DEVELOPING APPLICATIONS WITH XPAGES JAKARTA EE

JESSE GALLAGHER

CTO - I KNOW SOME GUYS
IP MANAGER - OPENNTF
HTTPS://FROSTILLIC.US
@JESSE@PUB.FROSTILLIC.US

Agenda

- What are Jakarta EE and MicroProfile?
- What is the XPages Jakarta EE Support project?
- Shared Components:
  - Expression Language
  - Managed Beans (CDI)
  - Data access (Jakarta NoSQL)
- UI Development Modes

Prerequisites

- Comfort with (or willingness to learn) Java
- Familiarity with annotations and Java 8 constructs (Optional, etc.) a plus
- Ability to install plugins into Designer and Domino

YOU DO NOT NEED:

- Knowledge of OSGi
- To start a new app from scratch

JAKARTA EE AND MICROPROFILE

What is Jakarta EE?

- The current form of Java EE
- Originally run by Sun, then Oracle, and now the Eclipse Foundation
    - Now fully open-source
- Releases 8 and 9 focused on open-sourcing and moving to jakarta.*
- Jakarta EE 10 made new spec changes and moved to Java 11 - we'll get that when Domino 14 is out
- https://jakarta.ee

What is MicroProfile?

- Eclipse project started during JEE's stagnation
- Now serves as a sort of focused incubator
- Targeted for microservice architectures, but most tools are useful generally
- https://microprofile.io/

The Standards And This Project

- Jakarta EE and MicroProfile are normally deployed in a server like GlassFish or Liberty as .war or .ear files
  - They're not needed here: Domino is our server and NSFs are our packages
- This project implements a large subset of both, but not all of either
  - Some specs - like Authentication - are largely inapplicable on Domino
  - Some - like EJB - are on the way out
  - Some - like WebSocket - face technical limitations
  - Some I just haven't gotten around to yet

Note on Naming

- With the move from Java EE to Jakarta EE, many specs changed their names, like:
  - JSP -> Jakarta Pages
  - JSF -> Jakarta Faces
  - JAX-RS -> Jakarta REST
  - JPA -> Jakarta Persistence
- I'll likely (and do in this slide deck) use the names interchangeably, out of habit
- In the short term, it's often useful to search by the old names when looking for documentation and Stack Overflow answers, but that is shifting over time

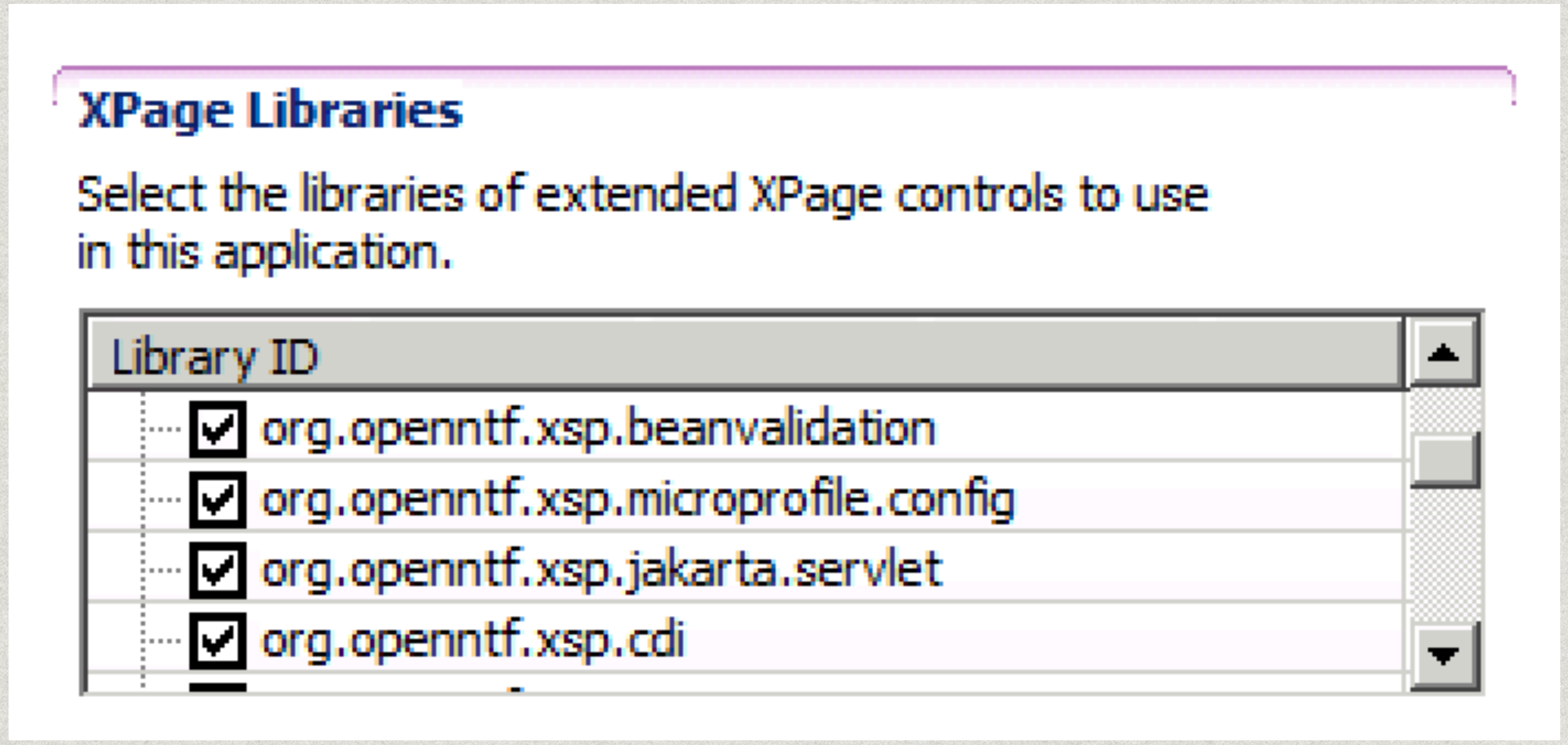
XPAGES JAKARTA EE SUPPORT

XPages Jakarta EE Support

- Began as adding a few utility specs: CDI for managed beans and JAX-RS for REST
- Grown to encompass a ton of specs, such as JSON-B, JSP, and Jakarta NoSQL
- It further expanded to include a selection of MicroProfile specs useful for Domino
- Primarily focuses on in-NSF development in Designer
    - Has some support for OSGi-based apps, but that takes extra knowledge

Usage

- Download from OpenNTF
- Install the plugins in Designer and the server
- Enable the libraries in the "Xsp Properties" editor
  - There's a ton - this will likely be simplified in 3.x
- Get to coding! (in Java, mostly)



XPage Libraries

Select the libraries of extended XPage controls to use in this application.

Library ID
☑ org.openntf.xsp.beanvalidation
☑ org.openntf.xsp.microprofile.config
☑ org.openntf.xsp.jakarta.servlet
☑ org.openntf.xsp.cdi

Examples

- The project has been gradually accumulating examples
- The "examples" directory in the repository and distribution ZIP contains ODPs for them
- For this presentation, I created a series of To-Do apps - more on that later
- The eclipse/nsfs directory contains "example" NSFs that serve as part of the integration-test suite. They're not useful as apps, but show a lot of capabilities in a technical way

SHARED COMPONENTS

Shared Components

- Regardless of your UI toolkit of choice, some components will be shared:
  - CDI for sure - "managed beans" but much better
  - Expression Language - a newer version than XPages ships with
  - REST - even in XPages or JSF apps, REST services come in handy
  - JSON-P and JSON-B - read/write JSON and convert objects
  - MicroProfile components - the Rest Client is a big one

EXPRESSION LANGUAGE

Expression Language

- Our old friend!
    - The current spec grew out of what started in JSF (as in XPages)
- Existing EL expressions will still work in XPages, including SSJS
- This EL interpreter is stricter about nulls, which is actually useful
- No configuration necessary: enable the library and it will take over in XPages
- EL also shows up in JSP, JSF, and other places (like more-esoteric CDI)

What you get

- All the same stuff as before!
  - `#{foo.bar}`, `#{foo[bar]}`, etc.
- Function calls
  - `${el:messages.format('helloMessage', session.effectiveUserName)}`
  - The "el:" prefix avoids an error marker in Designer - this is not needed in JSP or JSF
- String concatenation
  - `${'hi ' += session.effectiveUserName += '; good to see you!'}`

Examples

```
<xp:text value="#{managedBeanGuy.message}"/>


<xp:text value="#{el:functionClass.doFoo('I am from XPages')}"/>


<xp:dataTable id="issueList" value="#{el:issuesBean.get(viewScope.owner, viewScope.repo)}" var="issue">
   <!-- snip -->
</xp:dataTable>
```

Resources

- https://jakarta.ee/specifications/expression-language/4.0/
- https://www.baeldung.com/jsf-expression-language-el-3

CDI (MANAGED BEANS)

CDI (Managed Beans)

- The spec covering managed beans is CDI: Components & Dependency Injection
  - You don't have to care about why it's called that
  - You also don't have to care about EJB (don't ask if you don't know)
- Uses annotations instead of XML configuration (for our needs)
- Cooperates with EL and general XPages variable resolution
  - You can (and should) replace beans in faces-config.xml entirely

Example Bean

```
@ApplicationScoped
@Named("markdown")
public class MarkdownBean {
  private Parser markdown = Parser.builder().build();
  private HtmlRenderer markdownHtml = HtmlRenderer.builder()
      .build();

  public String toHtml(final String text) {
    Node parsed = markdown.parse(text);
    return markdownHtml.render(parsed);
  }
}
```

Example Bean - Injection

```java
@RequestScoped
@Named("encoder")
public class EncoderBean {

  @Inject @Named("dominoSession")
  private Session session;

  public String abbreviateName(String name) throws NotesException {
    Name dominoName = session.createName(name);
    try {
      return dominoName.getAbbreviated();
    } finally {
      dominoName.recycle();
    }
  }
}
```

Example Bean - Events and Scopes

```
@RequestScoped
@Named("requestGuy")
public class RequestGuy {
   @Inject
   private ApplicationGuy applicationGuy;
   private final long time = System.currentTimeMillis();

   public String getMessage() {
      return "I'm request guy at " + time + ", using applicationGuy: " + applicationGuy.getMessage();
   }

   @PostConstruct
   public void postConstruct() { System.out.println("Created requestGuy!"); }

   @PreDestroy
   public void preDestroy() { System.out.println("Destroying requestGuy!");  }
}
```

CDI Beyond Beans

- Managed beans are the "basic" case for CDI and most of what we'll use
- It goes beyond that, providing foundational layers for other techs:
  - Jakarta REST
  - MVC
  - Jakarta NoSQL
  - Pretty much all of MicroProfile
- Things get... weird when you dive in, but normal apps don't need that

Resources

- https://jakarta.ee/specifications/cdi/3.0/
- https://www.baeldung.com/java-ee-cdi
- https://openliberty.io/guides/cdi-intro.html

JAKARTA NOSQL

Jakarta NoSQL

- Data access layer similar to JPA for SQL databases
- Maps between Domino data and normal Java objects reasonably efficiently

Some Caveats

- The version of Jakarta NoSQL included in this project is specifically a beta version
- Newer versions have had several major changes:
  - They require Java versions higher than 8
  - The “Repository” concept moved to a new spec: Jakarta Data
- The plan is to move to a version of these when Domino 14 is out

Model Objects

- Models are “plain” Java objects (POJOs)
- The driver handles mapping between NSF documents and these objects
- There’s usually a one-to-one mapping between a form and a model object

```java
@Entity("To-Do") // Form name
public class ToDo {
  // Enums are stored as strings
  public enum State {
    Incomplete, Complete
  }

  @Id // Maps to UNID
  private String documentId;
  @Column("Title") // Field names
  private String title;
  @Column("Created")
  private OffsetDateTime created;
  @Column("Status")
  private State status;

  /* snip: getters/setters */
}
```

Repositories

- Repositories are just interfaces - you don’t provide the implementation
- The NoSQL layer parses method names and arguments to translate methods to queries
- The driver uses DQL and QRP internally to make this efficient
- Views and other Domino-specific behaviors (e.g. compute-with-form) are available

```
public interface ToDoRepository extends
   DominoRepository<ToDo, String> {

     Stream<ToDo> findAll(Sorts sorts);

     Stream<ToDo> findByStatus(State status,
Sorts sorts);

}
```

Usage

- Use CDI to inject a repository
  - Can be injected into CDI beans and REST resources
  - With XPages, you'll likely use a "broker" bean between XPages-type code and Jakarta-type
  - Can also be resolved programmatically, but not prettily

```java
@ApplicationScoped
@Named("ToDos") // Access by name in XPages
public class ToDosBean {
  @Inject
  private ToDoRepository repository;

  public List<ToDo> getAll() {
    return repository.findAll(
      Sorts.sorts().asc("created")
    ).collect(Collectors.toList());
  }

  public ToDo saveToDo(ToDo todo) {
    return repository.save(todo);
  }
}
```

THE UI PATHS

Example Apps: To-Do

- I made four versions of a bare-bones To-Do application:
  - XPages
  - REST with a basic HTML/JS UI
  - MVC with Jakarta Pages
  - Jakarta Faces
- Available at https://github.com/OpenNTF/org.openntf.xsp.jakartaee/tree/develop/examples/todo
  - Will be included in future distribution builds
  - I'll also add a README when I get a chance

Example Apps: To-Do

- Each uses Jakarta NoSQL for its data access
  - Single "To-Do" form with a few properties
  - Repository using DQL to load documents
- Just basic functionality: CRUD with "Complete" and "Incomplete" states
  - Don't take these as examples of fully-built apps, but rather just starting points

Preliminaries

- Though these are four apps, the mechanisms can be interwoven
    - For example, you can add REST services to an XPages app, or do an "admin" UI in MVC in an app that's otherwise just REST
    - Even XPages and Faces can mix, if you're careful
- These don't cover all the shared capabilities you'd likely use, such as the Rest Client and

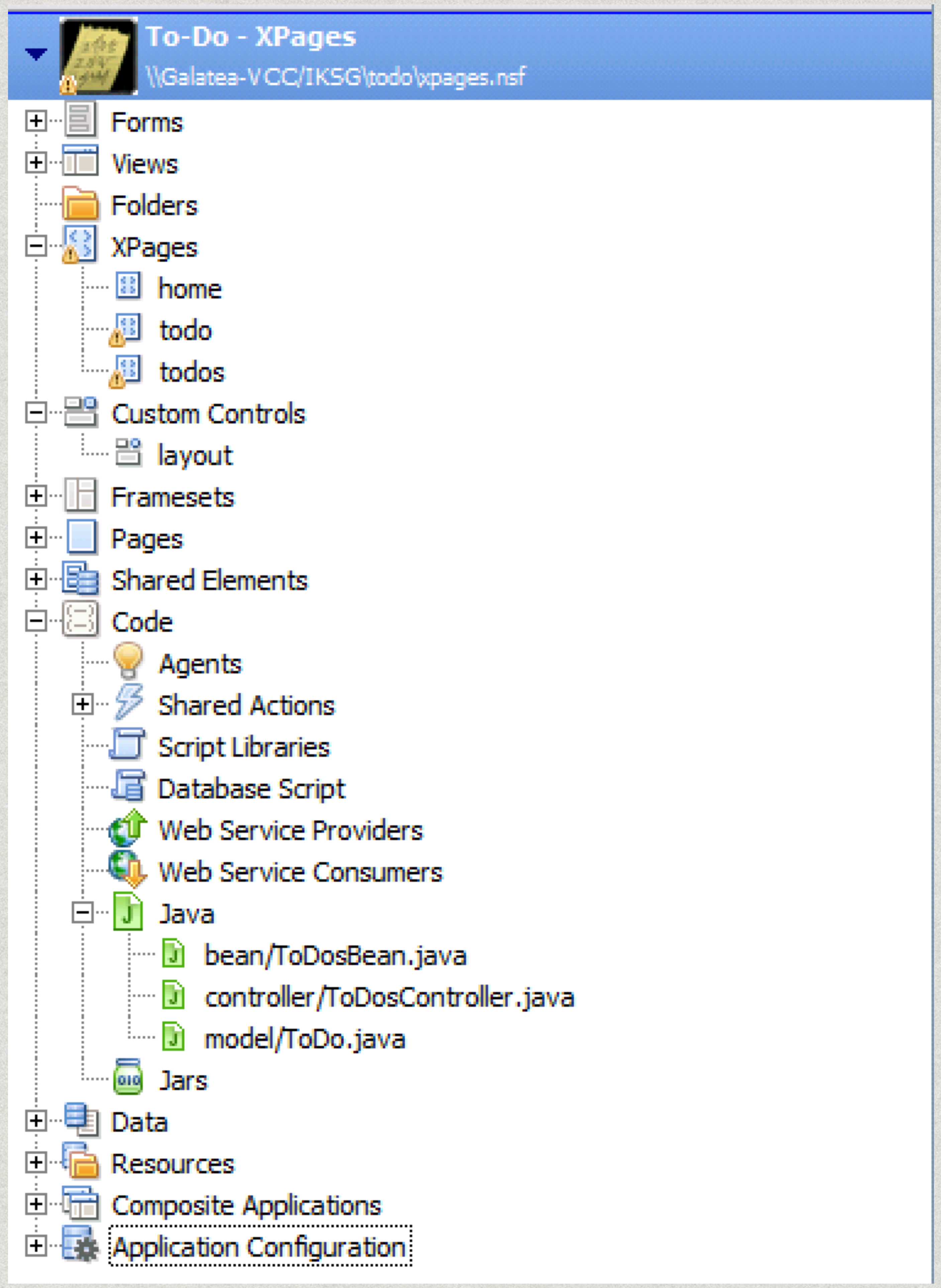
"XPAGES PLUS"

"XPages Plus"

- In this case, XPages remains your UI toolkit of choice
- Pros:
  - Retains the benefits of existing XSP markup, plus the tooling provided by Designer
  - EL and CDI bring direct improvements to the XPages experience
  - Can still mix with other types
- Cons:
  - XPages itself remains the same, and remains non-portable

Structure

- The structure here will be very similar to normal old XPages apps
  - "Normal", at least, if you write a lot of Java
- The "ToDosBean" here is a CDI bean that injects the Jakarta NoSQL repository
  - This is because XPages isn't "CDI native" the way some others are



To-Do - XPages
\\Galatea-VCC/IKSG\todo\xpages.nsf

- Forms
- Views
- Folders
- XPages
  - home
  - todo
  - todos
- Custom Controls
  - layout
- Framesets
- Pages
- Shared Elements
- Code
  - Agents
  - Shared Actions
  - Script Libraries
  - Database Script
  - Web Service Providers
  - Web Service Consumers
  - Java
    - bean/ToDosBean.java
    - controller/ToDosController.java
    - model/ToDo.java
  - Jars
- Data
- Resources
- Composite Applications
- Application Configuration

Development Experience

- Pretty much the same as you're used to when it comes to the UI
- The main change in XSP markup is that you can use newer EL syntax
- Business logic will be very heavily on the Java side
    - If you're not already doing this, it'll be a big change
    - If you *are* doing this, you can expect big improvements in convenience and capability
- Can also still use existing XPages constructs not available elsewhere, like xp:dominoDocument and xp:dominoView

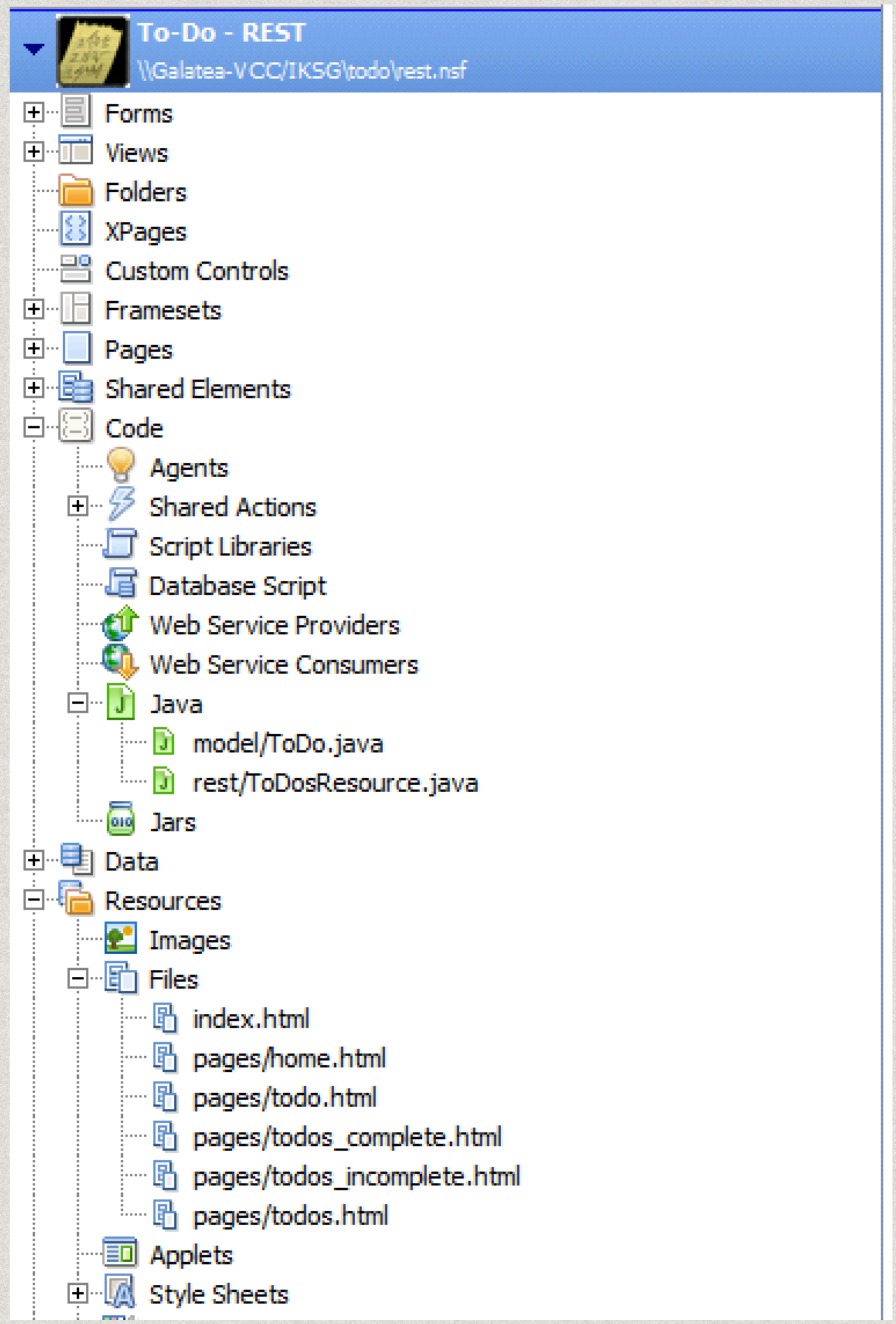
REST SERVICES

REST Services

- In this case, you focus primarily on writing REST services to be consumed by something else
- The "something else" would likely be a JavaScript app written in React or other toolkit
- Pros:
    - Jakarta REST (JAX-RS) provides very clean, declarative annotations for writing REST services
    - You get an OpenAPI spec "for free"
    - Scales very well for larger/split teams with front-end and back-end separation
- Cons:
    - A split app design like this is more complicated than a full-server-side toolkit
    - It introduces security/data-leakage concerns when you design your API

Structure

- Here, there are no XPages, but there are still Java classes
- ToDosResource defines the REST endpoints and needs no "broker" to access NoSQL
- Page UI files are in File Resources
  - These would likely be the build output from React/etc. in WebContent in a larger project
  - Could also potentially be a wholly-separate app



```
To-Do - REST
\\Galatea-VCC/IKSG\todo\rest.nsf
+ Forms
+ Views
  Folders
  XPages
  Custom Controls
+ Framesets
+ Pages
+ Shared Elements
- Code
    Agents
  + Shared Actions
    Script Libraries
    Database Script
    Web Service Providers
    Web Service Consumers
  - Java
      model/ToDo.java
      rest/ToDosResource.java
    Jars
+ Data
- Resources
    Images
  - Files
      index.html
      pages/home.html
      pages/todo.html
      pages/todos_complete.html
      pages/todos_incomplete.html
      pages/todos.html
    Applets
  + Style Sheets
```

Development Experience

- Will depend heavily on how you're writing the UI (or if you are at all)
- UI aside, all work will happen in Java, via model objects, beans, and REST services
- Designer will help with the Java syntax and source availability, but has no special knowledge of REST endpoints or CDI beans like e.g. IntelliJ does

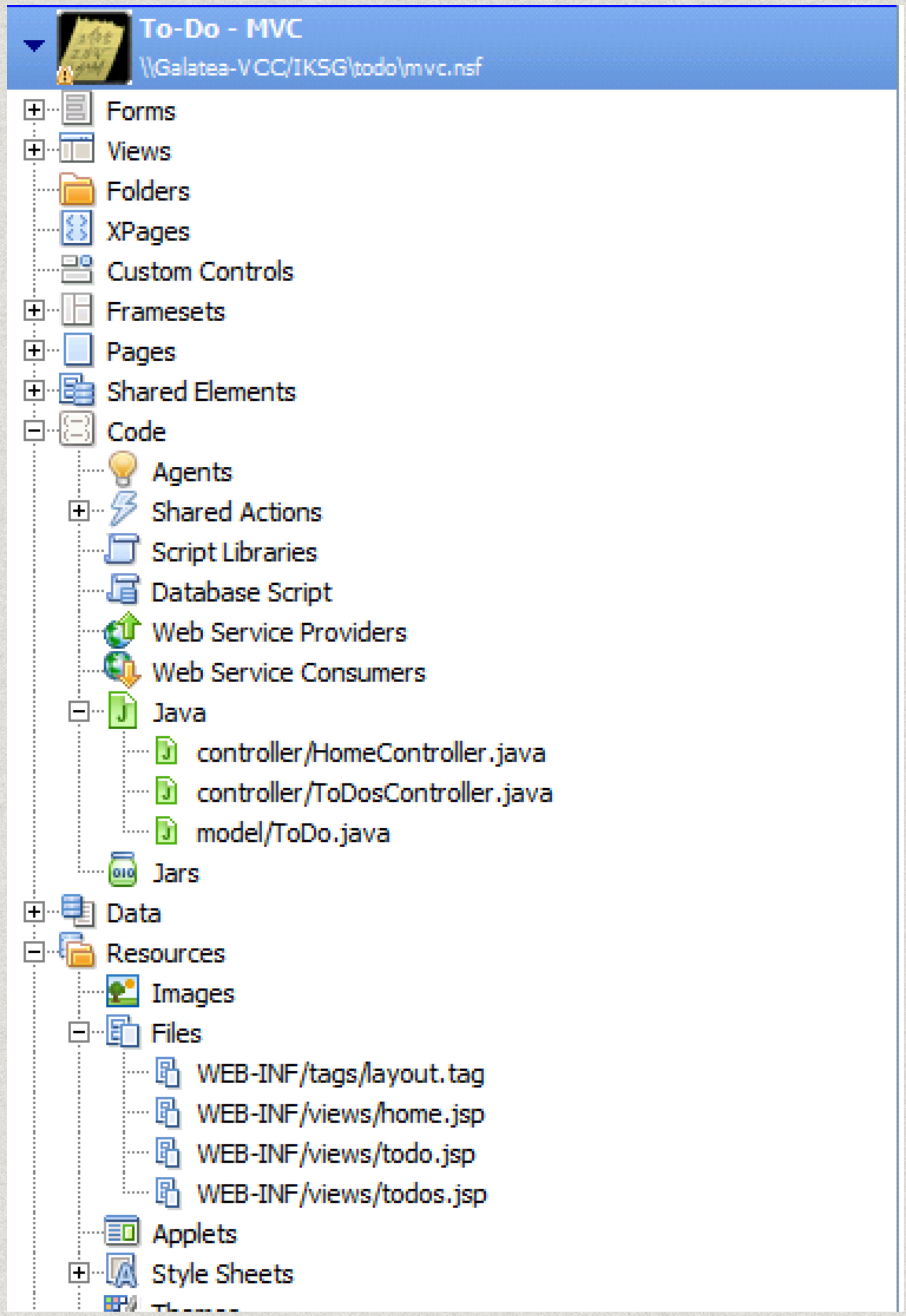
MVC AND JAKARTA PAGES (JSP)

MVC and Jakarta Pages

- In this case, you use the MVC spec on top of REST services and write your UI in Jakarta Pages (JSP)
- This is very well-suited to certain app types, such as document repositories, discussions, and blogs
- Pros:
  - Builds on the clean foundation of Jakarta REST
  - Pairs very well with a REST-based API for non-browser clients
  - "Back to basics" focus on HTML and HTTP
- Cons:
  - No server-side state makes complex forms difficult
  - Designer considers JSP pages HTML, so provides no help for JSP tags

Structure

- Looks similar to the REST version
- Here, "controller" refers to the MVC @Controller annotation
  - They are special kinds of REST resources
- Again, pages are in File Resources, but would likely be in WebContent in larger projects



To-Do - MVC
\\Galatea-VCC/IKSG\todo\mvc.nsf

- Forms
- Views
- Folders
- XPages
- Custom Controls
- Framesets
- Pages
- Shared Elements
- Code
  - Agents
  - Shared Actions
  - Script Libraries
  - Database Script
  - Web Service Providers
  - Web Service Consumers
  - Java
    - controller/HomeController.java
    - controller/ToDosController.java
    - model/ToDo.java
  - Jars
- Data
- Resources
  - Images
  - Files
    - WEB-INF/tags/layout.tag
    - WEB-INF/views/home.jsp
    - WEB-INF/views/todo.jsp
    - WEB-INF/views/todos.jsp
  - Applets
  - Style Sheets

Development Experience

- Writing JSP in Designer is *okay* - it knows about HTML well enough, but will not help with JSP tags
    - JSP is small enough that you'll likely quickly memorize the basics
- The Java side will be very similar to writing "normal" REST services

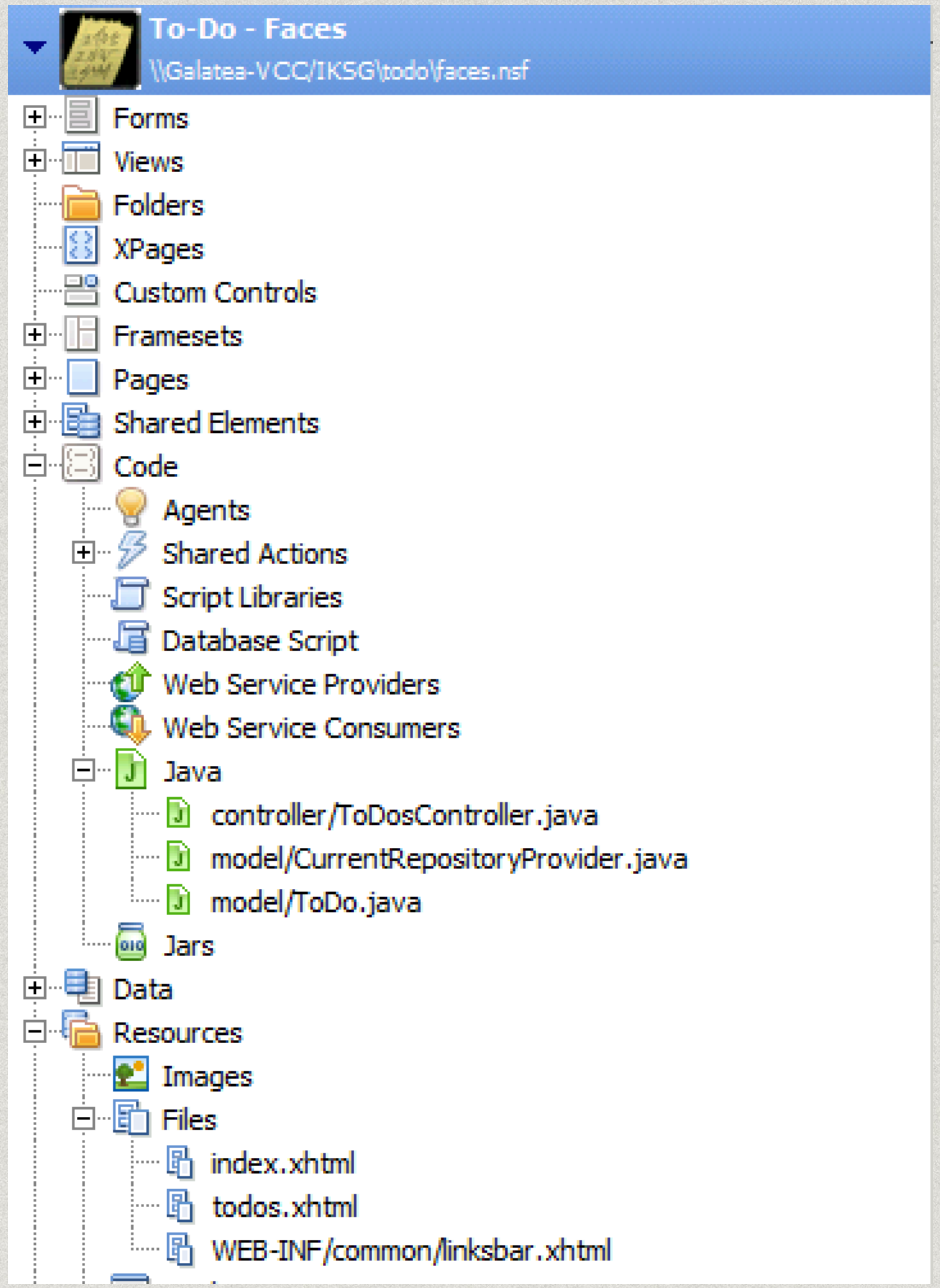
JAKARTA FACES (JSF)

Jakarta Faces

- In this case, you write Jakarta Faces (JSF) pages and access them similar to XPages
- Pros:
  - Very similar to XPages: individual pages with server-side state and a shared heritage
  - Very good for complex forms and other fiddly work
  - Faces is being actively developed
  - Can also use actively-developed third-party libraries like PrimeFaces
  - Lots of existing examples
- Cons:
  - Like JSP, Designer considers JSF pages HTML, so provides no help for JSF tags
  - Third-party libraries can require manual fiddling to avoid conflicts with the XPages runtime

Structure

- Same general idea as the XPages version
- Here, "controller" refers to a bean used to act like a controller as in XPages, not an official concept
  - Like in XPages, Faces has an implicit controller you don't write
- Faces files generally use ".xhtml", but can also use ".jsf"



```
To-Do - Faces
\\Galatea-VCC/IKSG\todo\faces.nsf
Forms
Views
Folders
XPages
Custom Controls
Framesets
Pages
Shared Elements
Code
    Agents
    Shared Actions
    Script Libraries
    Database Script
    Web Service Providers
    Web Service Consumers
    Java
        controller/ToDosController.java
        model/CurrentRepositoryProvider.java
        model/ToDo.java
    Jars
Data
Resources
    Images
    Files
        index.xhtml
        todos.xhtml
        WEB-INF/common/linksbar.xhtml
```

PROJECT INFORMATION

Project Information

- https://github.com/OpenNTF/org.openntf.xsp.jakartaee/
- https://www.openntf.org/main.nsf/project.xsp?r=project/XPages%20Jakarta%20EE%20Support
- YouTube series: https://www.youtube.com/playlist?list=PLaDSIoof-i96Nhho68wFsacBwwkCAmmVh

Requirements and Compatibility

- Domino 9.0.1FP10 for most pieces, Domino 12.0.1+ with FPs for NoSQL
- Should work with most or all existing libraries
  - Used in production alongside ODA and POI4XPages
- Can be used in OSGi bundles with some knowledge

Getting Involved

- Try it out!
  - Talk about it in the OpenNTF Discord: https://openntf.org/discord
- Report bugs and request features
- Documentation: guides, specific feature details, etc.
- Example applications
  - https://github.com/OpenNTF/org.openntf.xsp.jakartaee/issues/307
- Chip in on the code directly

THANK YOU
+ QUESTIONS