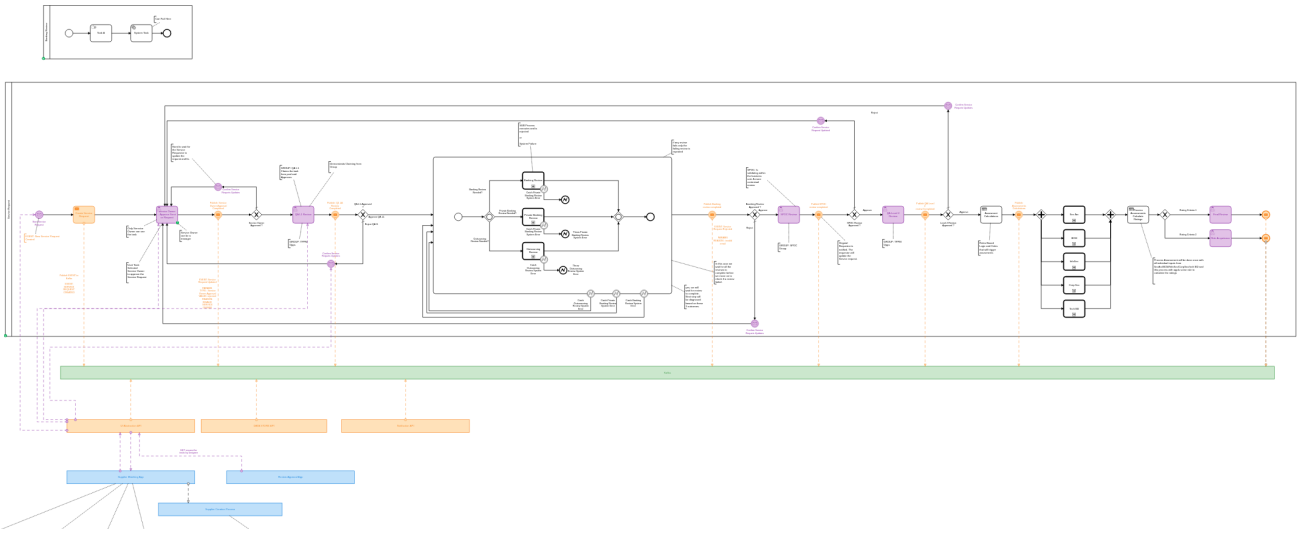
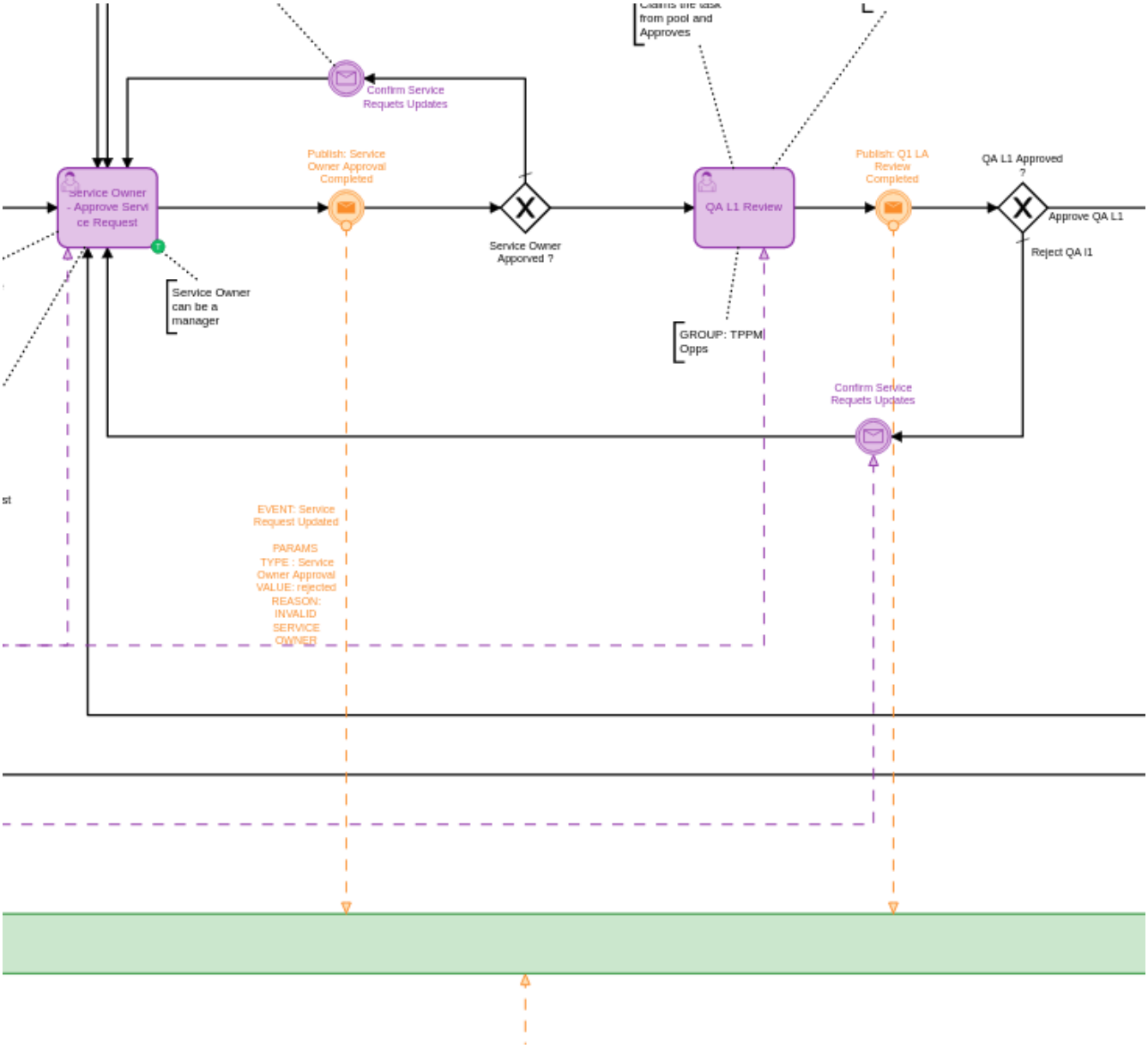
Business Case: Service Request with Multi-level Approval

Service Request use-case is focused on demonstrating the service request process through Web Based UI, Notifications and Camunda workflow. The service request process models several human review steps where a approval or rejection can happen. As well as system interactions through async over TCP communications with eventing system (Kafak) and synchronous communications through HTTP/REST in a point to point pattern.

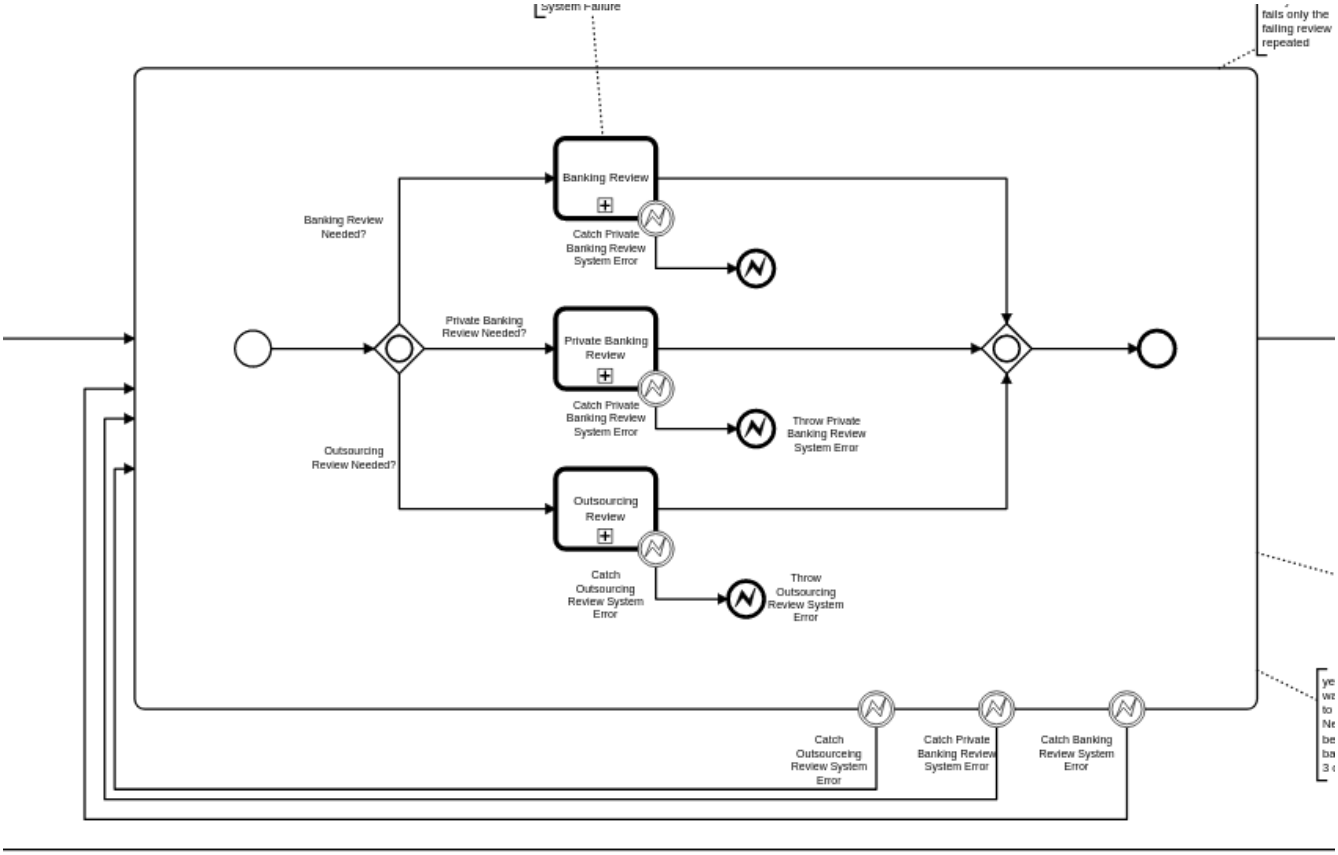


Some interesting points in the process

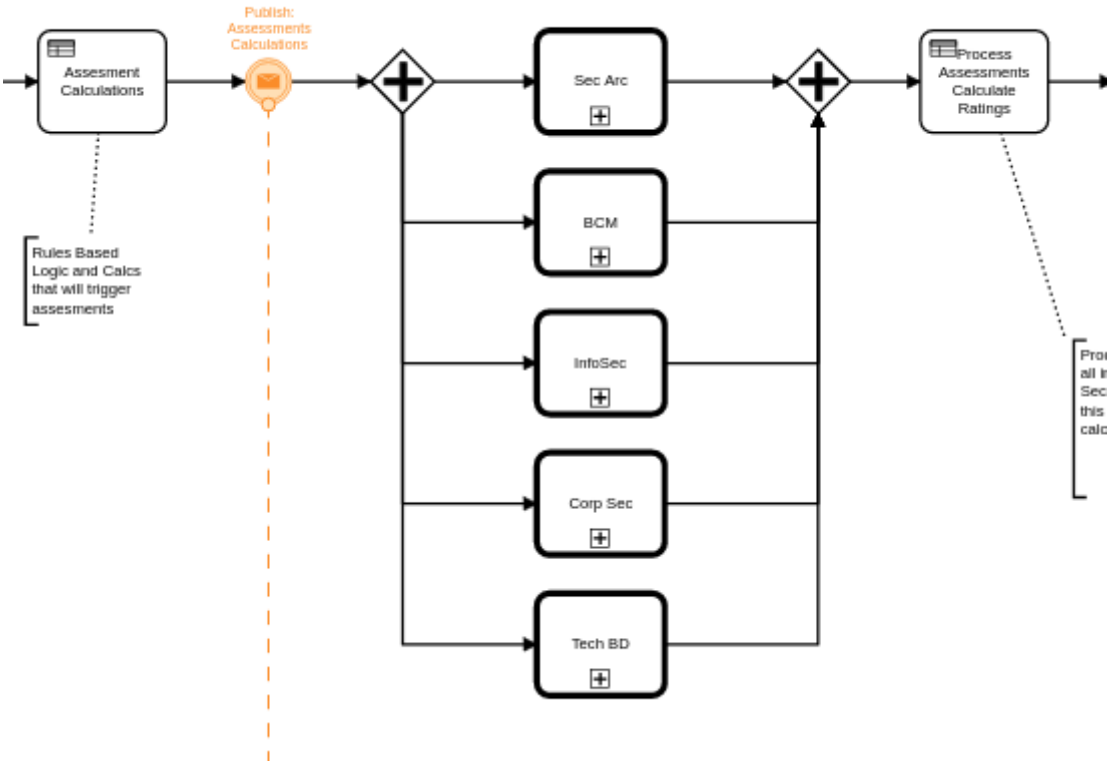


This section of the process is particularly interesting and packed with functionality. At the workflow level we are orchestrating user tasks for the explicit multilevel review of service requests. After each review we send a Event to Kafaka potentially notifying other systems, like a UI, about the completion of the task and update of the service request.

Notice on the rejection path no User Task is defined for Requester of the service request. This is intentional to signify that the Requestor can be notified in many ways about the rejection and the method of updating the service request is not important to the workflow itself. In technical terms this signifies a loose coupling for these events due to the workflow not orchestrating the interaction rather using a choreography approach. Essentially all the workflow cares about is the rejection is handled in some way but no idea how it happens.



This section of the process demonstrates the capability to explicitly handle errors and potentially compensate or retry certain activities in the workflow.



This section of the workflow demonstrates the possibility to use DMN and business rules to accomplish the assessment calculations.

Running the use-case

IMPORTANT	Running the app once with no profile is necessary to initalize the Camunda database.

```
mvn spring-boot:run
```

Profiles can be specified at the command line when the application starts. The notation is as follows. `-Dspring.profiles.active=servicerequest,integration,cors`

Or you can use the application.properties file to specify the profile.

```
spring.profiles.active: test,cors
```

Loading Test Data

TODO: Create REST end-point to parse csv and load service request data, start processes associating with a service id/business key

Testing with Postman

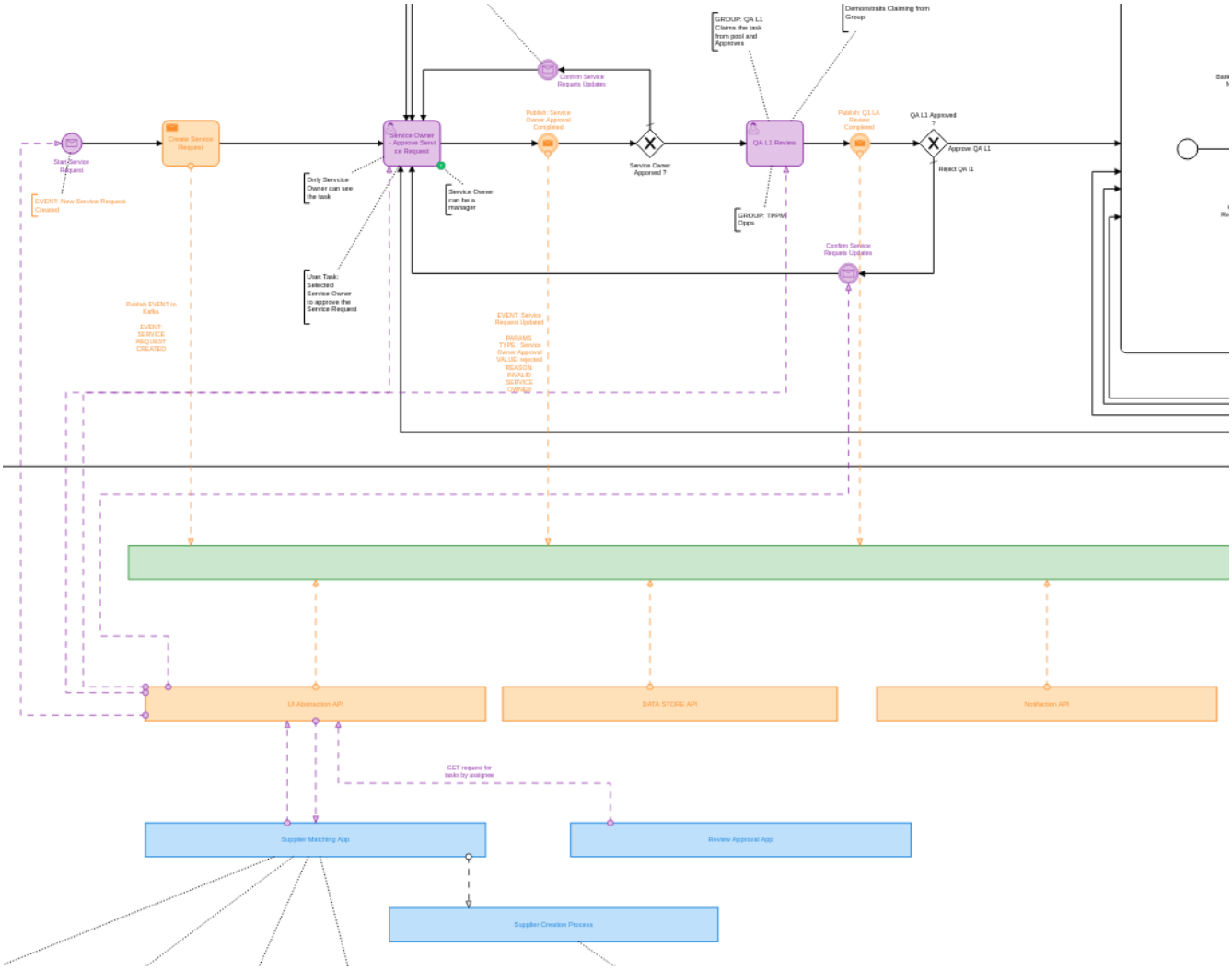
Use the postman collection `MOSA-PoC.postman_collection.json` in the `postman` folder.

With post-man you can move through the processes simulating REST requests to the app. - starting the process with new service request - completing user tasks - correlating messages

See it running

Visit `http://<server>:<port>/sr` to access the React app.

Architecture



The diagram above illustrates the interactions and logical components of the app. Note the app is all packed together into one artifact for easier development and PoC ing. But each component could be easily it's own deployable artifact.

The green bar signifies kafka. Events are published to Kafaka from the workflow Send Tasks. The Send tasks are implimented as java delegates. This pattern works well as we can utilize the Delegate and the Send task to control the execution of the workflow and potentially ack Kafaka and handle incidents when Publishing fails.

The orange boxes signify components that sub-scribe to Kafaka topinc and update other components based on the Events that they receive.

The blue boxes signify components that do specific work and are updated by Events from the subscription components.

NOTE This is a typical pattern for micro-service architectures though the level of abstraction between components will vary from use-case to use-case.

Kafka Integration

The spring-boot app is using spring cloud streams.

https://spring.io/projects/spring-cloud-stream

The app has a has a single publisher and a single subscriber for the service-request-events topic.

```
spring.cloud.stream.bindings.publishServiceRequest.destination=service-request-events
spring.cloud.stream.bindings.subscribeServiceRequest.destination=service-request-
events
```

See the `com.camunda.poc.starter.usecase.servicerequest.kafka.integration` package/folder for impl of publishers and subscribers.

> A single subsciber is implimented `ServiceRequestEventSubscriber.java`; it simply gets the message from the topic and serialized into memory. Then it saves/caches the Service Request into the local db based on the event type.

ReactJS UI Integration

The Maven frontend-maven-plugin configured in pom.xml is used to build the ReactJS app. The plugin creates a bundle.js file which ends up in `src/main/resources/static/built/bundle.js`. The static directory makes static resources such as JS and HTML available to the java app.

The Java application boot-straps the ReactJS App through Thymeleaf a java/spring frontend framework. The templates directory `src/main/resources/templates/app.html` has a HTML file app.html which calls the React app through a `<script />` tag loading the HTML into the react div `<div id="react"></div>`

Thymeleaf ties the Java frontend together using a Spring controller. `src/main/java/com/camunda/react/starter/controller/HomeController.java`. Mapping the app context to /home and calling the app.html.

The React Components are organized under the `src\main\js\reactjs` folder into a use-case folder then subdivided by component. Webpack and package.json define the structure and dependencies for the React App that allow node to build the app into the bundle.js which is later placed in the static directory as explained previously.

Developing with this PoC Starter Project

Settign up React for Dev

- run node and server.js by starting a node server in the home directory of the project. You may need to run `npm install` first.

```
nodemon server.js
```

also run the web-pack watch in the project home so you can update the bundle as you build reactjs

```
webpack -w
```

NOTE	you need to run the cors profile when using nodemon

- Also note you can use spring-dev-tools to build front and back-end component in dev mode providing faster restarts and live-reload.

for dev mode run the following with the appropriate profiles

```
mvn spring-boot:run
```

WARNING	spring-dev-tools affects the way serializes objects and will cause serialization errors in some cases. So it is commented out in pom.xml by default.