Solution Design DMD

1. At a glance

2. Overview

DMD which stands for Document Mess Detector lends its name from PMD, the famous Open Source Static Code Analyser, which helps developers to detect messy and risky code.

DMD leverages Salesforce's Einstein Prompt Templates to apply the concept to Natural Language Documents in daily business situations. This native approach allows administrators to adjust the AI model used and lets non-technical users modify prompts through Salesforce's trusted platform.

As in PMD, DMD allows Users to create sets of rules and then check documents against it. For the Rulecheck, the document text and rule definitions are processed using Einstein Prompt Templates which analyze the content and respond with structured feedback. For each rule, the app creates a Result record with a description of the status.

When the App is installed in a Subscriber's org:

1. an Admin can use it to enhance the Record pages of Salesforce objects
2. the App can read Files or Attachments
3. the App calls out to two External Services for Text Extraction (Extractor API) and Text Reasoning (OpenAI API)
4. the App writes into Custom Objects, Settings and Custom Metadata

```
@startuml

node SubscriberOrg {
    actor SysAdmin
    actor StandardUser

    component DmdManagedPackage #AntiqueWhite {
        database SettingsAndMetadata
        database CustomObjects
        component LowCodeProCode
    }

    database Files
}

node ExtractApi {
    boundary api
}

node "Einstein LLM" #AliceBlue {
    boundary promptTemplates
}

SysAdmin --> ExtractApi : manage account
SysAdmin --> DmdManagedPackage : installs & configures

StandardUser --> DmdManagedPackage : uses

LowCodeProCode -->  Files : reads
LowCodeProCode --> api : callout
LowCodeProCode --> promptTemplates : document analysis
LowCodeProCode <-up-> CustomObjects
LowCodeProCode <-up-> SettingsAndMetadata
@enduml
```

3. Roles, Use Cases and their Implementation

The Salesforce app implementing DMD has 3 User Personas with accompanied Permission Sets with the same name that perform multiple Use Cases.

```
    @startuml
    !theme plain
    left to right direction

    actor "AppAdministrator" as admin
    actor "RuleAuthor" as author
    actor "DocumentReviewer" as user

    package "DMD App" #AliceBlue {
        usecase configure as "Configure the App"
        usecase rules as "Author Rules/Rulesets"
        usecase export as "Export/Import Rulesets"
        usecase check as "Check Documents"
        usecase regression as "Regression Tests"
    }

    admin --> (configure)
    (export) <-- author
    (rules) <-- author
    (regression) <-- author
    user --> (check)
    @enduml
```

3.1. AppAdministrator

Users with the `AppAdministrator.permissionset` need to be a SysAdmin with rights to work in Salesforce Setup, Deploy Metadata changes and customize Salesforce.

3.1.1. Configure the App

As detailed in the User Guide a Sysadmin must install the App from the AppExchange and confirm that the app can use 2 external Endpoints for API communication.

He customizes the app by assigning the installed permission sets to users and adjusting Flexipages of packaged Custom but also Standard objects like Accounts and Opportunities.

Post install the Admin needs to configure the app from Apps Setup tab.

```
    @startuml
    !theme plain
    hide footbox

    actor AppAdmin
    participant SetupTab
    database DmdSettings__c

    == Extract API Step ==
    AppAdmin --> SetupTab : Provide Extract API credentials
    SetupTab --> DmdSettings__c : Store Protected
    @enduml
```

One Setup Step asks for Extract API credentials which on Save are securely stored in a Protected Custom Setting called `DmdSettings__c`. This is used to callout to an external API to extract text from PDF documents in Salesforce.

3.1.1.1. Relevant Packaged Components

- Custom Objects: `DmdSettings__c`
- Tab: `Setup__c.tab-meta.xml`
- Visualforce: `setup.page`, `setupStep.component`
- Classes: `setup/app-setup/*`, `SetupPageCtrl.cls`, `SetupExtractApi(_Test).cls`

3.1.1.2. Security Remarks

- All Authentication credentials are safely stored in Protected Settings or Metadata
- External Endpoints are approved during Package installation

3.2. RuleAuthor

Users with the `RuleAuthor.permissionset` can be Standard users. The Permset provides the right CRUD and FLS permissions to perform tasks of managing Rules and Rulesets.

3.2.1. Import/Export Rulesets

For easy exchange of Rules and Rulesets DMD has a Custom Export and Import tool that uses Andy Fawcetts SObjectdataloader Open Source. It serializes multiple `Ruleset__c` objects including related objects like `Rule__c`, `RuleInSet__c` into a single JSON file.

```
@startuml
!theme plain
hide footbox

actor RuleAuthor
database Ruleset__c
participant ExportQuickAction
participant ImportQuickAction
participant afawcett as "afawcett/\ndataloader"

== Export Rulesets ==
RuleAuthor --> ExportQuickAction : on Recordpage
RuleAuthor --> ExportQuickAction : on Listview
ExportQuickAction --> afawcett : Delegate
afawcett --> Ruleset__c : Serialize
RuleAuthor <-- afawcett : ruleset-export.json

== Import Rulesets ==
RuleAuthor --> ImportQuickAction : on Listview
RuleAuthor --> ImportQuickAction : ruleset-export.json
ImportQuickAction --> afawcett : ruleset-export.json
afawcett --> Ruleset__c : Deserialze
@enduml
```

3.2.1.1. Relevant Packaged Components

- Custom Objects/Tabs: `Rule__c`, `Ruleset__c`, `RuleInSet__c`
- LWC: `importRulesets`
- Visualforce: `exportRulesetButton.page`, `exportRulesetsButton.component`
- Classes: `/afawcett/apex-sobjectdataloader/*`, `ExportImportRulesetCtrl(_Test).cls`

3.2.1.2. Security Remarks

- We didn't change the OSS library to prevent PMD or Checkmarx issues but instead added False Positive comments there.
- Only users with the right permission set can see the related pages, call the Controllers and LWC components.
- As Rules are Public Read this works for mostly all Rules and Rulesets in an org.

- Rules and Rulesets of others should not be messed with by others, but reading and cloning them is fine.

3.2.2. Author Rules and Rulesets

Codifying Business requirements into reusable rule, grouping them into document- and context-specific ruleset is the main task of the Rule author. Using Topics for Objects Rules and Ruleset can use arbitrary Tags for grouping, versioning and filtering in Listview and Reports.

```
@startuml
!theme plain
hide footbox

actor RuleAuthor
database Rule__c
database Ruleset__c
database RuleInSet__c

== Create Rule ==
RuleAuthor --> Rule__c : New with Name, Description
RuleAuthor --> Rule__c : Formulate Rule Text for Gen AI
RuleAuthor --> Rule__c : Add Topics
RuleAuthor --> Rule__c : Create Listview based on Topics

== Create Ruleset ==
RuleAuthor --> Ruleset__c : New with Name, Description
RuleAuthor --> Ruleset__c : Describe Gen AI System Context
RuleAuthor --> Ruleset__c : Add Topics
RuleAuthor --> Ruleset__c : Create Listview based on Topics

== Add Rules to Ruleset ==
RuleAuthor --> RuleInSet__c : From Rule or Ruleset
@enduml
```

3.2.2.1. Relevant Packaged Components

- Custom Objects/Tabs: `Rule__c`, `Ruleset__c`, `RuleInSet__c`

3.2.2.2. Security Remarks

1. Rules, Rulesets and their Junction object use PublicRead as Sharing Setting. This allows other people to use and learn from existing rules but not modify them. Rule codify important Company knowledge. So having explicit owners and sharing tables for those who can evolve rules is important.

3.2.2. Run and Analyse Regression Tests

Generative AI is much more powerful but also less reliable for rule checking compared to deterministic methods. The AI Model itsself seems to change over time resulting in Analysis giving results of quality over time. Even if we use a temperature parameter on OpenAI we have seen the same rule to have different results on the same document when called multiple times.

To monitor such behavior we added a Regression Tester where a given Benchmark Analysis can be repeated over time (using Scheduled jobs) and then the results can be compared to get Accuracy insights.

```
@startuml
!theme plain
hide footbox
autonumber

actor RuleAuthor
box "Benchmark" #lightgreen
    database Analysis__c
end box

database RegressionTest__c
participant "Report/Dashboard" as report

box "Regression Runs" #DarkSalmon
    database Analysis1
    ...
    database AnalysisN
end box
participant Scheduler

== Setup a Regression Test ==
RuleAuthor --> Analysis__c : Find Benchmark Analysis
RuleAuthor --> RegressionTest__c : Create based on Benchmark
RuleAuthor --> RegressionTest__c : Set Cron Expression
RuleAuthor --> RegressionTest__c : Create

== Running Regression Test ==
Scheduler --> Scheduler : wait until Cron fires
Scheduler --> RegressionTest__c : Clone + Run: Benchmark Analysis
...
Scheduler --> RegressionTest__c : Clone + Run: many times

== Analyse Results ==
RuleAuthor --> report : See Regression in Record Dashboard
report --> Analysis__c : Compare with Regression runs
report --> AnalysisN : Compare with Benchmark
RuleAuthor --> Ruleset : Adjust AI config and Rule content

@enduml
```

3.2.2.1. Relevant Packaged Components

- Custom Objects/Tabs: `RegressionTest__c`
- Code: `RegressionTest.trigger`, `ScheduleRegressionTests.cls`

- Reports/Dashboards: All reports, types and Dashboard in the package
- LWC: `regressionResults`

3.3. DocumentReviewer

3.3.1. Check a Document

The main use case of the application is running document scan against selected file. Using a simple component, placed on sObject record page, reviewer can start scanning process. When the process is finished, users are informed about this fact via Custom Notifications.

```
@startuml
!theme plain
autonumber
hide footbox

actor DocumentReviewer
database "Object/File/Attachment" as file
participant "Selector LWC" as lwc
database "Analysis__c/Result__c" as analysis
participant "AnalyseDocument.cls" as code
boundary ExtractAPI
boundary "e.g. OpenAi API" as api

== Select Document ==
DocumentReviewer --> file : RecordPage
lwc --> file : Fetch allowed documents
DocumentReviewer --> lwc : Choose Document

== Create Analysis ==
DocumentReviewer --> lwc : "Analyse.."
DocumentReviewer --> analysis : Choose Ruleset/API
DocumentReviewer --> analysis : Create
analysis --> code : Insert Trigger

== Extract text from Document ==
code --> ExtractAPI : extractTextFrom(document)
code <-- ExtractAPI : plainText


== Create Result records ==
code <-- plugin : getCompletion()
code --> CustomNotif : display
@enduml
```

3.3.1.1. Relevant Packaged Components

- Custom Objects/Tabs: `Analysis__c`, `Result__c`
- Settings: `DmdSettings__c`
- LWC: `documentSelector`, `documentAnalysisHistory`

- Classes: `AnalyseDocument(_Test).cls`, `NotifyUserOnAnalysisChange(_Test).cls`, `ExtractApi(_Test).cls`

3.3.1.2. Security Remarks

1. Analysis record have a restrictive Private OWD Sharing model as otherwise their Result child objects could expose confidential information from the analysied documents. The Result's Justification field often explains by quoting the original document why a Rule failed or passed.
2. PDF Documents owned by the current user are sent to the Extract API endpoint authenticated by the `AppAdministrator` after installation
3. Extracted Text plus Rule Content is sent to the Open AI API during Analysis. The OpenAI API is authenticated by the `AppAdministrator` after installation.
4. For both Endpoints the Package contains `Remote Site Settings` that need to be approved on package install.
5. Errors and Exception that happen during Callouts are caught and persisted including Details into the Analysis__c.FailReason__c and Status__c fields.

4. Data Model

The App brings a broad range of Custom Objects for Application data. It doesn't rely on any data in Salesforce Standard Objects and only works on Files and Attachments.

It also comes with a Protected Custom Setting and a Public Custom Metadata Type with Protected Packaged records.

```
    @startuml
!theme plain

    object "File | Attachment" as file #DDDDDD {
        Binary with extractable text
    }

    package "DMD Managed Package" #AntiqueWhite {
        package "Check documents" <<Rectangle>> {
            object Analysis__c #ADD1B2 {
                Document
                Ruleset
                Status
                FailReason
            }
            object Result__c #ADD1B2 {
                Rule
                Status
                Justification
            }
        }
        package "Author Rules" <<Rectangle>> {
            object Rule__c #A9DCDF {
                Name
                Description
                Content
            }
            object RuleInSet__c #A9DCDF {
                Rule
                Ruleset
            }
            object Ruleset__c #A9DCDF {
                Name
                Description
                Context
            }
        }
        object RegressionTest__c #EB937F {
            Benchmark Analysis
            Cron Expression
        }
    }

    Rule__c "0..*" -left- "1..*" Ruleset__c
    (Rule__c, Ruleset__c) . RuleInSet__c
    Ruleset__c o-left- Analysis__c
    Analysis__c *-- Result__c
    Analysis__c -right-o file
    Result__c --o Rule__c

    RegressionTest__c --o Analysis__c : Benchmark
    RegressionTest__c o-- Analysis__c

@enduml
```

4.1. Custom Settings : `DmdSettings__c`

For safely storing global Application settings the app comes with a Protected Custom Setting. Currently we only store API credentials for the Extract API there. This is managed by the custom Setup page.