enonic

academy

WIFI

Network name:

enonic

password:

enonic12

XP: Developer 201

Welcome

- Instructor
- Our offices
- Computers - preinstalled with JDK 8 (u92+)
- You?
  - What other CMS have you worked with?
  - Any experience with Enonic XP? (or the old CMS)

Course prerequisite

Attended XP Developer 101

or

A few days of experience building with XP

(+ JSON, JavaScript, HTML, and XML)

What we’ll learn today

Tasks to improve your knowledge from Developer 101 course (or similar experience):

- More in-depth XP features
  - Basic privileges theory, library stacking,
  - Page Contributions, site-wide settings, Mixins (x-data and inline)
  - Thymeleaf fragments
- XP functionality and libs
  - Filters, services (Rest), Controller Mappings, Error-handling
  - Install libraries, make your own (local and external)
  - Images, Menus, webapps
- Architecture in XP
  - Best practices, common mistakes
  - Querying, performance, request object

Schedule

- 09:00 - Introduction
- 09:05 - XP setup
- 09:10 - XP roles, site-settings, page contributions, mixins
- 09:30 - XP libraries (install and use local and external)
- 11:30 - Lunch
- 12:00 - XP, more functions
- 14:00 - XP, architecture and beyond
- 15:15 - Closing words, discussion
- 15:30 - Gives us **feedback** and you can go

A “Smörgåsbord” structured course

40%

enonic

Getting started

Task - **Setup**

Did you just attend the Developer 101 course? Then you are already set up, enjoy a micro-nap instead.

- Download and install XP
- Setup the **$XP_HOME** variable
- Start XP
- Download and deploy “starter-academy”
  - Find it on Enonic’s Github page
- Publish the entire Site in Content Studio
  - With child content

EASY

Privileges

Control access to XP and your contents

enonic academy

Users and roles - basics

Each user/visitor requesting data from Enonic XP is assigned one or more roles. Use these to handle privileges.

- **Everyone** - the role 100% of all visitors get.
  - You will want all your content to give “Read” access to this role.
- **Anonymous** - a “placeholder” user for all unknown traffic
  - when you log in to the admin interface (or any internal login system) you are no longer anonymous!

💡 ***Protip**! Add settings to the Site content and inherited down.*

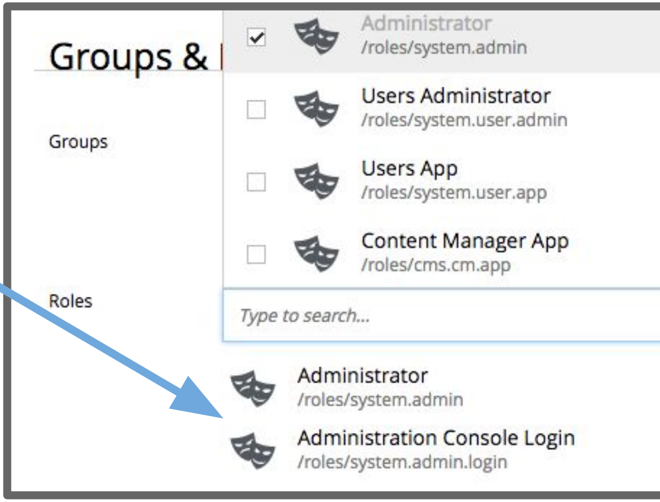
Roles - the basics (cont.)

Docs: / Admin Guide / Users / **Roles /**

- Roles are assigned to users (or groups) in Users admin tool

  Protip! Assign roles to groups, not users!

- Roles for an "admin" (full access)
- Limit access as much as possible
- **Best practise**: assign Roles to Groups and add multiple users to the Group



Groups &

Groups

Roles

✓ Administrator
/roles/system.admin

Users Administrator
/roles/system.user.admin

Users App
/roles/system.user.app

Content Manager App
/roles/cms.cm.app

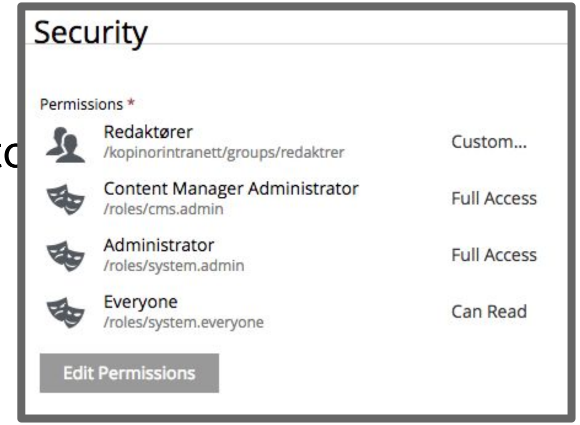
Type to search...

Administrator
/roles/system.admin

Administration Console Login
/roles/system.admin.login

Roles - the basics (cont.)

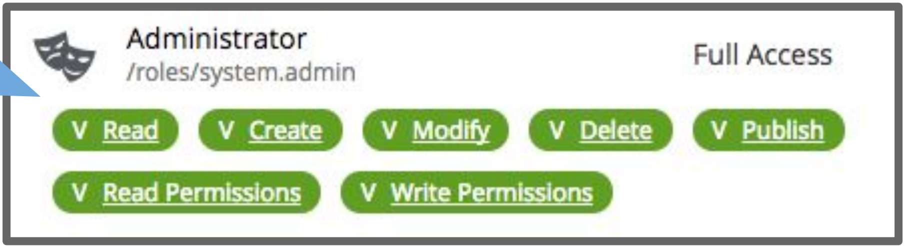
- Content Studio is used to assign roles + groups + users different privileges.
  - From predefined lists:
    - "Can read", "Can publish", "Can write" etc
  - Or Custom!
- Set per content
  - Can be inherited
- XP has these common privileges



Security

Permissions *

Redaktører	/kopinorintranett/groups/redaktrer	Custom...
Content Manager Administrator	/roles/cms.admin	Full Access
Administrator	/roles/system.admin	Full Access
Everyone	/roles/system.everyone	Can Read



Administrator /roles/system.admin — Full Access

XPloring

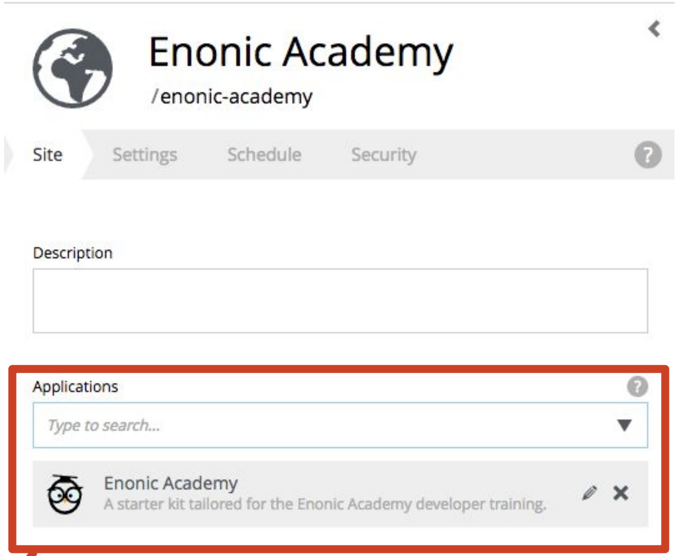
Some bits and pieces of XP functionality

enonic academy

Site-wide settings

Like config for a Part, we can add settings for our entire app - global ones. They are configurable for each Site the app is used on!

Edit the **Site content** in Content Studio and look for the pen-icon on the app in the Applications list.



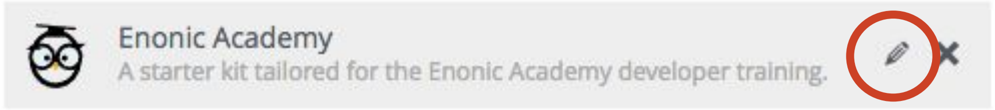
Enonic Academy
/enonic-academy

Description

Applications

Type to search...

Enonic Academy
A starter kit tailored for the Enonic Academy developer training.



Enonic Academy
A starter kit tailored for the Enonic Academy developer training.

Task - **Site-wide settings**

Docs: / Developer Guide / Sites / **Site Descriptors /**

Site.xml can store global settings for an App. Unique for each **site** the app is used on. We call it the Site descriptor.

- Add a TextLine called “copyright” in site.xml
  - Add inside the <config>
- Open your Site in edit mode (reload if already open)
  - Write in the new field
- Edit your page component /pages/default/
  - **Controller**: Extract with getSiteConfig()
  - **View**: Output this text in the <footer>

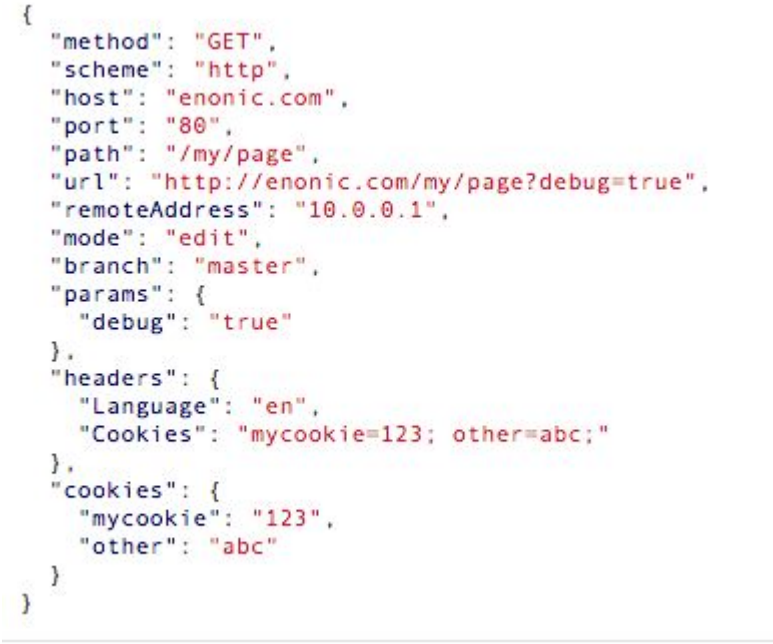
EASY

The Request

Docs: / Developer Guide / Serverside JavaScript / **Http Request /**

All controllers running will get the request object. Contains host data, method, current path, remote IP, querystring parameters (params), headers, and cookies.

The returning response object is similar, and can be manipulated.



```
{
  "method": "GET",
  "scheme": "http",
  "host": "enonic.com",
  "port": "80",
  "path": "/my/page",
  "url": "http://enonic.com/my/page?debug=true",
  "remoteAddress": "10.0.0.1",
  "mode": "edit",
  "branch": "master",
  "params": {
    "debug": "true"
  },
  "headers": {
    "Language": "en",
    "Cookies": "mycookie=123; other=abc;"
  },
  "cookies": {
    "mycookie": "123",
    "other": "abc"
  }
}
```

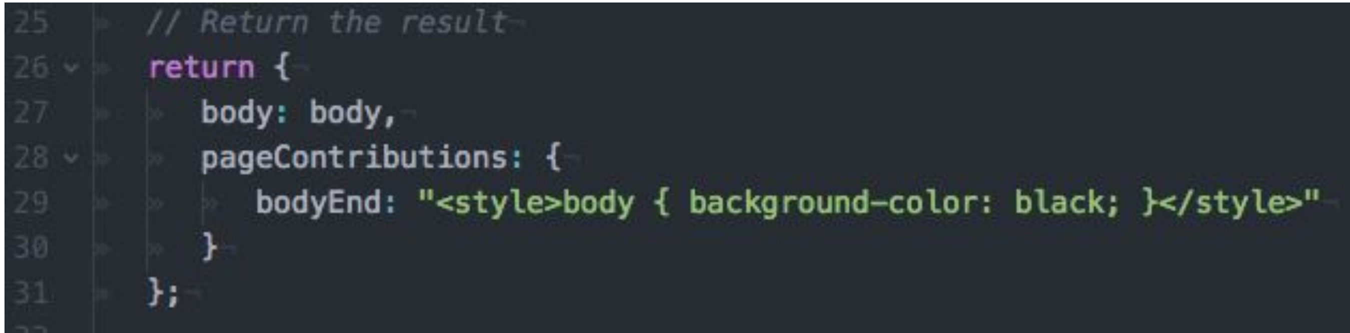
enonic

Contribute to page html

Each content (page/url) is rendered by the Page Component's code. This is usually where you have the <head> and <body>.

With **Page Contributions**, any Part (or other component) can contribute with custom html-code to the Page Component. Added only when that component is used on a content. This is a part of the JS object each component returns.

(Send HTML-code!)



```
// Return the result
return {
  body: body,
  pageContributions: {
    bodyEnd: "<style>body { background-color: black; }</style>"
  }
};
```

enonic

Task - **Page Contributions**

Docs: / Developer Guide / Sites / **Page Contributions** /

Let’s use Page Contributions to add the required CSS file whenever a specific Layout Component is used.

- Download **bootstrap.min.css** to folder /assets/css/
  - (or use a CDN direct link to skip downloading)
- Open the layout “2col” in your code editor
  - Update controller to send the html for adding a CSS file to the <head> of our pages.
  - (return the pageContribution in the response)

MEDIUM

Task - **Using mixins**

Docs: / Developer Guide / Schemas / **Mixins /**

Mixins are used to re-use sets of input types between **content types**, like different fields for address, or contact detail fields.

- Create a mixin “population”
  - In / site / mixins / **population** / **population**.xml
  - Add input type TextLine “population” to it *(steal it from city.xml)*
  - Also add another TextLine, called “Population density” to mixin
- In both **city**.xml and **country**.xml:
  - Remove the population TextLine field
  - Add the new mixin (of type “inline”)

EASY

Libraries

Using local and external libraries

Local libraries

Share functionality between controllers with local JavaScript libraries. Don’t repeat yourself! DRY-principle.

- Uses the /lib/ folder.
- For example, /lib/local.js:
    - exports.**myFunctionName** = function() { /* BODY */ };
- Add any numbers of files / functions / parameters
- Require where needed, like so:
    - var myLib = **require**(“**/lib/local**”) };
    - var something = myLib.**myFunctionName**(params);

Task - **Create and use a local library**

- Create file “local.js” in folder /lib/
- Write one simple JavaScript function using the format for exporting it, example:
  - exports.whatever = function() { return 1 + 2; };
- Include your library in another controller
  - var libs = { **local**: require(“/lib/local”) };
  - var testingLocal = libs.**local**.whatever();
  - log.info(testingLocal);

EASY

External libraries

Like local libraries, but these are bundled and shipped as their own apps. You use Gradle to include them in your app.

- Included when Gradle builds your jar
- You've already used a few of them!
  - lib-portal & lib-thymeleaf
- They too exports functions from the /lib/-folder, just like you just did with the local lib!
- Published to a Maven repo

Libraries: lib-util & lib-menu

We provide these libs to make day-to-day work easier for XP developers.

It’s our “STK” and gives you multiple functions for assisting you when coding your controllers (**lib-util**), and when building menus (**lib-menu**). Loads of features!

github.com/enonic/**lib-util**   github.com/enonic/**lib-menu**

Functions for working with JSON, like “trimArray”, “forceArray”, “isInt”, “isString”, check for values, and more!

Task - **Installing an external library**

- Find our **Lib Util** and **Lib Menu** on GitHub
- Edit your “build.gradle” and install them
  - add these lines to the dependencies-block:
    *include 'com.enonic.lib:util:**1.3.1**'*
    *include 'com.enonic.lib:menu:**2.0.1**'*
- “gradlew deploy” (“gradlew deploy -t” can’t detect these changes)

Now, all functions in lib-util and lib-menu are available for your controllers (that require them)!

EASY

Task - **Using an external library**

- Try the log function from **lib util**
- Open the Country part’s controller
  - Add *util: require(‘/lib/enonic/util’)* to libs object in top
  - Inside exports.get(), add *libs.util.log(req);*
  - Trigger a request to this part (visit a page with it on)

Stuck? Check the installation description
in the lib-util readme.

- Locate this function in lib-util source!

EASY

Stacking of libraries

When you use libraries in your app, everything inside them will merge into your app when Gradle builds it. Meaning, things inside a lib’s /lib/ folder will at runtime exist in your app’s /lib/ folder.

Doesn’t stop with libraries and controllers - also components (parts, layouts), content types, and assets stacks the same way!

⚠ Be aware of name collisions as the app’s code will always overwrite a library’s code.

Menus

Doing menus in XP (with libraries)

enonic academy

lib-menu

- Uses x-data mixins on its site.xml to add “Show in menu?” checkboxes to all content types in Content Studio.
- JS-functions to fetch site-menu, submenu or breadcrumb.
- Thymeleaf that outputs a menu with all the classes you’d need, like “active”, ”has-children”, “in-path”.
- Bring your own styling with CSS
- Or, “fork” our examples and write your own Thymeleaf!

From lib-menu 2.0.0 we introduced Thymeleaf fragments and mixins into the library itself. By just installing it, you will be able to use these from your app code. No copy pasting!

Task - **Create a menu** (1 of 2)

This is a task with many steps, so take it slow. **Verify each step** before you proceed!

- Make sure you installed lib-menu and rebuilt app
- Add to your site.xml the **x-data** mixin menu-item
  - Bundled with the library, read the main readme
- Build app and open a new contents
  - You should see new Menu fieldset
  - Update a few contents to “Show in menu”
  - Verify this with the Content Viewer app
    - Data is stored in the “x” property

HARD

Task - **Create a menu** (2 of 2)

- Create new part: /parts/**menu**/
  - With a menu.js and a menu.html
- Go to Github again and main readme:
  - Copy the code for the Thymeleaf fragment
    <div data-th-replace="/site/views/fragments/enonic-lib-menu/menu :: main-menu"></div>
  - Add this Thymeleaf to your menu.html
- In your controller menu.js:
  - Add lib-menu: require("/lib/enonic/menu")
  - Invoke it: var **menuItems** = libs.menu.getMenuTree(1);
  - Return this data in the model as **menuItems**
- Deploy
- Add Part to "Country" Page Template

HARD

Thymeleaf fragments

With the menu task we brushed on a powerful Thymeleaf feature: **fragments**.

- Like functions but for HTML
- With or without parameters
- Created anywhere in your app
- Called on with its path and name
- Libraries with fragments can also be used! (like **lib-menu**)

```
<nav data-th-fragment="main-menu">
  My <strong>repeating</strong> code
</nav>
```

```
<div data-th-replace="/site/views/menu :: main-menu">
</div>
```

1. Define the fragment
2. Use by pointing to it

Query

Theory on querying and performance

enonic academy

Theory - Querying

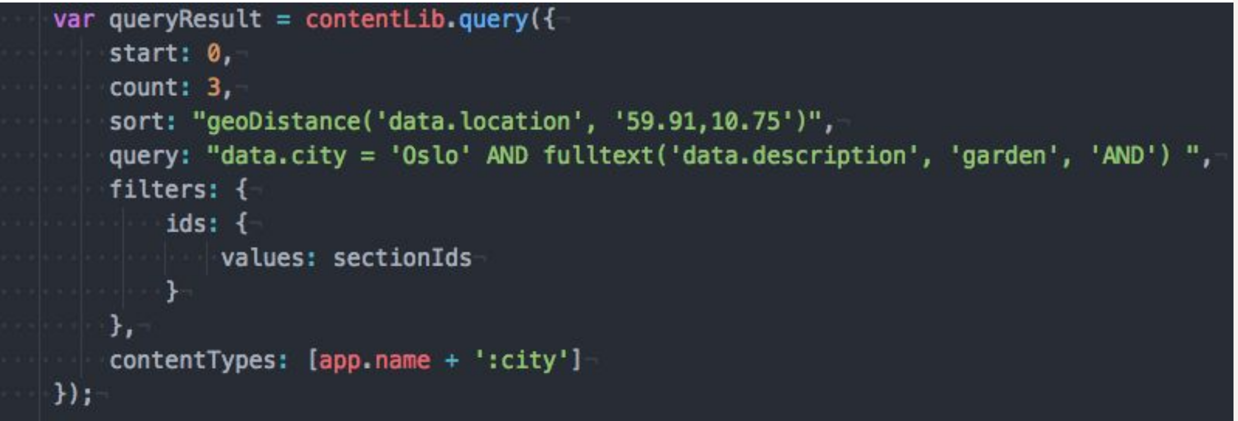
Docs: / Developer Guide / **Search** /

We’ve used .getContent() and .getComponent() so far. But in the real world you’ll be using other functions to accomplish more advanced querying.

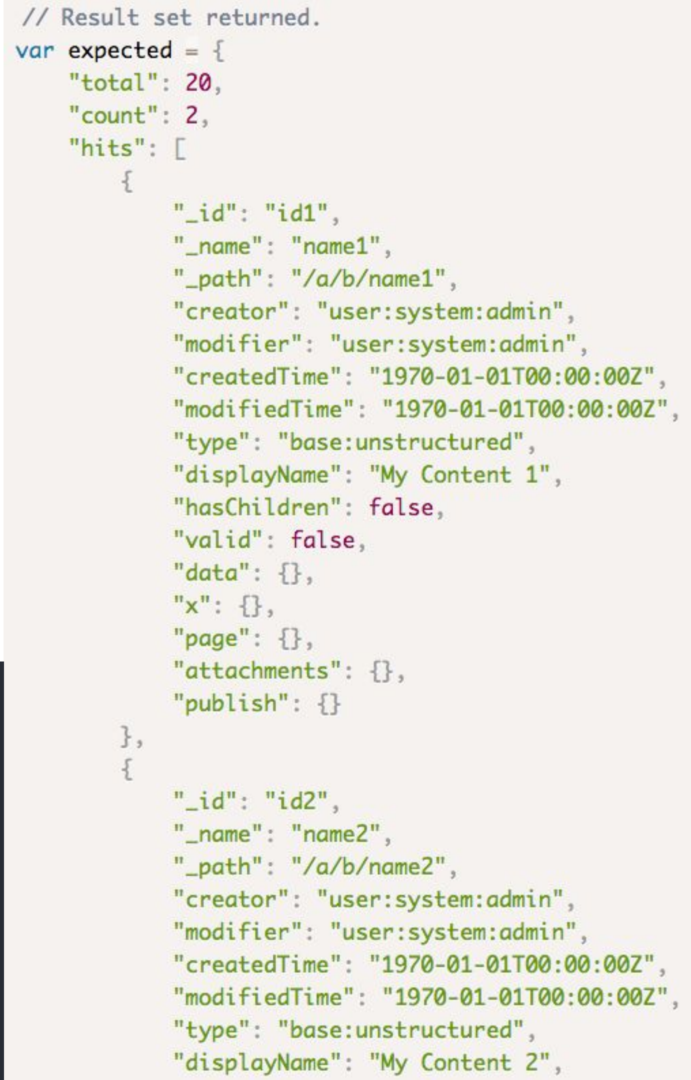
Query on any stored data, on date, time, path, relationships, geo location, and more. You can also aggregate data. Sort data. A query can even be filtered based on set values (very fast)!

Theory - Querying (cont.)

lib-content introduces the most capable query methods. It has simpler ones like .getChildren() for getting current content's children's data. Or .query() for more advanced queries.



```
var queryResult = contentLib.query({
    start: 0,
    count: 3,
    sort: "geoDistance('data.location', '59.91,10.75')",
    query: "data.city = 'Oslo' AND fulltext('data.description', 'garden', 'AND') ",
    filters: {
        ids: {
            values: sectionIds
        }
    },
    contentTypes: [app.name + ':city']
});
```



```
// Result set returned.
var expected = {
    "total": 20,
    "count": 2,
    "hits": [
        {
            "_id": "id1",
            "_name": "name1",
            "_path": "/a/b/name1",
            "creator": "user:system:admin",
            "modifier": "user:system:admin",
            "createdTime": "1970-01-01T00:00:00Z",
            "modifiedTime": "1970-01-01T00:00:00Z",
            "type": "base:unstructured",
            "displayName": "My Content 1",
            "hasChildren": false,
            "valid": false,
            "data": {},
            "x": {},
            "page": {},
            "attachments": {},
            "publish": {}
        },
        {
            "_id": "id2",
            "_name": "name2",
            "_path": "/a/b/name2",
            "creator": "user:system:admin",
            "modifier": "user:system:admin",
            "createdTime": "1970-01-01T00:00:00Z",
            "modifiedTime": "1970-01-01T00:00:00Z",
            "type": "base:unstructured",
            "displayName": "My Content 2",
```

Querying - performance

Enonic XP is very fast, but still consider speed when doing any type of query.

- A .getChildren() on your site content, using the query param to filter the results is significantly (tens of times) slower than a .query using the filter param to do the same.
- Always be careful with queries inside loops.
- Don’t fetch more than you need (don’t filter data in JavaScript - use query filter-parameter).
- A query result can easily be cached (using **lib-cache**).

Special functionality

Handle errors, intercepting pages/code, Rest APIs

enonic academy

Extending your app

You can easily extend functionality in XP with libraries. Need to send emails? Use the lib-mail. Need to do caching? Use lib-cache. And so on.

Other functions needs a few “ingredients” to work. Let’s look at the theory behind things like **Error handling**, **filters** (intercept pages), **controller mappings** (act on patterns) and how to set up **Rest APIs** in XP.

Error handling

Docs: / Developer Guide / Sites / **Error Handling /**

XP supports custom error handling scripts that can intercept any error code, like 401, or 404.

- Needs the file error.js in /site/error/
- Export functions for different errors (*handle404*)
- In the errorHandler, use a separate view file
- Capture any error and act upon it
  - Redirect, return a new body, etc

Filters - intercept traffic

Docs: / Developer Guide / Sites / **Response Filters /**

XP can intercept any code being sent to the client. You can "catch" and manipulate the HTML before sending it further.

- Add a JS-filter in /filters/
- Initiate new filter in site.xml
- Combined with Page Contributions we can insert code snippets into existing HTML
- Heavily used in apps on our Market

Controller Mappings - catch patterns

Docs: / Developer Guide / Sites / **Controller Mappings /**

For other ways of intercepting code we'll be using Controller Mappings. Lets you map different patterns to any JS-file.

- Runs specific Controllers on specific patterns:
  - On URL "xx/*/xx" (like /myapp/user/insert)
    - Doesn't need to exist in Content Studio!

```
<match>type:'portal:fragment'</match>
<match>_path:'/features/.*'</match>
<match>data.employee.type:'developer'</match>
<match>data.product.category:42</match>
<match>x.com-enonic-myapp.menuItem.show:true</match>
```

Services - setup Rest APIs

Docs: / Developer Guide / **Services** /

XP can expose functions via URLs, we call these Services. These can, for example, be used to create your own REST service!

- Needs a js-file inside a folder with same name
  - Like /resources/services/**counter**/**counter**.js
- Will expose a URL for this service
  - Like /_/service/com.enonic.app/**counter**
- Use exports.get (or other method) to catch the request
- In the return also set type:
  - return { **contentType: 'application/json'** }

Webapps

Using XP to run webapps (without CMS)

academy

Webapps in XP

Docs: / Developer Guide / **Application Controller /**

(Full guide https://developer.enonic.com/guides/hello-xp)

- Using the “main.js” file in root of any XP app will turn it into a webapp.
- A webapp will handle its own routing, but does not need the Content Studio at all!
- Combined with routing or Services turns it into a Rest API.
  - Use XP as the backend for your mobile app, PWA, etc!

Task - **My First webapp**

- Create the main.js file
- Export a get-handler
- In it, return the body with HTML
  - Set a <title> and some dummy <body> content
- Deploy
- Go to Application admin tool
  - Locate link to this new Webapp
- Test your app

EASY

Lessons learned

Experiences and other notes

academy

Preventing common mistakes (1 / 2)

- Don’t use Content Types for everything =S
- Use layouts for layout-tasks
- Understand and use the Page Templates
  - Use “supports” for preview and automation!
- Remember: content hierarchy generates paths/urls!
  - URL triggers content rendering. Think about structure in Content Studio!
- Repeating code instead of re-using and configuring
  - Don’t make nearly identical Parts
  - Use “require” in controllers, and reuse views!
  - Create / use libraries in your controllers
  - Use Thymeleaf fragments with params

Preventing common mistakes (2 / 2)

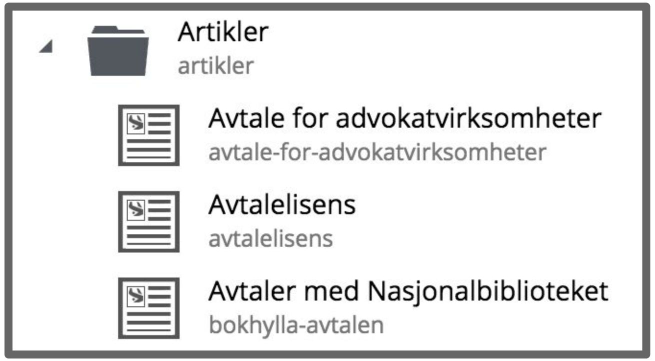
- Don’t fear refactoring, especially in early stages
- Get to know XP
  - Spend some time learning the terminology (Content Type, Component, etc). It will fundamentally change admin and the way web editors work.
- Prototyping: try different setups
  - Suggest different setups to the client, let them test and then decide.
- Use the data toolbox
  - Exports / imports, changing node structure, backup, migration...
- Explore Enonic Market
  - DRY, and learn from code
- Layer your site with apps

Architecture - best practices

Content first! Use **natural content structure**! The XP-way.

Each content is possible to visit (url exposed) anyway.

“Wrong”:



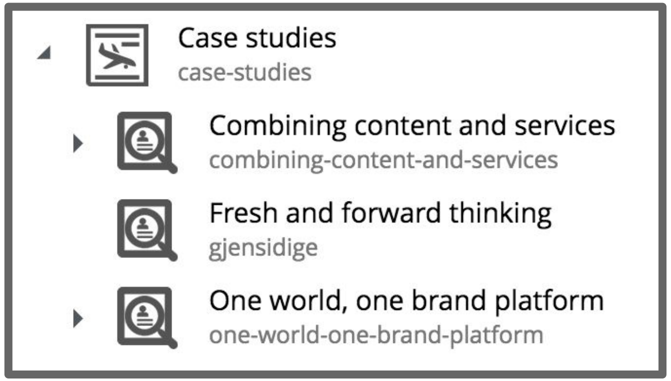
Artikler
artikler

Avtale for advokatvirksomheter
avtale-for-advokatvirksomheter

Avtalelisens
avtalelisens

Avtaler med Nasjonalbiblioteket
bokhylla-avtalen

Optimal:



Case studies
case-studies

Combining content and services
combining-content-and-services

Fresh and forward thinking
gjensidige

One world, one brand platform
one-world-one-brand-platform

Real world example

www.webagility.com

- One Page Component
- Two Layouts
- Multiple parts
- Content Types - authors, posts
- Multiple Apps (uses 9 apps!)
  - The site, SEO Meta Fields, Disqus, SumoMe, Google Analytics, Google Tag Manager, Sitemap.xml, RSS app, Content Viewer
- Multiple Libraries
  - Lib-menu, util, auth, context, encoding, http-client, and more!

Storage

Theory on XP storage

Storage in XP

- Data you store in XP is instantly indexed and searchable
  - ElasticSearch (bundled) used for searching
- XP custom logic for persisting data to disc
  - File based storage (stored in **$XP_HOME/repo**)
- Every content data (a “data entity”) is instantly indexed when created
  - And made ready for search
- Each “data entity” becomes a **Node**
  - You can access and manipulate its raw data using the **Node API** - even create your own custom storage!
  - Content is just a selected part from the Node Layer, has its own API. Content Viewer shows this part.

Nodes and Content

- A Node is accessed via the **Node API**
  - contains **all** the data about this entity.
- A Content is an extension of a Node, with schemas
  - (content type with validations, mandatory fields, scope)
- Work with Content using the Content API (lib-content)
  - Content Viewer uses the Content API

Example of two Nodes:

```
_id = 1001
_name = 'oslo'
_parent = ROOT

displayName = 'Oslo'
data.population = 647676
data.area = 454.03
location = geoPoint('59.9127300,10.7460900')
```

```
_id = 102131
_name = 'enonic'
_parent = '/oslo'

displayName = 'Enonic'
category = 'Software company'
employees = 20
```

Storage (cont.)

- By default, data is indexed based on the data-type:
  - Text-type properties are fulltext-indexed and aggregated into a special field "_allText" (commonly used when searching data in controller code).
  - Other properties are indexed according to the data-type (e.g GeoPoint, Double) and a String-representation.
- When using the **Node API**, the user can specify how the data will be indexed.
  - With the Node API you can edit anything in any way
- When using the **Content API**, Enonic XP decides how to index using the content-type schemas you developed.
  - The Content API controls what you can and cannot do

Storage (cont.)

- SQL-like syntax to query for data.
- Each Node is stored in a **Repo**.
- Using the **Repo API** you can create and manipulate repos
  - Like “silos” to store Nodes in
- Enonic XP comes with the system and cms repos
  - System repo is used for user data
  - Cms repo is used for all the content, issues, and more
- Create your own repo and fill it using the Node API

Misc

Useful Market apps, export/import, backup

enonic academy

Enonic Market

A few useful apps & libs:

- SEO Meta fields, Favicon, Robots txt
- Google Analytics, Google Tag Manager
- Re-captcha, Google Custom Search
- Data Toolbox, Content Viewer, lib-util and lib-menu

Nice apps with examples (see what can be done):

- PWA Starter, XPhoot (Websockets)

Export/import

- Securing data
- Migrating between installations
- Copies one specific branch (draft/master)
- Can “cherry pick” paths
- Stored on disc **$XP_HOME/data/**

Backups

Docs: / Operations Guide / **Backup and Restore /**

- “Dump” - used for bigger upgrades
    - And full backup
- “Snapshot” - stop floating index and make copy
    - Best way to do backup
    - Requires manual copy of blobs (cannot be restored while server is running)
- Market has apps for this!

Certification

Details about getting certified

enonic
academy

Improve and establish your competence

Why?

- ✓ Demonstrate your commitment to your profession as a developer
- ✓ Establish your credibility
- ✓ Empover your CV
- ✓ Get an advantage in building websites and applications on the Enonic XP Platform

Preparing for the exam

✓ The certification is based on Enonic XP. Attending our training is a great start!

✓ Study and go through the Enonic XP documentation, all answers can be found there.

About the exam

- ✓ The exam is conducted online, whenever you're ready!
- ✓ Multiple choice based test
- ✓ **30** minutes, **50** questions, **75%** correct!
- ✓ Your screen and face is monitored
- ✓ Questions will have 2 to 4 alternatives

Psst.. a few extra tips

- ✓ Study the overall parts of /Operations Guide
  - ○ Basics around Enonic XP server config, clustering
- ✓ Study! Take your time doing all tasks in the training slides, try to do them in different ways. *Don't do the test too soon*!
- ✓ Understand our available APIs and concepts.
- ✓ Understand how Content Studio is used by a developer
- ✓ Components, Layouts, Parts, Content Types - the XP lingo

Get certified

Visit our Certification portal

enonic.com/learn/**developer-certification**

Use this coupon code (one time):

XPDEVELOPER18

Need help?

Our brilliant community and dedicated crew of senior developers is here to help you through your struggles.

- ✓ discuss.enonic.com
- ✓ slack.enonic.com

What’s next?

Future XP versions

enonic academy

What’s next?

- XP 7.0

Wanna know what’s coming in the future?
Go to Discuss and find category called “**PAB**”
[https://discuss.enonic.com/c/**pab**](https://discuss.enonic.com/c/pab)

Want latest news? [https://discuss.enonic.com/c/**news**](https://discuss.enonic.com/c/news)

Have ideas? [https://discuss.enonic.com/c/**features**](https://discuss.enonic.com/c/features)

Got bugs? [https://discuss.enonic.com/c/**bugs**](https://discuss.enonic.com/c/bugs)

Enonic Meetup

http://www.meetup.com/enonic-oslo/

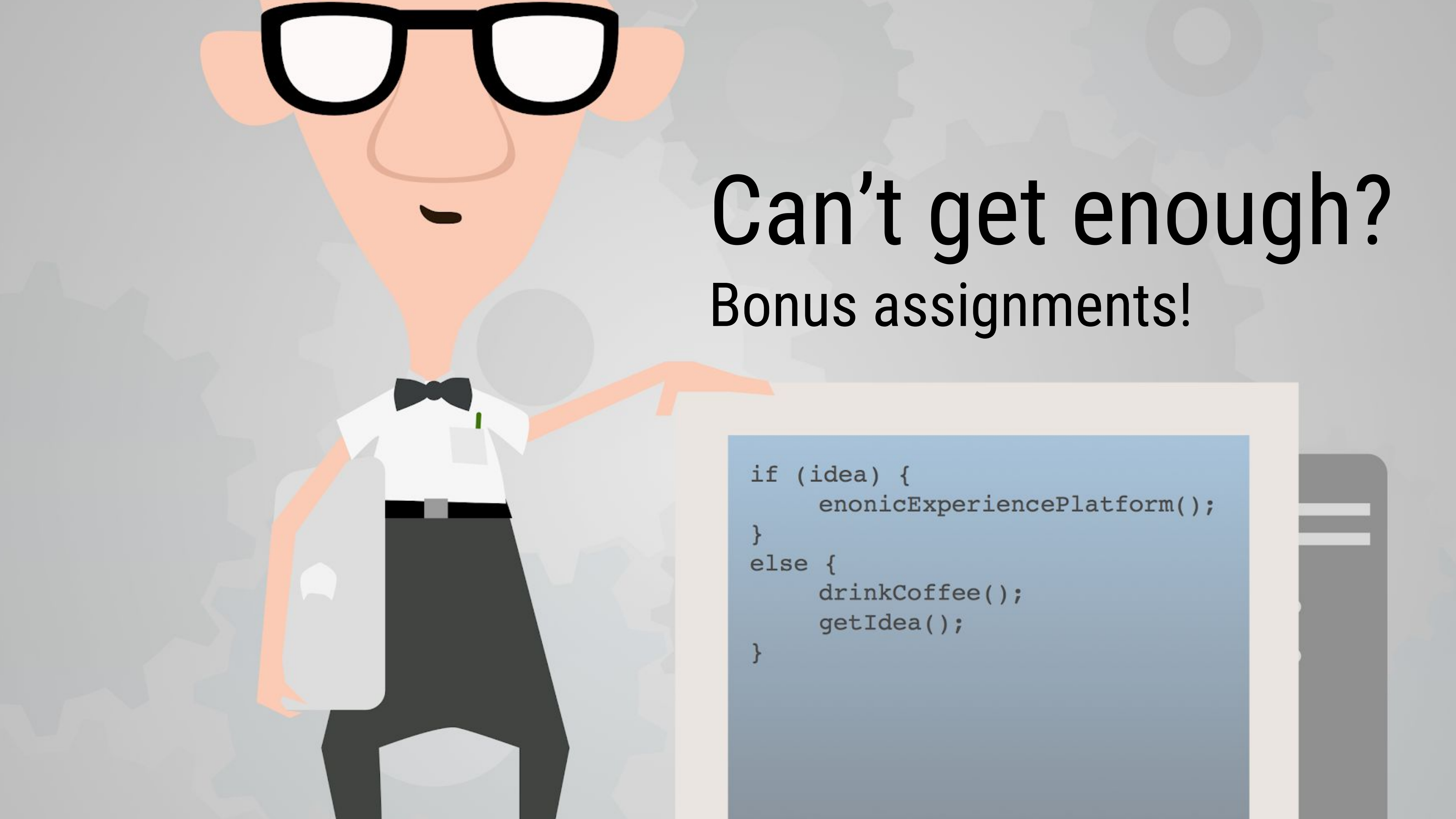
Feedback

https://www.surveymonkey.com/r/enonicfeedback

enonic
academy

Can’t get enough?

Bonus assignments!



```
if (idea) {
     enonicExperiencePlatform();
}
else {
     drinkCoffee();
     getIdea();
}
```

App Features

Try the developers own “app-features” from our github account. They use it to test all functions. Everything you can do in Content Studio is there.

- Fragments
- Filters
- Item sets
- Option sets
- Every input type

Task - **Multiple images**

Only if you attended and finished the task from **Dev 101**.

Extend the ImageSelector in the content type “Celebrity” to support **more than one image**.

- How does the data change?
  - Use Content Viewer widget to see
- How to loop this in your controller?
  - Hint: use a new array and the .push function
  - Don’t forget portal.imageUrl first!
- Use data-th-each to loop the json in View.
- **Extra**: How link each image to a larger image?

HARD

Task - **Be local**

Docs: / Developer Guide / **Localization** /

Locale settings exists on all content (check out edit mode). With “phrases”-files we can serve different strings depending on this setting.

- Add English phrases file with labels for the Celebrity content type
- Now duplicate this for Norwegian (or other)
- Output in Thymeleaf
- Switch locale on content to test

MEDIUM

Task - **Http-client**

XP can "call" any other URL out there, like APIs. From 6.11 this is a library shipped outside of XP core.

- Install the "http-client" library
- Add it to a part
- Try one of the JSON collections here: http://jsonplaceholder.typicode.com/
- Try outputting any data in Thymeleaf

MEDIUM

Task - **Configurable layouts**

The size of columns in a Layout are all CSS, so it’s easy to let users control this from admin!

- Add config to control Left/Right ratio
  - Add this config to layout model (xml)
  - Perhaps radiobuttons with 30/70, 50/50, etc
- Access these settings in your controller
- Update the view accordingly
  - Add inline style, or Bootstrap CSS classes

MEDIUM

Task - **Item sets**

Docs: / Developer Guide / Schemas / **Item Sets** /

Group input fields, possible to "add more". You can limit the amount of items. Stored as arrays in JSON.

- Create a "books" item-set
  - Item-set not mandatory
- But if a book is added, then these fields:
  - title (mandatory) - TextLine
  - author (mandatory) - TextLine
- Take this for a spin in admin
- Let's discuss how to extract data

MEDIUM

Task - **Cache**

XP has a programmable cache that is simple, yet powerful. It works in two steps: initiate, and call. Not a part of XP core.

- “Install” the cache library
- Initiate it in any part - before exports.get()
- Define a cache with 10 items for 1 minute
- Store “new Date();” in here
  - Make sure “key” is unique
- Output cached data in Thymeleaf

HARD

Task - **Cache your request**

Using both cache lib and http-client lib to achieve a powerful combo!

- Take the http-request from earlier task and store its “body” node into a cache.
- Make sure the http request is only done ones per minute even if you get a lot of traffic.

HARD

Task - **Multiple projects**

In the real world you'll be working on multiple Apps, spanning a number of different XP versions. With a good file structure this is easy to maintain.

- One folder for all XP installs
  - **/dev/xp/servers/**6.9.2
- One folder for all "home"-folders (one per App)
  - **/dev/xp/homes/**my-first-app
- One folder for each app
  - **/dev/xp/apps/**my-first-app
- Change $XP_HOME variable each time

HARD

Task - **More things to try** (1 of 2)

Running out of tasks? Use Docs and try these:

- More advanced use of content types and input types
  - Add icons
  - Use the <default> functionality
  - Try <help-text>
  - Try the <option-set> feature
  - Use Regex to validate a TextLine element
  - Add advanced filters to your ContentSelectors
  - Create your first Custom Selector input type (check tutorial!)
- Controller Mappings
- Try a XP snapshot - bleeding edge!

MEDIUM

Task - **More things to try** (2 of 2)

And even more things:

- Creating and changing content from code
  - lib-content contains functions for this
- Build your own fragments in Thymeleaf
- WebSockets
- Ajax and portal.componentUrl();
- Connect with Java
  - Explore the “__”-bridge from any controller

MEDIUM