SCHOOL OF COMPUTER SCIENCE ENGINEERING AND INFORMATION SYSTEMS

FALL SEMESTER 2025-2026

ANDROID PROGRAMMING

SLOT: A2+TA2

TITLE: QUEUE MANAGEMENT SYSTEM (ONLINE RATION SHOP)

FINAL PROJECT REVIEW

Under the Guidance of prof VENKETESH P

TEAM MEMBERS:

DRISHA .P -22MIS0603

Sri Pradeepaa KS -22MIS0582

ABSTRACT

The **Online Ration Shop Management System** is a smart application designed to digitalize and simplify the process of managing public distribution activities. Traditionally, ration distribution is handled manually, leading to inefficiencies such as long queues, poor record management, and lack of prioritization for special citizens. This project introduces an Android-based solution that makes the process more transparent, efficient, and user-friendly for both customers and administrators.

The proposed system enables customers to register using their basic details and ration card numbers, select the items they need (such as rice, dal, wheat, sugar, oil, and kerosene), and receive a unique digital token. The system gives special priority to elderly citizens and pregnant women, ensuring fair service for all. Administrators can view the queue, serve customers in the correct priority order, and reset or update records directly from the admin dashboard.

This application enhances convenience, reduces manual errors, and minimizes waiting time at ration shops. It provides a paperless and transparent way of managing ration distribution. The project demonstrates how technology can bring efficiency and accountability to government-supported food distribution systems, improving the overall experience for both staff and citizens.

⚙️ PROBLEM STATEMENT

The manual ration distribution system in India and many other regions faces several issues such as time delays, inaccurate record-keeping, and lack of priority handling for vulnerable groups. Customers often have to wait in long queues for hours, and maintaining proper transaction records manually becomes challenging for the shop administrators. There is also limited transparency in how tokens are managed, often resulting in customer dissatisfaction and disputes.

There is currently no simple and centralized platform that allows both customers and ration shop administrators to interact efficiently. Most ration shops still rely on physical tokens and registers, which can be lost, duplicated, or manipulated. Moreover, customers have no way to know their queue position or receive special consideration for genuine needs like pregnancy or old age.

The **Online Ration Shop Management System** aims to solve these problems by offering a digital solution where customers can quickly register, obtain digital tokens, and receive service in the proper order of priority. By automating token management and introducing a database-driven queue, the system ensures fairness, reduces human intervention, and enhances transparency in ration distribution.

💡 INTRODUCTION

The **Online Ration Shop Management System** is an Android-based mobile application developed to streamline the operations of ration shops. This project bridges the gap between technology and essential government services by allowing digital management of ration distribution.

It provides separate login modules for administrators and customers, thereby maintaining data security and smooth access control. Customers can register for tokens, select required commodities, and receive their allotted turn without unnecessary waiting.

From the administrator's perspective, the system offers complete control over the queue. The admin can view all registered customers, check their details (including priority status), and serve them in a systematic order.

Special logic is used to ensure that pregnant women and elderly citizens receive priority service, thus aligning the project with inclusivity and welfare goals. All customer data and transactions are stored in an SQLite database for easy access and future record verification.

The system not only saves time but also encourages transparency and accountability in ration distribution. It eliminates the risk of human error, duplication, and mismanagement.

The traditional manual process of ration management often results in long queues, miscommunication, and human errors, affecting both the customers and shopkeepers.

By developing this Android-based application, the project bridges the gap between technology and social welfare.

Overall, the project demonstrates how digital transformation can improve public welfare services through automation, efficiency, and fairness — making it an ideal solution for modernizing ration shop operations.

Furthermore, the **Online Ration Shop Management System** contributes to the government's digital initiatives and e-Governance goals. With features such as token generation, priority handling, and data storage, it acts as a model for other civic distribution services.

This project demonstrates how technology can be effectively utilized to promote transparency, reduce corruption, and enhance citizen satisfaction.

Through its practical approach and scalable design, the system can be implemented across multiple ration shops, making it a reliable step toward a smarter and more efficient distribution network.

SYSTEM ARCHITECTURE

The **Online Ration Shop Management System** follows a **client–server architecture**, where the Android application serves as the client interface and SQLite acts as the local database server.

The system is composed of two main modules — the **Customer Module** and the **Admin Module** — both interacting with a shared database through structured queries.

This modular design ensures that each user group has specific functionalities, maintaining clarity, scalability, and security within the application.

The customer module allows users to register their details, select ration items, and generate tokens, while the admin module handles queue management, viewing of customers, and priority-based serving.

At the core of the system lies the **Database Layer (SQLite)**, which handles all persistent data operations such as storing customer details, item lists, tokens, and priority status.

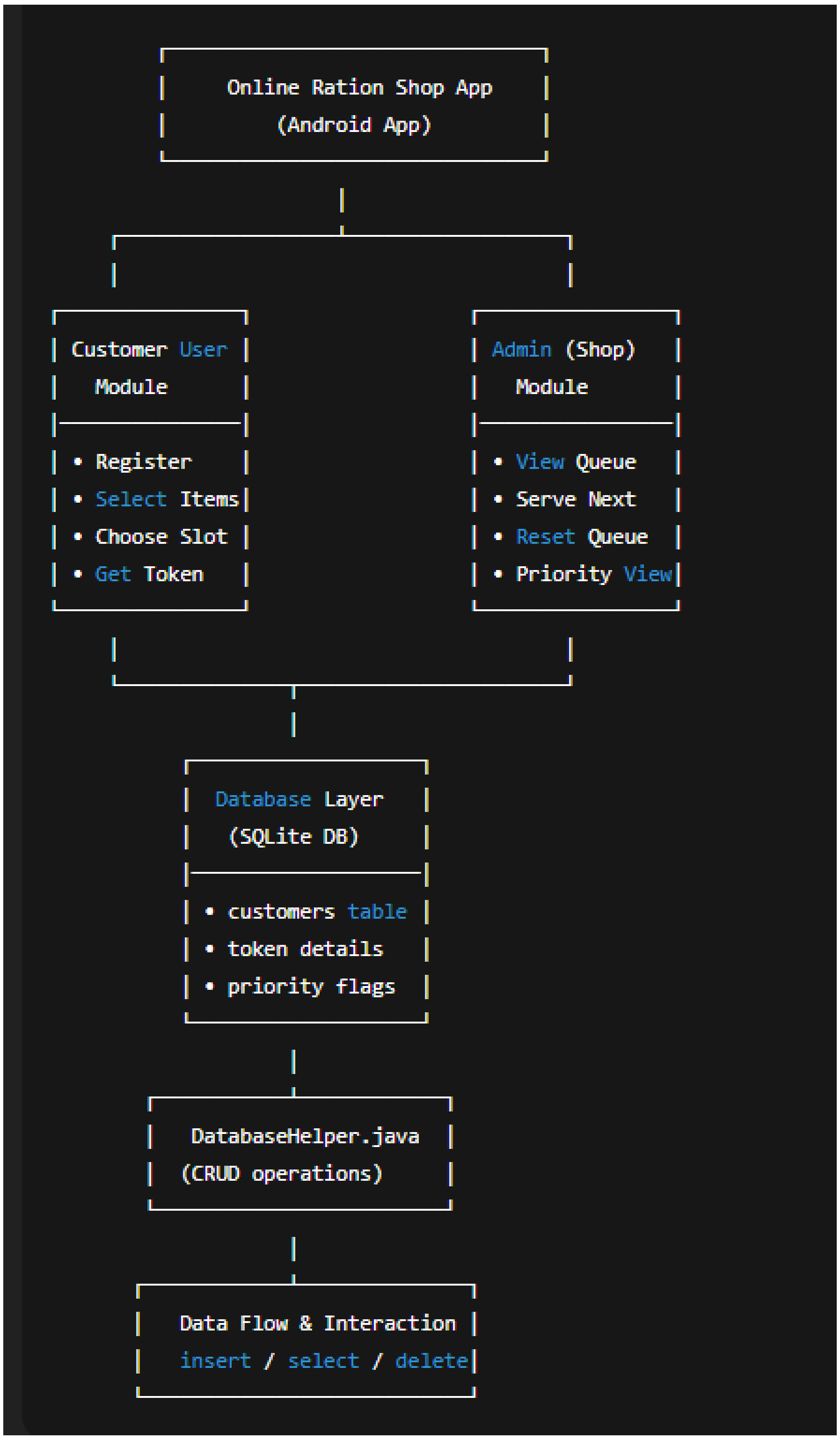
The database acts as a bridge between the frontend interface and backend logic, ensuring that data integrity and synchronization are maintained throughout the user sessions.

Through SQL operations like INSERT, SELECT, and DELETE, the application efficiently retrieves and manages data without relying on internet connectivity, making it ideal for use in rural or low-network areas.

The **overall architecture** is designed to maintain a smooth flow of information between components. When a customer registers and selects items, the details are inserted into the database and a token is generated.

The admin interface retrieves this data and displays it in priority order — pregnant women and elderly customers appear first. Upon serving, the system updates the queue and reflects the changes instantly.

This layered and event-driven architecture ensures low latency, high reliability, and simple maintainability, making the system efficient for real-time ration shop operations.



```
             ┌──────────────────────────┐
             │   Online Ration Shop App │
             │      (Android App)       │
             └──────────────────────────┘
                         │
         ┌───────────────┴───────────────┐
         │                               │
  ┌──────────────┐             ┌──────────────┐
  │Customer User │             │ Admin (Shop) │
  │   Module     │             │   Module     │
  │──────────────│             │──────────────│
  │ • Register   │             │ • View Queue │
  │ • Select Items│            │ • Serve Next │
  │ • Choose Slot│             │ • Reset Queue│
  │ • Get Token  │             │ • Priority View│
  └──────────────┘             └──────────────┘
         │                               │
         └───────────┬───────────────────┘
                     │
            ┌────────────────┐
            │ Database Layer │
            │  (SQLite DB)   │
            │────────────────│
            │ • customers table│
            │ • token details│
            │ • priority flags│
            └────────────────┘
                     │
          ┌─────────────────────┐
          │ DatabaseHelper.java │
          │ (CRUD operations)   │
          └─────────────────────┘
                     │
         ┌──────────────────────┐
         │ Data Flow & Interaction│
         │ insert / select / delete│
         └──────────────────────┘
```

⚙ METHODOLOGY USED

The development of the **Online Ration Shop Management System** follows a **modular and iterative methodology** based on the *Waterfall Model* blended with agile principles for flexibility. The process begins with requirement gathering, where both customer and admin needs were analysed to define clear system functionalities.

The primary goal was to design a system that digitizes ration shop management while maintaining simplicity for end users.

Each feature — such as customer registration, token generation, and admin queue management — was first defined conceptually and then translated into technical specifications during the design phase.

In the **design and implementation phase**, the system was divided into two core modules: the *Customer Module* and the *Admin Module*.

The customer interface was developed using XML layouts for the UI and Java for backend logic. SQLite was chosen as the local database for offline storage, ensuring that the application functions smoothly even without internet connectivity.

Data flow diagrams, entity-relationship models, and UML diagrams were prepared to visualize interactions between system components. The development process was carried out iteratively — coding, testing, and refining each component to achieve optimal performance and usability.

Finally, in the **testing and evaluation phase**, multiple testing techniques were applied, including *unit testing*, *integration testing*, and *user acceptance testing (UAT)*.

Each module was validated individually to ensure reliability and accuracy of data transactions. Emphasis was placed on testing user inputs, database queries, and token priority logic.

The feedback collected during testing was used to enhance interface design and improve system response time.

This structured methodology ensured a stable, efficient, and user-friendly system that meets the objectives of digitizing ration distribution while ensuring fairness, accessibility, and transparency.

🧩 MODULE DESCRIPTIONS

1. High-level overview

The app is an Android client that runs locally on the device and stores data in an embedded SQLite database. Two main user roles exist: **Customer** (mobile UI to request tokens) and **Admin** (manage queue and serve customers). The app is offline-first — all operations use the local DB; later you can add server sync for multi-terminal sharing. The architecture is simple, robust and event-driven: user actions (take token, serve next, reset) trigger DB writes/reads that update the UI (RecyclerView) immediately.

2. UI / Front-end design (detailed)

Breakdown of screens and components:

1. **MainActivity (Landing)**
    - Title (app name), two large buttons: Customer and Admin.
    - Nice background, accessible fonts, responsive layout.
2. **CustomerActivity**
    - Title bar: "Customer Token Registration".
    - Form fields:
        - EditText etName — full name.
        - EditText etRation — ration card number (unique identifier).
    - Item selection: six CheckBoxes (Rice, Dal, Sugar, Wheat, Oil, Kerosene).
        - Place them in a vertical group or 2-column grid for compactness.
        - Each CheckBox has a corresponding price label (small TextView) so users know cost.
    - Priority toggles:
        - CheckBox cbPregnant
        - CheckBox cbElderly
    - Slot selection:
        - RadioGroup rgSlot with RadioButton Morning/Afternoon/Evening.
    - Action:

        - Button btnTakeToken — validates inputs, computes total, inserts record, shows Toast with Token # and Total.
    - Visual feedback: clear success/failure Toasts and field reset on success.
3. **AdminActivity**
    - Title: “Admin - Queue Management”.
    - Control buttons: Serve Next, Refresh, Reset Queue.
    - Queue display: RecyclerView with item_customer.xml cards showing:
        - Token #, Name, Ration Card, Items, Total, Slot.
        - Priority label area: visible only when isPregnant or isElderly. Color-coded (pink for pregnant, orange for elderly).
    - Behavior:
        - Serve Next removes the top-priority row (pregnant > elderly > normal, within same priority by token ascending).
        - Refresh reloads from DB.
        - Reset clears DB table after confirmation dialog.

Accessibility/UI tips:

- Use readable colors/contrast; ensure title color contrasts with animated background.
- Large touch targets for checkboxes/buttons.
- Use content descriptions for icons and support talkback.

3. Database design (SQLite) — schema & SQL

Single table design (simple, denormalized — sufficient for small shops). If you later want itemized billing per quantity, convert to two tables (customers + customer_items).

Table: customers

- id INTEGER PRIMARY KEY AUTOINCREMENT
- name TEXT NOT NULL
- ration_card TEXT NOT NULL
- items TEXT — comma-separated list like "Rice, Dal, Oil"
- slot TEXT — "Morning"|"Afternoon"|"Evening"
- isPregnant INTEGER — 0 or 1
- isElderly INTEGER — 0 or 1

- total INTEGER — total amount in rupees
- token INTEGER — sequential token number

Create SQL

```
CREATE TABLE customers (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  ration_card TEXT NOT NULL,
  items TEXT,
  slot TEXT,
  isPregnant INTEGER DEFAULT 0,
  isElderly INTEGER DEFAULT 0,
  total INTEGER DEFAULT 0,
  token INTEGER
);
```

Important queries

- Insert:

```
INSERT INTO customers (name, ration_card, items, slot, isPregnant, isElderly, total, token)
VALUES (?, ?, ?, ?, ?, ?, ?, ?);
```

- Next token:

```
SELECT IFNULL(MAX(token),0) + 1 FROM customers;
```

- Get all (priority order):

```
SELECT * FROM customers
ORDER BY isPregnant DESC, isElderly DESC, token ASC;
```

- Get next to serve:

```
SELECT id FROM customers
ORDER BY isPregnant DESC, isElderly DESC, token ASC
LIMIT 1;
```

- Delete by id:

```
DELETE FROM customers WHERE id = ?;
```

4. Data flow & sequence (textual diagrams)

Data flow (high-level)

[Customer UI] --(submit details)--> [CustomerActivity] --(compute total, priority)--> [DatabaseHelper.insertCustomer] --> [SQLite customers table]

[AdminActivity] --(load queue)--> [DatabaseHelper.getAllCustomers] --> [RecyclerView -> QueueAdapter display]

[AdminActivity Serve] --(serve)--> [DatabaseHelper.serveNextCustomer] --> update DB --> reload UI

Sequence: Customer takes token

1. User fills form and taps Take Token.
2. Activity validates inputs (non-empty, slot selected).
3. Activity computes total by summing selected item prices.
4. Activity calls DatabaseHelper.insertCustomer(name, rationCard, items, slot, isPregnant, isElderly, total).
    - Inside DB helper: compute nextToken = SELECT MAX(token)+1.
    - Insert values with token.
5. DB returns success; Activity shows Toast: Token #<token> - Total ₹<total>.
6. Admin RecyclerView should reflect new row after loadQueue() / Refresh.

Sequence: Admin serves next

1. Admin taps Serve Next.
2. Admin calls DatabaseHelper.serveNextCustomer().
    - DB finds top id with ordering and deletes it.
3. App reloads queue and UI updates.

5. Core algorithms & pseudocode

Token generation (atomic within single DB connection)

- Query SELECT MAX(token) FROM customers -> maxToken
- nextToken = (maxToken == NULL ? 1 : maxToken+1)
- Insert record with nextToken.

Priority ordering (SQL ORDER BY)

- Use ORDER BY isPregnant DESC, isElderly DESC, token ASC.

- This ensures pregnant first, then elderly, then others, with older tokens ahead.

Pseudocode: insertCustomer(...)

```
function insertCustomer(name, rationCard, items, slot, isPregnant, isElderly, total):
    db = getWritableDatabase()
    nextToken = db.query("SELECT IFNULL(MAX(token),0)+1 FROM customers")
    values = { name, rationCard, items, slot, isPregnant?1:0, isElderly?1:0, total, nextToken }
    result = db.insert("customers", values)
    return result != -1
```

Pseudocode: serveNextCustomer()

```
function serveNextCustomer():
    db = getWritableDatabase()
    cursor = db.rawQuery("SELECT id FROM customers ORDER BY isPregnant DESC, isElderly DESC, token ASC LIMIT 1")
    if cursor.moveToFirst():
        id = cursor.getInt(0)
        db.delete("customers", "id = ?", [id])
        return true
    else:
        return false
```

6. Classes & Responsibilities (Java/Android)

- **DatabaseHelper (SQLiteOpenHelper)**
    - Create/upgrade DB, insertCustomer, getAllCustomers, serveNextCustomer, resetQueue, getNextToken.
- **CustomerActivity (AppCompatActivity)**
    - Input validation, compute selected items & total, call DB insert, show Toasts.
- **AdminActivity (AppCompatActivity)**
    - Load queue from DB, configure RecyclerView, manage serve/refresh/reset operations.
- **QueueAdapter (RecyclerView.Adapter)**

    - Bind Customer model to item_customer.xml, show/hide priority label and set colors.
- **Customer (POJO)**
    - Fields: id, token, name, rationCard, items, slot, isPregnant, isElderly, total; getters/setters.

7. UI components mapping to code

- EditText -> findViewById(R.id.etName)
- CheckBox -> findViewById(R.id.cbRice) etc.
- RadioGroup -> findViewById(R.id.rgSlot) + getCheckedRadioButtonId().
- RecyclerView -> findViewById(R.id.recyclerViewQueue) + LinearLayoutManager.
- Adapter -> new QueueAdapter(customersList).

8. Error handling & input validation

- Validate non-empty name and rationCard.
- Ensure at least one slot selected (radio button). If not, show Toast.
- For items: allow zero items? Prefer to require at least one item; else show warning.
- DB insert: check return value != -1. Show error Toast on failure.
- Protect against SQL injection: using ContentValues avoids manual concatenation.
- When serving/resetting, show confirmation dialog (Admin) to prevent accidental deletes.

9. Security & privacy considerations

- Sensitive data: ration card numbers are personal identifiers. If storing locally, protect DB:
    - Use app-level encryption for DB (SQLCipher) if needed.
    - Restrict backups or set android:allowBackup="false" if sensitive.
- Authentication: add Admin login (username/password stored securely — hashed) if multiple users.
- Use least-privilege patterns; avoid sending sensitive data without HTTPS if sync added.

- Logging: avoid printing ration numbers in logs.

10. Testing & QA

- **Unit tests** for DB helper: insert, getAll, serveNext, resetQueue.
- **Integration tests**: flow from CustomerActivity to DB and AdminActivity displays the same.
- **Manual UAT**: simulate multiple customers, mark pregnant/elderly, ensure order changes.
- **Edge cases**:
  - No items selected.
  - Very long name/ration strings.
  - Two customers with same ration number (should be allowed but consider unique constraint if needed).
  - Database upgrade path — drop or migrate table safely.
- **Performance**: For small queues (<1000), SQLite is fine. For scaling, consider remote server.

11. Extensibility & future enhancements

- Add server-client sync to centralize queues across multiple terminals.
- Add item quantities and dynamic pricing (customer chooses quantity per item).
- Integrate Aadhaar or government API for verification (requires strict security & legal compliance).
- Add notifications (SMS or push) to customers when their token is near.
- Analytics dashboard for monthly distributions.

12. Appendix — Example class skeletons (quick reference)

DatabaseHelper

```
public boolean insertCustomer(..., int total) { ... }

public List<Customer> getAllCustomers() { ... }

public boolean serveNextCustomer() { ... }

public void resetQueue() { ... }
```

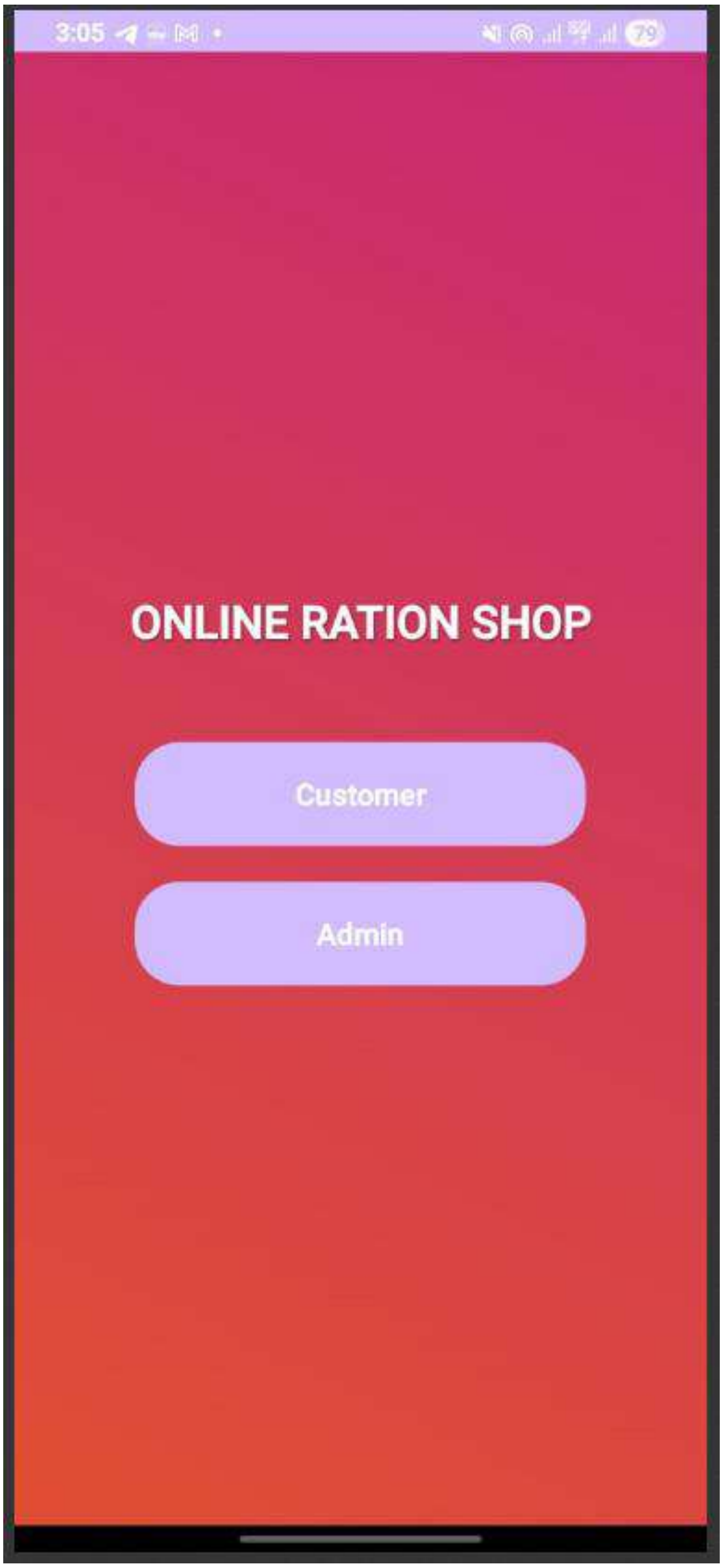
CustomerActivity (key snippet)

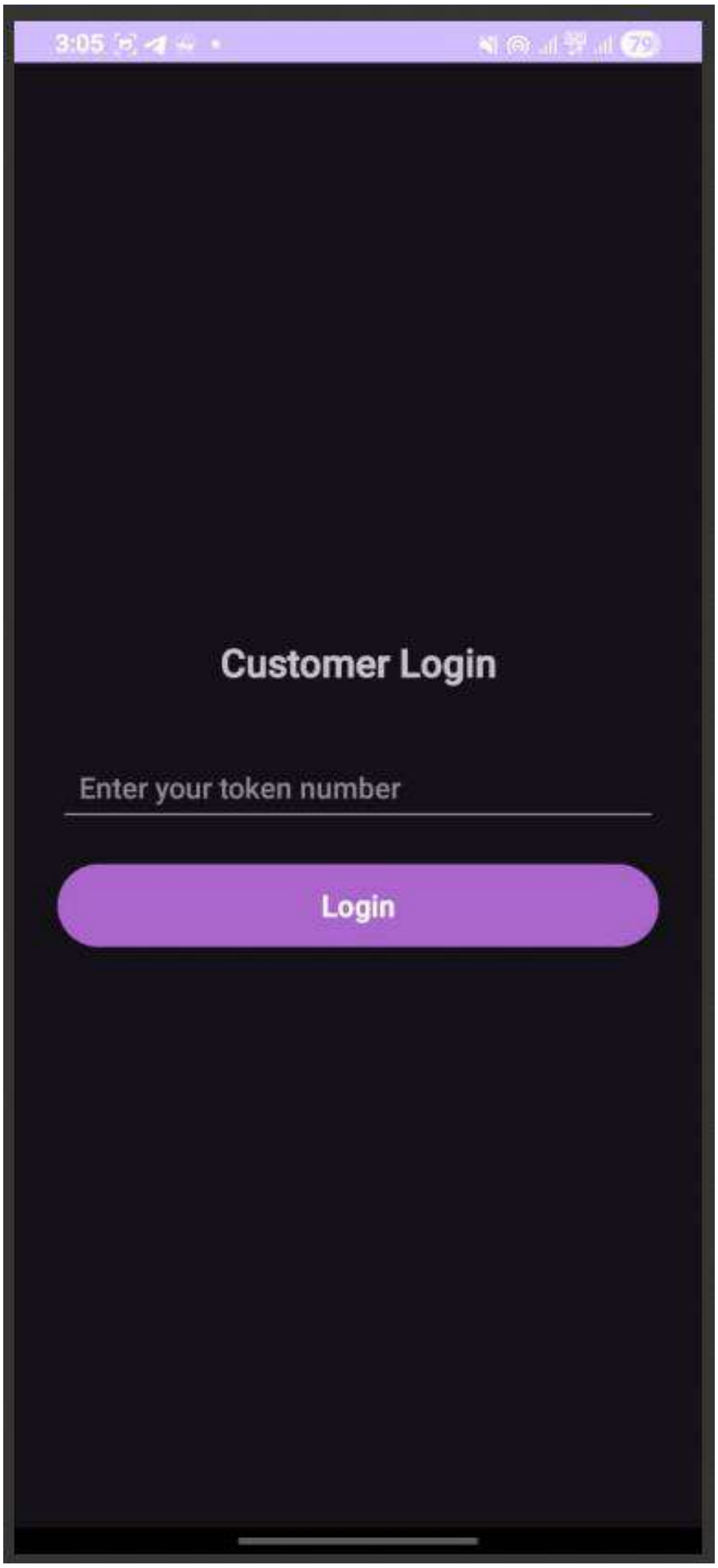
```
int total = calculateTotalFromCheckboxes();

db.insertCustomer(name, ration, items, slot, cbPregnant.isChecked(), cbElderly.isChecked(), total);
```

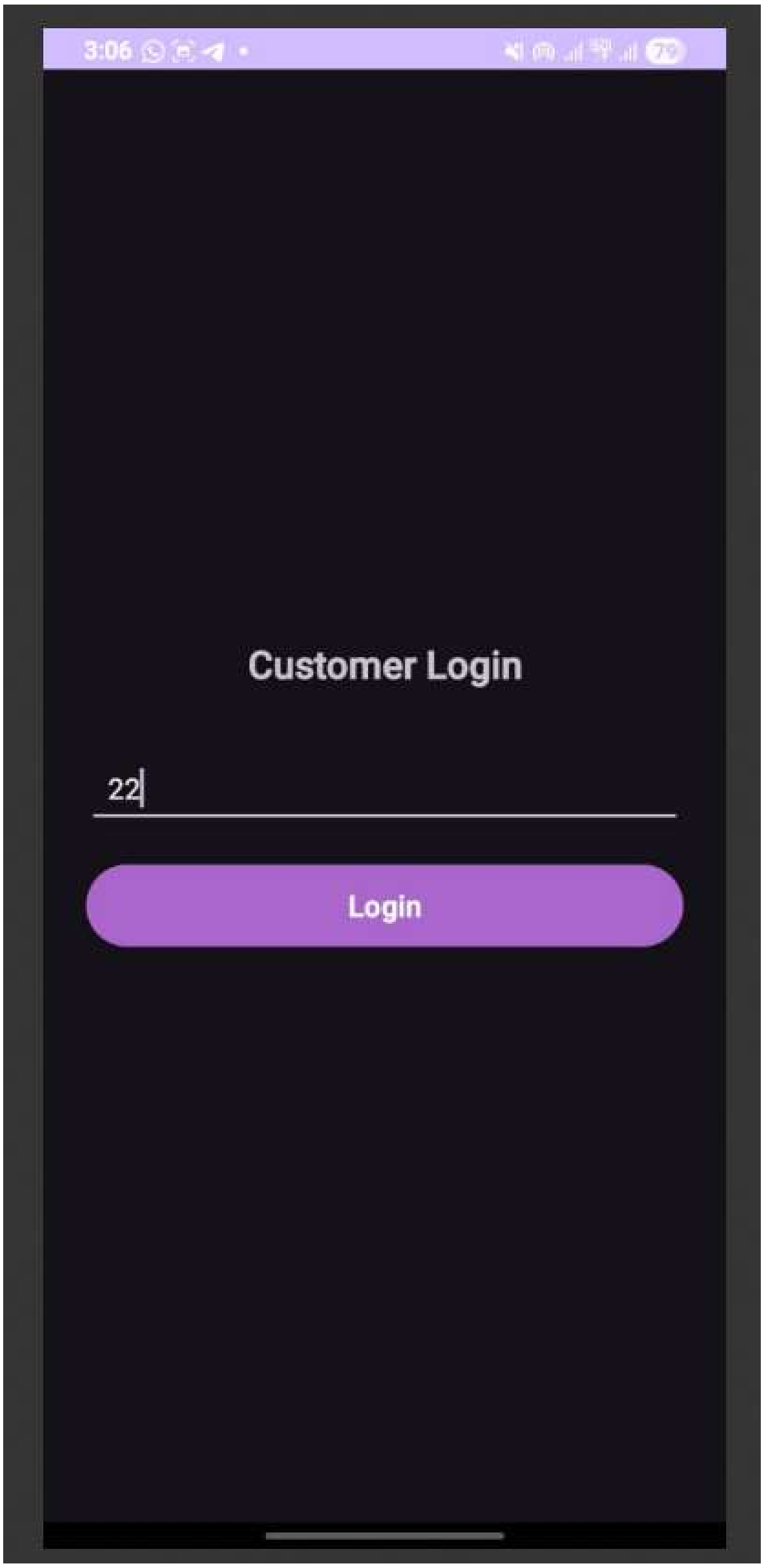
QueueAdapter

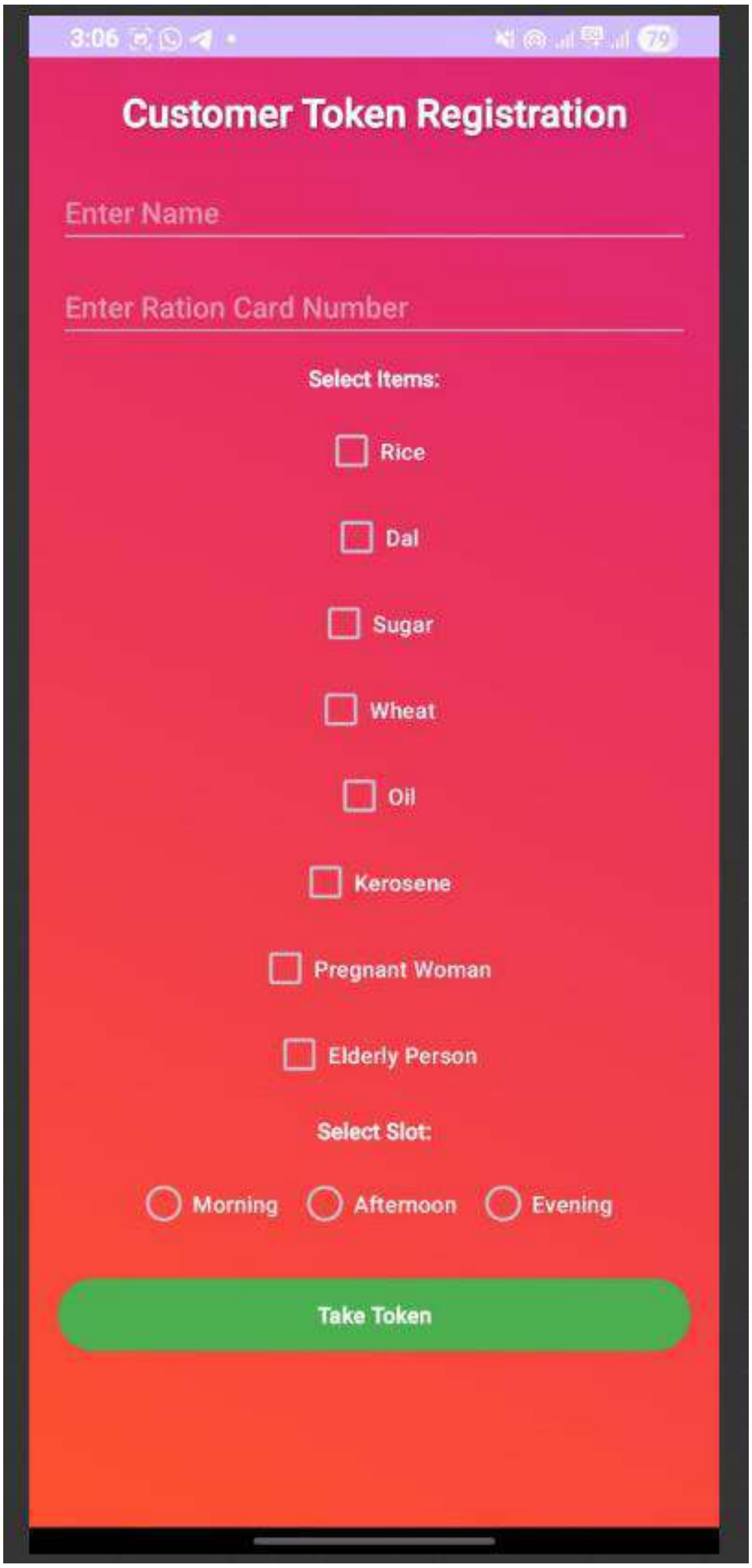
```
if (c.isPregnant()) { showPriority("Pregnant", R.color.priority_pink); }

else if (c.isElderly()) { showPriority("Elderly", R.color.priority_orange); }

else { hidePriority(); }
```

RESULTS:




3:05
Customer Login
Enter your token number
Login


3:06
Customer Login
22
Login


3:06

Customer Token Registration

Enter Name

Enter Ration Card Number

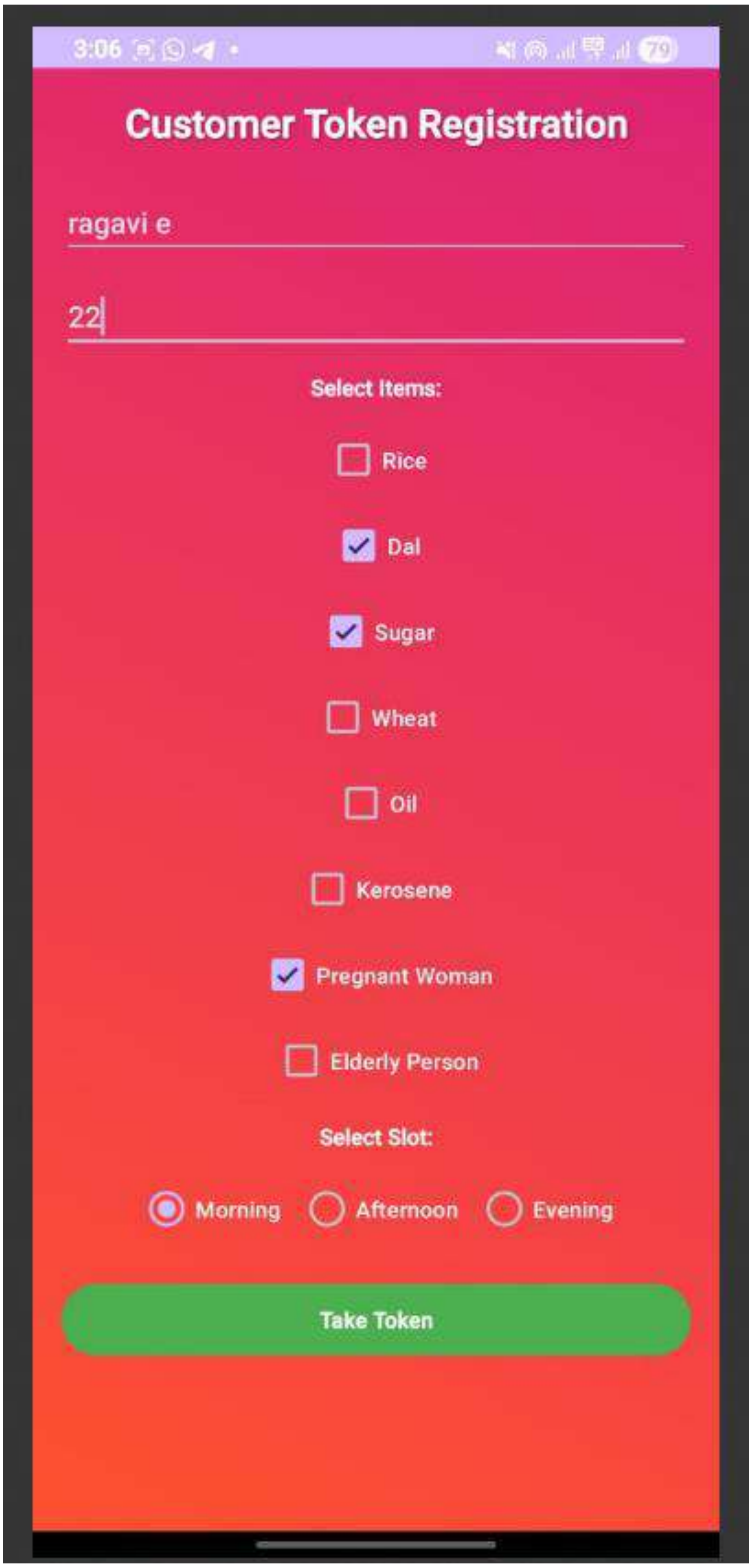
Select Items:

- [ ] Rice
- [ ] Dal
- [ ] Sugar
- [ ] Wheat
- [ ] Oil
- [ ] Kerosene
- [ ] Pregnant Woman
- [ ] Elderly Person

Select Slot:

○ Morning ○ Afternoon ○ Evening

Take Token



3:06

Customer Token Registration

ragavi e

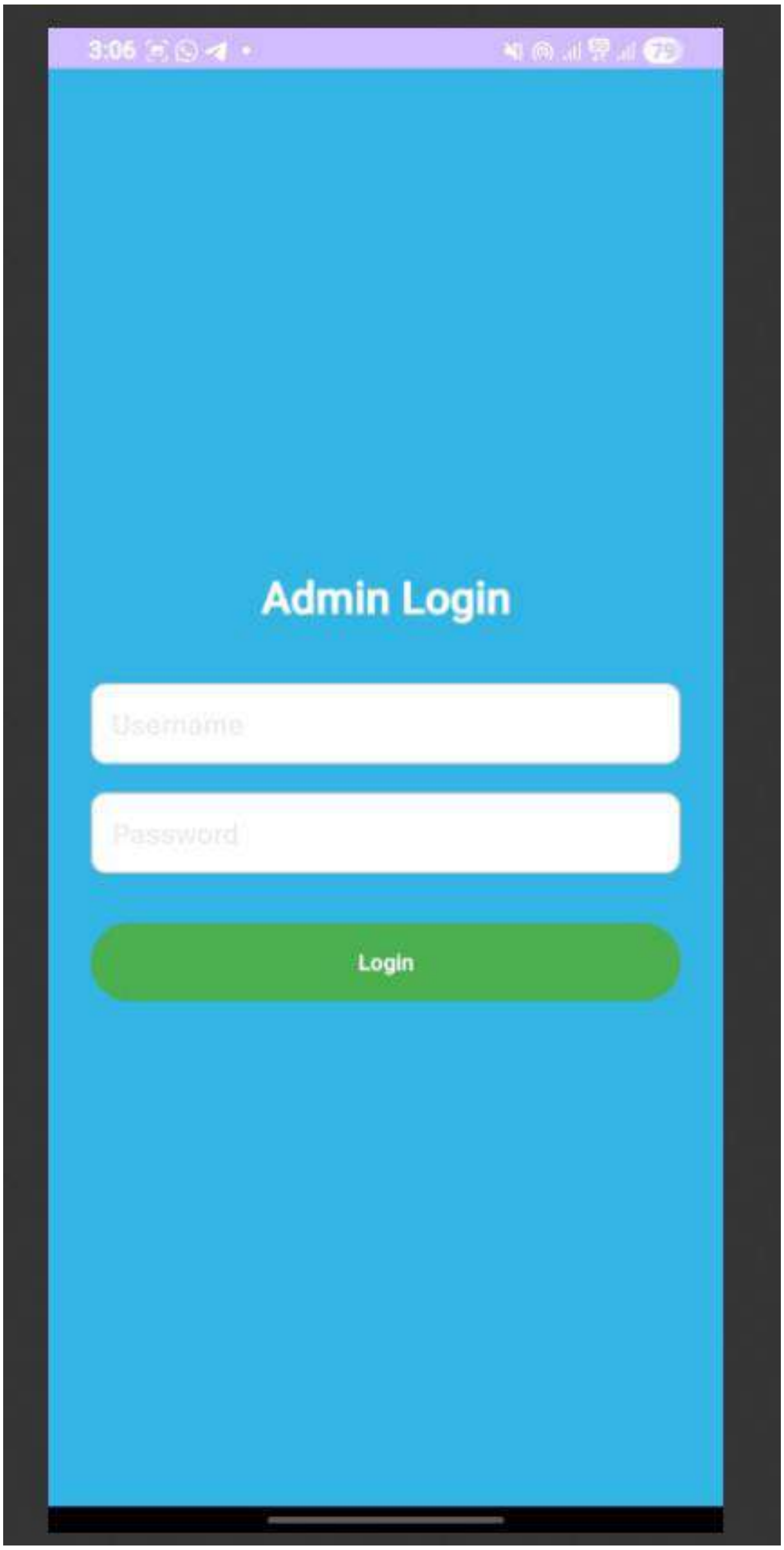
22

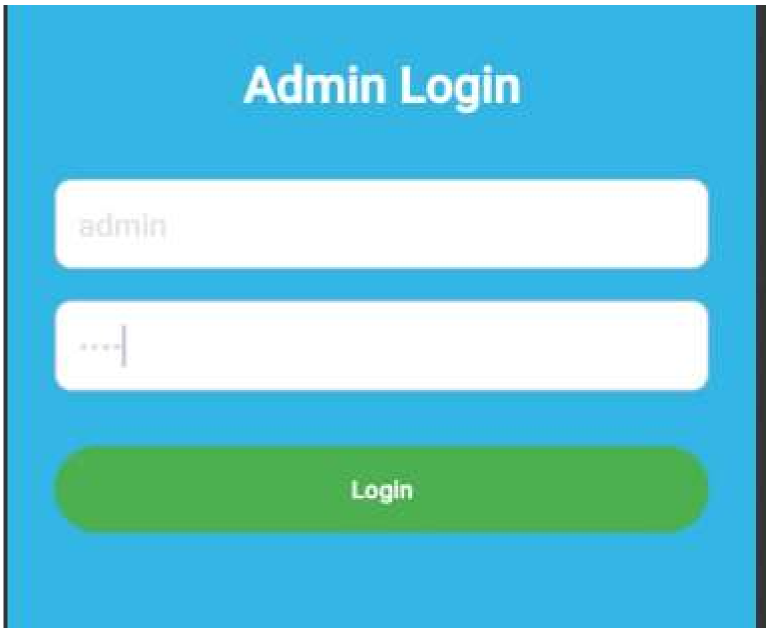
Select Items:

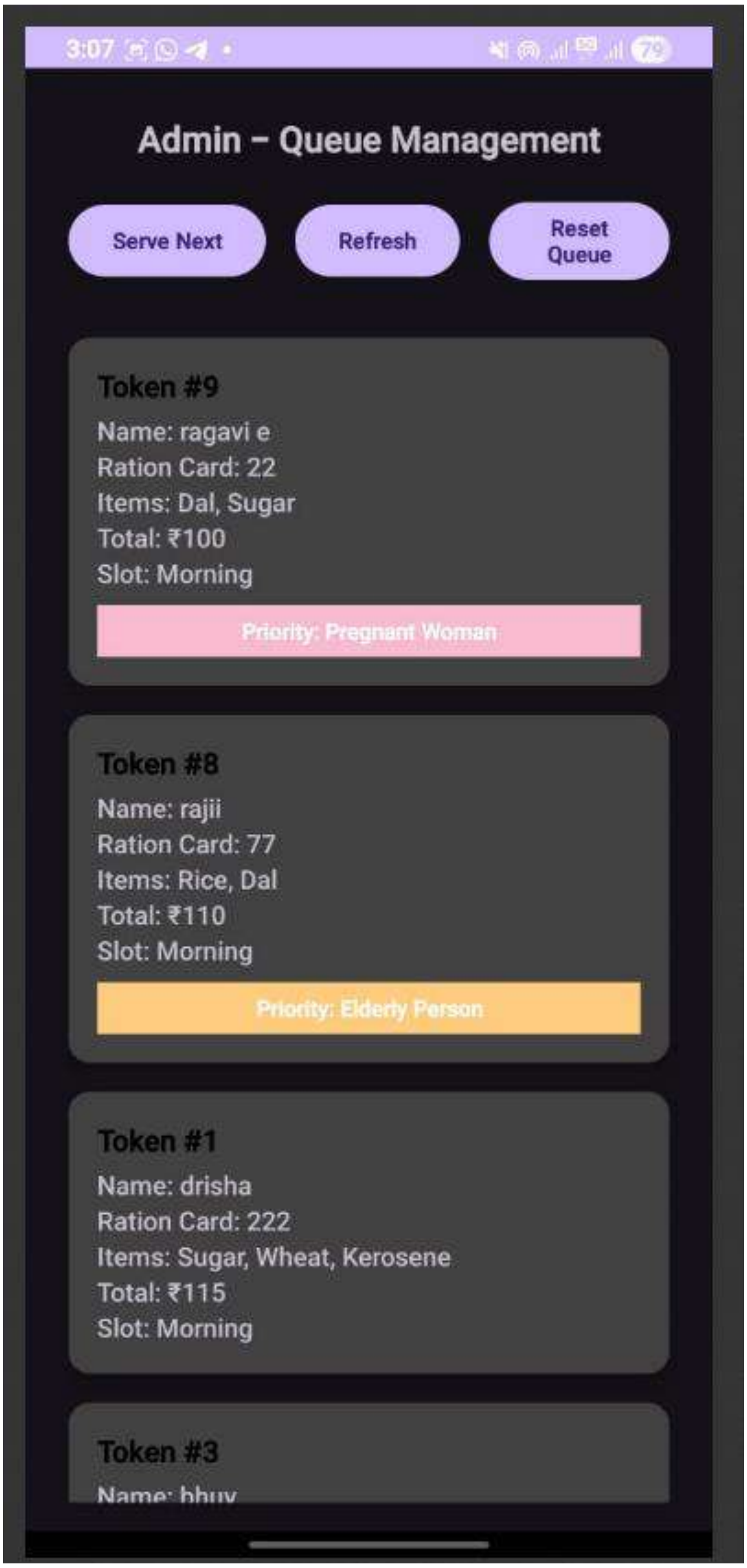
- [ ] Rice
- [✓] Dal
- [✓] Sugar
- [ ] Wheat
- [ ] Oil
- [ ] Kerosene
- [✓] Pregnant Woman
- [ ] Elderly Person

Select Slot:

[✓ Morning] Afternoon Evening


3:06
Admin Login
Username
Password
Login


Admin Login
admin
••••
Login

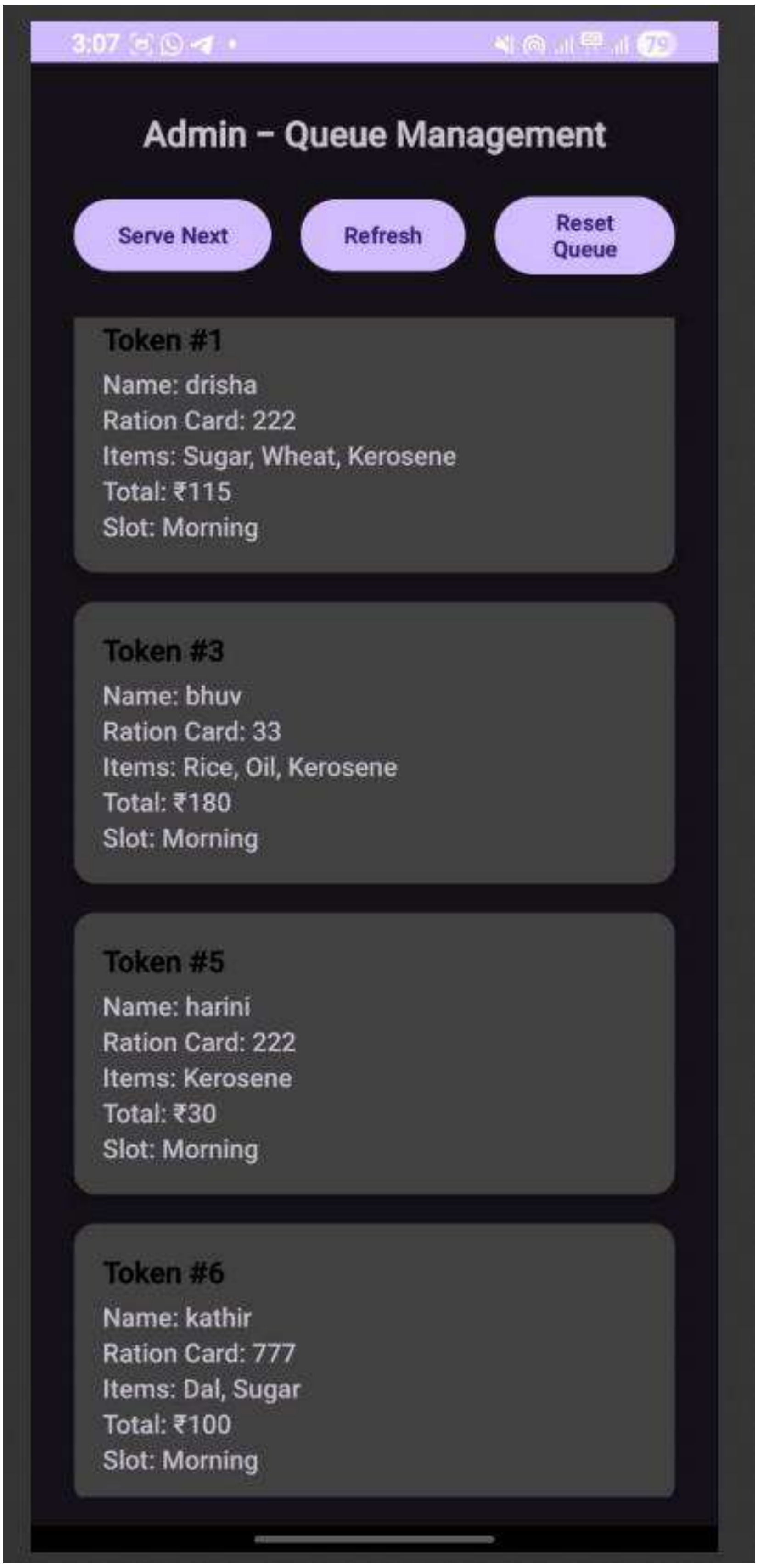


Token #9
Name: ragavi e
Ration Card: 22
Items: Dal, Sugar
Total: ₹100
Slot: Morning
Priority: Pregnant Woman

Token #8
Name: rajii
Ration Card: 77
Items: Rice, Dal
Total: ₹110
Slot: Morning
Priority: Elderly Person

Token #1
Name: drisha
Ration Card: 222
Items: Sugar, Wheat, Kerosene
Total: ₹115
Slot: Morning

Token #3
Name: bhuv



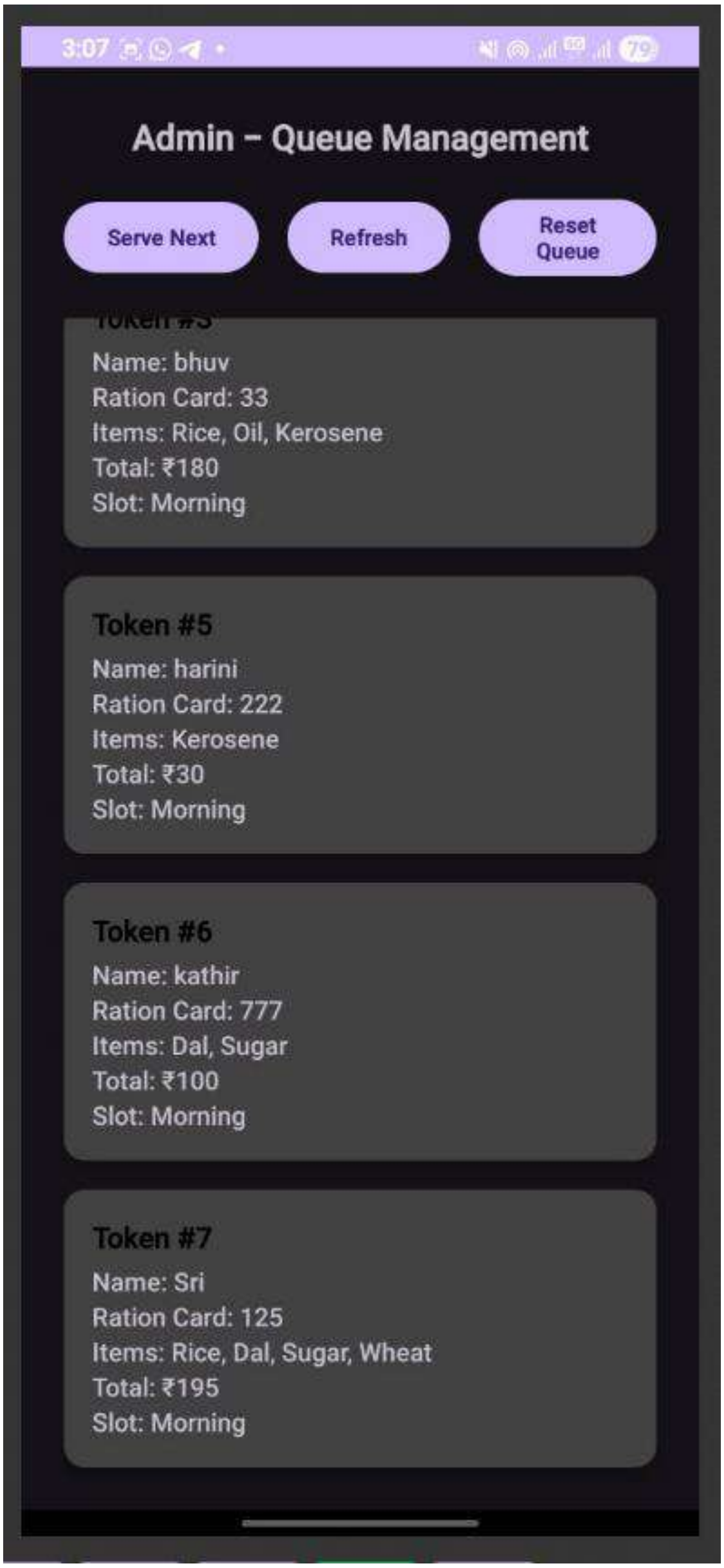
Admin – Queue Management

Token #1
Name: drisha
Ration Card: 222
Items: Sugar, Wheat, Kerosene
Total: ₹115
Slot: Morning

Token #3
Name: bhuv
Ration Card: 33
Items: Rice, Oil, Kerosene
Total: ₹180
Slot: Morning

Token #5
Name: harini
Ration Card: 222
Items: Kerosene
Total: ₹30
Slot: Morning

Token #6
Name: kathir
Ration Card: 777
Items: Dal, Sugar
Total: ₹100
Slot: Morning



Admin – Queue Management

Name: bhuv
Ration Card: 33
Items: Rice, Oil, Kerosene
Total: ₹180
Slot: Morning

Token #5
Name: harini
Ration Card: 222
Items: Kerosene
Total: ₹30
Slot: Morning

Token #6
Name: kathir
Ration Card: 777
Items: Dal, Sugar
Total: ₹100
Slot: Morning

Token #7
Name: Sri
Ration Card: 125
Items: Rice, Dal, Sugar, Wheat
Total: ₹195
Slot: Morning

🚀 FUTURE SCOPE

The **Online Ration Shop Management System** currently operates as a standalone Android application using a local SQLite database. In the future, this system can be upgraded to a **centralized, cloud-based architecture** that allows synchronization of multiple ration shops under a single network.

By integrating with an online server and government APIs, the application can automatically verify ration card details, fetch family member data, and maintain real-time stock updates across all distribution centers.

This would enable the system to function seamlessly at the state or national level, bringing transparency and uniformity to the public distribution process.

Additionally, the system can incorporate **smart card and biometric authentication** for secure user verification. Integration with **Aadhaar** or **digital ration cards** can prevent duplication and fraud.

The app can also support **SMS or push notifications** to inform customers about token status, ration availability, or government subsidy updates.

These improvements would enhance accessibility and communication, particularly in rural areas where manual information sharing is still a challenge.

The system can also include multi-language support to ensure inclusivity for users from different linguistic backgrounds.

In the long run, the project can evolve into a **complete e-governance solution** for the public distribution system. Features like **AI-based demand prediction**, **inventory analytics**, and **data visualization dashboards** can help the government forecast requirements and reduce wastage.

With a scalable backend and secure authentication mechanisms, the system can handle millions of users efficiently.

These advancements would not only streamline the ration distribution process but also support the government's vision of a **Digital India**, ensuring efficiency, transparency, and accountability in every transaction.

✅ CONCLUSION

The **Online Ration Shop Management System** successfully demonstrates how digital technology can modernize traditional ration distribution processes.

By replacing manual methods with an automated Android-based solution, this system simplifies the workflow for both customers and administrators.

Customers can easily register, choose ration items, and receive a digital token, while the admin can efficiently manage the queue, prioritize customers, and serve them systematically.

The application promotes transparency, eliminates manual errors, and reduces waiting time, thereby improving overall service quality in ration shops.

Through the implementation of SQLite as a lightweight local database, the system ensures data integrity, reliability, and offline functionality — making it suitable even for areas with limited internet connectivity.

The inclusion of special provisions for pregnant women and elderly citizens reflects the system's commitment to fairness and inclusivity.

The modular design, simple interface, and efficient backend logic make it both user-friendly and technically robust, meeting the key objectives of the project — efficiency, accessibility, and transparency.

In conclusion, this project proves to be a practical and socially beneficial application that can be further expanded for large-scale deployment across multiple ration shops.

With minor enhancements like cloud integration, biometric authentication, and real-time analytics, it can serve as a foundation for a nationwide digital ration management platform.

The **Online Ration Shop Management System** not only automates distribution but also empowers users through technology — supporting the broader vision of a **smart, digital, and corruption-free India**.