



adaptTo()

EUROPE'S LEADING AEM DEVELOPER CONFERENCE

27th – 29th SEPTEMBER 2021

Promoting Solid Code Architecture in AEM

Daniel Strmečki, ecx.io - part of IBM iX



Director
Digital Platforms
at ecx.io,
part of IBM iX

- Web development, Java, AEM, software craftsmanship, testing, reusability, architecture, coaching
 - <https://www.linkedin.com/in/strmecki/>
 - <https://www.baeldung.com/author/danielstrmecki/>
 - daniel.strmecki@ecx.io

Agenda

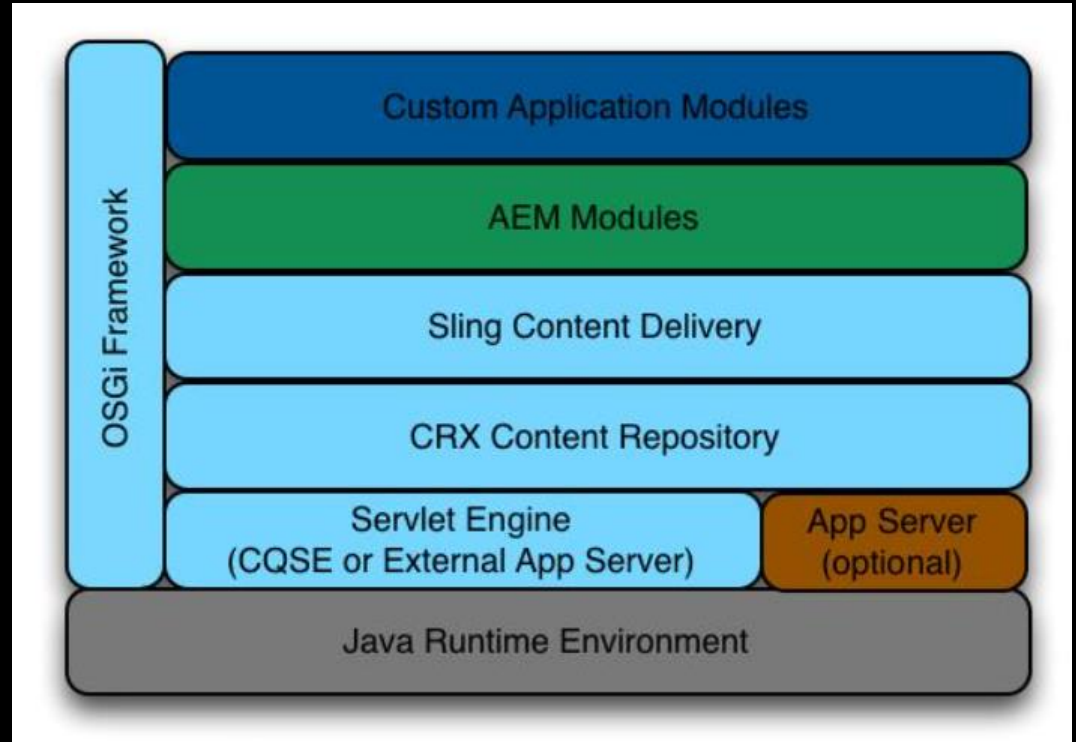
- 1) Learning AEM
- 2) Architecture Patterns
- 3) The Glue for Our Code
- 4) Architecture Rules
- 5) Best Practices
- 6) Coding Guidelines
- 7) Custom Quality Rules
- 8) Conclusion
- 9) Questions



Learning AEM

- Adobe Experience Manager is not the simplest Web development framework to master
 - Sling works quite differently compared to most popular choices like Spring or Java EE
 - You cannot simply apply MVC pattern
 - You won't find a controller Java class

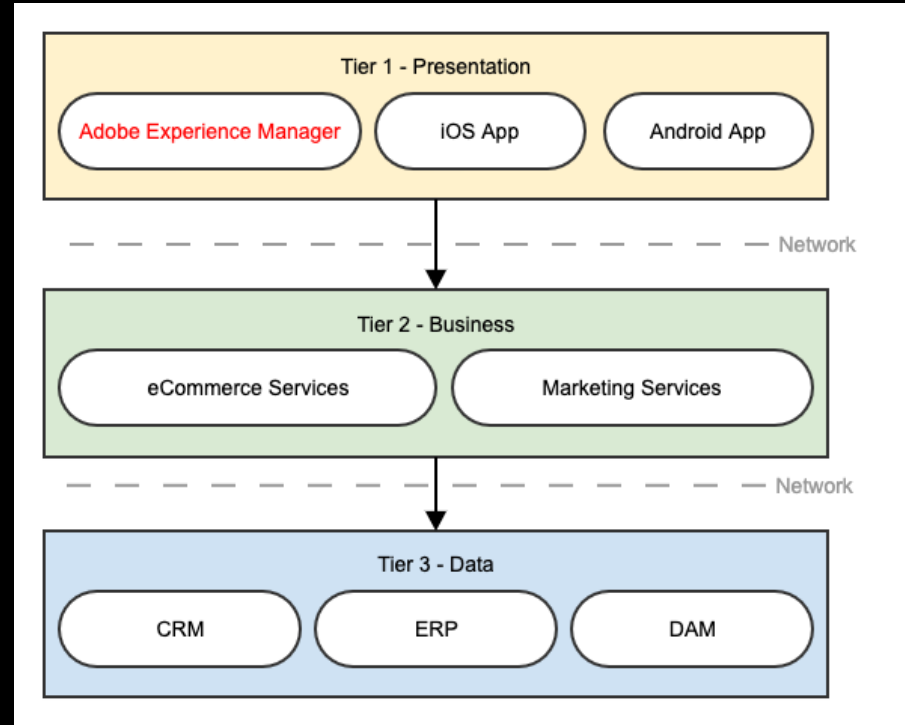
- Adobe teaches us AEM runtime architecture
- But what does that mean for our code?



Architecture Patterns

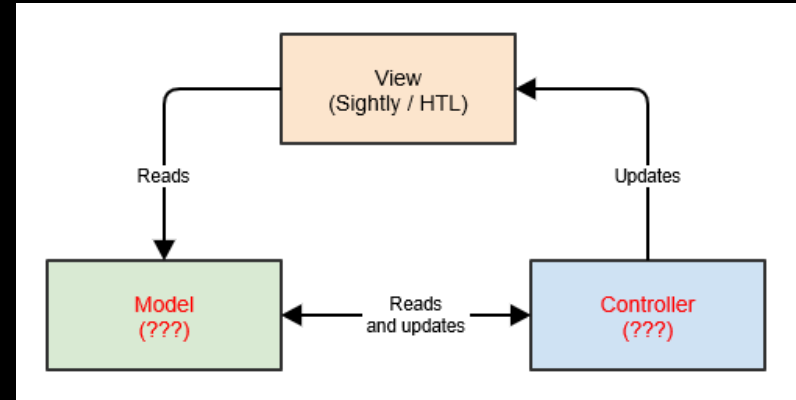
The N-Tier Pattern

- In an enterprise architecture, AEM is usually in the top layer / tier (presentation)



The MVC Pattern

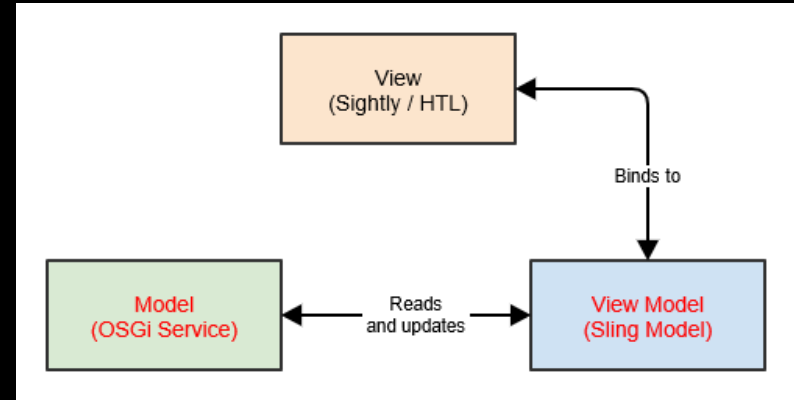
- MVC cannot be applied to AEM components
- Sling Models are not Controllers, because Apache Sling handles script resolution



- ✓ io.ecx.cms
 - ✓ controllers
 - CollageController
 - ✓ models
 - CollageModel

The MVVM Pattern

- The View (HTL) binds to the methods provided by the View-Model (Sling Model)
- The View-Model (Sling Model) calls the business logic implemented in dedicated components (OSGi Services)





The Glue for Our Code

- Sling Models are the glue between our UI, database and business logic
- Heavily misused on a lot of AEM projects
 - Can easily become very large if separation of concerns is not applied properly
 - Injecting data, providing view methods, implementing logic, injecting other models

Example – Sling Models

- Connect data with business logic
- Keep it simple
- Move business logic to dedicated OSGi services

```
@Slf4j
@Model(adaptables = SlingHttpServletRequest.class,
      defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL)
public class ArticlesListModel {

    @SlingObject
    private ResourceResolver resourceResolver;

    @ScriptVariable
    private Page currentPage;

    @OSGiService
    private TextExcerptsService textExcerptsService;

    @OSGiService
    private ImageFinderService imageFinderService;

    @OSGiService
    private LinkService linkService;

    @Getter
    @ValueMapValue
    private List<ArticleItem> articles;
```

Example – OSGi Services

- Highly reusable
- Easily testable
 - TDD
 - BDD

```
@Component(service = TextExcerptsService.class, immediate = true)
public class TextExcerptsServiceImpl implements TextExcerptsService {

    @Override
    public String getTextExcerpts(final Page articlePage, final int maxLength) {
        final List<Resource> mainContentResources = ResourceUtils.getMainContentResources(articlePage);
        return mainContentResources
            .stream() Stream<Resource>
            .filter(resource -> resource.getResourceType().equals(TEXT_COMPONENT_RESOURCE_TYPE))
            .findFirst() Optional<Resource>
            .map(resource -> resource.adaptTo(TextModel.class)) Optional<TextModel>
            .map(TextModel::getText) Optional<String>
            .map(text -> StringUtils.abbreviate(htmlTagsPattern
                .matcher(text)
                .replaceAll(StringUtils.EMPTY), maxLength))
            .orElse(StringUtils.EMPTY);
    }
}
```

Architecture Rules

- Library for checking the architecture of your Java code using JUnit
 - <https://www.archunit.org/>
- ArchUnit can check dependencies between packages and classes, layers and slices, check for cyclic dependencies and more
 - <https://github.com/dstr89/aem-archunit>

Example – OSGi Services

- Only interfaces allowed
- Force a naming convention
- Classes are annotated

```
classes()
    .that().resideInAPackage(SERVICES_PACKAGE)
    .should().beInterfaces();

classes()
    .that().resideInAPackage(SERVICES_PACKAGE)
    .should().haveSimpleNameEndingWith(s: "Service");

classes()
    .that().resideInAPackage(SERVICES_IMPL_PACKAGE)
    .should().beAnnotatedWith(Component.class);
```

Example – Sling Models

- Dependencies between packages
- No static methods allowed

```
noClasses()
    .that()
    .resideInAnyPackage(COMPONENTS_PACKAGE, MODELS_PACKAGE)
    .should()
    .dependOnClassesThat()
    .resideInAPackage(SERVLETS_PACKAGE);

methods()
    .that().areDeclaredInClassesThat()
    .resideInAnyPackage(COMPONENTS_PACKAGE, MODELS_PACKAGE)
    .should().notBeStatic()
    .orShould()
    .haveModifier(JavaModifier.SYNTHETIC);
```

Best Practices

Value of Documentation

- Agile value misconception
 - „Working software over comprehensive documentation”
- Make notes on the good practices that you see on different projects
 - We always refer to previous projects and lessons learned

Example – OSGi Services

- Place business logic in Services, not Utils
 - Single Responsibility and Open-Closed Principle
- Prefer Sling APIs over JCR
- Close service resource resolver

```
try (ResourceResolver resourceResolver = getServiceResourceResolver()) {  
    // Use resource resolver here  
} catch (RuntimeException e) {  
    log.error("Error creating base names", ex);  
}
```

Example – Sling Models

- Use injector-specific annotations
 - Instead of using *@Inject* everywhere
- Log runtime exceptions in *@PostConstruct*

```
try {  
    this.query = Optional.ofNullable(this.request.getParameter(PARAMETER_QUERY)).orElse(StringUtils.EMPTY);  
    this.articles = this.searchArticles();  
} catch (final RuntimeException e) {  
    log.error("Error while initiating model", e);  
}
```



Example – Avoid *adaptTo*

- Doesn't throw exceptions, harder to mock

```
ValueMap valueMap = this.resource.getValueMap(); // Recommended
ValueMap valueMap = this.resource.adaptTo(ValueMap.class); // Not recommended

@Reference
private PageManagerFactory pageManagerFactory;

PageManager pageManager = pageManagerFactory.getPageManager(resourceResolver); // Recommended
PageManager pageManager = resourceResolver.adaptTo(PageManager.class); // Not recommended

@Reference
private QueryBuilder queryBuilder;

this.queryBuilder.createQuery(predicates, session); // Recommended
QueryBuilder queryBuilder = resourceResolver.adaptTo(QueryBuilder.class); // Not recommended
```

Coding Guidelines

Promoting Best Practices

- With time, a set of notes can become company coding guidelines
 - Share them with everyone in the company
 - Anyone can and should contribute
 - Use them to align with other developers
 - Use them to coach junior developer

Benefits of Guidelines

- Boost quality and align codebases on all your AEM projects
- Learn on mistakes others made
- Easy to switch people on projects
- Brings competitive advantage through quality KPIs and expanding existing client engagements

- ▼ AEM - Coding Guidelines
 - 01 Java Coding Guidelines
- ▼ 02 AEM Coding Guidelines
 - 02-01 Sling Models
 - 02-02 OSGi Services
 - 02-03 Sling Servlets
 - 02-04 Sling Filters
 - 02-05 Naming Conventions
 - 02-06 Components
 - 02-07 Sightly / HTL
 - 02-08 Dialogs
- 03 Testing Guidelines

Custom Quality Rules

- Coding guidelines can be automated using a set of custom SonarQube rules
- Automated checks, together with pull-request decoration feature, helps us ensure that guidelines really do get applied on all projects
 - If it not automated, it probably won't be used

Example – Rule Definition

```
@Override
public void visitMethod(@NonNull final MethodTree methodTree) {
    if (JavaFinder.containsAnnotation(methodTree, PackageConstants.POST_CONSTRUCT_ANNOTATION)) {
        final BlockTree blockTree = methodTree.block();
        if (blockTree != null) {
            final List<StatementTree> blockElements = blockTree.body();
            if (isTryStatementFirst(blockElements)) {
                this.context.reportIssue( javaCheck: this, methodTree.simpleName(),
                    s: "Wrap all the code in a try-catch clause");
            } else {
                final TryStatementTree tryStatementTree = (TryStatementTree) blockElements.get(0);
                this.checkCaughtExceptions(tryStatementTree);
            }
        }
    }
    super.visitMethod(methodTree);
}
```

Example – Rule Usage

- SonarLint
- BitBucket

```
@PostConstruct
public void init() {
    final PageManager pageManager = resourceResolver.adaptTo(PageManager.class);
    this.homePage = pageManager.getPage("home");
}
```

SonarLint: Wrap all the code in a try-catch clause

SonarLint: Show rule description 'ecxio-aem:PostConstructException'

core / src / main / java / io / ecx / aem / ktm / website2020 / core / servlets / BikeModelsServlet.java **MODIFIED**

81 +
82 +

```
final PageManager pageManager = resourceResolver.adaptTo(PageManager.class);
```

⌵ Low · Code smell · Avoid using adaptTo method [View report](#)

Conclusion

- AEM doesn't follow the MVC pattern, instead it more aligned with the MVVM
- Define a clear separation of concerns
 - Just because we can write all the code in the Sling Model, doesn't mean we should
 - Use OSGi Services as much as possible, as they promote reusability and testability

- Coding guidelines help boost quality and alignment between project teams
 - Architectural and SonarQube rules ensure that guidelines actually do get applied on projects
- Creating internal guidelines and assets requires a significant team effort, but it pays off in the long term



Thank you! Questions?