User Manual: Webshop Case Study for Software Product Line Engineering

Version 1.0

June 12, 2025

Contents

List of Code Listings

1 Introduction

This document serves as a comprehensive technical reference and practical manual for generating and running the webshop product line, targeting developers, students, and researchers interested in feature-based software product line engineering. It provides step-by-step guidance from initial plugin installation and project setup through the complete process of generating WinVMJ modules, integrating payment gateways, and configuring microservices with API gateway environments. By following this guide, users can effectively scaffold, connect, and operate modular Java systems using model-driven development techniques to create fully functional webshop applications. To begin, ensure that the Eclipse application is installed via the following link: Eclipse Application. If you have already installed Eclipse, you just need to follow the plugin installation instructions for each tool. If not, you can skip that part.

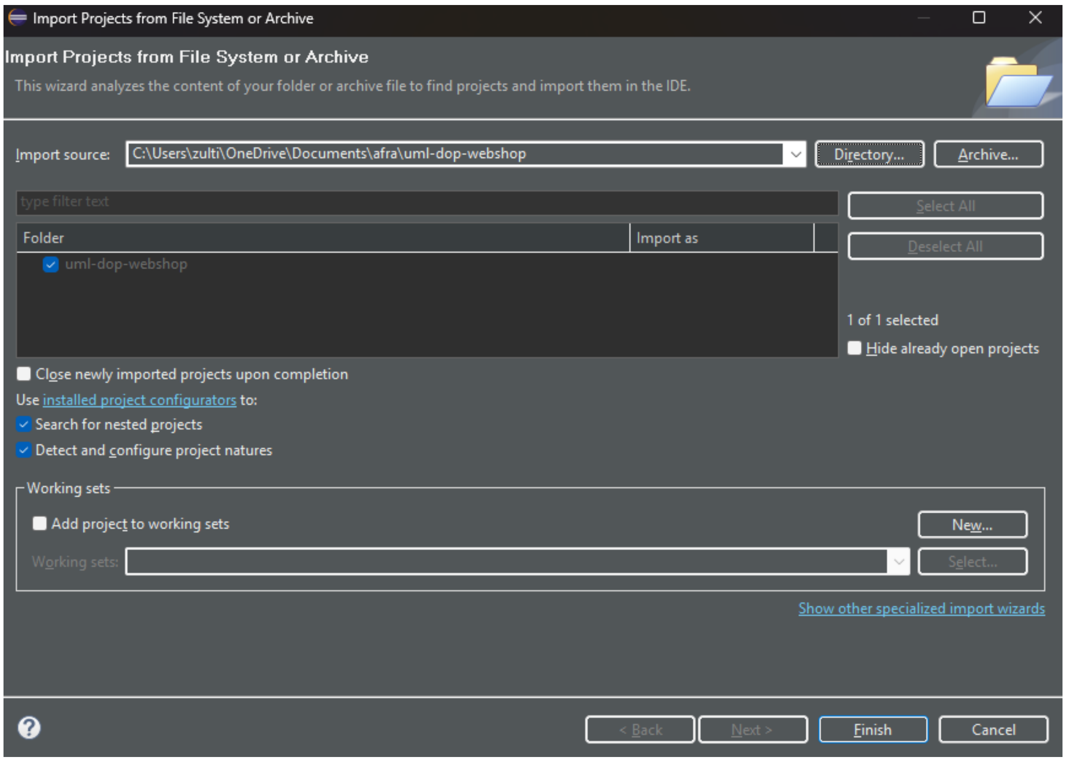
2 UML to WinVMJ

This tutorial will guide you in creating UML projects and applying UML-DOP profiles in Papyrus within Eclipse. We have a sustainable product line that can be used as a base for development or as a learning tool to deepen your understanding of coding with WinVMJ. This product line is the Webshop product line and the UML-DOP can be accessed through the following link: Webshop UML-DOP Repository .

2.1 Plugin Installation

This step is optional if you don't use our Eclipse.

1. On the top toolbar, click Help → Install New Software.
2. Click Add. In the `Name` field, add `UML to WinVMJ`, and add `https://amanah.cs.ui.ac.id/priceside/uml-to-winvmj/staging/` in the `Location` field. Click Save.
3. Open the Work with dropdown menu, and select `UML to WinVMJ`. Select UML to WinVMJ and then click Next.
4. Approve all licenses, and then click Finish.


Import Projects from File System or Archive

Import Projects from File System or Archive

This wizard analyzes the content of your folder or archive file to find projects and import them in the IDE.

Import source: C:\Users\zulti\OneDrive\Documents\afra\uml-dop-webshop

type filter text

Folder	Import as
✓ uml-dop-webshop	

1 of 1 selected

☐ Hide already open projects

☐ Close newly imported projects upon completion

Use installed project configurators to:

✓ Search for nested projects

✓ Detect and configure project natures

Working sets

☐ Add project to working sets

Working sets:

Show other specialized import wizards

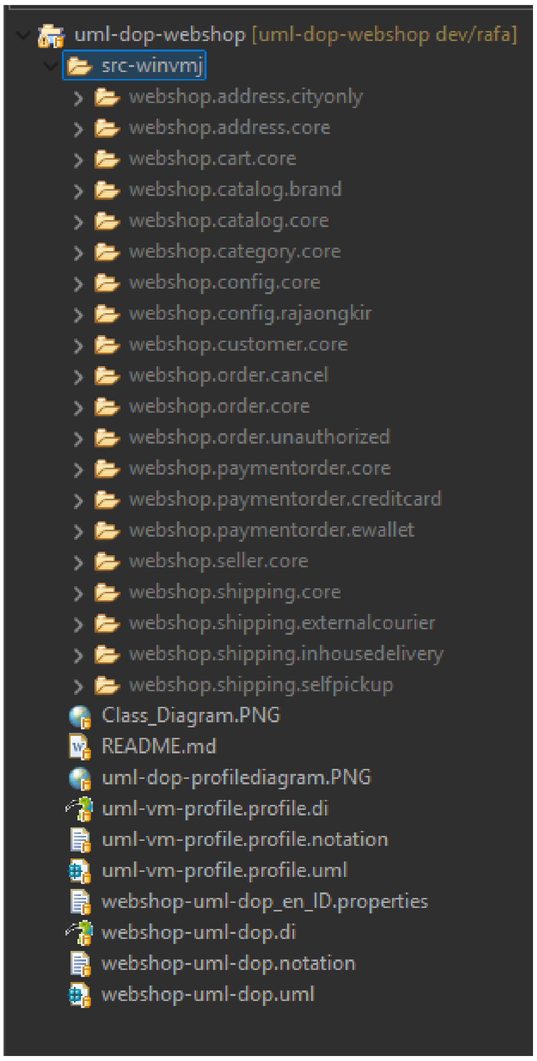
Figure 1: Open Project in Eclipse

2.2 Open UML project

1. On the top toolbar, click File $\rightarrow$ Open Projects From File System.
2. Click Directory, — in this case, the directory of the cloned repository from:
   Webshop UML-DOP Repository. Click Finish
3. If desired, you can open the `webshop-uml-dop.di` file and select the class diagram to view the UML model.

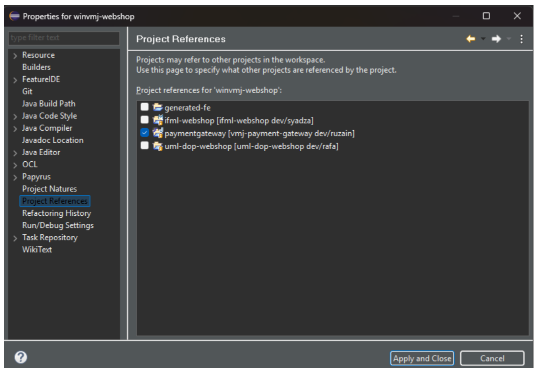
2.3 Generate WinVMJ Modules

1. Right-click on the UML model file named `webshop-uml-dop.uml` within your project.
2. Select `Acceleo Model to Text > Generate WinVMJ`.
3. Check the `src-gen` directory in your project; the `WinVMJ` files will be generated here as shown by Figure 2.



uml-dop-webshop [uml-dop-webshop dev/rafa]
src-winvmj
webshop.address.cityonly
webshop.address.core
webshop.cart.core
webshop.catalog.brand
webshop.catalog.core
webshop.category.core
webshop.config.core
webshop.config.rajaongkir
webshop.customer.core
webshop.order.cancel
webshop.order.core
webshop.order.unauthorized
webshop.paymentorder.core
webshop.paymentorder.creditcard
webshop.paymentorder.ewallet
webshop.seller.core
webshop.shipping.core
webshop.shipping.externalcourier
webshop.shipping.inhousedelivery
webshop.shipping.selfpickup
Class_Diagram.PNG
README.md
uml-dop-profilediagram.PNG
uml-vm-profile.profile.di
uml-vm-profile.profile.notation
uml-vm-profile.profile.uml
webshop-uml-dop_en_ID.properties
webshop-uml-dop.di
webshop-uml-dop.notation
webshop-uml-dop.uml

Figure 2: Generated WinVMJ folders



Project References

Projects may refer to other projects in the workspace.
Use this page to specify what other projects are referenced by the project.

Project references for 'winvmj-webshop':

- [ ] generated-fe
- [ ] ifml-webshop [ifml-webshop dev/syadza]
- [✓] paymentgateway [vmj-payment-gateway dev/ruzain]
- [ ] uml-dop-webshop [uml-dop-webshop dev/rafa]

Figure 3: Add Reference to Payment Gateway

4. These generated modules in the WinVMJ Project require additional implementation to be complete. For reference, you can examine a fully implemented example by cloning the repository at:
   Webshop WinVMJ Repository

3 Payment Gateway Integration

1. Clone the *Payment Gateway* project from the repository at:
   Payment Gateway Repository.
2. Add a project reference to the *Payment Gateway* by:
   - Right-clicking on the WinVMJ-webshop project and selecting `Properties`.
   - Opening the `Project Reference` menu and adding a reference to the `Payment Gateway` project.
3. Create a folder named `interfaces`, then copy the `model.uvl` file from the `Payment Gateway` into this folder.
4. Import the UVL file from `Payment Gateway` into the `Webshop Product Line` UVL file by adding the following lines, as shown in lines 3–4 and 13 of Code 1, in the `source tab` of the `model.uvl` file:

Listing 1: Webshop's Feature Diagram

```
1  namespace webshop
2
3  imports
4      interfaces.PaymentGateway as PG
```

```

features
    webshop {abstract true}
        mandatory
            Catalog
                ...
        optional
            ...
            PG.PaymentGateway
constraints
    ...
```

5. Create a configuration file for inter-product relationships named `inter_spl_product.json`.
6. Copy the JAR file from the `Payment Gateway` into the `external` folder to simplify dependency management.

4 WinVMJ Composer

This tutorial will guide you through applying WinVMJ to run the backend as a Java project.

4.1 Plugin Installation

This step is optional if you don't use our Eclipse.

1. On the top toolbar, click Help → Install New Software.
2. Click Add. In the Name field, add `WinVMJ Composer`, and add `https://amanah.cs.ui.ac.id/priceside/winvmj-composer-microservice/staging/` in the `Location` field. Click Save.
3. Open the Work with dropdown menu, and select `WinVMJ Composer`. Select `WinVMJ Composer` and then click Next.
4. Approve all licenses, and then click Finish.

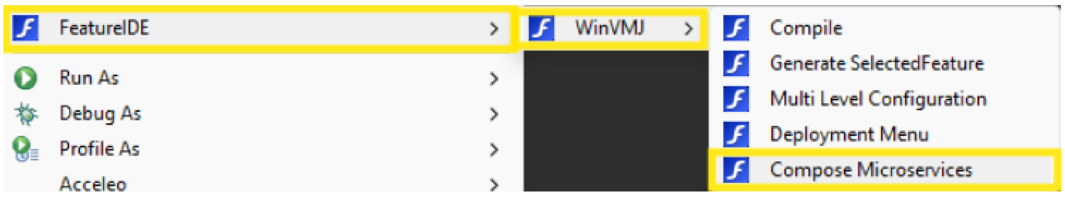


Figure 4: Compose Microservice

4.2 Compose Microservice

1. The configuration is defined in the `services-def.json` file. You can clone or download it from any product in the Generated Product Repository.
2. Right click on the services-def.json file, then select FeatureIDE -> WinVMJ -> Compose Microservices.
3. Generated modules are available in the src directory. If the generated modules do not appear, ensure you have generated the modules. You should see multiple projects:
   - `ApiGateway`
   - One or more microservice modules (e.g., `ServiceAuth`, `ServiceUser`, etc.)

4.3 Compile Microservice

1. Right click on the src directory.
2. Select FeatureIDE > WinVMJ > Compile.

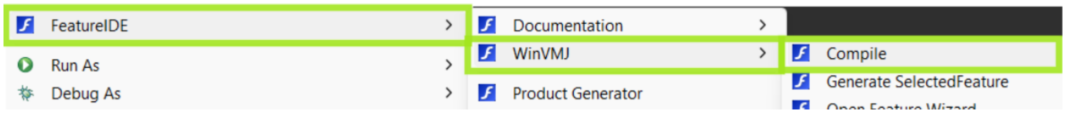


Figure 5: Compile Product

3. Check the WinVMJ Console to monitor the compilation process.
4. The compiled application will be placed in the src-gen directory.

4.4 Running a Product

1. On the top toolbar, click on `Run > External Tools > External Tool Configuration`.
2. Right-click on `Program` and select `New Configuration`.
3. Set the script location to `src-gen/[Product Name]/run.bat` (for Windows) or `run.sh` (for Linux/Mac).
4. Set the working directory to `src-gen/[Product Name]/`.
5. Click Run.

4.4.1 Set Environment Variables

Add the following environment variables for **each service**:

Variable Name	**Description**	**Example Value**
`AMANAH_PORT_BE`	Backend service port	7777, 7778, etc.
`APP_ID`	Unique identifier for the service	`auth-service`, `user-service`
`RABBITMQ_HOST`	RabbitMQ host address	`localhost`
`RABBITMQ_USER`	RabbitMQ username	`guest`
`RABBITMQ_PASS`	RabbitMQ password	`guest`

Ensure each service has a unique `AMANAH_PORT_BE` and `APP_ID`.

4.5 Configure Run Settings for `ApiGateway`

Set up the run configuration for the `ApiGateway` project similarly:

4.5.1 Set the Run Command

- **Command:** `run.bat`
- **Working Directory:** Folder containing the `run.bat` file

4.5.2 Set Environment Variables

Variable Name	**Description**

`AMANAH_PORT_BE`	Port for `ApiGateway`
`ServiceAuth_URL`	URL for any one service (e.g., Auth Service)
`<ServiceName>_URL`	One for each service, based on `service-def.json`

Note: Replace `$<ServiceName>$` with the actual service name from `service-def.json`.

Make sure the ports in these URLs match the `AMANAH_PORT_BE` values configured for each service.

4.6 Example Configuration

ServiceUser Environment Variables

```
AMANAH_PORT_BE=7777
APP_ID=service-user
RABBITMQ_HOST=localhost
RABBITMQ_USER=guest
RABBITMQ_PASS=guest
```

ApiGateway Environment Variables (if ServiceUser is on port 7777)

```
AMANAH_PORT_BE=8888
ServiceAuth_URL=http://localhost:7777
ServiceUser_URL=http://localhost:7777
```

5 IFML Generator

This tutorial will guide you in creating a modeling project and applying IFML-DOP profiles so it can be generated into a React app.

5.1 Plugin Installation

This step is optional if you don't use our Eclipse.

1. On the top toolbar, click Help $\rightarrow$ Install New Software.

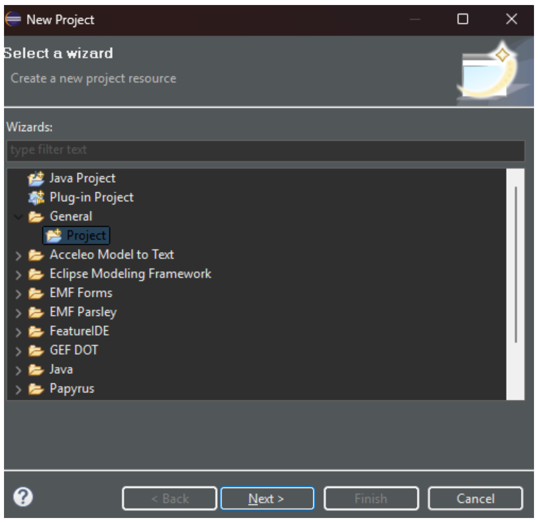


Figure 6: Create New Project

2. Click Add. In the `Name` field, add `IFML UI Generator`, and add https://amanah.cs.ui.ac.id/priceside/ifml-ui-generator/staging in the `Location` field. Click Save.
3. Open the Work with dropdown menu, and select `IFML UI Generator`.
4. Select IFML UI Generator and then click Next.
5. Approve all licenses, and then click Finish.

5.2 Generate React Application

1. Clone IFML model from Webshop IFML Repository, then open it in Eclipse using the same steps as in Subsection 2.2.
2. Create a new Project folder that will contain the generated front-end application code.
3. Right click in the new project, than click `New` → `File` to add an empty file named SelectedFeature (case-sensitive) inside the project folder that was already created.
4. Fill in the `SelectedFeature` file with the features to be generated from your model, listing one feature per line. An example is shown in Figure 7.

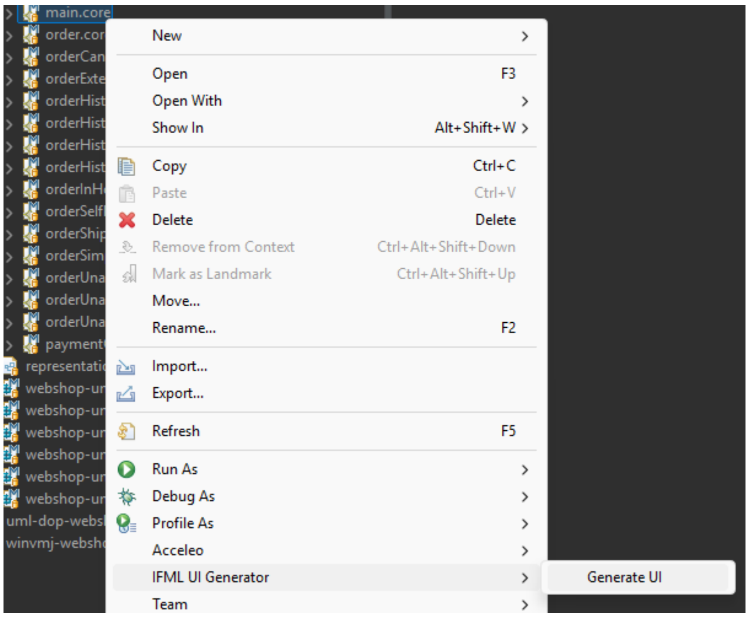
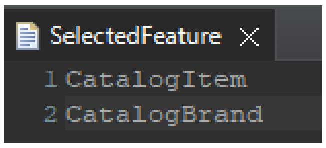


Figure 8: Generate UI

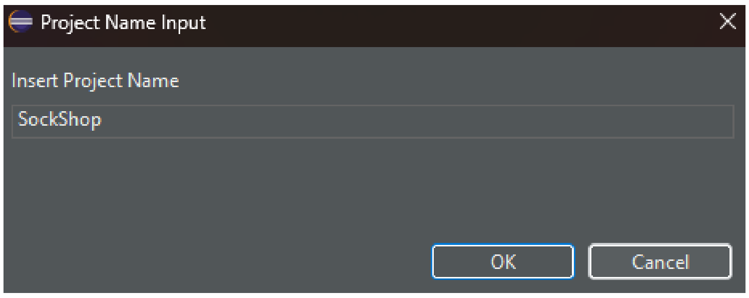


```
CatalogItem
CatalogBrand
```

Figure 7: Example of features listed in the `SelectedFeature` file

Don’t forget to save the file by pressing `Ctrl+S`.

5. In the IFML project, inside the `model-modules` folder, right-click on the core IFML model you want to generate (a `.core` file). For example, you can select `main.core`.
6. Choose IFML UI Generator → Generate UI as shown in Figure 8.
7. Select your app folder and click OK.
8. Select one of the available UI variants. For the default UI, choose Standard.
9. Input your app name and click OK. The name will appear on the Navbar and Footer of each page.
10. Once completed, find the generated front-end code in your app folder.


Project Name Input

Insert Project Name

SockShop

Figure 9: Input App Name

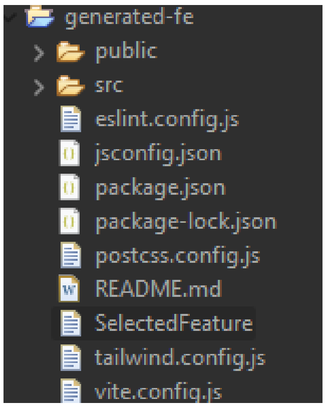

```
generated-fe
  public
  src
  eslint.config.js
  jsconfig.json
  package.json
  package-lock.json
  postcss.config.js
  README.md
  SelectedFeature
  tailwind.config.js
  vite.config.js
```

Figure 10: Generated Front End Application

5.3 Running the Generated React Application

1. To run the application for the first time, install dependencies with the following command:

   ```
   npm install
   ```

2. Create a .env file with the following contents:

   ```
   VITE_PORT=<YOUR_FRONTEND_PORT>
   VITE_BACKEND_URL=<YOUR_BACKEND_URL>
   VITE_STATIC_SERVER_URL=<YOUR_STATIC_SERVER_URL>
   ```

3. Start the React application with the following command:

   ```
   npm run start
   ```

4. Open a new terminal and start the Static Server on `localhost:3003` (only required if modifying static pages) with the following command:

   ```
   npm run json:server
   ```

5. Your website will be accessible at

   ```
   <YOUR_FRONTEND_URL>:<YOUR_FRONTEND_PORT>
   ```

   (eg. http://localhost:3000).