OPENNTF WEBINARS

August, 2022 OpenNTF Webinar

XPages Jakarta EE Support In Practice

AGENDA

- Welcome
- Presentation – Jesse Gallagher
- Q and A - All

THANKS TO THE OPENNTF SPONSORS

- HCL made a contribution to help our organization
  - Funds these webinars!
  - Contests like Hackathons
  - Running the organization
- Prominic donates all IT related services
  - Cloud Hosting for OpenNTF
  - Infrastructure management for HCL Domino and Atlassian Servers
  - System Administration for day-to-day operation

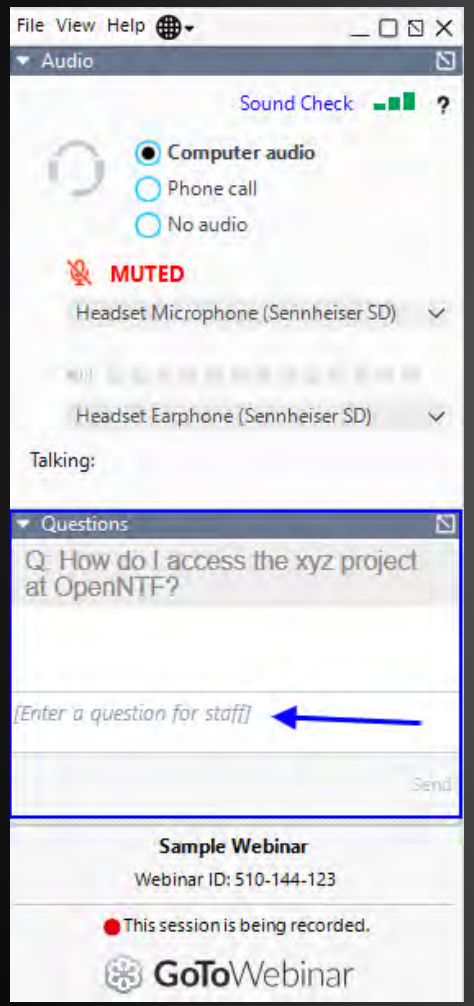
THIS IS OUR COMMUNITY

- Join us and get involved!
- We are all volunteers
- No effort is too small
- If your idea is bigger than you can do on your own, we can connect you to a team to work on it
- Test or help or modify an existing project
- Write guides or documentation
- Add reviews on projects / stars on Snippets

NEXT WEBINAR

- Watch https://www.openntf.org/webinars for more information

ASKING QUESTIONS



- First Question – Will this be recorded?
  - Yes, view on YouTube!!!
  - https://www.youtube.com/user/OpenNTF
- Use the Questions Pane in GoToWebinar
- We will get to your questions at the end of the webinar
- The speakers will respond to your questions verbally
  - (not in the Questions pane)
- Please keep all questions related to the topics that our speakers are discussing!!!
- Unrelated Question => post at:
  - https://openntf.org/discord

XPAGES JAKARTA EE SUPPORT IN PRACTICE

Jesse Gallagher

XPAGES JAKARTA EE IN PRACTICE

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

JESSE GALLAGHER

AGENDA

- What are Jakarta EE and MicroProfile?
- What is the XPages Jakarta EE Support project?
- Components:
  - Expression Language
  - Managed Beans (CDI)
  - Data access
  - Producing REST Services
  - Consuming REST Services
- User Interface Options

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

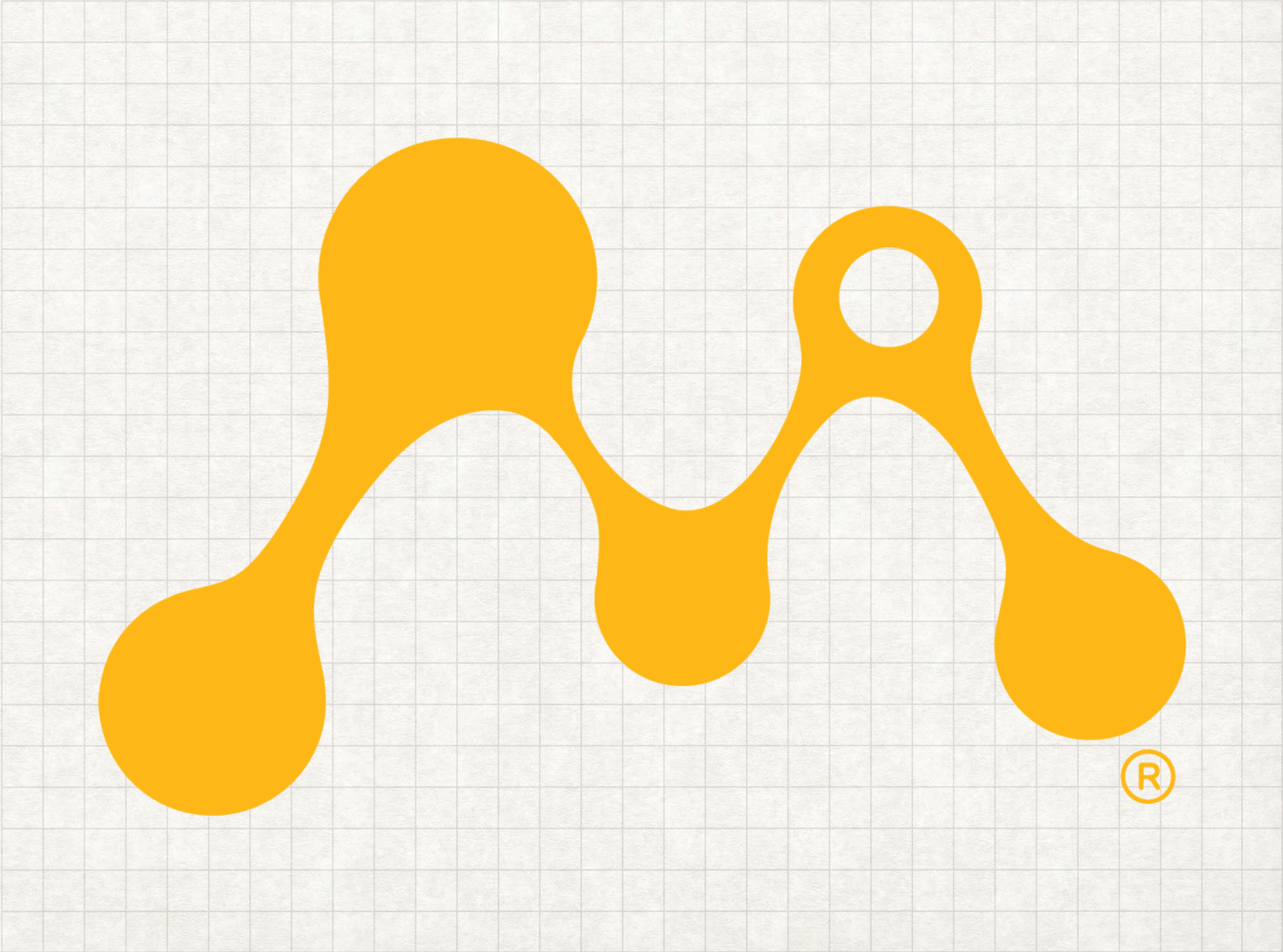
JAKARTA EE AND MICROPROFILE

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, releasing this month, makes new spec changes and moves to Java 11
- 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

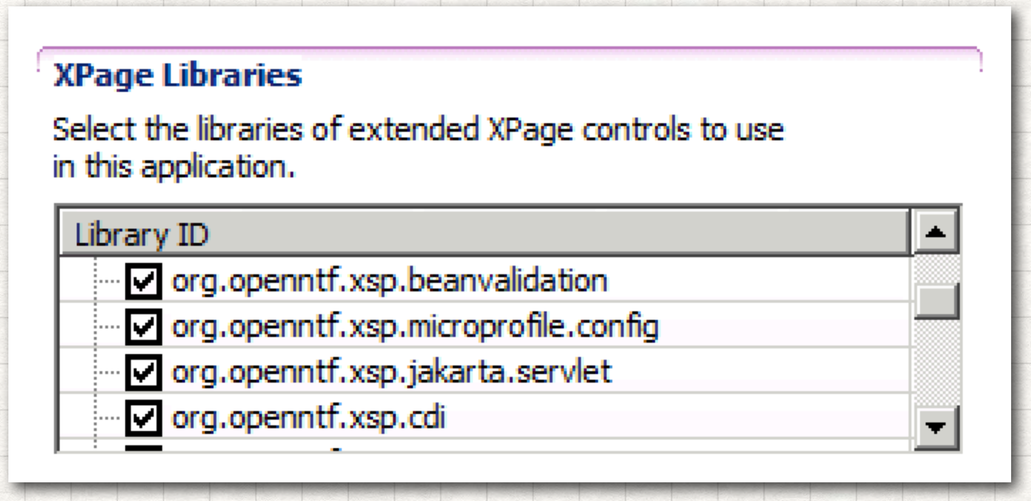
XPAGES JAKARTA EE SUPPORT

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)



EXAMPLES

- Almost all code in this presentation is from the in-development OpenNTF home DB
  - It's not publicly available yet, but I'll aim to make it so
- The XPages JEE project contains a DB in eclipse/nsfs/nsf-example, though it's a bit packed
  - (It doubles as the DB for the integration-test suite)
- Fortunately, most examples online of each spec should work - JAX-RS here is the same JAX-RS as on Stack Overflow

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, including SSJS
- This EL interpreter is stricter about nulls, which is actually useful
- No configuration necessary: enable the library and it will take over

WHAT YOU GET

- All the same stuff as before!
  - `#{foo.bar}`, `#{foo[bar]}`, etc.
- Function calls
  - `${el:messages.format('helloMessage', session.effectiveUserName)}`
  - The "el:" prefix avoids an error marker in Designer
- String concatenation
  - `${'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

MANAGED BEANS

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

```
@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:
  - JAX-RS
  - 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

JAX-RS (REST)

JAX-RS

- JAX-RS, officially "Jakarta RESTful Web Services" or "Jakarta REST", is a long-standing framework for REST services
- Primarily serves JSON, but can work with anything
- Domino ships with an ancient implementation - Wink - that powers DAS in the ExtLib
- JAX-RS focuses on using annotations and implicit conversion to keep code clean and meaningful

JAX-RS EXAMPLE

```java
@Path("/config")
public class ApplicationConfigResource {

  // CDI managed bean
  @Inject
  ApplicationConfig config;

  @GET
  @Produces(MediaType.APPLICATION_JSON)
  public ApplicationConfig get() {
    // The @Produces above causes automatic
    // JSON conversion
    return config;
  }

}
```

curl http://server.com/foo.nsf/xsp/app/config | jq


```
{
  "blogDbPath": "blogs\\openntf.nsf",
  "configEntries": {
    "links_resources": {
      "comments": "",
      "key": "links_resources",
      "name": "links_resources",
      "value1": [
        "Twitter",
        "YouTube",
        "Facebook",
        "OpenNTF Guides",
        "XPages Downloads & Resources",
        "OpenNTF Stash",
        "OpenNTF JIRA",
        "OpenNTF Confluence",
        "OpenNTF GitHub",
        "OpenNTF Connections GitHub",
        "IBM Connections Downloads",
        "DominoHelp (external)"
      ],
      "value2": [
        "https://twitter.com/openntf",
        "http://www.youtube.com/openntf",
        "http://www.facebook.com/pages/OpenNTF/159464360754911?ref=ts",
        "https://wiki.openntf.org/display/OPH/Community+Guides",
        "https://xpages.info",
        "https://stash.openntf.org/",
        "https://jira.openntf.org/",
        "https://wiki.openntf.org",
```

JAX-RS EXAMPLE - POSTING FORMS

```
// Takes a standard HTML form format and returns JSON
// URL like "/foo.nsf/xsp/app/people/create"
@Path("create")
@POST
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces(MediaType.APPLICATION_JSON)
public Person createPerson(
    @FormParam("firstName") @NotEmpty String firstName,
    @FormParam("lastName") String lastName
) {
  Person person = new Person();
  person.setFirstName(firstName);
  person.setLastName(lastName);

  return personRepository.save(person);
}
```

JAX-RS EXAMPLE - POSTING JSON

```
// Consumes and returns JSON, validating the object on input
// URL like "/foo.nsf/xsp/app/people/some-person-id"
@Path("{id}")
@PUT
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Person createJson(@PathParam("id") String id, @Valid Person person) {
  person.setUnid(id);
  return personRepository.save(person, true);
}
```

RESOURCES

- https://jakarta.ee/specifications/restful-ws/3.0/
- https://www.baeldung.com/eclipse-microprofile
- https://openliberty.io/guides/rest-intro.html

MICROPROFILE REST CLIENT

MICROPROFILE REST CLIENT

- Uses JAX-RS annotations to make it easy to access remote services
- Pairs with JSON-B to translate between remote JSON and local Java classes
- Tools like https://openapi-generator.tech/ can generate bindings for it automatically
  - (These may need translation from javax.* to jakarta.*)

MICROPROFILE REST CLIENT

restclient/GitHubIssues.java

```java
@RegisterRestClient(baseUri="https://api.github.com")
@Path("repos/{owner}/{repo}/issues")
public interface GitHubIssues {
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    List<Issue> get(
        @PathParam("owner") String owner,
        @PathParam("repo") String repo
    );

    class Issue {
        private int id;
        private String url;
        private String title;
        private String state;
        @JsonbProperty("created_at")
        private Date created;


        // Getters and setters
    }
}
```

bean/IssuesBean.java

```java
@ApplicationScoped
@Named
public class IssuesBean {

   @Inject
   private GitHubIssues client;

   public List<GitHubIssues.Issue> get(String owner, String repo) {
      if(StringUtil.isEmpty(owner) || StringUtil.isEmpty(repo)) {
         return Collections.emptyList();
      }
      return client.get(owner, repo);
   }
}
```

gitHubIssues.xsp

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

RESOURCES

- https://github.com/eclipse/microprofile-rest-client/releases/tag/3.0
- https://openliberty.io/guides/microprofile-rest-client.html

JAKARTA NOSQL

JAKARTA NOSQL

- Jakarta NoSQL is a beta specification not yet officially included in JEE releases
- It's meant to be similar to JPA, but suited to various kinds of NoSQL databases
- Thanks to DQL, Domino is now a practical data source for it
- Provides standard behavior for databases, but encourages per-DB customization
- The Domino driver is extended with support for item flags, views, etc.
- https://jakarta.ee/specifications/nosql/1.0/
- https://www.baeldung.com/eclipse-jnosql

ENTITY OBJECTS

```java
@Entity("Project") // Maps to Form value
public class Project {
   @RepositoryProvider("projectsRepository") // Pull from a different NSF
   public interface Repository extends DominoRepository<Project, String> {
      // Auto-synthesized query based on method name
      Optional<Project> findByProjectName(String projectName);
   }

   @Id
   private String id;
   @Column("ProjectName")
   private String name;
   @Column("ProjectOverview")
   private String overview;
   @Column("Details")
   @ItemStorage(type=ItemStorage.Type.MIME) // Domino-specific extension
   private String details;
   @Column("DownloadsProject")
   private int downloads;
   @Column("MasterChef")
   private List<String> chefs;
   @Column("Entry_Date")
   private OffsetDateTime created;

   // Getters and setters
}
```

USING A REPOSITORY

```
@Inject
Project.Repository projectRepository;

@Path("{projectName}")
@GET
@Produces(MediaType.APPLICATION_JSON)
public Project getProject(@PathParam("projectName") String projectName) {
  String key = projectName.replace('+', ' ');

  // java.util.Optional includes .orElseThrow(...), perfect for this case.
  // Throwing NotFoundException leads to a 404
  Project project = projectRepository.findByProjectName(key)
    .orElseThrow(() -> new NotFoundException("Unable to find project for name: " + key));

  return project;
}
```

USING REPOSITORIES

- By default, JNoSQL repositories have a few methods for CRUD operations
- DominoRepository adds a few more, such as methods to add/remove from folders and options to call computeWithForm on save
- The Domino driver also includes a number of extensions for reading from views

USER INTERFACE

OPTION 1: XPAGES

- XPages works as well as ever in an NSF using these libraries
- Applicable specs work here: Expression Language, CDI beans, MP REST Client, etc.
- Other than EL improvements, the act of writing XSP markup is the same, with the same components and capabilities
- XPages can work alongside JAX-RS and the other UI technologies without issue
  - JAX-RS can be a bit more pleasant than the ExtLib components

OPTION 2: REST + CLIENT JS

- You can write all of your server logic in JAX-RS
- Use React, Angular, vanilla JS, etc.
  - Heck, use C if you want to
- The app could live outside of the NSF or inside as design elements
  - (Try the NSF ODP Tooling project for automated-build options!)
- Inside an NSF, you can enforce access with an ACL and share the login with pages

OPTION 3: MVC + JSP

- MVC is a newer spec, not in the full release but not in beta
- It builds on top of JAX-RS
- It's an action-oriented framework, as opposed to XPages's component-based approach
- In general, you're "closer to the metal"
- MVC can work with multiple UI techs, but JSP is in this project

OPTION 3: MVC + JSP

controller/HomeController.java

```java
// URL like /foo.nsf/xsp/app
@Path("/")
@Controller
public class HomeController {

   @Inject Models models;

   @Inject ProjectReleases releases;

   @Inject BlogEntries blogEntries;

   @GET
   @Produces(MediaType.TEXT_HTML)
   public String get() {
      // Put objects you need in "models", like viewScope
      models.put("recentReleases", releases.get(30));
      models.put("blogEntries", blogEntries.getEntries(5));

      // Return the name of the JSP file to render
      return "home.jsp";
   }

}
```

WebContent/WEB-INF/views/home.jsp

```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="t" tagdir="/WEB-INF/tags" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<t:layout>
   <section class="main-content">
      <div class="home-layout">
         <section id="blog">
            <c:forEach items="${blogEntries}" var="entry">
               <t:blogEntry value="${entry}"/>
            </c:forEach>
         </section>
         <section id="recent-releases" class="activity-feed">
            <h2><c:out items="${translation.recentReleases}"/></h2>

            <ol>
               <c:forEach items="${recentReleases}" var="release">
                  <!-- snip -->
               </c:forEach>
            </ol>
         </section>
      </div>
   </section>
</t:layout>
```

FUTURE OPTIONS

- XPages + MVC?
  - I did an early trial, but there are parts of the XPages stack that need workarounds
- Jakarta Faces (JSF)?
  - JSF 3.0 is present in the project, but not PrimeFaces or Apache Tobago
  - It generally works as-is, but doesn't have a lot of niceties
- Other view engines, like Thymeleaf?
  - MVC has extensions for several of these, so I may bring them in

RESOURCES

- https://jakarta.ee/specifications/mvc/2.0/
- https://www.baeldung.com/java-ee-mvc-eclipse-krazo
- https://jakarta.ee/specifications/faces/3.0/

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!
- 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

QUESTIONS?

Use the GoToWebinar Questions Pane

Please keep all questions related to the topics that our speakers are discussing!!!

Unrelated Question => post at:

https://openntf.org/discord