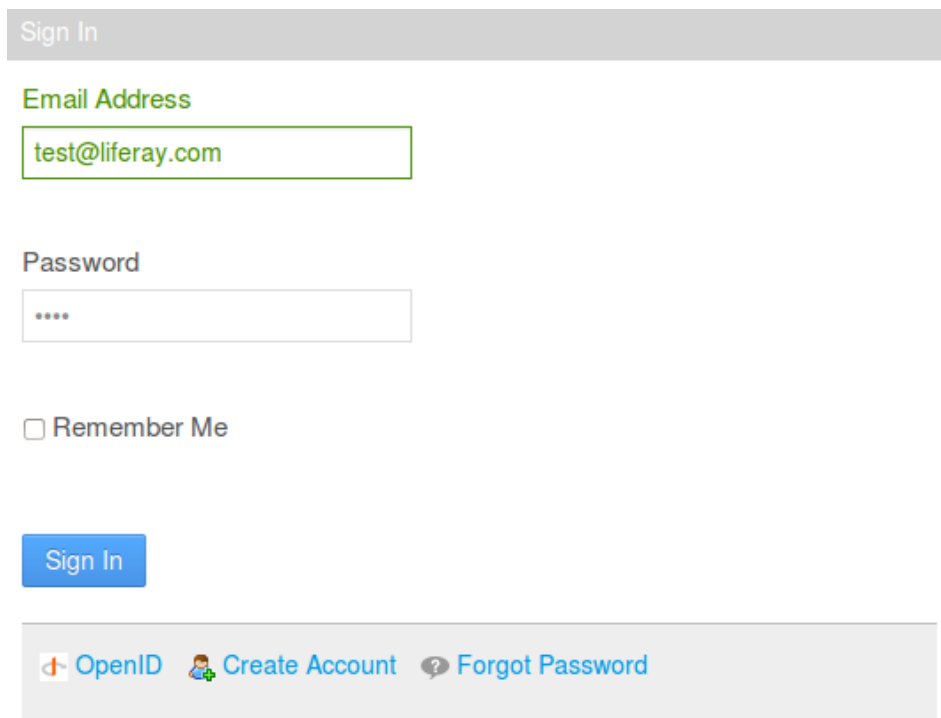
Suppose, we should display in bold '**Login to AimProSoft**' in the '*Sign In*' portlet, and then after login - '**Welcome to AimProSoft**'.

Defining JSP-page, for which hook should be installed

One of the main technology that is used in Liferay, is Struts.

Therefore, it is possible to define which *jsp* in which portlet is used via *struts action* (file / ROOT/WEB-INF/struts-config.xml).

To do it, go on *Login* portlet:



Sign In

Email Address




test@liferay.com

Password

....

☐ Remember Me

Sign In

 OpenID  Create Account  Forgot Password

In *FireBug* we examine URL forms.

```

<div id="yui_patched_v3_11_0_1_1427296338161_290" class="portlet-body">
  <form id="_58_fm" class="form sign-in-form"
    " autocomplete="on" name="_58_fm" method="post" action="http://localhost:
    8080/web/guest/welcome?p_p_id=58&p_p_lifecycle=1&p_p_state=normal&
    p_p_mode=view&p_p_col_id=column-1&p_p_col_pos=2&p_p_col_count=3&
    58_struts_action=%2Flogin%2Flogin">
  <div class="navigation">
</div>

```

As we see, one of the parameters in this URL is *struts_action=/login/login*.

We should find the appropriate action in *struts-config.xml*:

```

<action path="/login/login" type="com.liferay.portlet.login.action.LoginAction">
  <forward name="portlet.login.login" path="portlet.login.login" />
  <forward name="portlet.login.login_redirect" path="portlet.login.login_redirect" />
</action>

```

We examine *path* for *forward*:

path="portlet.login.login"

Using this *path* we should find the required *jsp* in file */ROOT/WEB-INF/tiles-defs.xml*:

```

<definition name="portlet.login.login" extends="portlet.login">
  <put name="portlet_content" value="/portlet/login/login.jsp" />
</definition>

```

That is, required *jsp* for hook - is */portlet/login/login.jsp*.

Creating hook on JSP

One of the main technology that is used in Liferay, is Struts.

Therefore, it is possible to define which *jsp* in which portlet is used via *struts action* (file / ROOT/WEB-INF/struts-config.xml).

To do it, go on *Login* portlet:

Sign In

Email Address

test@liferay.com

Password

....

☐ Remember Me

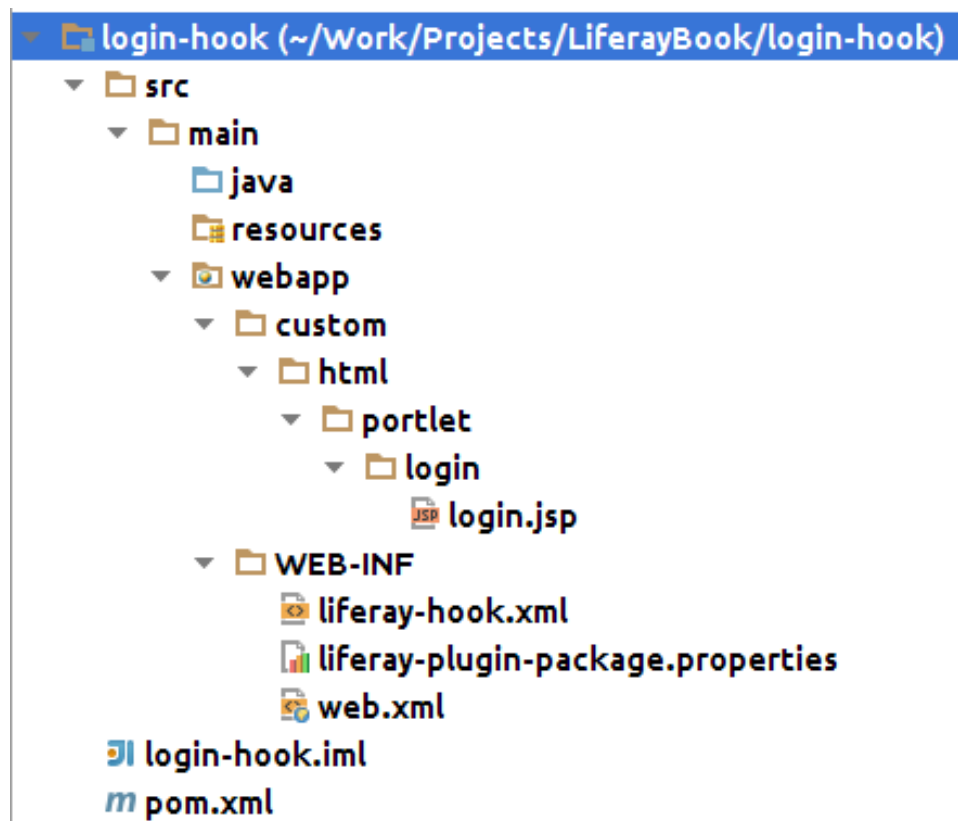
Sign In

OpenID
 Create Account
 Forgot Password

In liferay-hook.xml we register custom-jsp-dir:

```
<?xml version="1.0"?>
<!DOCTYPE hook PUBLIC "-//Liferay//DTD Hook 6.2.0//EN" "http://www.liferay.com/dtd/liferay-hook_6_2_0.dtd">
<hook>
  <custom-jsp-dir>/custom</custom-jsp-dir>
</hook>
```

We create the folder *custom* inside of *webapp*, and put into it /ROOT/html/portlet/login/login.jsp:



We change *jsp* according to requirements:

```
<c:choose>
  <c:when test="<%= themeDisplay.isSignedIn() %>">
    <%
      String signedInAs = HtmlUtil.escape(user.getFullName());

      if (themeDisplay.isShowMyAccountIcon() && (themeDisplay.getURLMyAccount() != null)) {
        String myAccountURL = String.valueOf(themeDisplay.getURLMyAccount());

        if (PropsValues.DOCKBAR_ADMINISTRATIVE_LINKS_SHOW_IN_POP_UP) {
          signedInAs = "<a class=\"signed-in\" href=\"javascript:Liferay.Util.openWindow({
        }
        else {
          myAccountURL = HttpUtil.setParameter(myAccountURL, "controlPanelCategory", Portl
          signedInAs = "<a class=\"signed-in\" href=\"\" + HtmlUtil.escape(myAccountURL) +
        }
      }
    %>
    <hl>Welcome to AimProSoft!!!</hl>
    <%= LanguageUtil.format(pageContext, "you-are-signed-in-as-x", signedInAs, false) %>
  </c:when>
  <c:otherwise>
    <hl>Login to AimProSoft</hl>
  </c:otherwise>
  <%
    String redirect = ParamUtil.getString(request, "redirect");

    String login = LoginUtil.getLogin(request, "login", company);
    String password = StringPool.BLANK;
    boolean rememberMe = ParamUtil.getBoolean(request, "rememberMe");
  %>
</c:choose>
```

Then we deploy the hook via *Maven* and check what we have as result:

Sign In




Login to AimProSoft

Email Address

Password

☒ Remember Me

Sign In

 OpenID  Create Account  Forgot Password

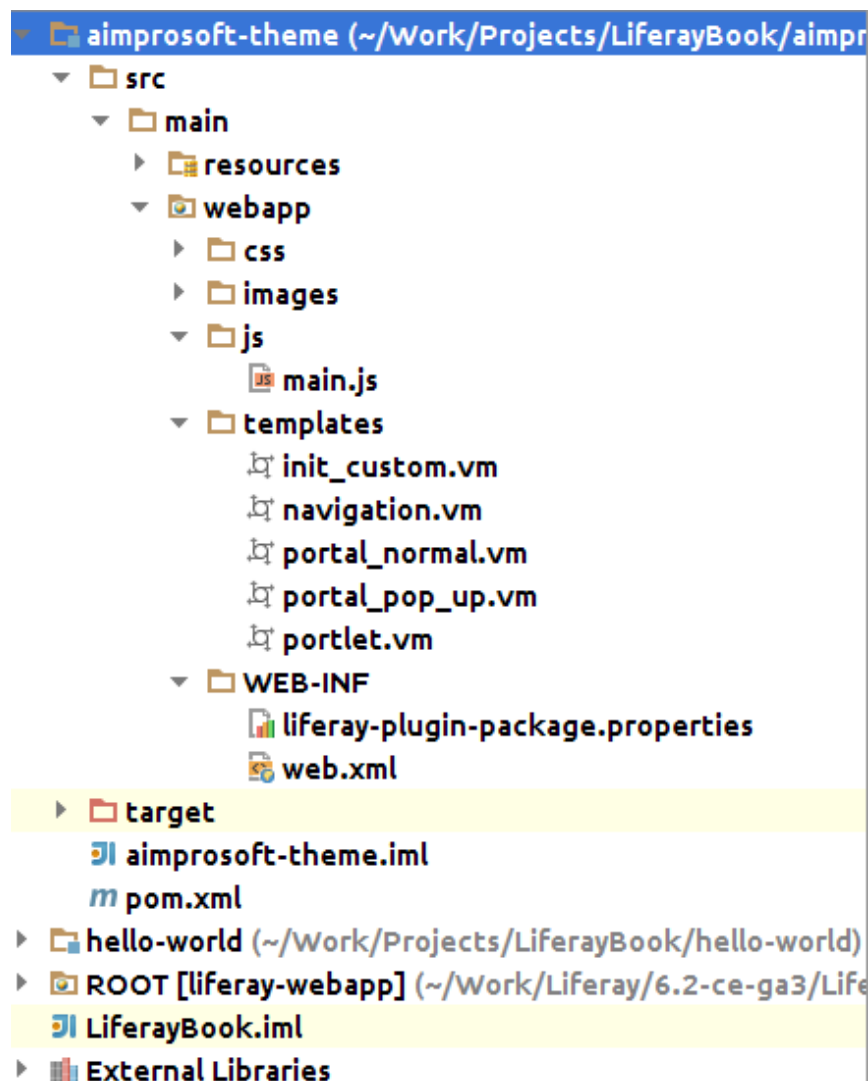
Chapter 7. More about Themes

Perform all more nicely...

In the fourth chapter we created our first Liferay-theme. Now we consider the themes development more detail.

Themes structure

After creating the theme we have the structure:



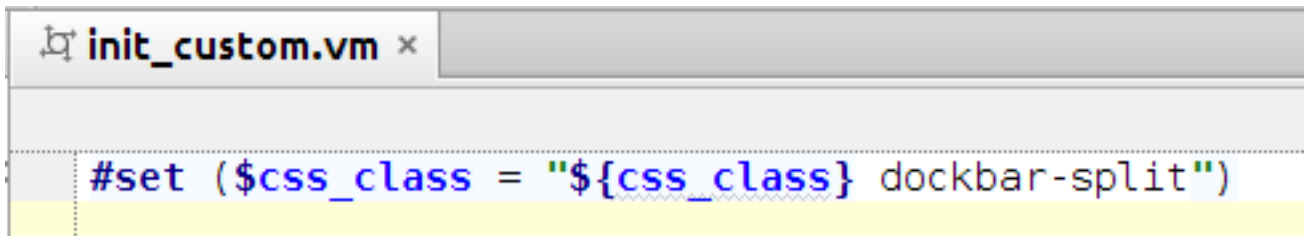
/css - css-files of the theme

/images - images, that are used in the theme

/js - js-files of the theme

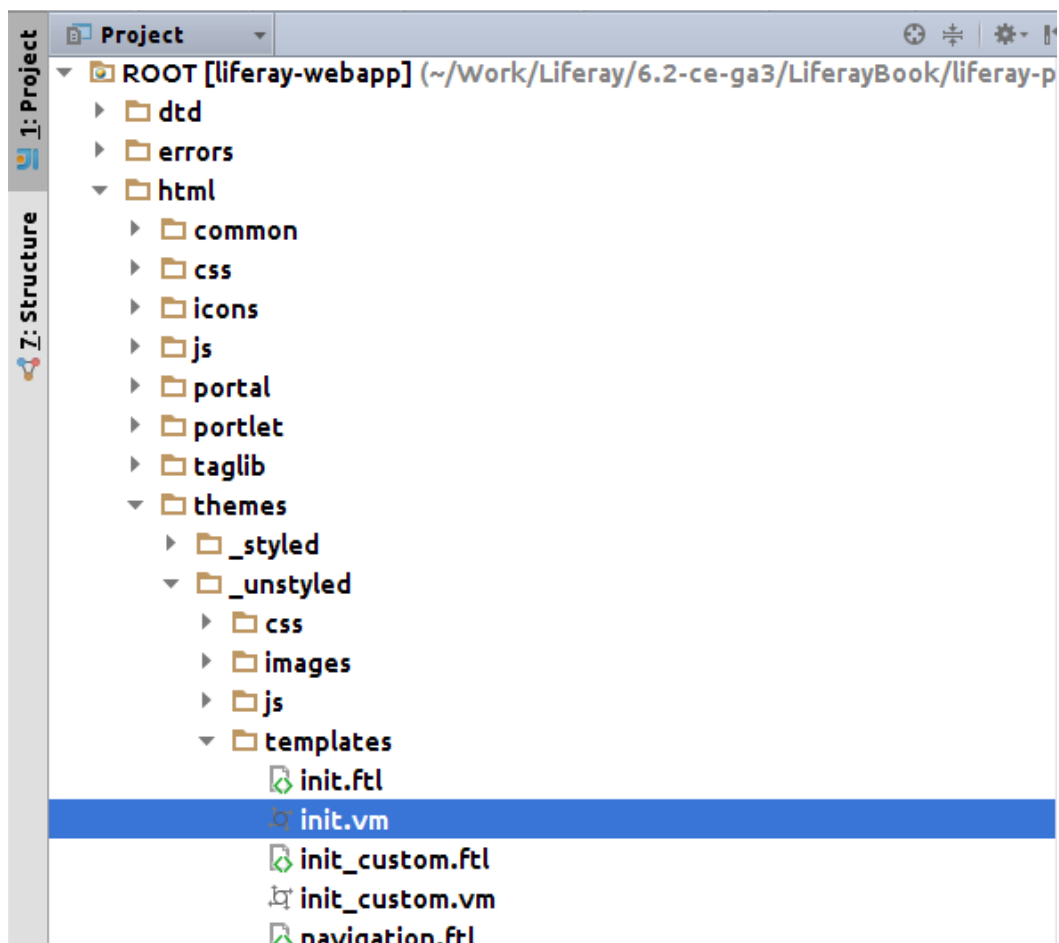
/templates - folder with templates for theme consists of such files:

- **init_custom.vm** - in this file custom velocity-variables are created, which then can be used in the theme template. Variables are specified via command **#set (\$name = value)**. Usually a name of a variable begins with the sign '\$'. In the classic theme we see how the value of the variable **\$css_class** is changing:



```
#set ($css_class = "${css_class} dockbar-split")
```

List of all predefined variables is located in the file **ROOT/html/themes/_unstyled/templates/init.vm**:



In this file we see the following:

```

init.vm x
## ----- Common variables ----- ##

#set ($theme_display = $themeDisplay)
#set ($portlet_display = $portletDisplay)

#set ($theme_timestamp = $themeDisplay.getTheme().getTimestamp())
#set ($theme_settings = $themeDisplay.getThemeSettings())

#set ($root_css_class = "au" + $languageUtil.get($locale, "lang.dir"))
#set ($css_class = $htmlUtil.escape($theme_display.getColorScheme().getCssClass()) + " yui3-skin-sam")

#set ($liferay_toggle_controls = $sessionClicks.get($request, "liferay_toggle_controls", "visible"))

#if ($layout)
    #set ($page_group = $layout.getGroup())

    #if ($page_group.isStagingGroup())
        #set ($css_class = $css_class + " staging local-staging")
    #elseif ($theme_display.isShowStagingIcon() && $page_group.hasStagingGroup())
        #set ($css_class = $css_class + " live-view")
    #elseif ($theme_display.isShowStagingIcon() && $page_group.isStagedRemotely())
        #set ($css_class = $css_class + " staging remote-staging")
    #end

    #if ($page_group.isControlPanel())
        #set ($liferay_toggle_controls = "visible")
    #end
#end

```

All variables which are asserted in this file, you can use in your themes.

- **navigation.vm** - in this file is defined the template for menu of the navigation in the theme. In the classic theme has already been created the navigation for pages and subpages of first level.

```

navigation.vm x
<nav class="$nav_css_class navbar site-navigation" id="navigation" role="navigation">
    <div class="navbar-inner">
        <div class="collapse nav-collapse">
            <ul aria-label="Language ('site-pages')" class="nav nav-collapse" role="menubar">
                #foreach ($nav_item in $nav_items)
                    #set ($nav_item_attr_selected="")
                    #set ($nav_item_attr_has_popup="")
                    #set ($nav_item_caret="")
                    #set ($nav_item_css_class="lfr-nav-item")
                    #set ($nav_item_link_css_class="")

                    #if ($nav_item.isSelected())
                        #set ($nav_item_attr_selected="aria-selected='true'")
                        #set ($nav_item_css_class="$nav_item_css_class selected active")
                    #end

                    #if ($nav_item.hasChildren())
                        #set ($nav_item_attr_has_popup="aria-haspopup='true'")
                        #set ($nav_item_caret="<span class='lfr-nav-child-toggle'><i class='icon-caret-down'></i></span>")
                        #set ($nav_item_css_class="$nav_item_css_class dropdown")
                        #set ($nav_item_link_css_class="dropdown-toggle")
                    #end

                    <li class="$nav_item_css_class" id="layout_$nav_item.getLayoutId()" $nav_item_attr_selected role="presentation">
                        <a aria-labelledby="layout_$nav_item.getLayoutId()" $nav_item_attr_has_popup class="$nav_item_link_css_class" href=
                        <span>$nav_item.icon() $nav_item.getName() $nav_item_caret</span>
                        </a>

                        #if ($nav_item.hasChildren())
                            <ul class="dropdown-menu child-menu" role="menu">
                                #foreach ($nav_child in $nav_item.getChildren())
                                    #set ($nav_child_attr_selected="")
                                    #set ($nav_child_css_class="lfr-nav-item")

                                    #if ($nav_child.isSelected())
                                        #set ($nav_child_attr_selected="aria-selected='true'")
                                        #set ($nav_child_css_class="selected")
                                    #end

                                    <li class="$nav_child_css_class" id="layout_$nav_child.getLayoutId()" $nav_child_attr_selected role="pr
                                    <a aria-labelledby="layout_$nav_child.getLayoutId()" href="$nav_child.getURL()" $nav_child.getTarge
                                    </li>
                                #end
                            </ul>
                        #end
                    </li>
                #end
            </ul>
        </div>
    </div>
</nav>

```

It is possible to use this file as a basic for navigation development in our own themes.

- **portal_normal.vm** - in this file is created a template of the theme. If we open a template for classic theme, we will see that it has the following structure:

```
<body class="$css_class">  
  <a href="#main-content" id="skip-to-content">#language ("skip-to-content")</a>  
  $theme.include($body_top_include)  
  
  #dockbar() ← Докбар  
  
  <div class="container-fluid" id="wrapper">  
    <header id="banner" role="banner">  
      <div id="heading"...>  
        <if ($has_navigation || $is_signed_in)>  
          #parse ("{$full_templates_path/navigation.vm}") ← Проверка,  
                                                         залогинен ли  
                                                         пользователь  
          <end>                                     Навигация  
        </if>  
      </div>  
    </header>  
  
    <div id="content">  
      <nav id="breadcrumbs">#breadcrumbs()</nav> ← Бредкрамб  
  
      <if ($selectable)>  
        $theme.include($content_include)  
      <else>  
        $portletDisplay.recycle()  
  
        $portletDisplay.setTitle($the_title) ← Портлеты  
  
        $theme.wrapPortlet("portlet.vm", $content_include)  
      </if>  
    </div>  
  
    <footer id="footer" role="contentinfo">  
      <p class="powered-by">  
        #language ("powered-by") <a href="http://www.liferay.com" rel="external">Liferay</a>  
      </p>  
    </footer>  
  </div>  
  
  $theme.include($body_bottom_include)  
  
  $theme.include($bottom_include)  

```

Dockbar is displayed above, then *wrapper* follows, consisting of header, content and footer. The logotype, site name and navigation are placed in the header (only for logged users). Breadcrumb and the portlets are displayed in the content. The standard caption ‘**Powered By Liferay**’ is displayed in the footer.

- **portal_pop_up.vm** - template for pop-up;
- **portlet.vm** - template for portlet has such structure.

```
#set ($portlet_display = $portletDisplay)

#set ($portlet_id = $htmlUtil.escapeAttribute($portlet_display.getId()))
#set ($portlet_title = $htmlUtil.escape($portlet_display.getTitle()))
#set ($portlet_back_url = $htmlUtil.escapeHref($portlet_display.getURLBack()))

<section class="portlet" id="portlet_$portlet_id">
  <header class="portlet-topper">
    <h1 class="portlet-title">
      $theme.portletIconPortlet() <span class="portlet-title-text">$portlet_title</span>
    </h1>

    <menu class="portlet-topper-toolbar" id="portlet-topper-toolbar_$portlet_id" type="toolbar">
      #if ($portlet_display.isShowBackIcon())
        <a class="portlet-icon-back" href="$portlet_back_url">#language ("return-to-full-page")</a>
      #else
        $theme.portletIconOptions()
      #end
    </menu>
  </header>

  <div class="portlet-content">
    $portlet_display.writeContent($writer)
  </div>
</section>
```

Portlets are displayed on the portal page according to this template.

/WEB-INF - folder with the configuration files

- **liferay-plugin-package.properties** - description properties of the theme;
- **web.xml** - deployment descriptor.

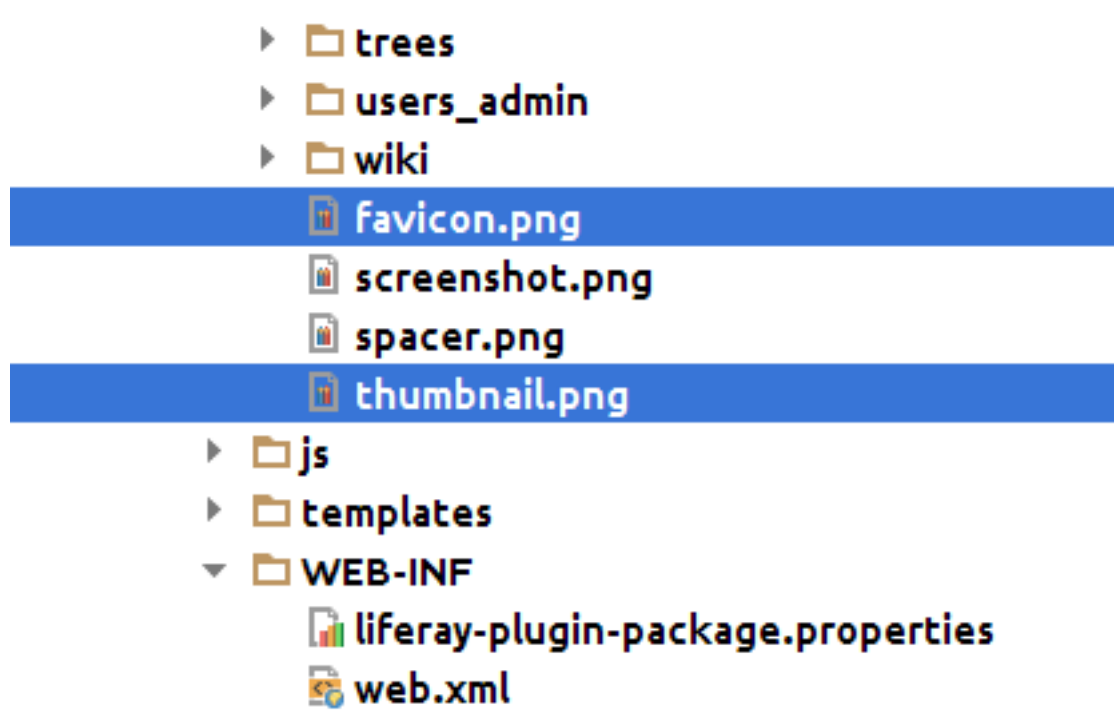
Creating own theme

After getting acquainted with theme structure, we create own theme '**aimprosoft-theme**' on basis of classic theme:

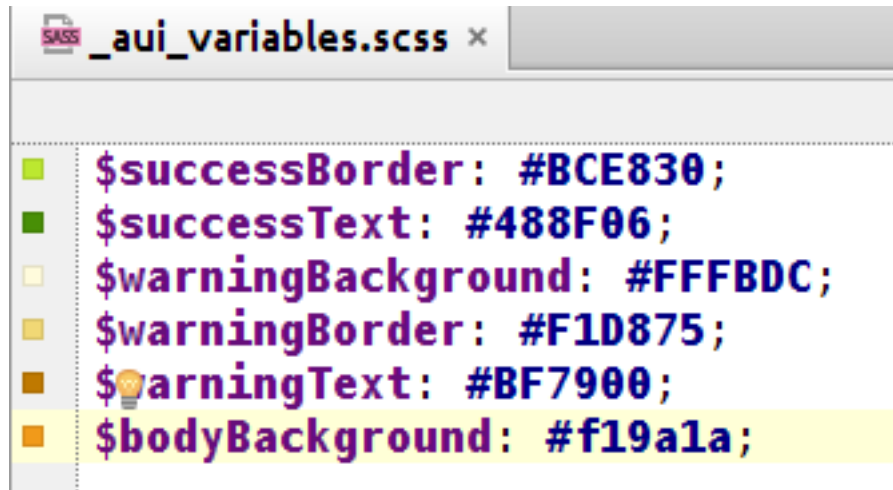
- 1) Change standard inscription '**Powered By Liferay**' to '**Powered By AimProSoft**' in *portal_normal.vm*

```
<footer id="footer" role="contentinfo">
  <p class="powered-by">
    #language ("powered-by") <a href="http://www.aimprosoft.com" rel="external">AimProSoft</a>
  </p>
</footer>
```

- 2) Replace *favicon.png* and *thumbnail.png* files on logotype of our theme:



- 3) Change the color of the background. To do it, add the variable *\$bodyBackground*: *#f19a1a*; in the file *css/_aui_variables.css*:

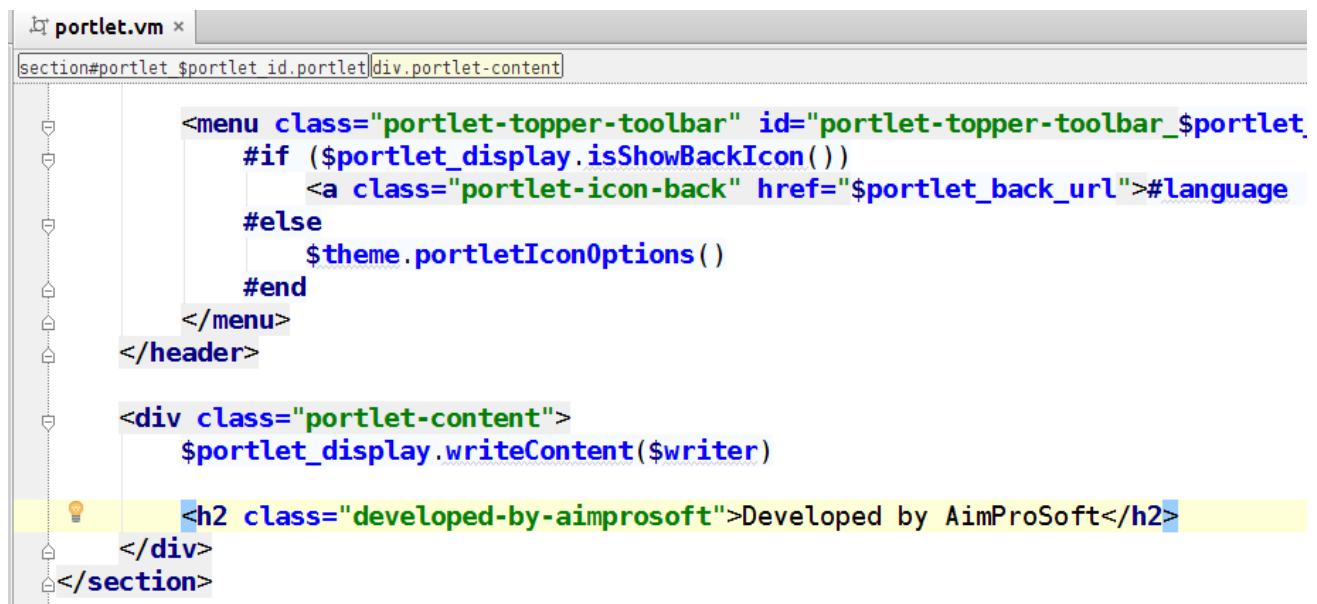


```

$successBorder: #BCE830;
$successText: #488F06;
$warningBackground: #FFFBDC;
$warningBorder: #F1D875;
$warningText: #BF7900;
$bodyBackground: #f19a1a;

```

4) We do so that the inscription 'Developed by AimProSoft' will be displayed in each portlet. To do that, we change the file *templates/portlet.vm*:



```

<menu class="portlet-topper-toolbar" id="portlet-topper-toolbar_$portlet
  #if ($portlet_display.isShowBackIcon())
    <a class="portlet-icon-back" href="$portlet_back_url">#language
  #else
    $theme.portletIconOptions()
  #end
</menu>
</header>

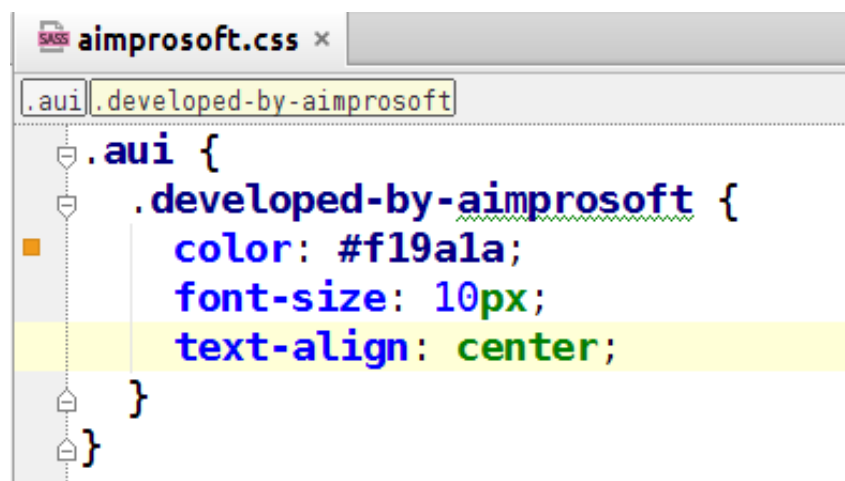
<div class="portlet-content">
  $portlet_display.writeContent($writer)

  <h2 class="developed-by-aimprosoft">Developed by AimProSoft</h2>
</div>
</section>

```

5) Assign our own styles for the theme.

Add the new file *aimprosoft.css*:

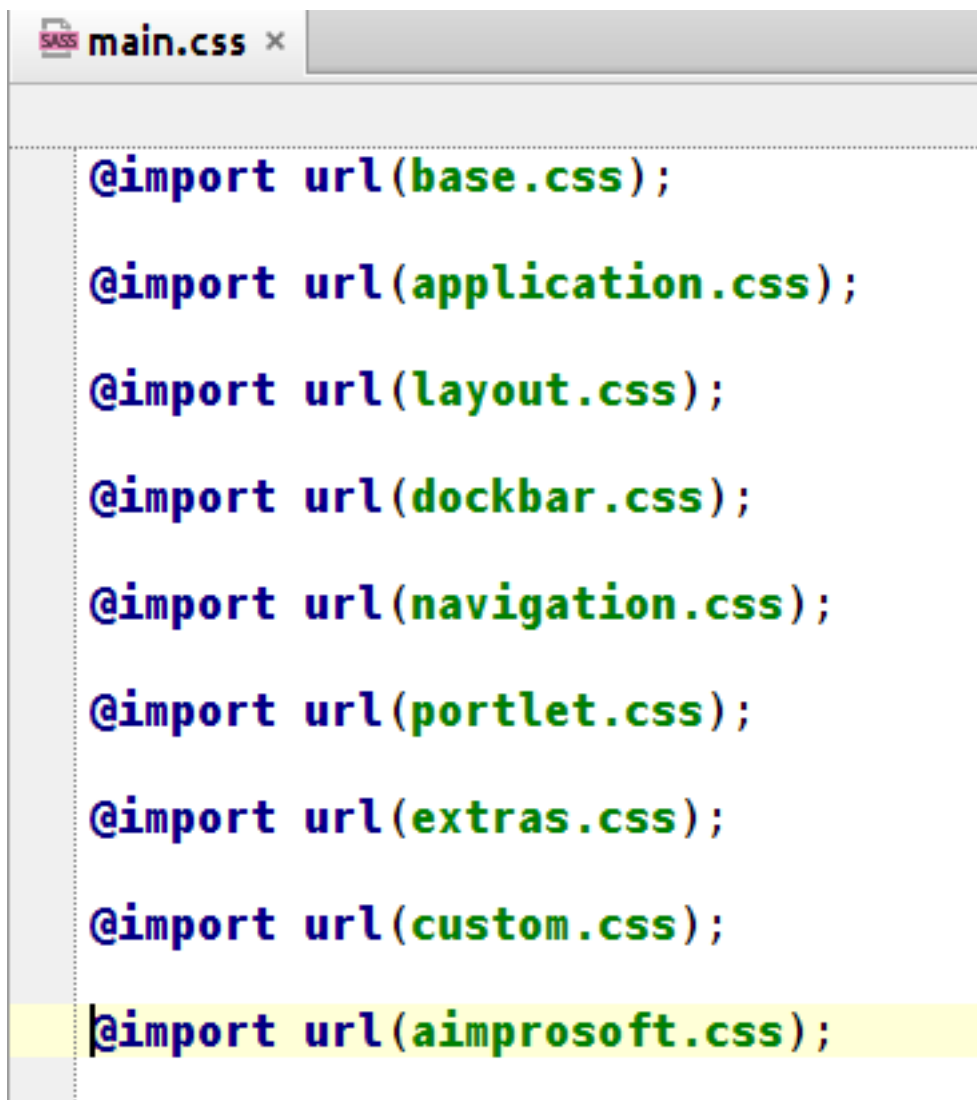


```

.aui {
  .developed-by-aimprosoft {
    color: #f19a1a;
    font-size: 10px;
    text-align: center;
  }
}

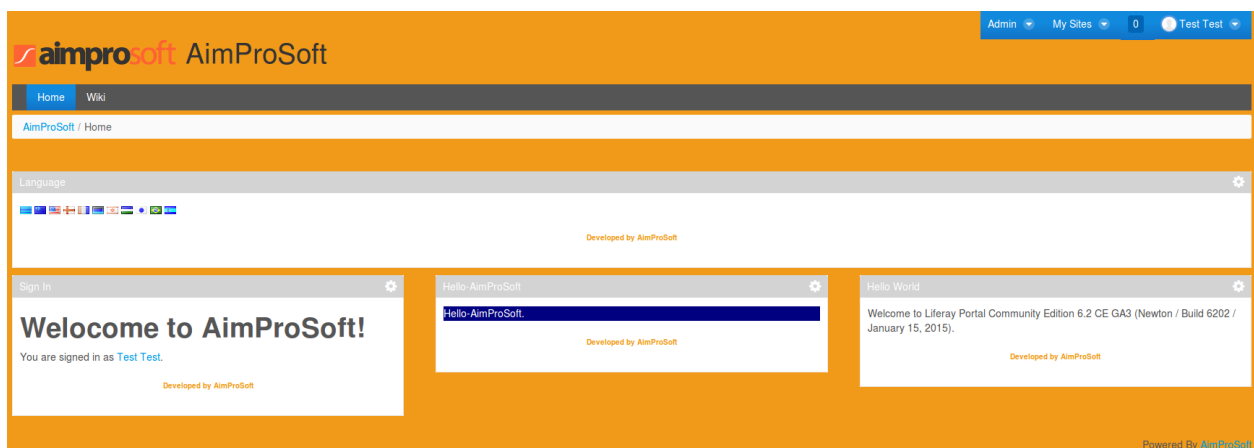
```

and register it in *main.css*:



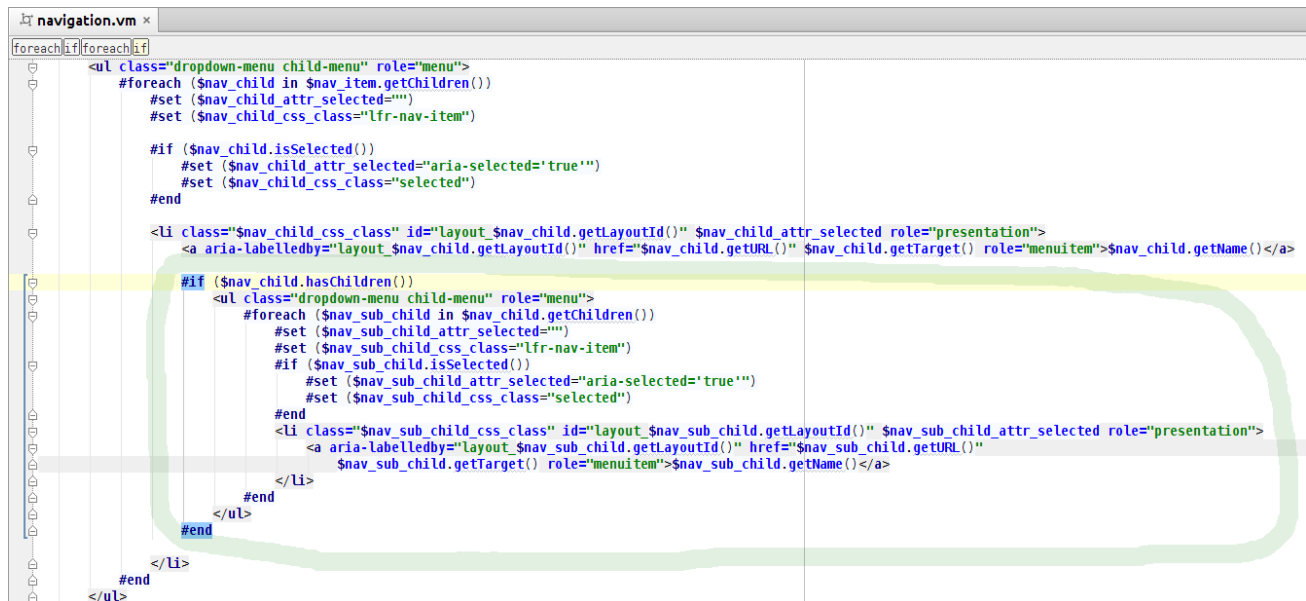
```
main.css x
@import url(base.css);
@import url(application.css);
@import url(layout.css);
@import url(dockbar.css);
@import url(navigation.css);
@import url(portlet.css);
@import url(extras.css);
@import url(custom.css);
@import url(aimprosoft.css);
```

Deploy the theme and look what is the result:



Changing navigation

In the classic theme the navigation displays the pages and subpages of the first level. We do so, that the subpages of the second level will be displayed in the our theme. To do that, edit the file *navigation.vm* and add there the nested iteration for passing the subpages of the second level (similarly the iteration for passing the subpages of the first level):



```

navigation.vm
foreach ($nav_child in $nav_item.getChildren())
  #set ($nav_child_attr_selected='')
  #set ($nav_child_css_class='lfr-nav-item')

  #if ($nav_child.isSelected())
    #set ($nav_child_attr_selected='aria-selected="true"')
    #set ($nav_child_css_class='selected')
  #end

  <li class="$nav_child_css_class" id="layout_$nav_child.getLayoutId()" $nav_child_attr_selected role="presentation">
    <a aria-labelledby="layout_$nav_child.getLayoutId()" href="$nav_child.getURL()" $nav_child.getTarget() role="menuitem">$nav_child.getName()</a>

    #if ($nav_child.hasChildren())
      <ul class="dropdown-menu child-menu" role="menu">
        #foreach ($nav_sub_child in $nav_child.getChildren())
          #set ($nav_sub_child_attr_selected='')
          #set ($nav_sub_child_css_class='lfr-nav-item')
          #if ($nav_sub_child.isSelected())
            #set ($nav_sub_child_attr_selected='aria-selected="true"')
            #set ($nav_sub_child_css_class='selected')
          #end
          <li class="$nav_sub_child_css_class" id="layout_$nav_sub_child.getLayoutId()" $nav_sub_child_attr_selected role="presentation">
            <a aria-labelledby="layout_$nav_sub_child.getLayoutId()" href="$nav_sub_child.getURL()" $nav_sub_child.getTarget() role="menuitem">$nav_sub_child.getName()</a>
          </li>
        #end
      </ul>
    #end
  </li>
#end
</ul>

```

Add also the css-styles for subpages of the second level:



```

aimprosoft.css
.aul {
  .developed-by-aimprosoft {
    color: #f19a1a;
    font-size: 10px;
    text-align: center;
  }

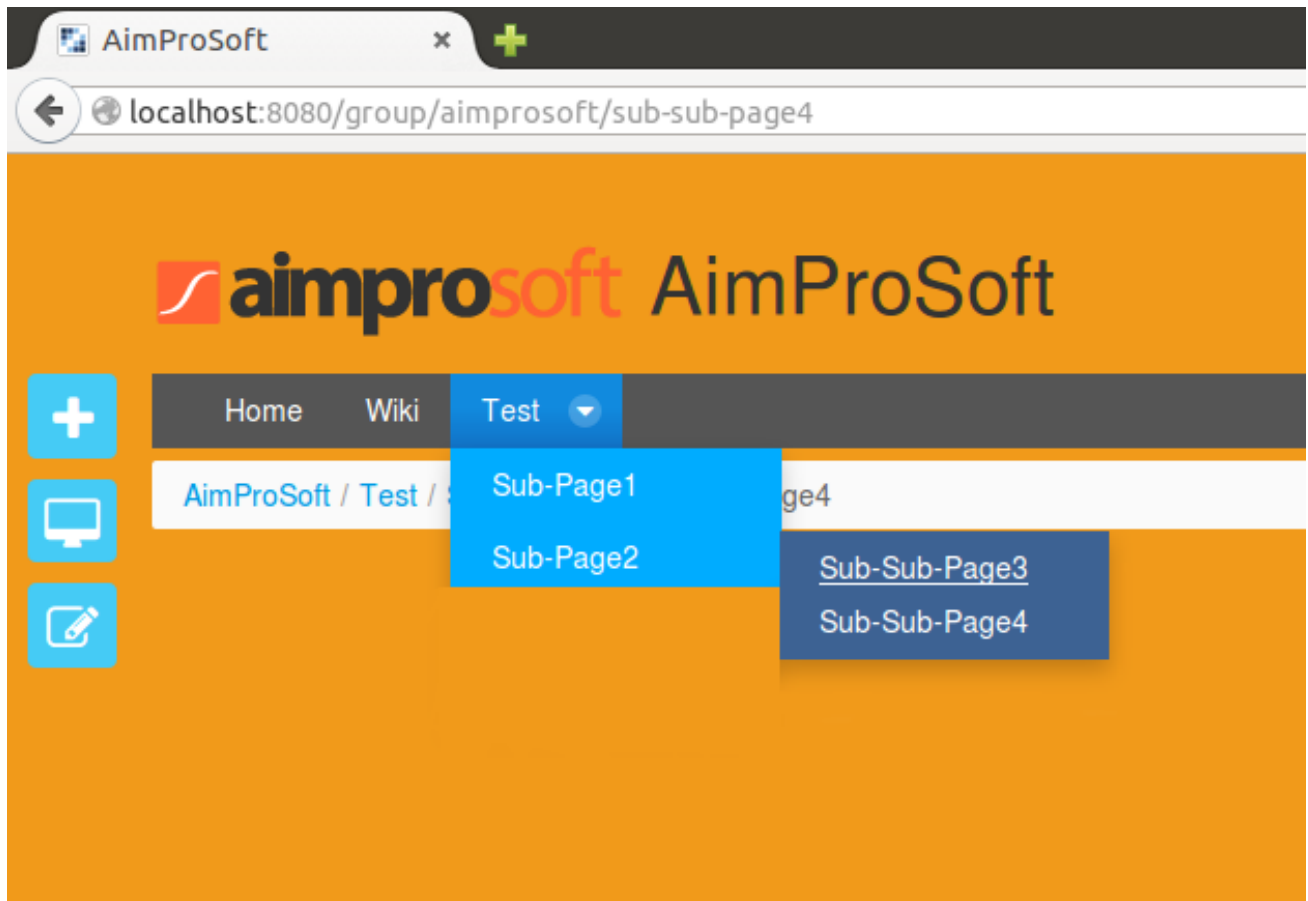
  #navigation {
    ul.dropdown-menu {
      li {
        clear: both;
        display: block;
        margin: 0;
        padding: 0;
        position: relative;

        a {
          color: #fff;
          display: block;
          width: 100%;
        }

        &:hover {
          ul {
            background-color: #3d6293;
            display: block;
            position: absolute;
            left: 100%;
            top: 0;
          }
        }
      }
    }
  }
}

```

Add some pages/subpages and see what we have as a result:



Embedding the portlets into the theme

The portlets can be added to a page in a conventional manner, or can be embedded into a theme. In this case, the portlet will be displayed on the pages of such theme by default.

The portlet is embedded into the theme via the command:

```
$theme.runtime("PORTLET_ID"),
```

where *PORTLET_ID* – is **id** of the necessary portlet.

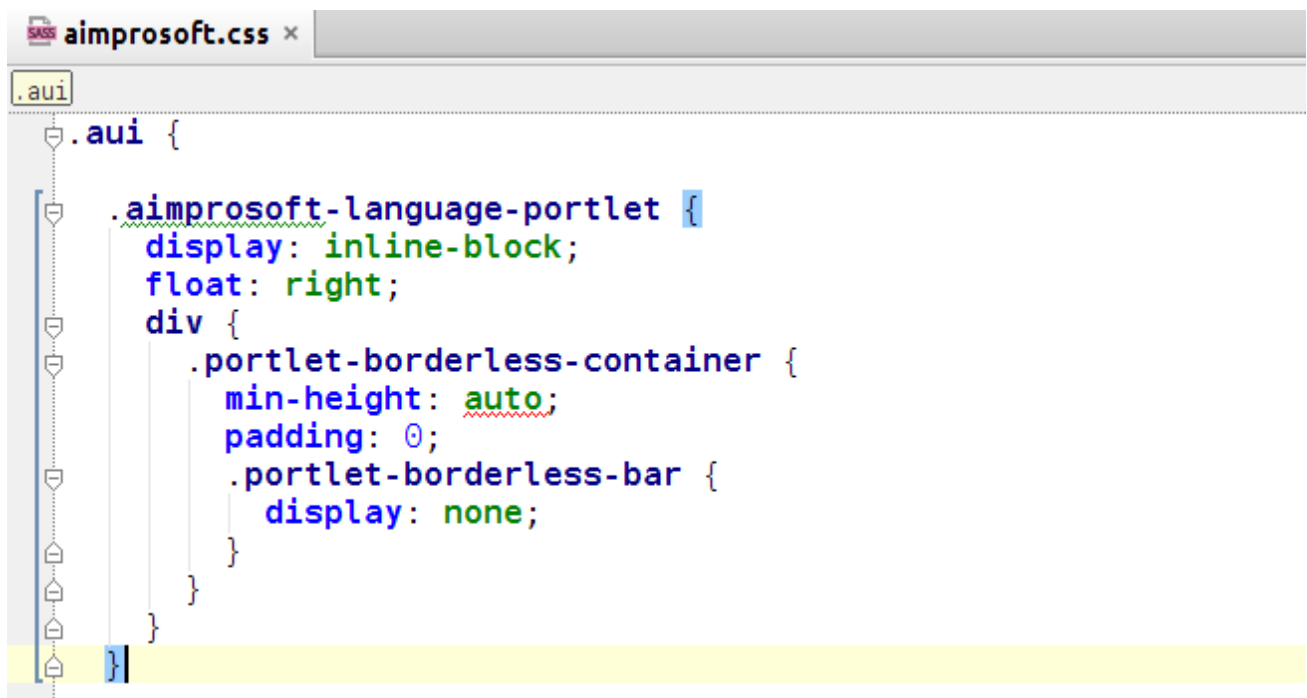
We add to our theme the *Language* portlet to be able to change the language (do it in *portal_normal.vm*):

```
<div class="container-fluid" id="wrapper">
  <header id="banner" role="banner">
    <div id="heading">
      <h1 class="site-title">
        <a class="$logo_css_class" href="$site_default_url" title="#language">
          
          #end
        </a>
      </h1>
    </div>
    <div class="aimprosoft-language-portlet">
      $theme.runtime("82")
    </div>
  </div>
```

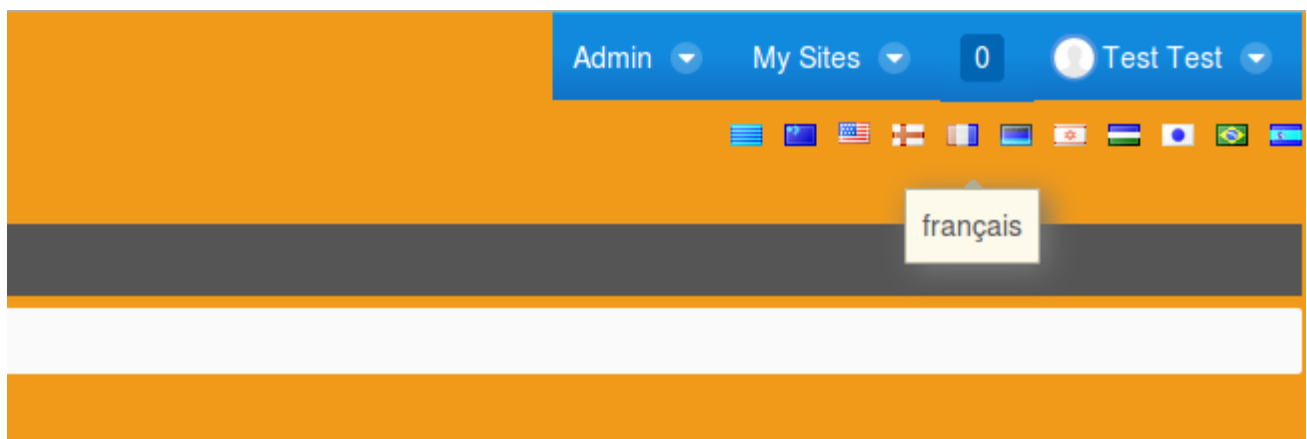
If we recompiling the theme, we will see that the *Language* portlet will be added to the header of the theme (but will be displayed with the title/borders and it is not beautifully). To fix this, remove the **portletSetupShowBorders** via **\$velocityPortletPreferences**:

```
<div class="aimprosoft-language-portlet">
  $velocityPortletPreferences.setValue("portletSetupShowBorders", "false")
  $theme.runtime("82", "", $velocityPortletPreferences.toString())
  $velocityPortletPreferences.reset()
</div>
```

Also assign the styles for this portlet:



After redeploying of the theme we will see that *Language* portlet is displayed normally on the right in the header:



Now on the pages of our theme we can easily change the language.

Similarly to this, you can add any other portlet. **ID** of required portlet can be defined in the interface `com.liferay.portal.util.PortletKeys`:

```
PortletKeys.java ×
package com.liferay.portal.util;

/**
 * @author Brian Wing Shun Chan
 */
public class PortletKeys {

    public static final String ACTIVITIES = "116";

    public static final String ADMIN = "9";

    public static final String ADMIN_INSTANCE = "135";

    public static final String ADMIN_PLUGINS = "136";

    public static final String ADMIN_SERVER = "137";

    public static final String ALERTS = "83";

    public static final String ANNOUNCEMENTS = "84";

    public static final String ASSET_BROWSER = "172";

    public static final String ASSET_CATEGORIES_NAVIGATION = "122";

    public static final String ASSET_PUBLISHER = "101";

    public static final String ASSET_TAGS_NAVIGATION = "141";

    public static final String BACKGROUND_TASK = "189";

    public static final String BLOGS = "33";
```

Using VELOCITY-macros

Now we consider the use of [velocity-macros](#) in the template of a theme.

What is the VELOCITY-macro?

Velocity-macro admits the creation of the iterative fragment of the template, that then can be reused. Also, it can take one or more parameters that can be used in this fragment of the template. Macro in *Velocity* is created by using the command:

```
#macro(macros_name param1 ... paramN)
MACROS_BODY
#end
```

Then it can be called as:

```
#macros_name(arg1 ... argN)
```

The parameters (and, accordingly, the arguments) can be absent.

How velocity-macros are used in Liferay?

During the development the template of the theme we have already used the macros that was created in *Liferay* by default, for example:

1) dockbar:

```
$theme.include($body_top_include)
#dockbar()
<div class="container-fluid" id="wrapper">
```

2) breadcrumb:

```
<div id="content">
  <nav id="breadcrumbs">#breadcrumbs()</nav>
```


These macros have been declared in the file /ROOT/WEB-INF/lib/portal-impl.jar/

VM_liferay.vm:

```

VM_liferay.vm
macro 'dockbar'

#macro (breadcrumbs $control_panel)
    #set ($breadcrumb_tag = $theme.getBreadcrumbTag())

    #if ($control_panel == "control_panel")
        $breadcrumb_tag.setShowGuestGroup(false)
        $breadcrumb_tag.setShowParentGroups(false)
    #end

    $breadcrumb_tag.runTag()
#end

#macro (dockbar)
    $theme.runtime("145")
#end

#macro (silently $foo)
    #set ($foo = $foo)
#end

```

and can be used during the theme development.

Development of the own velocity-macro

Earlier in this chapter we added to this theme *Language* portlet. Now we do it with the help of the velocity-macro.

```

/* Navigation Macro */
#macro(navigation_macro $nav_elem)

    #set ($nav_elem_attr_selected="")
    #set ($nav_elem_css_class="lfr-nav-item")

    #if ($nav_elem.isSelected())
        #set ($nav_elem_attr_selected="aria-selected='true'")
        #set ($nav_elem_css_class="selected")
    #end

    <li class="$nav_elem_css_class" id="layout_$nav_elem.getLayoutId()"
        $nav_elem_attr_selected role="presentation">
        <a aria-labelledby="layout_$nav_elem.getLayoutId()" href="$nav_elem.getURL()"
            $nav_elem.getTarget() role="menuItem">$nav_elem.getName()</a>
        #if ($nav_elem.hasChildren())
            <ul class="dropdown-menu child-menu" role="menu">
                #foreach ($nav_elem_child in $nav_elem.getChildren())
                    #navigation_macro($nav_elem_child)
                #end
            </ul>
        #end
    </li>
#end

```

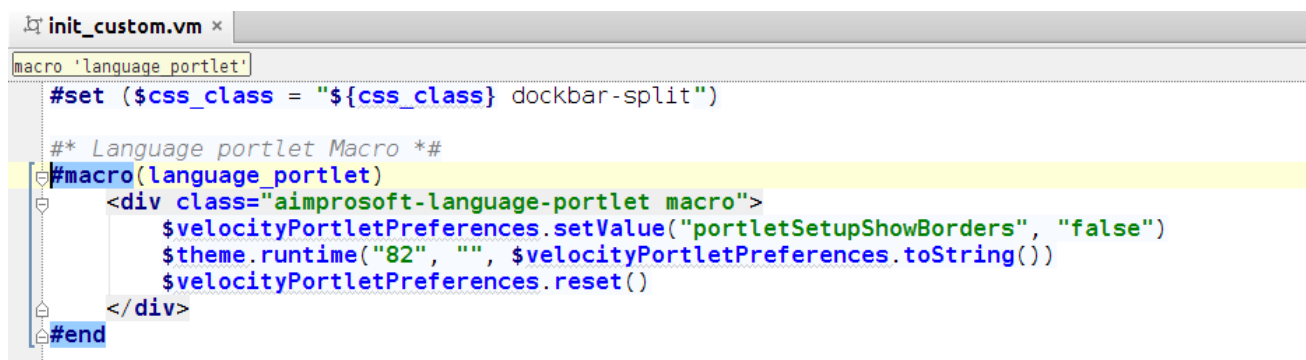


```

portal_normal.vm x
language portlet
<div id="heading">
  <h1 class="site-title">
    <a class="$logo_css_class" href="$site_default_url" title="#langua
      
        #end
      </a>
    #language_portlet()
  </h1>

```

We take out the code for displaying *Language* portlet into separate macro (call it 'language_portlet') in *init_custom.vm*:



```

init_custom.vm x
macro 'language portlet'
  #set ($css_class = "${css_class} dockbar-split")
  /* Language portlet Macro */
  #macro(language_portlet)
    <div class="aimprosoft-language-portlet macro">
      $velocityPortletPreferences.setValue("portletSetupShowBorders", "false")
      $theme.runtime("82", "", $velocityPortletPreferences.toString())
      $velocityPortletPreferences.reset()
    </div>
  #end

```

Now we can use the created macro in the template of the theme:

As we can see, after creating macro for *Language* portlet, we can use it just as well as *#dockbar()* or *#breadcrumbs()*. It reduces code of theme template and admits reuse it.

More elaborate Velocity-macro

Now we develop more elaborate Velocity-macro. Earlier in this chapter we have changed the navigation menu in the theme so that it displays the pages of second level. We did this by using a nested cycle. But suppose that it will need to display the pages of the third, fourth level, etc. Then we have to write a nested cycle for each level of nesting. Not to do so, it is possible to use a [recursion](#). Similarly to the [recursion on jsp](#), it is possible to develop a velocity-macro, that will take the current page as a parameter, display the current level of the drop-down menu, and call itself for each of the subpages. Thus, the macro will call itself and display the next level of the menu as long as the subpages will be there. Add the macro for the navigation in *init_custom.vm*:

As we can see, the macro takes the parameter `$nav_elem` and calls itself for each of `$nav_elem_child`.

Now we use this macro in the navigation:

```

navigation.vm <
foreach($nav_elem)foreach($nav_elem_child)
<nav class="$nav_css_class navbar site-navigation" id="navigation" role="navigation">
  <div class="navbar-inner">
    <div class="collapse nav-collapse">
      <ul aria-label="#language ('site-pages')" class="nav nav-collapse" role="menubar">
        #foreach ($nav_item in $nav_items)
          #set ($nav_item_attr_selected='')
          #set ($nav_item_attr_has_popup='')
          #set ($nav_item_caret='')
          #set ($nav_item_css_class="lfr-nav-item")
          #set ($nav_item_link_css_class='')

          #if ($nav_item.isSelected())
            #set ($nav_item_attr_selected="aria-selected='true'")
            #set ($nav_item_css_class="$nav_item_css_class selected active")
          #end

          #if ($nav_item.hasChildren())
            #set ($nav_item_attr_has_popup="aria-haspopup='true'")
            #set ($nav_item_caret='<span class="lfr-nav-child-toggle"><i class="icon-caret-down"></i></span>')
            #set ($nav_item_css_class="$nav_item_css_class dropdown")
            #set ($nav_item_link_css_class="dropdown-toggle")
          #end

          <li class="$nav_item_css_class" id="layout_$nav_item.getLayoutId()" $nav_item_attr_selected role="
            <a aria-labelledby="layout_$nav_item.getLayoutId()" $nav_item_attr_has_popup class="$nav_item_
              <span>$nav_item.icon() $nav_item.getName() $nav_item_caret</span>
            </a>

            #if ($nav_item.hasChildren())
              <ul class="dropdown-menu child-menu" role="menu">
                #foreach ($nav_child in $nav_item.getChildren())
                  #navigation_macro($nav_child)
                #end
              </ul>
            #end

          </li>
        #end
      </ul>
    </div>
  </div>
</nav>

```

Also, we fix up the css-styles for navigation:

```

aimprosoft.css <
.navigation {
  ul.dropdown-menu {
    li {
      clear: both;
      display: block;
      margin: 0;
      padding: 0;
      position: relative;

      a {
        color: #fff;
        display: block;
        width: 100%;
        padding: 2px 7px;
      }

      &:hover {
        > ul {
          background-color: #009ae5;
          display: block;
          position: absolute;
          left: 100%;
          top: 0;
          margin: 0;
          padding: 0;
          li {
            a {
              padding: 2px 7px;
            }
          }
        }
      }
    }
  }
}

```

Add subpages of the several levels of nesting:

Site Pages

Your request completed successfully.


Public Pages Private Pages


- Private Pages
 - Home
- Services
 - Software Development
- Liferay Development
 - Themes Development
 - Portlets Development
- Liferay Customization
 - Hooks
 - EXT Plugins
 - Other customization
- Alfresco Development
- Mobile Development
- Portfolio
- Testimonials


+ Add Child Page Permissions Delete Copy Applications


Details

Name (Required)

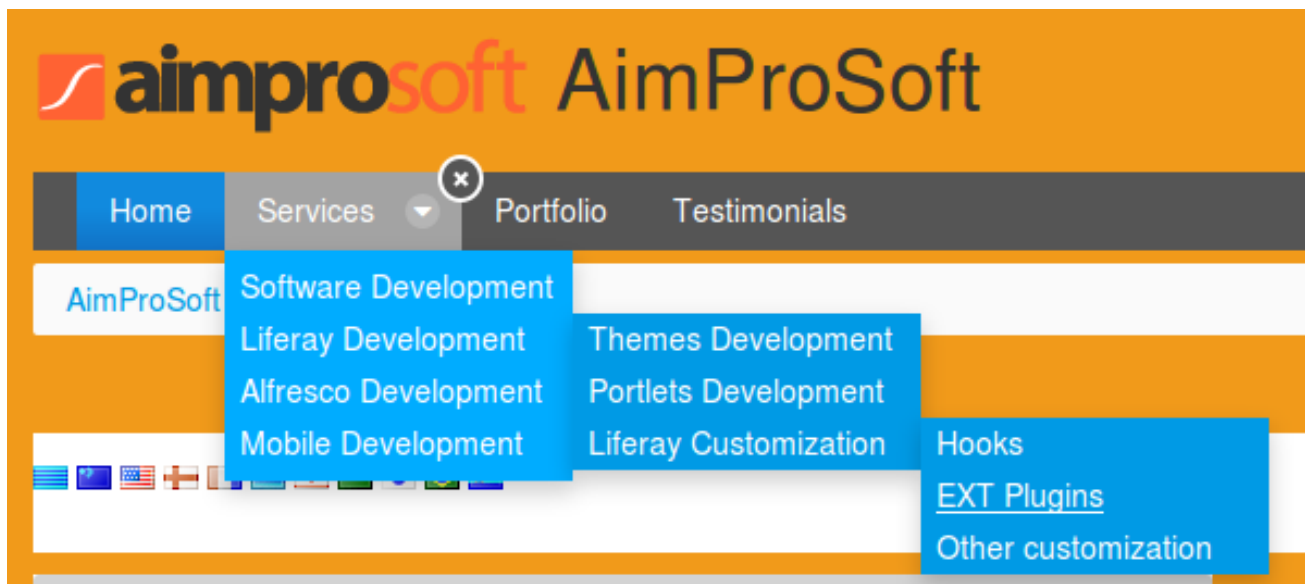


☐ Hide from Navigation Menu 

Friendly URL 



And look at the result:



As we can see, with the help of the developed macro the submenu is displayed for any level of nesting pages.

Using Liferay-services in theme

Now we consider the use of Liferay-services in a theme.

What is Liferay-services?

Liferay-services – classes that allow to work with liferay models (create, retrieve from the database, save, etc.). For each **[Entity] model** in Liferay its own **[Entity] Service** was created.

[Liferay-services](#) can be used in your theme by using the service locator:

```
#set($myService = $serviceLocator.findService("[Entity]Service"))
```

After obtaining the service by such way, it is possible to call its methods for receiving the required values in the theme. For example, we can obtain the service for users and for the layout in the following manner:

```
##User Service
#set($userLocalService = $serviceLocator.findService("com.liferay.portal.service.UserLocalService"))

##Layout Service
#set($userLocalService = $serviceLocator.findService("com.liferay.portal.service.LayoutLocalService"))
```

In Liferay the services can be **local** and **remote**. Local services:

- Classes com.liferay.portal.service.[Entity]LocalService;
- Do not contain the checking of permissions before calling.

Remote services:

- Classes com.liferay.portal.service.[Entity]Service;
- Before the call, they check permissions , and put the **PrincipalException** if the user does not have enough permissions for method calling.

It is better to use **local services** for avoiding the appearance of **PrincipalException**.

Example of Liferay-services

We will do it so that the page **‘/administration’** will be displayed in the navigation only for users of **‘Aim ProSoft’** organization, which have there the role **‘AimproSoft Admin’**. To do this, we will connect the necessary services in **init_custom.vm**:

```
##Services
#set($organizationLocalService = $serviceLocator.findService("com.liferay.portal.service.OrganizationLocalService"))
#set($userGroupRoleLocalService = $serviceLocator.findService("com.liferay.portal.service.UserGroupRoleLocalService"))
```

Using these services, we will define, whether the current user has the role **"AimProSoft Admin"** in the **"AimProSoft"** organization.

```
##Custom Variables
#set($companyId = $themeDisplay.getCompanyId())
#set($aimOrganization = $organizationLocalService.getOrganization($companyId, "AimProSoft"))
#set($aimOrgGroupId = $aimOrganization.getGroup().getGroupId())
#set($isAimAdmin = $userGroupRoleLocalService.hasUserGroupRole(
    $themeDisplay.getUserId(), $aimOrgGroupId, "AimProSoft Admin"))
```

In **navigation.vm** for checking we use the variable **\$isAimAdmin** that was declared in **init_custom.vm**.

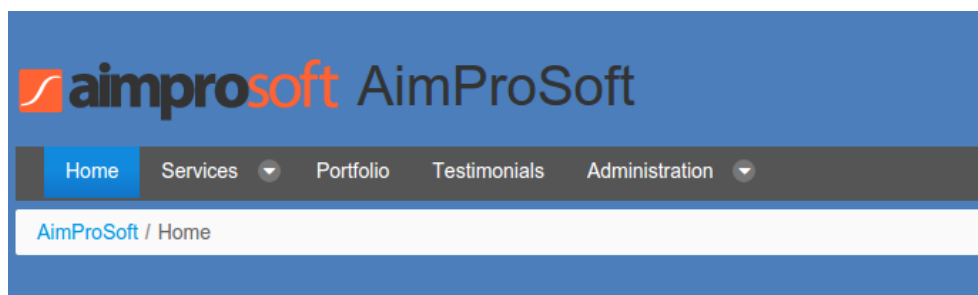
```
#foreach ($nav_item in $nav_items)

#set($friendlyUrl = $nav_item.layout.friendlyURL)  /* Get friendlyUrl of current $nav_item */
#set($showPage = true)                          /* Set 'showPage' as true by default */
#if ($friendlyUrl == "/administration")           /* Check if friendlyUrl is '/administration' */
    #set($showPage = $isAimAdmin)                /* Show '/administration' for AimAdmin only */
#end

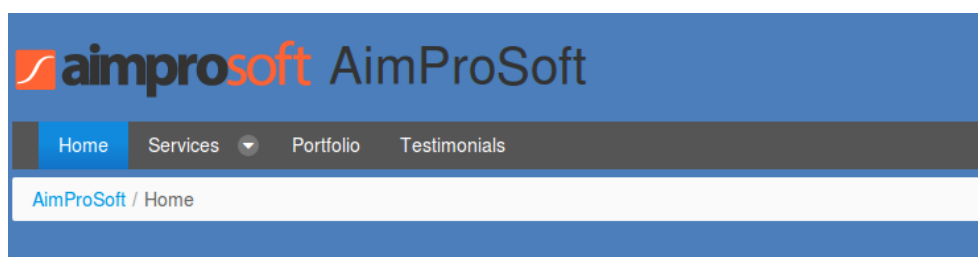
#if ($showPage == true)

#set ($nav_item_attr_selected="")
#set ($nav_item_attr_has_popup="")
#set ($nav_item_caret="")
#set ($nav_item_css_class="lfr-nav-item")
#set ($nav_item_link_css_class="")
```

Here the checking **isAimAdmin** for the page **'/administration'** is executed. That is, only users with the role **'Aim ProSoft Admin'** will be able to see this page. After deployment we see that **'AimProSoft Admin'** can see the page **'Administration'**,



but a user without this role - can not.



(P.S. The same functional can be realized via display of the permissions on the page; here it was so done for demonstration of capabilities of using the services in the theme).

Using the custom attributes

Custom attribute can be gained in the topic via **expandoBridge**:

```
$entity.getExpandoBridge().getAttribute("[ATTR_NAME]"),
```

where **\$entity** - an entity, of it attribute we should receive (it can be user, organization, page, etc.), **"[ATTR_NAME]"** - the attribute name.

For the page we create custom attribute **"bgColor"** (for setting the page color in the navigation menu):

[Portal Settings](#)
[Custom Fields](#)
[Server Administration](#)
[Portal Instances](#)
[Workflow](#)

[←](#) Page: New Custom Field

Key

Type

Resource	Custom Fields	
Blogs Entry		Edit
Bookmarks Entry		Edit
Bookmarks Folder		Edit
Calendar Event		Edit
Document		Edit
Documents Folder		Edit
Message Boards Category		Edit
Message Boards Message		Edit
Organization		Edit
Page	Bgcolor	Edit
Role		Edit
Site		Edit
User		Edit
User Group		Edit
Web Content Article		Edit
Wiki Page		Edit

Now, in the **'Manage Pages'** we can specify the value of a custom attribute for the different pages:

Site Pages

Then, we change the navigation using a custom attribute for the background color:

```

## Navigation Macro ##
#macro(navigation_macro $nav_elem)

    #set ($nav_elem_attr_selected="")
    #set ($nav_elem_css_class="lfr-nav-item")

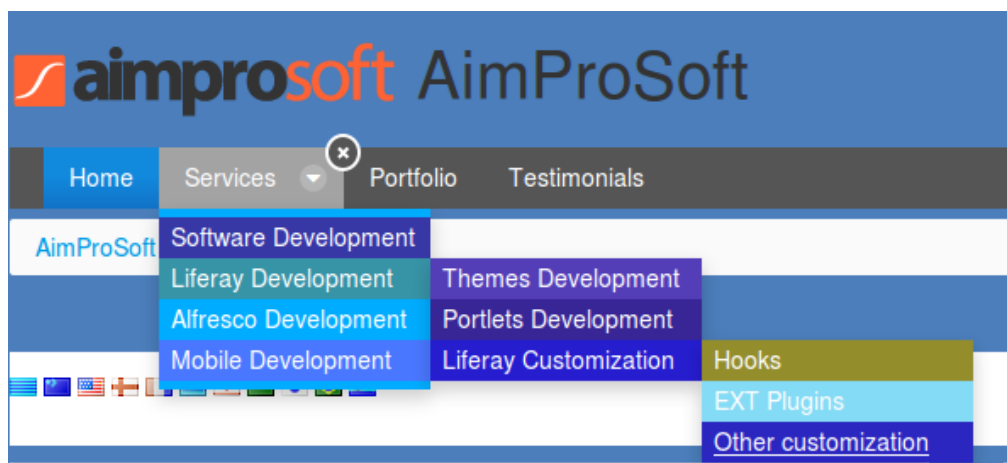
    #if ($nav_elem.isSelected())
        #set ($nav_elem_attr_selected="aria-selected='true'")
        #set ($nav_elem_css_class="selected")
    #end

    #set($bgColor = $nav_elem.layout.getExpandoBridge().getAttribute("bgColor"))

    <li class="$nav_elem_css_class" style="background-color: $bgColor" id="layout_$nav_elem.
    $nav_elem_attr_selected role="presentation">
        <a aria-labelledby="layout_$nav_elem.getLayoutId()" href="$nav_elem.getURL()"
        $nav_elem.getTarget() role="menuitem">$nav_elem.getName()</a>
        #if ($nav_elem.hasChildren())
            <ul class="dropdown-menu child-menu" role="menu">
                #foreach ($nav_elem_child in $nav_elem.getChildren())
                    #navigation_macro($nav_elem_child)
                #end
            </ul>
        #end
    </li>
#end

```

After that we will get an multicolored menu:



Chapter 8. Color Schemes

Add the colors...

What is the color schemes?

Color Scheme - is one of the variants of Liferay-theme (that has its proper colors, styles, borders, etc.). For the same theme the several color schemes can be developed.

Color schemes for CLASSIC theme

Color schemes for theme are specified in the file **liferay-look-and-feel.xml**. For classic theme we see the following:



As we see, for theme '**classic**' were created three color schemes: **Default**, **Dark** and **Light**. When we go to '*Look and Feel*' and select '*Classic*' theme, we will see, that three color schemes are available for this theme:



☒ Classic

Description

Portlets, themes, and layout templates included with Liferay Portal.

Author

Liferay, Inc.

Color Schemes (3)



☒ Dark



☐ Default

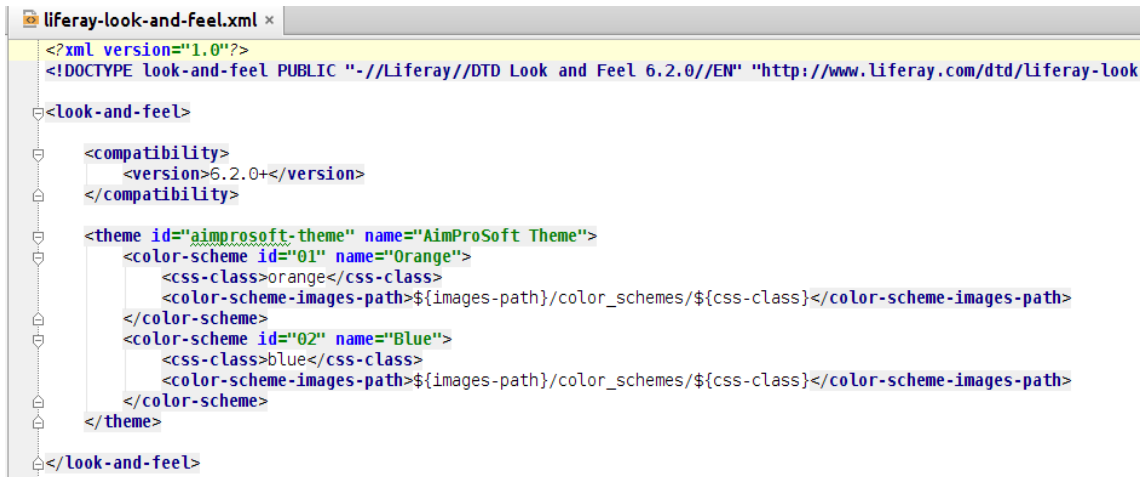


☐ Light

Applying the different color schemes for **classic** theme, we will get the different styles for theme.

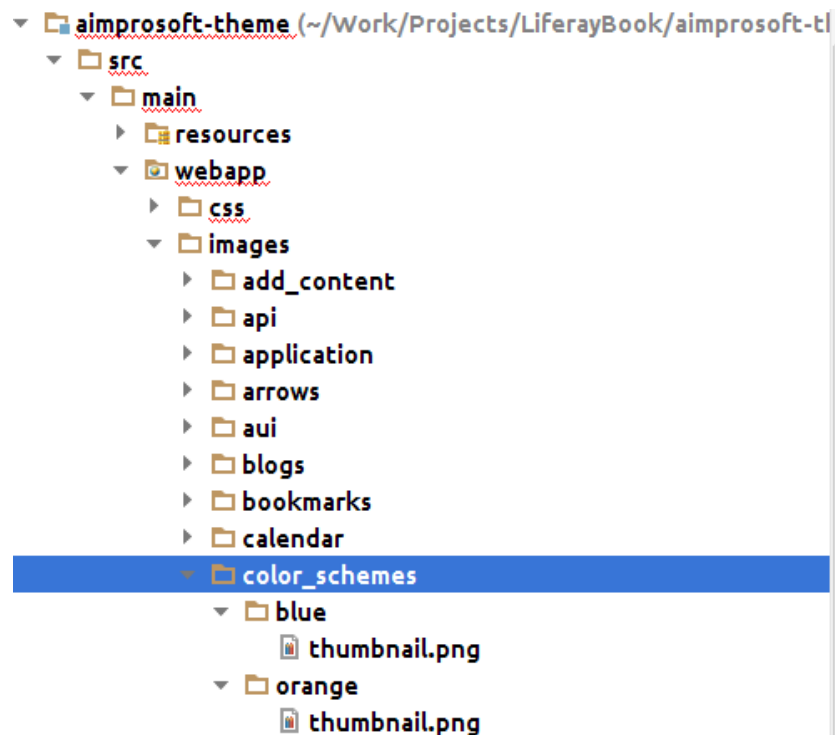
Creating our own COLOR SCHEMES

For created theme '**aimprosoft-theme**' we will create two color schemes: **orange** and **blue**. We create the file `liferay-look-and-feel.xml` (on basis of **classic** scheme):



Here we described two color schemes for **aimprosoft-theme**, indicating for each of them css-class and basic path to images.

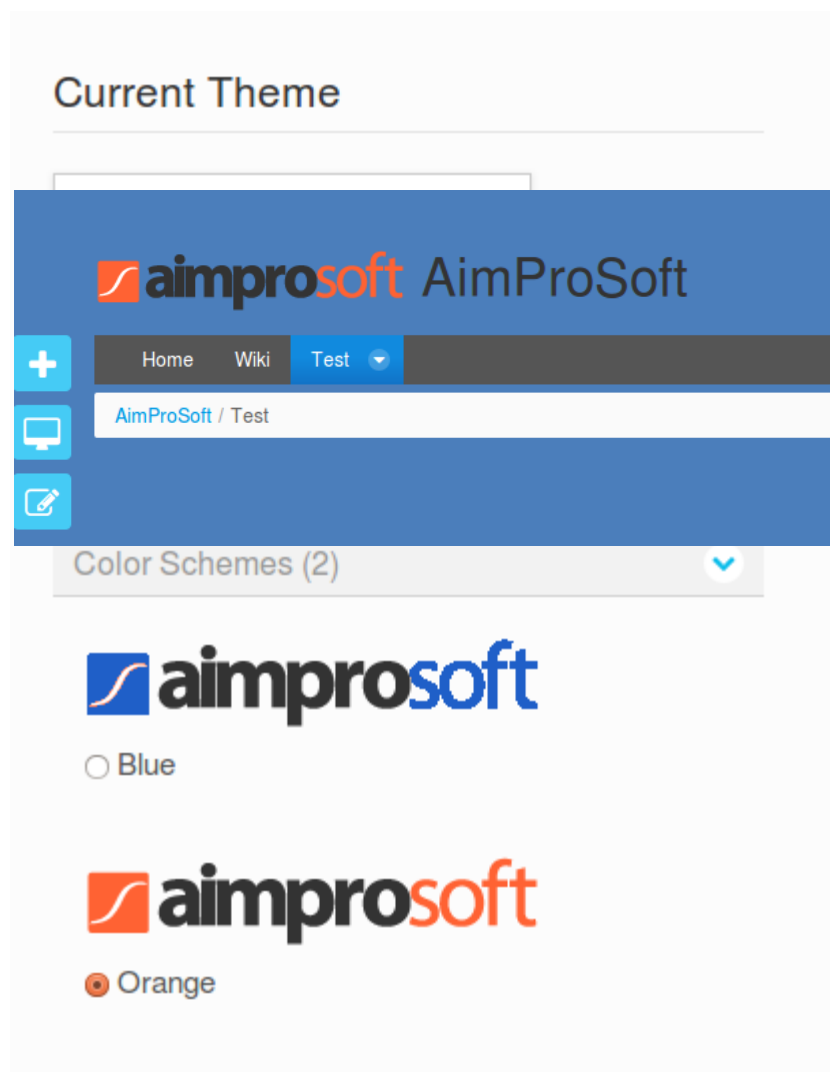
In the folder `aimprosoft-theme/src/main/webapp/images/color_schemes` we will create the folders for each of color schemes. We will deposit the file **thumbnail.png** in each of these folders.



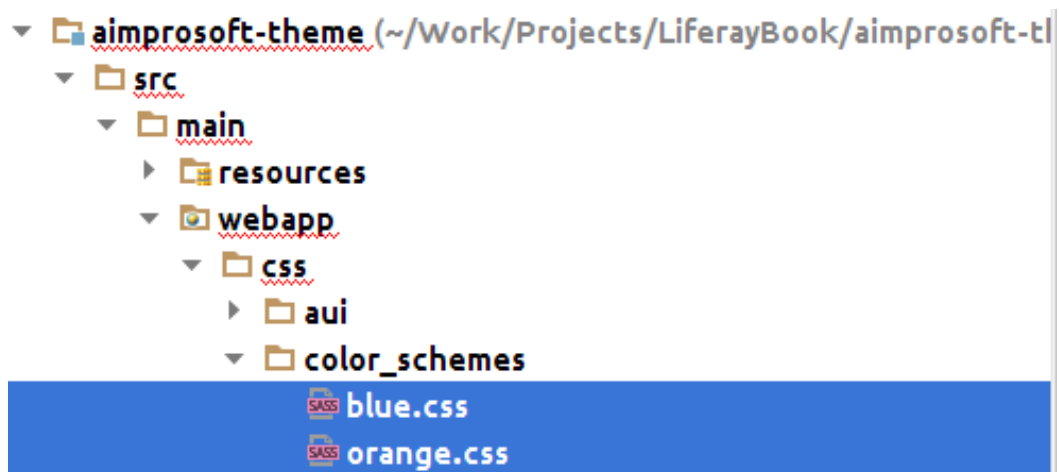
After recompiling of the theme we will see that the two color schemes became available



for “aimprosoft-theme”:



Now we specify the styles for color schemes. We create the files `css/color_schemes/blue.css` and `css/color_schemes/orange.css`:



and specify various colors of background:

Similarly we can specify another styles inside of classes `.blue` and `.orange`. We apply the created color schemes and check the result.

1) Blue Color Scheme

2) Orange Color Scheme

