Pakwan Restaurant Management System

Complete Project Report

Project name	GKB / Pakwan Restaurant Management System
Project folder	/Users/hikmatullah/Documents/GKB
Distributable	kebabish.jar (≈ 826 KB fat-JAR)
Language / Stack	Java + JavaFX 25 (FXML + CSS)
Architecture	Single-window desktop app, MVC via FXML controllers
Persistence	Plain CSV files in the project directory
Branches managed	GKB Pakwan • Kebabish Marki • Outer Pakwan
Cross-cutting modules	Khata (credit ledger) • Sales • Expenses
Lines of Java code	~ 13,200 across 13 source files
Report generated	2026-05-08

Contents

1. Project Overview

GKB (also referred to as the **Pakwan Restaurant Management System**) is a desktop application built with Java and JavaFX that manages day-to-day operations for a multi-branch catering / pakwan business. The owner runs three sales channels — **GKB Pakwan**, **Kebabish Marki**, and **Outer Pakwan** (walk-in / small orders) — and uses the same app to track customer credit (**Khata**), sales reporting, and expenses.

The app is distributed as a single fat-JAR (**kebabish.jar**) that bundles all third-party libraries. All business data is stored as CSV files alongside the JAR, so the system can be moved between machines simply by copying the project folder.

Functional scope at a glance

- Take and edit catering orders for two branded channels (GKB & Kebabish).
- Take walk-in / outer orders with separate fields and pricing.
- Maintain a per-person *khata* ledger (debits, credits, running balance).
- Track expenses by type, item, KG, cost-per-KG, supplier and bill number.
- Compute sales reports across channels.
- Save instantly with F8 or Cmd/Ctrl+S from any page.
- Cache loaded pages so re-navigating is instantaneous.

2. How the App Runs

Easiest path

Open the project folder in Finder and **double-click** `GKB.command`. The app launches automatically. This shell wrapper is what end-users on the family machine use day-to-day.

VS Code

Open the project, switch to the **Run and Debug** sidebar (Ctrl+Shift+D), pick **Launch GKB** from the dropdown, and press the green Play button. **Do not** use the small Run button in the editor's top-right corner — it bypasses the JavaFX module configuration and fails with *"JavaFX runtime components are missing"*.

Command line

From the project root:

```
./compile_and_run.sh
```

That script wipes `bin/`, recompiles every `.java` file under `src/`, copies FXML/CSS resources next to the classes, and launches `Launcher` with the JavaFX module path pointed at `/Applications/javafx-sdk-25/lib`.

Why is there a Launcher.java?

Since Java 11, JavaFX is shipped as separate modules. The JVM checks for them *before* starting any class that extends `javafx.application.Application`. If the modules aren't perfectly visible at that early stage, you get the infamous *"JavaFX runtime components are missing"* crash. `Launcher.java` is a plain class with a `main()` that calls `Main.main()`, sidestepping that early check and letting JavaFX load through the configured module path / bundled libs.

3. Folder & File Layout

Path	Purpose
GKB.command	Double-click launcher (zsh wrapper around the JAR)
compile_and_run.sh	Dev script: clean, compile, copy resources, run
build_jar.sh	Builds the distributable kebabish.jar fat-JAR
run_gkb.sh	Minimal helper to run the JAR
Manifest.txt	Manifest used when packaging the JAR (Main-Class: Launcher)
sources.txt	Generated list of .java sources (used by javac @sources.txt)
kebabish.jar	Built fat-JAR — what end users actually run
lib/	ZXing JARs (core-3.5.2, javase-3.5.2) for barcode/QR support
src/	Java sources, FXML pages, CSS
src/Main.java	JavaFX Application entry — loads EntryPage.fxml
src/Launcher.java	Plain main() shim that calls Main.main()
src/EntryPage.fxml	Landing page with one card per module
src/MainPage.fxml	Top nav + content area shell
src/controllers/	11 controllers — one per page + 2 interfaces + nav
src/pages/	FXML for gkb, kebabish, outer, khata, sales, expenses
src/styles/main.css	Single global stylesheet
bin/	Compiled classes + copied resources (regenerated by build)
orders.csv	Outer / general orders log
gkb_orders.csv	GKB Pakwan order history
kebabish_orders.csv	Kebabish Marki order history
expenses.csv	Expense ledger
khata_records.csv	Khata (credit) transactions
persistent_khata_payments.csv	Persistent khata payment log
pakwan.zip	Archived snapshot of an earlier version
README.md	Run instructions for end users

4. Application Architecture

The app is a single-window JavaFX desktop application that uses FXML for layout, CSS for styling, and Java controllers for logic. Persistence is intentionally simple: every module reads and writes its own CSV file in the project root.

Startup flow

- **Launcher.main()** → **Main.main()** → `Application.launch()`.
- **Main.start()** loads `/EntryPage.fxml`, applies `/styles/main.css`, sets the title to *Pakwan Restaurant System* and a 1000×700 minimum size.
- **EntryPageController** shows four restaurant cards (GKB, Kebabish, Outer, Khata) plus top-right Sales / Expenses buttons. Clicking any of them swaps the scene to `MainPage.fxml` and tells **MainPageController.setInitialPage()** which page to land on.
- **MainPageController** is the host: a top nav bar plus a `StackPane` that swaps in module pages from `/pages/*.fxml`.

Page caching

MainPageController keeps two parallel maps — `pageCache` (FXML root nodes) and `controllerCache` (their controllers). First navigation to a page loads the FXML; later navigations just reuse the cached node. If the cached controller implements **Refreshable**, its `refreshData()` is called so the view is up-to-date without paying the FXML-load cost again.

Two tiny but important interfaces

Saveable — implemented by controllers that own data. Pressing F8 or Cmd/Ctrl+S anywhere in the app calls `save()` on the currently active controller via **MainPageController.handleGlobalSave()**.

Refreshable — controllers implement `refreshData()` so the host can ask them to re-read their CSVs when they're brought back from cache.

Cross-page navigation with payload

Controllers can hand an *Order* object to another page so the user can edit an existing record. **MainPageController** exposes `loadOuterPageWithOrder(...)`, `loadGkbPageWithOrder(...)`, and `loadKebabishPageWithOrder(...)`; each loads the target FXML fresh (bypassing the cache for that one navigation), then calls `controller.loadOrderForEditing(order)`.

NavigationService is a small static helper that holds the active `MainPageController` reference so deeply-nested controllers can request a navigation without having to plumb the host through their constructors.

Where each controller lives

Controller	LoC	Owns FXML	Role
EntryPageController	63	EntryPage.fxml	Landing page; routes to MainPage with an initial module.
MainPageController	307	MainPage.fxml	Top nav, page cache, global save shortcut, cross-page pay
NavigationService	43	—	Static accessor for the live MainPageController.
GkbOrderController	1,675	pages/gkb.fxml	GKB Pakwan order entry / edit / save.

Controller	LoC	Owns FXML	Role
KebabishOrderController	1,634	pages/kebabish.fxml	Kebabish Marki order entry / edit / save.
OuterOrderController	3,465	pages/outer.fxml	Walk-in / outer orders — the largest module.
KhataController	1,585	pages/khata.fxml	Per-person credit ledger and payments.
SalesController	2,130	pages/sales.fxml	Sales reports across modules.
ExpensesController	2,258	pages/expenses.fxml	Expenses ledger by type / item / supplier.
Saveable (interface)	5	—	Marker interface — has save().
Refreshable (interface)	5	—	Marker interface — has refreshData().

5. Modules / Pages in Depth

5.1 Entry / Landing page

Defined by **EntryPage.fxml** + **EntryPageController**. The user sees a centered title, four large clickable cards (GKB Pakwan, Kebabish Marki, Outer Pakwan, Khata), and two top-right buttons (Sales, Expenses). Each click forwards into **MainPage.fxml** already focused on the chosen module.

5.2 GKB Pakwan and Kebabish Marki orders

Two near-twin modules implemented by **GkbOrderController** and **KebabishOrderController**. Each manages its own CSV (*gkb_orders.csv*, *kebabish_orders.csv*) with the same schema:

```
Timestamp, Customer Name, Order Date, Total Amount, Discount, Paid Amount, Remaining Amount, Payment Method, KG Items, Status
```

The **KG Items** column is a single string holding the entire order: items separated by `;` , fields per item encoded as `Name(KG|P:price-units@RsRate|T:total)`. Example from the live data:

```
Mutton Rosh(240.0kg|P:240.0@Rs2700|T:648000); Mutton Saji(187.0kg|P:187.0@Rs3700|T:691900); …
```

Both controllers expose **loadOrderForEditing(Order)** so the host page can re-open an existing order in edit mode (used for the cross-page navigation from sales / khata views).

5.3 Outer Pakwan (walk-in / outer orders)

OuterOrderController is by far the largest module (~3,500 LoC). It writes to **orders.csv**, which extends the schema with extra columns for walk-in workflow:

```
Timestamp, Customer Name, Contact Number, Order Date, Total Amount, Discount, Cash, Online, Bank, Check, Check Bank, Check Number, Total Paid, Remaining Amount, Payment Method, KG Items, Extra Items, Status, Payment History
```

Notice the multi-tender split (**Cash / Online / Bank / Check** with a separate Check Bank and Check Number), the **Extra Items** string for non-KG line items, and a **Payment History** column that lets the controller keep a running log of partial payments on a single order. This is also the module that integrates most tightly with Khata when an order is left partially paid.

5.4 Khata (credit ledger)

KhataController tracks a per-person ledger of debits (purchases on credit) and credits (payments). It writes to **khata_records.csv**:

```
Date, Person, Type, Description, Amount, Balance, TransactionType
```

(plus a few trailing fields used internally for unit pricing and a UUID per row, as visible in the existing data). It also reads / writes **persistent_khata_payments.csv** for payments that should survive ledger rebuilds.

5.5 Sales

SalesController aggregates across the three order CSVs to produce sales reports. Because every order CSV uses a similar schema, the controller can sum totals, filter by date range, and compare branches. There is no separate sales CSV — sales is a read-only view over the order history.

5.6 Expenses

ExpensesController writes to **expenses.csv**:

```
Date, Type, Item, KG, CostPerKG, TotalAmount, Supplier, BillNo
```

Type is a free-form category (e.g. GENERAL). KG / CostPerKG / TotalAmount support both per-KG purchases (meat, rice) and flat-amount expenses (petrol, naan) — the existing data shows both styles co-existing in the same file.

6. Data Files (CSV Persistence)

All persistent data lives next to the JAR. There is no database, no migration tool, and no schema file other than the headers in the CSVs themselves. This is by design — the system is meant to be portable by simply copying the folder.

File	Owned by	Approx size	Notes
orders.csv	OuterOrderController	≈ 46 KB	Walk-in orders, full multi-tender + payment histo
gkb_orders.csv	GkbOrderController	≈ 3.6 KB	Catering orders for GKB Pakwan
kebabish_orders.csv	KebabishOrderController	≈ 3.8 KB	Catering orders for Kebabish Marki
expenses.csv	ExpensesController	≈ 15 KB	Daily expenses by type / item / supplier
khata_records.csv	KhataController	small	Credit ledger transactions
persistent_khata_payments.csv	KhataController	small	Payments that survive ledger rebuilds

Schema summary

CSV	Header
orders.csv	Timestamp, Customer Name, Contact Number, Order Date, Total Amount, Discount, Cash, Onli
gkb_orders.csv / kebabish_orders.csv	Timestamp, Customer Name, Order Date, Total Amount, Discount, Paid Amount, Remaining Am
expenses.csv	Date, Type, Item, KG, CostPerKG, TotalAmount, Supplier, BillNo
khata_records.csv	Date, Person, Type, Description, Amount, Balance, TransactionType (+ trailing internal fields inc

7. Build & Distribution

compile_and_run.sh (dev)

- Wipes `bin/` and recreates it.
- Generates `sources.txt` with all `.java` paths.
- Compiles with `javac --module-path /Applications/javafx-sdk-25/lib --add-modules javafx.controls,javafx.fxml -cp "lib/*" -d bin @sources.txt`.
- Copies all resources (`*.fxml *.css *.properties *.png *.jpg`) into `bin/` via `tar` piping (preserves directory structure on macOS).
- Runs `java --module-path /Applications/javafx-sdk-25/lib --add-modules javafx.controls,javafx.fxml -cp "bin:lib/*" Launcher`.

build_jar.sh (release)

- Compiles the same way as the dev script.
- Copies all resources into `bin/`.
- Extracts `lib/*.jar` contents into `bin/` (fat-JAR style) and removes `META-INF/` to avoid signature conflicts.
- Packages everything with `jar --create --file kebabish.jar --main-class Launcher -C bin .`.
- Result: a single `kebabish.jar` that embeds the libraries and can be moved between machines.

Manifest.txt

```
Main-Class: Launcher
```

Required local toolchain

- Java JDK with `javac` / `java` / `jar` on PATH (Java 11+ for the module-path syntax).
- JavaFX SDK installed at `/Applications/javafx-sdk-25/lib` (the build scripts hard-code this path).
- macOS — the scripts use `tar` pipes that are written for macOS / BSD `find` + `tar`.

8. Keyboard Shortcuts & UX Notes

Global save

MainPageController.registerF8Shortcut() wires three save triggers, all of which call the same `handleGlobalSave()`:

- **F8** — registered both as a scene-level event filter and as an accelerator (so it fires regardless of which control has focus).
- **Cmd+S** on macOS, **Ctrl+S** on Windows / Linux — via a `SHORTCUT_DOWN` key combination, which JavaFX maps to the platform-correct meta key automatically.

If the active controller doesn't implement **Saveable**, the shortcut prints a console warning and is otherwise a no-op.

Navigation behavior

- First load: FXML is parsed and the controller is constructed.
- Subsequent loads: the cached node + controller are reused; `refreshData()` is called if available.
- "Edit existing order" deep-links: bypass the cache so the page is loaded fresh, then receive an **Order** via `loadOrderForEditing(...)`.
- Active nav button is highlighted via the `nav-button-active` CSS class swap.

Window sizing

Initial window has a 1000×700 minimum. Once a module is opened from the entry page, the stage is set to maximized (**EntryPageController.loadMainPage()**).

Stylesheet

A single global stylesheet, **src/styles/main.css** (~200 lines), is attached to every scene. It defines the look of `top-button`, `restaurant-card`, `card-icon`, `card-title`, `main-title`, `nav-button`, `nav-button-active`, etc. — the classes referenced from the FXML files.

9. External Libraries

Library	Version	Purpose
JavaFX (controls + fxml)	25	UI toolkit; not bundled — supplied via /Applications/javafx-sdk-25/lib
ZXing core	3.5.2	Barcode / QR encoding (in lib/core-3.5.2.jar)
ZXing javase	3.5.2	Java SE bridge for ZXing (in lib/javase-3.5.2.jar) — bundled into the fat-JAR

Everything else is from the JDK standard library — there is no Maven / Gradle build, no Spring, no logging framework. CSV reading and writing is hand-rolled inside the controllers.

10. Known Quirks & Notes

- **Hard-coded JavaFX SDK path.** Both build scripts assume `/Applications/javafx-sdk-25/lib`. On another machine you'll need either to install JavaFX 25 there or to edit the scripts.
- **Window title says "Pakwan Restaurant System".** Internally the project is called GKB and the JAR is *kebabish.jar*. All three names refer to the same app — Pakwan is the umbrella, GKB & Kebabish are branches.
- **Big controllers.** The largest controllers are 1,500 – 3,500 lines. They mix UI wiring, CSV I/O, and business rules. If you plan to extend the app, splitting them into model + service + controller is the highest-leverage refactor.
- **CSVs are the source of truth.** No locking, no schema version, no header validation beyond what each controller does ad-hoc. If two copies of the app run at the same time on the same folder, last-write-wins.
- **khata_records.csv has more columns than its header advertises.** The current data row contains trailing fields (unit price, another amount, a label, and a UUID) that the controller knows about but the header line does not. Keep that in mind before hand-editing the file.
- **pakwan.zip** in the project root is an archived snapshot of an earlier version, not part of the active build.
- **Test.class / TestGkb.class** sit in `src/` with no matching `.java` files — old throwaway test artifacts. Safe to delete.

End of report — generated for hikull16@gmail.com on 2026-05-08.