1. **6.1 to 6.5 key features**

* ***Adobe Experience Manager (AEM) 6.1 2015***

@ Improved Touch UI Authoring interface (usable)

@ Introduced Cold standby Topology for disaster recovery.

@ Introduced transient workflows.

@ Lot of improvements for Touch UI Interface.

* ***Adobe Experience Manager (AEM) 6.2 2016***

@ Lot of User Interface Improvements.

@ Dashboard for Monitoring Queues.

@ Introduction of content fragments.

@ Introduced editable templates using template editor.

* ***Adobe Experience Manager (AEM) 6.3 2017***

@ Introduction to Production-ready components (Core Components)

@ Experience fragments

@ Stable online revision Clean up(online tar compaction)

@ Bulk workflow

@ Performance and scale enhancements

@ Community ideation

@ Source user generated content

@ 3D assets

@ Asset templates

@ High-quality file format and color management

* ***Adobe Experience Manager (AEM) 6.4 2018***

@ Style System

@ Short Codes

@ Experience Fragments Building Blocks and Experience Fragments integration with Adobe Target

@ Structured Content Fragments and Summarization

@ TouchUI Workflow Editor

@ Metadata Export and Import

@ Cascading Metadata Rules

@ Asset Reports

@ AEM Smart Tags and Smart Translation Search

1. **AEM technology stack**

* ***Apache Sling RESTful Application Framework:***

RESTful framework to access a jcr over http protocol. It maps the request url to the node in jcr.

* ***OSGi(Apache Felix):***

Framework for modular application development using java. Each module called bundle can be independently started and stopped. OSGi container which provides implemention classes for OSGi framework

* ***Java Content Repository (JCR):***

Java specification for accessing a content repository JSR-283 specification jcr 2.0 , cq5 uses its own implementation of jcr called CRX. Apache Jackrabbit is an open-source implementation of jcr 2.0 specification. Java Content Repository (JCR) persistence layer with Access Control, Versioning, Search Observation etc. Implementation of Apache Jackrabbit Oak, is various persistence options supported such as TAR files, MongoDB, Relational Database and S3.

1. **Layers present in OSGI**

* "***Bundles***" – Bundles are normal jar components with extra manifest headers.
* "***Services***" – The service layer, which hold the service-side of the framework, keeps the service registry and manages it.
* "***Life-Cycle***" – The lifecycle layer manages and keeps track of the frameworks and bundles lifecycle state. It is used to install or uninstall framework objects and start or stop them.
* "***Modules***” – The module layer, which is the bundle space, holds the bundles that are installed on the framework and are managed through the lifecycle layer.
* "***Security***" – The security layer, which extends the jave 2 security architecture, is optional. When active, it validate the bundle signatures and controls the component access rights .
* "***ExecutionEnv***" – The execution environment layer, which is the bottom layer on which the bundles live, is selected to fit the underlying hardware or operating system.

1. **Benefits of OSGI**

Advantages over using standalone jars.

* "***Reduced Complexity***" – Developing with OSGi technology means developing bundles. They hide their internals from other bundles and communicate through well-defined services. It makes bundles simpler to develop because correctly sized bundles implement a piece of functionality through well-defined interfaces.
* "***Reuse***" – The OSGi component model makes it very easy to use many third-party components in an application. An increasing number of open source projects provide their JARs ready-made for OSGi. We just need to install them in our application OSGi container.
* "***Real World***" – The OSGi framework is dynamic. It can update bundles on the fly and services can come and go.
* "***Dynamic Updates***" – The OSGi component model is a dynamic model. Bundles can be installed, started, stopped, updated, and uninstalled without bringing down the whole system.
* "***Versioning***" – OSGi technology solves JAR hell. JAR hell is the problem that library A works with library B;version=2, but library C can only work with B;version=3. In standard Java, you’re out of luck. In the OSGi environment, all bundles are carefully versioned and only bundles that can collaborate are wired together in the same class space. This allows both bundle A and C to function with their own library.

1. **Difference between bundle & Jar file**

* OSGi bundles are jar files with metadata inside. Much of this metadata is in the jar’s manifest, found at META-INF/MANIFEST.MF. This metadata, when read by an OSGi runtime container, is what gives the bundle its power.
* With OSGi, just because a class is public doesn’t mean you can get to it. All bundles include an export list of package names, and if a package isn’t in the export list, it doesn’t exist to the outside world. This allows developers to build an extensive internal class hierarchy and minimize the surface area of the bundle’s API without abusing the notion of package-private visibility. A common pattern, for instance, is to put interfaces in one package and implementations in another, and only export the interface package.
* All OSGi bundles are given a version number, so it’s possible for an application to simultaneously access different versions of the same bundle (eg: junit 3.8.1 and junit 4.0.). Since each bundle has it’s own classloader, both bundles classes can coexist in the same JVM.
* OSGi bundles declare which other bundles they depend upon. This allows them to ensure that any dependencies are met before the bundle is resolved(that are activated). Because bundles have versions, versioning can be included in the dependency specification, so one bundle can depend on version junit version 3.8.1 and another bundle depend on junit version 4.0.
* In OSGi bundle, there will be an Activator.java class in OSGi which is an optional listener class to be notified of bundle start and stop events.

1. **Sling Resolution:**

The priority of resolution in Descending order(Highest to lowest):

Selector +Extension.jsp –> test.myComponent.html.jsp

Selector.jsp –> test.jsp

Extension.jsp –> html.jsp

Node Name.jsp –> myComponent.jsp

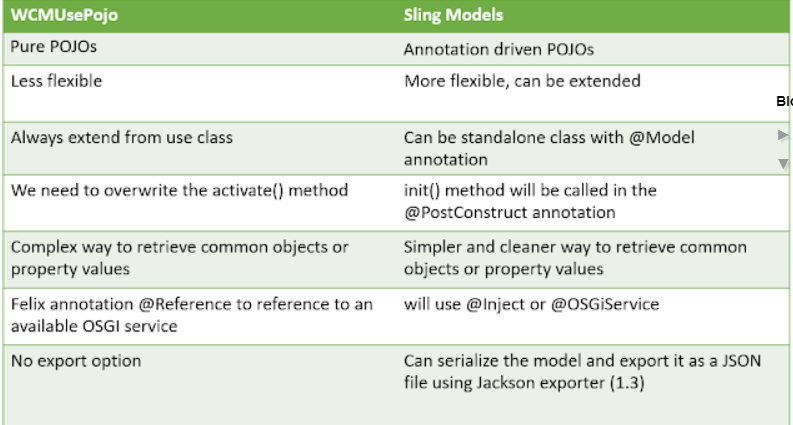
method name.jsp -> (Based on type of request we make Either GET or POST) GET.jsp or POST.jsp

1. **Sling Model and its benefit when compared to wcmpojo class**

Sling Models are "pure" POJOs which maps Sling objects (resources, request objects etc.).

Since Sling Models are annotation-driven Plain Old Java Objects (POJOs), annotations are used a lot.

They allow you to map resource properties, assign default values, inject OSGi services and much more.



1. **Sling api’s used in your projects**
2. **Dialog & Design dialog in AEM**

* ***Dialog*** :  -> Page level changes
* These components will be available in the sidekick. if not can be added to the sidekick
* These components will be available in the edit mode.
* Gives control for the authors to use/delete these components in the page.
* ***Design dialog*** : -> Template level changes
* These components are available will not be available for the authors in sidekick and will not be able to add them to the page.
* These component should be included directly in the page component (Template ref.) to use.
* These components can be edited only in the design mode. By default only admin will have to access to this.

1. **Overlay and Override**

* ***Extend and Overlay*** involves copying the component from /libs/ (or other base library) to /apps node and changing the behaviour.
* ***Override*** is similar to extend but the only difference is that you change the sling:resourceSuperType of the component so that it's behaviour completely changes (hence overridden).

1. **ResourceType & ResourceSuperType**

* ***sling:resourceType*:** It is a path, which locates the script to be used for rendering the content. Path used can be absolute or relative.
* ***sling:resourceSuperType***: It is used to achieve inheritance in cq. When set, it inherits the specified component to this component.

1. **Editable & Static Templates**

* ***AEM Static Templates***
* Static templates are created by developers ( It defines page rendering component , availability of template for authors to create the page )
* Stored in /apps directory of CRX
* Once a page is created using templates, after that if we make any change in static template it wouldn't affect the page.
* Design mode is available for applying Page design.
* ***AEM Editable Templates***
* Editable Templates can be created by template-authors ( non-developers ).
* Stored in /conf directory of CRX
* Pages created using editable templates would affect if any changes on editable templates are done.
* Page design is controlled by content and page policies. Hence no design mode available.

1. **Page policies can be set in editable template but not in static template**
2. **Implicit objects in sightly**

* ***Enumerable objects*** – properties, pageProperties, inheritedPageProperties
* ***Java Backed objects*** – component, currentPage,currentDesign, resource, resourceDesign, resourcePage, wcmmode, request,
* ***JavaScript Backed objects*** - It is possible to back HTL logic with JavaScript. However the preferred or recommended method is by using [Sling Models.](https://sling.apache.org/documentation/bundles/models.html)

1. **Sightly tags you have used in your project**

* ***Data-sly-resource*** - includes the result of rendering the indicated resource through the sling resolution and rendering process.
* ***data-sly-test*** - conditionally removes the host element and it’s content. A value of false removes the element; a value of true retains the element.
* ***data-sly-list*** - repeats the content of the host element for each enumerable property in the provided object.
* ***data-sly-repeat*** - you can repeat an element multiple times based on the list that is specified.
* **data-sly-include** - replaces the content of the host element with the markup generated by the indicated HTML template file (HTL, JSP, ESP etc.) when it is processed by its corresponding template engine. The rendering context of the included file will not include the current HTL context (that of the including file)
* ***data-sly-use*** - initializes a helper object (defined in JavaScript or Java) and exposes it through a variable.

1. **How to register servlet in AEM**

* ***sling.servlet.paths***: A list of absolute paths under which the servlet is accessible as a Resource. The property value must either be a single String, an array of Strings or a Vector of Strings.

A servlet using this property might be ignored unless its path is included in the Execution Paths (servletresolver.paths) configuration(Apache Sling Servlet/Script Resolver and Error Handler) setting of the SlingServletResolver service.

* ***sling.servlet.resourceTypes***: The resource type(s) supported by the servlet. The property value must either be a single String, an array of Strings or a Vector of Strings.

Note: Either this property(sling.servlet.resourceTypes) or the sling.servlet.paths property must be set, or the servlet is ignored. If both are set, the servlet is registered using both ways.

1. **Difference b/w SlingSafeMethod & SlingAllMethod Servlet**

* ***public class SlingAllMethodsServlet extends SlingSafeMethodsServlet***
* Helper base class for data modifying Servlets used in Sling. This class extends the SlingSafeMethodsServlet by support for the POST, PUT and DELETE methods.
* ***public class SlingSafeMethodsServlet extends javax.servlet.GenericServlet***
* Helper base class for read-only Servlets used in Sling. This base class is actually just a better implementation of the Servlet API HttpServlet class which accounts for extensibility. So extensions of this class have great control over what methods to overwrite.
* If any of the default HTTP methods is to be implemented just overwrite the respective doXXX method. This servlet by itself does not support the POST, PUT and DELETE methods. Extensions of this class should either overwrite any of the doXXX methods of this class or add support for other read-only methods only.

1. **Which is preferred(Registering servlet by path or ResourceType)**

Binding servlets by paths have several disadvantages when compared to binding by resourceTypes:

* Path bound servlets cannot be access-controlled using the default JCR repository ACLs while the resourceType bound servlets can be controlled.
* path-bound servlets can only be registered to a path and not a resource type (i.e. no suffix handling)
* If a path bound servlet is not active, e.g. if the bundle is missing or not started, a POST request might result in unexpected results, usually creating a node at /bin/xyz or many such issues.
* The mapping is not transparent to the developer who is just looking at the repository.
* When we register a servlet using path, we must be specific what all paths are allowed as if we define something randomly, our servlet might not work or give unexpected results.
* Only a limited set of paths are allowed and rest are blocked. Though we can add more paths in the OSGi configuration Apache Sling Servlet / Script Resolver and Error Handler.

1. **What are the annotations to be used to create a service**

* ***@Component*** – defines the class as a component
* ***@Service*** - defines the service interface that is provided by the component

1. **How to call service in our class**

* Use SCR annotations to let SCR inject the service in your component:(Generally in Servlet)  
  ***@Reference  
  private MyService myService;***
* Use Bundle Context to get the service in your Java/Jsp file  
  ***BundleContext bundleContext = FrameworkUtil.getBundle(MyService.class).getBundleContext();  
  ServiceReference factoryRef = bundleContext.getServiceReference(ResourceResolverFactory.class.getName());  
  ResourceResolverFactory resolverFactory = (ResourceResolverFactory) bundleContext.getService(factoryRef);***
* Use sling.getService() method(Generally in JSP)  
  ***MyService service=sling.getService(MyService.class);***
* Use SlingScriptHelper.getService() method (into Java Use-API class ie Handler class )  
  ***MyService service=getSlingScriptHandler().getService(MyService.class);***

1. **What are the annotations used in your project**
2. **Run modes in AEM**

Run modes allow you to tune your AEM instance for a specific purpose. For example author/publish, QA, development, intranet or others.

***Why Run Modes?***

* Uniquely identify an environment and instances
* Unique configurations based on environment
* OSGI Component Creation for a specific environment
* Bundle Creation for a specific environment

***Types of run modes:***

* ***Primary Run Modes:***
* ***Author***: This instance is used for the complete development and authoring purpose.
* ***Publish***: This is the actual environment which can be accessed by end users.
* ***nosamplecontent***: This instance is having no sample content(like geometrixx,we-retail not available).It is highly recommended in production environment because it is very secure and it provides no sample configurations.lt makes your instance production ready, by disabling CRXDE lite, WebDAV etc.
* ***samplecontent***: having sample content like geometrixx-all package. This is just for the help of developers, not required on any server.

Combinations of primary run modes also happen:

* Author+samplecontent
* Author+nosamplecontent
* Publish+samplecontent
* Publish+nosamplecontent
* ***Secondary Run Modes:***
* Dev Server
* QA Server
* UAT Server
* Prod Server

1. **Client libs and its purpose in AEM**

It allows us to not only manage our client side resources like ( JavaScript, CSS, images, fonts etc. ), but also provide options to debug, minify, merge and gzip the client-side code.

* ***What are Client-Side Libraries?***
* Modern websites rely heavily on client-side processing driven by complex JavaScript and CSS code. Organizing and optimizing the serving of this code can be a complicated issue.
* To deal with this issue, AEM provides the concept of Client-Side Library Folder which allows you to store client side code in the repository, organize it into categories and decide when and how each category code is served to the client.
* Client side libraries play a very crucial role to provide rich functionality to end user. Clientlibs is used to manage these libraries in AEM.
* ***Why Client-Side Libraries?***
* Client-Side Libraries include JS and CSS files. In traditional way of using these on any web page, there will be separate network call for each js and css file.
* The overall performance will be reduced as load time of a web page will increase gradually if there are hundreds of files.
* So AEM came up with a better approach. Only two calls will be there for single Clientlibs, one for js and another for CSS.
* Irrespective of number of CSS file present, only single CSS call will be there and all css file will be embedded in same call and same applicable for JS file as well.

1. ***Feature or Properties of Clientlibs***

* ***Categories***: This property uniquely identifies the Clientlibs and a Clientlibs can be included in a page using categories. The categories property being multi-valued, allows a library folder to be the part of more than one category.
* ***Dependencies***: There is a list of other client-side libraries on which the library folder depends. To identify dependencies, add a property to your cq:ClientLibraryFolder node with the following attributes:
  + *Name*: dependencies
  + *Type*: String[]
  + *Values*: The value of the categories property of the cq:ClientLibraryFolder node that the current library folder depends on.
* ***Embed***: It is used to embed code from other libraries. Use the categories property to identify the client library folder to embed. To embed the library, add a property to the embedding cq:ClientLibraryFolder node, using the following property attributes:
* *Name*: embed
* *Type*: String[]
* *Value*: The value of the categories property of the cq:ClientLibraryFolder node that the current library folder Embeds upon.

1. **How to debug Clientlibs in browser**

In AEM projects, There is always a challenge in debugging clientlibs. So There are a lot of tools provided by aem to help developers :

* ***?debugClientLibs=true:*** If there is a need to debug how many clientlibs are getting embedded in the main clientlibs,
* Just add a query parameter in the page URL. “?debugClientLibs=true”, there will be separate network calls for each clientlibs. So debugging of js/css individual files will be easy.
* For Example: There are two clientLibraryFolders.

/etc/clientlibs/clientlib2

/apps/aem-developer/clientlib1

Clientlib1 is embedded in clientlib2.

* When we add “?debugClientLibs=true” in the page URL and go to the source tab of browser:

All the embed files of css can be seen while going in clientlib2.css?debug=true.

All the embed files of js can be seen while going in clientlib2 .js?debug=true.

* There is a component available in ***/libs/cq/ui/components/dumplibs/dumplibs.test*** that shows all the clientlibs folder of the system in a page.
* You can directly go to the page by using this url http://localhost:4502/libs/granite/ui/content/dumplibs.html and see all the clientlibs, its dependencies and embed clientLibraries.
* If you want to know about a particular clientlibs, which css and js files are getting loaded, add a selector "test" and see the results here: http://localhost:4502/libs/granite/ui/content/dumplibs.test.html
* There is always a cache issue with clientlibs, when you make any change in files (CSS/JS),the changes doesn’t reflect on pages, because AEM cache the clientlibs under “/var/clientlibs”. If you want to rebuild the clientlibs or clear cache, Go here: http://localhost:4502/libs/granite/ui/content/dumplibs.rebuild.html
* The below link shows the table of all the clientlibs, its dependencies and embed client Libraries and validates it (whether the embed and dependencies will exist or not, or they exist for a particular category type i.e., css/js or not) with different color codes.

<http://localhost:4502/libs/granite/ui/content/dumplibs.validate.html>

***WORKFLOW & API RELATED QUESTIONS***

1. **Workflow console and the tabs present in it**

The Workflow console is the centralized location for workflow management in AEM. It can be accessed via the Workflows link on the AEM Welcome page (Classic UI), and the Tools section of the touch-optimized UI.

Within the Workflow console, there are several tabs:

* ***Models*** - Lists the workflow models currently available. Here you can create, edit or delete workflow models.
* ***Instances*** - Shows you details of workflow instances which are currently active. These instances are also version dependent.
* ***Archive*** - Enables you to access details of workflow instances which have terminated, for whatever reason.
* ***Launcher*** - Allows you to define a workflow to be launched if a specific node has been updated.
* ***Failures*** - Enables you to monitor and manage failed workflow instances.

1. **How to create custom workflow process step**

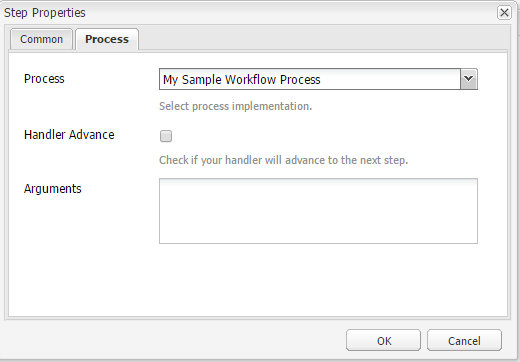
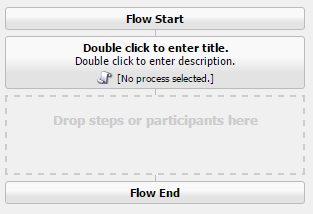
To define a process step as an OSGI service component.

* The OSGI component needs to implement the WorkflowProcess interface with it’s execute() method.
* Add the SCR property "process.label" and set the value as you please. This will be the name which your process step is listed as when using the generic Process Step component.

import org.apache.felix.scr.annotations.Component;  
import org.apache.felix.scr.annotations.Properties;  
import org.apache.felix.scr.annotations.Property;  
import org.apache.felix.scr.annotations.Service;  
import org.osgi.framework.Constants;  
import com.adobe.granite.workflow.WorkflowException;  
import com.adobe.granite.workflow.WorkflowSession;  
import com.adobe.granite.workflow.exec.WorkItem;  
import com.adobe.granite.workflow.exec.WorkflowProcess;  
import com.adobe.granite.workflow.metadata.MetaDataMap;  
@Component  
@Service  
@Properties({  
   @Property(name = Constants.SERVICE\_DESCRIPTION, value = "An example workflow process implementation."),  
   @Property(name = Constants.SERVICE\_VENDOR, value = "my vendor"),  
   @Property(name = "process.label", value = "My Sample Workflow Process") })  
  public class MyProcess implements WorkflowProcess {  
  public void execute(WorkItem arg0, WorkflowSession arg1, MetaDataMap arg2 )throws WorkflowException {  
  // TODO Auto-generated method stub  
  }  
}

Execute method has three parameters :-

* ***WorkItem*** - A WorkItem is the unit that is passed through a Workflow instance of a WorkflowModel. It contains the WorkflowData the instances acts on and a reference to the WorkflowNode that describes the underlying workflow step. A WorkItem can be considered as a token in a petri net that flows through the net during execution.
* ***WorkflowSession*** - The WorkflowSession class provides all functionality (depending on the user’s rights) for managing WorkflowModels, Workflow instances and their execution.[More](https://docs.adobe.com/docs/en/cq/5-6-1/javadoc/com/day/cq/workflow/WorkflowSession.html)
* ***MetaDataMap*** - A value map for generic access to Meta data values.
* In the CQ Workflow console, add the process step to the workflow using the generic Process Step component.(**Fig**-1)
* In the edit dialog, go to the Process tab and select your process implementation. (Here My Sample Workflow Process).(**Fig**-2)
* If you use arguments in your code set the Process Arguments
* Save the changes



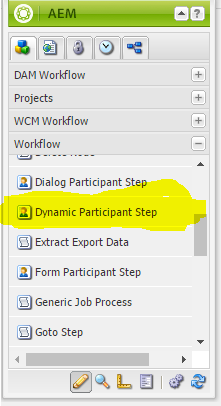
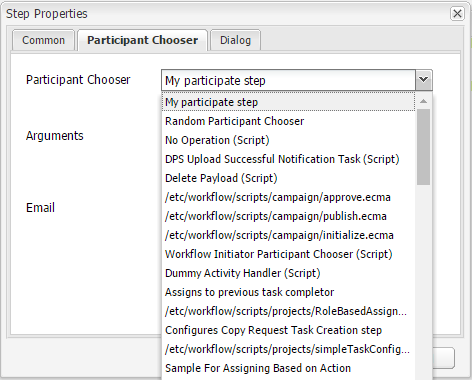
1. [**What is dynamic Participant step and how to create dynamic Participant step?**](http://www.aeminterviewquestions.com/workflow.html#collapse4)

The Dynamic Participant Step component is similar to Participant Step except the participant is selected automatically at run time.

* To define a dynamic participate step as an OSGI service component. The OSGI component needs to implement the ParticipantStepChooser interface with its getParticipant() method.
* Add the SCR property "chooser.label" and set the value as you please.
* This will be the name which your Participant chooser is listed as when using the Dynamic Participant Step component.

import org.apache.felix.scr.annotations.Component;  
import org.apache.felix.scr.annotations.Properties;  
import org.apache.felix.scr.annotations.Property;  
import org.apache.felix.scr.annotations.Service;  
import org.osgi.framework.Constants;  
import com.day.cq.workflow.WorkflowException;  
import com.day.cq.workflow.WorkflowSession;  
import com.day.cq.workflow.exec.ParticipantStepChooser;  
import com.day.cq.workflow.exec.WorkItem;  
import com.day.cq.workflow.metadata.MetaDataMap;  
@Component  
@Service  
@Properties({  
    @Property(name = Constants.SERVICE\_DESCRIPTION, value = "An example of a dynamic participant chooser."),  
    @Property(name = ParticipantStepChooser.SERVICE\_PROPERTY\_LABEL, value = "My participate step") })  
  public class InitiatorParticipantChooser implements ParticipantStepChooser {  
    public String getParticipant(WorkItem arg0, WorkflowSession arg1,MetaDataMap arg2) throws WorkflowException {  
      return "admin";  
}  
}

* In the CQ Workflow console, add the participant step to the workflow using the dynamic participant Step component.(**Fig**-1)
* In the edit dialog, go to the Participant chooser tab and select your Participant chooser implementation.(**Fig**-2)
* If you use arguments in your code set the Arguments
* Save the changes.



1. **Different workflows you have worked on**
2. **What are workflow api’s (workflow session, workflow model, workflow service, workflow data)**
3. **Launchers in AEM and when it is used.**

Workflow launcher lets you automatically invoke a workflow based on conditions.  
That is, a change to AEM JCR content. The Workflow Launcher monitors changes in the content repository to launch workflows dependent on the location and resource type of the changed node.

Using the Launcher you can:

* See the workflows already launched for specific nodes.
* Select a workflow to be launched when a certain node/node-type has been created/modified/removed.
* Remove an existing workflow-to-node relationship.

1. **Page Manager API and what is the use.**
2. **JCR api’s**

* Allows us to query data located in the AEM Java Content Repository (JCR). To query data, you use a javax.jcr.query.Query instance that belongs to the JCR Query API. This API supports both searching and querying operations
* Jackrabbit is a complete, and fully compliant implementation of the Content Repository API for Java Technology (JCR) and therefore its primary API is defined by JCR. For a developer this means that most operations required are defined by the JCR API. The classes and interfaces within Apache Jackrabbit are only needed when accessing functionality that is not specified in JCR.

1. **Event listeners and event handlers**

There is actually no difference at application level. While theoretically you can say the following statement :

* A ***listener*** is an object that listens, (and takes actions) upon certain events. I.e. it "listens" for events.
* A ***handler*** is an object that handles certain things that the client class don't want to deal with. I.e. it "handles" events. A typical scenario is that I provide a handler for a specific event/task as an argument to a constructor.

So, you can have multiple listeners (just call addXxxxListenerseveral times) while you're only supposed to have a single handler.

***DISPATCHER RELATED QUESTIONS***

1. **Dispatcher in AEM**
2. **How dispatcher performs load balancing**
3. **What is .stat file, dispatcher. Any in AEM**
4. **How caching is performed in dispatcher**