Mortgage Processing Platform

# ING Bank Australia

There’s the done thing and then there’s the ING way. And the fact that we’re Australia’s Most Recommended Bank makes us smile a thousand smiles.

# ING Bank Engineering

Our engineering team is a diverse group of people who share the same aim. We want to build a secure, scalable system which delivers for our customers. We want to achieve this with respect for each other taking into account different perspectives. We try to have open minds in relation to implementations and will look at the best technology for a given problem.

# The context

ING has a strong presence in the business of Mortgages and Lending. As such, the bank requires robust and flexible systems to

* Capture new loan applications and View existing applications and their status
* Progress them through a workflow (approved or declined by bank representatives with different privileges)
* Integrate with third party services to verify details before granting credit to a customer.

# Your mission

Real systems for mortgages have many screens, complex workflows and integrations. For this exercise however, we would like you to build a **prototype** **application** for a mortgages platform with the following components in one **single** “Dashboard/Screen”, split horizontally in three main areas. There are few “mandatory services” for the screen to integrate with in the back end, and a few “**optional** services” if you have time to implement them



**Main Dashboard**

1. **Top Navigation Bar**: simple navigation action menu for our prototype application. The supported actions will be:
   1. **Create a new application**
   2. **View an existing application** (can edit on this screen, also approve/decline or fund)
   3. Application User **Profile Control**: A hard coded dropdown to alternate between “different user profiles with different permissions” to control certain features of the application. A real application would have login/permissions functionality, and that would control what you can and cannot do in the application. You can assume another team would build this module, and **you only have to build a dropdown that instantly changes the “permissions profile”** of the current user according to three permission profiles:
      1. Credit Assessor: Can only enter new applications, and edit existing
      2. Credit Approver – Can enter new applications, edit existing, **and Approve**
      3. Lending Manager: Superuser that can do anything: Enter new, edit, approve, but is also the only type of user able to click the “Fund Application” button to complete the application and release the funds.
2. **Main Panel**: Displays a form with fields related to a mortgage application. We will use only six fields in our prototype. Status of the application is displayed on the form heading.
   1. First Name (text field)
   2. Last Name (text field)
   3. Address (text field can contain numbers for street addresses e.t.c)
   4. Applicant Income (numeric field)
   5. Loan Amount (numeric field)
   6. Credit Status: Approved/Not Approved (read only label field). This value will be derived from a backend call depending on values entered in other fields
3. **Workflow queue panels**: Three sub panels in the bottom section of the screens holding the application numbers, depending on which of these***Three States*** the application is in
   1. New Application (sub panel 1)
   2. Approved Application (sub panel 2)
   3. Funded application (Complete and Funds released to Customer) (sub panel 3)

**Services**

To implement the functionality for the Dashboard, you will need to consider implementing the following back end services, some are mandatory (described below), others optional and described in later Use Cases:

* **LoanApplicationService** [Mandatory]: With the following operations
  + **void saveApplication(Application obj):** Persist Details captured in the Main Panel (for new and existing applications)
  + **Application loadApplication()**: Retrieve the same details, as an Application object for display by frond end
* **WorflowService.setStatus** [Mandatory]: The main panel and form will contain buttons to move an application between the three states, in the back end and onscreen. When clicking each of these buttons, besides performing the button action (e.g. Save), you should call the “setStatus(long:applicationNumber)” method on the workflow service according to the following
  + Save Button: should set status to “New”
  + Approve Button: should set status to “Approved”
  + Fund Button: should set status to “Funded”
* **List WorkflowService.getQueueList(optional state)** [Mandatory] – another method on the Workflow service to retrieve read-only lists, you can poll to display lists of applications in the “Workflow queue panel bottom section of the screen, depending what state they are in. If not state is provided, all applications should be retrieved.

**Technology**

* You need to implement the Front End using Polymer, ASP.NET Application (WebForms or MVC) or .NET WinForms
* You need to implement the Back End using Java.
* SpringBoot is highly desirable for the back end, but if you want to use another Java based solution, that is acceptable.
* Calls from front end to back end should use REST over HTTP passing a JSON message.
* **Optional**: If you can have some simple logging in the backend to demonstrate what your application is doing when interacting with it, this will be highly regarded. It will also help you debug it.

Read carefully all the use cases below before starting to ensure you fully understand the scope of this assignment. **You must complete first on the** **mandatory requirements for every use case** before attempting any optional use cases. Once you fulfill all mandatory requirements for every use case you can start thinking to improve the scope of the exercise with the nice to have requirements.

# Use case 1 – Capturing New Applications and then displaying them [Mandatory]

Build the Dashboard screen as described above, with the three horizontal sections suggested (see screenshot above) Top Navigation, Main Panel, and Worflow Queues (with at least 1 panel display list of deals, or nice to have 3 subpanels for each of the states)

You will need to build a database to store this data in the backend. You can use an in-memory database if you wish to avoid spending time on database set up. We do want to see some simple SQL queries to persist/retrieve the details and a simple DB structure.

The message for saving new and updating existing applications will take the JSON form of:

{ **"firstName"**:"Greg",  
   **"lastName"**:"Fraser",  
   **"address"**:”54 Bellavista Rd, Stanmore, Sydney NSW”,  
   **"creditStatus"**:"Approved",

**"applicantIncome"**:"80000",

**"loanAmount"**:"720000",

**"timeCreated"**:"19-11-2017 10:27:44"}

{ **"firstName"**:"Kaivan",  
   **"lastName"**:"Murthy”,  
   **"address"**:”7 Coronation St, Ryde, Sydney NSW”,  
   **"creditStatus"**:"Not Approved",

**"applicantIncome"**:"60000",

**"loanAmount"**:"350000",  
   **"timeCreated"**:"19-11-2017 10:27:44"}

## Use case requirements

1. Frontend: As per requirements above
2. Capture a Timestamp when saving a new application

### Mandatory

1. Implement **Saving** and **Retrieving (display)** of the 6 Fields for a Mortgage application
2. Generate a simple and Application ID upon saving a new application. This should be a unique number. Optionally it would be nice if this was a 7 digit number.
3. After saving, an application can be retrieved using the “View Existing Application” text field in Top Nav Bar (you search entering the application ID)
4. When Application is saved, its status = New. Hence display it in the “New” workflow queue panel at the bottom of the screen or display in common list with status clearly shown. It would be nice if this updated dynamically once the deal is created.
5. Implement the LoanApplicationService in the Back End with two operations: Save and Load. An “Application” object or value map should be transferred to represent the state.
6. “Approve” button to change status of application to “Approved”.
7. “Fund” button to change status of application to “Funded”.

### Nice to have

1. Retrieving an existing application by clicking on a hyperlink (its application number in the workflow queue lists)
2. CSS Customisation of the UI
3. Validation of User Inputs
4. Automatic update of the lists when deals are added/changed
5. 7 digit deal numbers

### Out of scope

1. User authentication is not required
2. User permissions not yet required for this first use case.
3. For Use Case 1, you only need to get The Workflow Queue “New” panel displaying the new applications (bottom of screen). The others (Approved / Funded) are not needed yet.

# Use case 2 – Implement User Profiles, Approvals and Workflow [Mandatory]

Once applications can be saved and retrieved, now we want to evolve the logic of our application to allow for three different user profiles. These are hard-coded into a dropdown, in the top nav bar, which we are using to test our prototype application. Depending on the user profile, some of the buttons on the application form will be enabled or disabled. The bottoms controlled by these permissions are “Approve” and “Fund” (complete application).

With Approve and Fund Actions implemented, we need to permission the buttons so that they are only enabled the buttons for the correct user profile as outlined below.

Optional Requirement: If implementing the three state based application panels then when an application state is updated the application should move from one panel to the next. For example, a “New” application once we click Approve, should disappear from the “New” queue panel and appear in the “Approved Panel”. Then when a “Lending Manager” clicks “Fund” the application will disappear from the “Approved Queue” and move to the Final “Fund” Workflow Queue panel.

## Use case requirements

### Mandatory

1. Implement the “Profile Control” dropdown in the top nav bar. This can be hardcoded, but should control dynamically whether which buttons are enabled.
   1. Save Button: Can be used by any profile
   2. Approve Button: Can be used by Credit Approver or Lending Manager
   3. Fund Button: Can only be used by a Lending Manager
2. Implement WorkflowService in the back end to set/get the workflow state on an application

### Nice to Have

1. The application should move from New => Approve => Funded Workflow Queue lists (panels) dynamically in the bottom of the screen as its state changes

### Out of scope

1. User authentication and login is not required – profiles are hardcoded in the top dropdown
2. If you begin saving many applications and the list grows large, it is OK if they don’t all fit in the bottom three workflow queue panels. It would be nice to have a scroll bar if required, but not strictly necessary, unless you have time

# Use case 3 – [Advanced and Nice to have] Implement Credit Status as a separate endpoint and service

Only attempt this part if you have time and have completed Use Cases 1 and 2

Upon collecting details about the applicant, a real mortgage processing system would use those parameters and run some rules to determine if the bank can lend the “loan amount” requested, based on the applicant’s details. In our simple example, we could **Implement Credit Status = “Approved” or “Not Approved”, depending on the applicant’s income as a percentage of the loan amount requested**. For this Use Case, that logic should be simple, but we’d like to implement it as a separate service, and make the call using REST and a JSON message

## Use case requirements

1. Build a separate service as **another** “Spring Boot” application or similar. This should be **separate** from the web application back end and polymer front end that you implemented in the first use cases
2. New Service and method:   
   Status CreditStatusService.assessApplication(applicationNumber, loanAmount, income)
3. Service takes 3 parameters as input, returns a single String as Approved/Not Approved
4. The actual calculation to decide if Approved / Not approved will be:  
    String status = Income is at least 1/3 of the Loan Amount Requested  
   While this is a simple calculation, should the rules for this need to change in the future, they can be done easily and separate in the “CreditStatusService” without modifying the main web application

### Mandatory

1. Implement Separate Service using REST and separate HTTP call
2. Display results on your Application Form (in the web app) as the “Credit Status” field

### Nice to have

1. The service is invoked realtime: once income and loan amount are entered, you should make an asynchronous call, get the result and update the UI in realtime
2. Error Handling Incorrect Params: What would your CreditStatusService do if it was given invalid parameters for loanAmount or income? How can you implement some error handling given it’s a REST call across HTTP?
3. Error Handling when service is down: What would your Web Application do if it tried to contact the CreditStatusService, and it was down and unavailable?

### Out of scope

1. No authentication or security needs to be applied to the new CreditStatusService

# Use case 4 – Non-functional performance of loadApplication service

It’s important that our application is performing well. As such, build a performance test to run the service to retrieve/load applications at 30tps for a minimum of 10mins.

## Use case requirements

1. Test to be built and executed using JMeter

### Mandatory

1. Tests must run for 30tps for a minimum of 10mins
2. Results presented as histogram
3. Summary of min, max, average, 95th percentile performance

### Nice to have

1. Stress test to find breaking point

### Out of scope

1. Performance testing UI. Standard functional UI testing is required however.

# What we look for (Assessment criteria topics)

There is no right or wrong solution, we will assess what you did and your approach according to these criteria

## The architectural solution

As we said there is no right or wrong, several approaches would lead you to complete the mission successfully.

The implementation

The code must be well structured, with enough clarity and follow good coding principles. Implementation is expected to be in Polymer and Java. Usage of Spring Boot is desirable but not essential.

Testing

A balanced testing strategy is a key point in software engineering. A significant amount of effort will be invested in testing so it’s important to have a clear testing approach. We are not expecting you to have fully implemented full-fledged testing solution covering each layer of testing. We expect you to use judgement to determine what testing is required for this solution but would expect to see at least some unit testing, UI testing and non-functional testing. The service based functional and non-functional testing will be the key area for our assessment.

Best of luck!