Introduction

This document contains detail information for a developer experience evaluation of the DCM-RUST architecture. This architecture is developed to facilitate runtime code mobility in Rust-based web applications. The general workflow is as shown in the figure below:

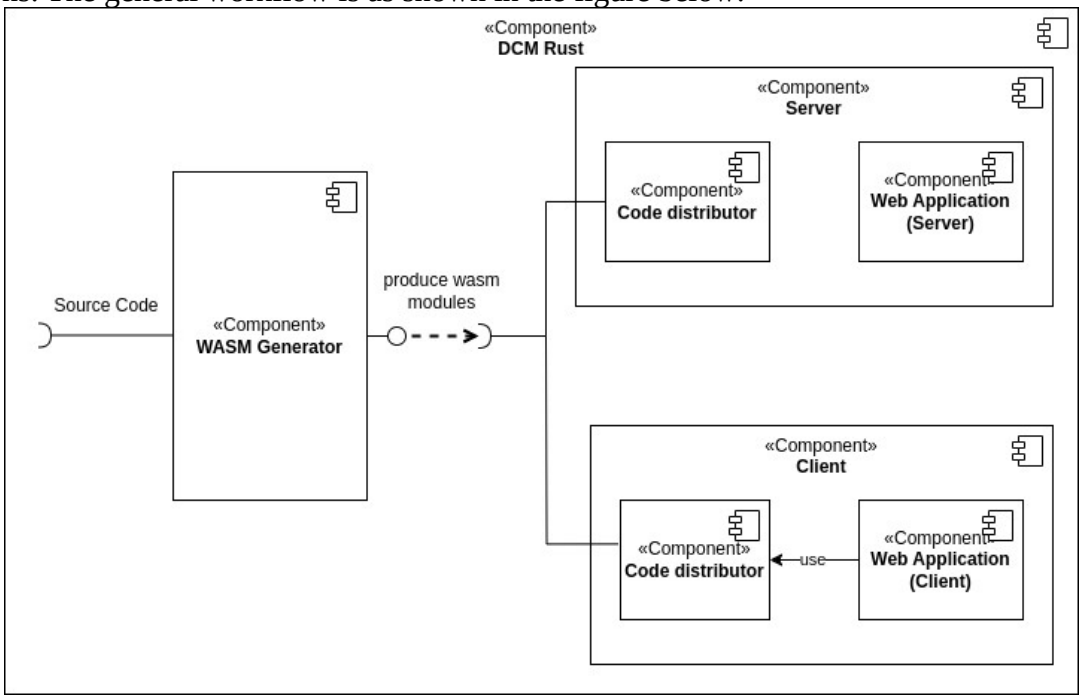

Figure 1: DCM-RUST Overview

1. In the first step, the WASM generator application generates WebAssembly modules from the project's source code and moves them to the proper directories in client and server side code distributors.
2. Alongside, it also generates JavaScript glue code and moves them to the client side code distributor's directory.
3. Now, when the developer uses this glue code instead of regular JavaScript function, the invocation is handled by the code distributor component on either the client or server side depending on the fragment's current execution location, which is decided by the decision system.
4. The decision system is not in the scope of this thesis, so, a chrome browser extension is provided to manually emulate the decision system. Same interface is also attached to the demo-app UI as a shorthand.

The demo application

A demo project is developed, with our solution architecture integrated into it. The project is a simple online shopping application with basic features like browsing products, adding them to the cart, and placing orders. Since the decision system is not part of this thesis, an interface is provided to manually simulate the execution location of mobile functions and objects. Figure below shows the main page of the demo application.

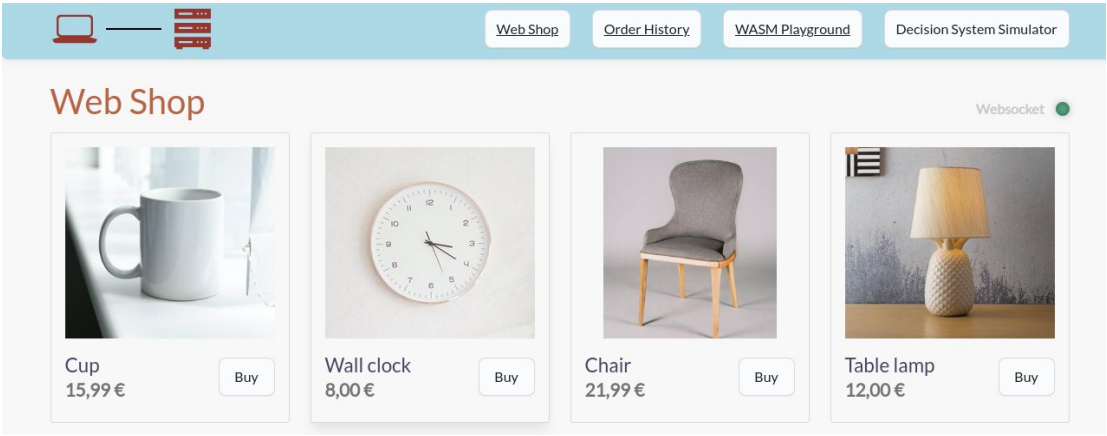

Figure 2: The demo application (Shopping page)

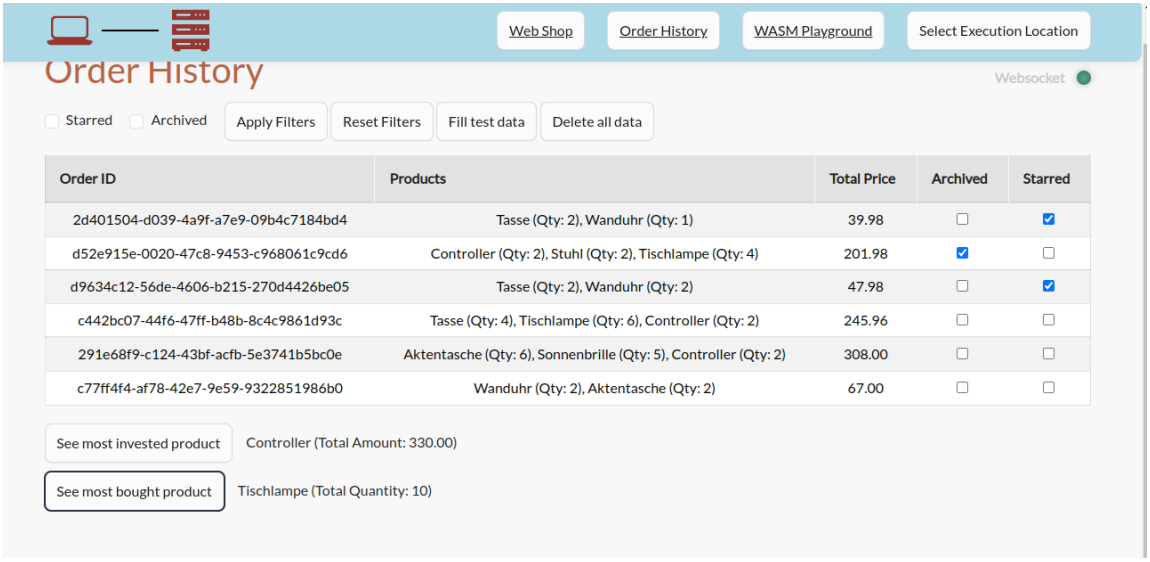

Figure 3: The demo application (Order history page)

After ordering some items, the user can then see their order history, as shown in figure above. Orders in the history can be starred or archived for later reference. The view can be filtered to show only the starred or archived orders. Below the order history table, there are two buttons labeled ’See most invested product’ and ’See most bought product’. When clicked, these buttons display the respective product along with relevant details like the total amount invested in that product or the total quantity

bought just to its right. The most invested product is the one that the user has spent the most money on, and the most bought product is the one that is bought the most in terms of total quantity.

Task and Experiment

All participants of this evaluation are asked to perform some pre-defined tasks to get familiar with the system. The participants can access the required codebase and infrastructures in the cloud from the browser with these credentials:

url: http://34.174.167.236:8080/?folder=/home/dx/project/dx/web-application{participant-number}

password: 26434b954bc74c77c11aa545

Shown in figure 3, the functions to find the ’most invested product’ and ’most bought product’ are currently implemented in JavaScript. The participants are asked to implement these functions in Rust, make them mobile and then replace their usage in JavaScript with our exported functions. They are free to use any IDE or text editor of their choice, along with any external resources like the Internet. The actual logic of the functions is already provided in the form of helper functions in Rust. These helper functions take reference of the input as parameters, which prevents them from being made mobile (because references are not supported as parameters for mobile functions). So, the actual task that has to be completed by the participants are explained step-by-step as below:

1. Open the file: project_root/runtime-code-mobility-demo/src/shared/webshop /developer_experience_evaluation.rs.
2. At the bottom of this file, create two new functions: most_bought_product and most_invested_product. The functions signature will be similar to get_most_bought_product and get_most_invested_product in the same file respectively. Only difference is that these new functions must take the actual value as parameter rather than references (remove the &).
3. From these functions, call their respective helper functions by supplying the reference of the parameter, and return the value.
4. Mark the two newly created functions as mobile /// @mobile doc comment on top of functions.
5. Now run the wasm-generator application from the web-ide’s terminal. Just type ‘wasm-generator’ and press enter.
6. When it finishes, the two functions from rust will be now available in the client side.
7. Navigate to this path: runtime-code-mobility-demo/src/frontend/static/js/OrderHistory.
8. Open the file: project_root/runtime-code-mobility-demo/src/frontend/static/js/OrderHistory/ index.js
9. In this file, replace the function invocations for get_most_bought_product and get_most_invested_product with most_bought_product and most_invested_product respectively by importing them from initCodeDistributor.js.

10. Restart the demo application and the code distributor.
11. Clear the browser cache and hard reload the webpage. The two actions shown in figure 3 should not be able to change their execution location.