# Emergency Citizen App: Complete Flow Explanation for Beginners

Imagine you're building a super helpful app for your city, like a digital helper that can assist people during emergencies and daily life. Let me explain every single part of this app like you've never seen anything like it before!

## 🏠 What is This App?

Think of this app like a \*\*digital Swiss Army knife\*\* for citizens. Just like how a Swiss Army knife has different tools (knife, scissors, screwdriver), our app has different "tools" (features) to help people in various situations.

---

## 📱 The Complete User Journey

### 1. \*\*First Time Opening the App (Onboarding Flow)\*\*

\*\*What happens:\*\* When someone downloads and opens the app for the first time.

\*\*Real-life example:\*\*

- Sarah downloads the app on her phone

- She sees a welcome screen explaining what the app does

- The app asks for permissions (like accessing her location to find nearby hospitals)

\*\*Technical Flow:\*\*

```plaintext

User opens app → Check if user exists → If new user → Show onboarding screens → Request permissions → Go to registration

```

\*\*Endpoints needed:\*\*

- `GET /api/user/check-first-time` - Check if this is first app launch

- `POST /api/user/permissions` - Save user permission preferences

---

### 2. \*\*User Registration Flow\*\*

\*\*What happens:\*\* New users create their account.

\*\*Real-life example:\*\*

- Sarah fills out a form with her name, phone number, email

- She creates a password

- The app sends her a verification code via SMS

- She enters the code to verify her phone number

\*\*Technical Flow:\*\*

```plaintext

Fill registration form → Submit to server → Server validates data → Send SMS verification → User enters code → Account created → Login automatically

```

\*\*Endpoints needed:\*\*

- `POST /api/auth/register` - Create new user account

- `POST /api/auth/send-verification` - Send SMS verification code

- `POST /api/auth/verify-phone` - Verify the SMS code

- `POST /api/user/profile` - Save additional user details

\*\*Database operations:\*\*

- Create new record in `User` table

- Create record in `UserProfile` table

- Create record in `VerificationCode` table

---

### 3. \*\*Login Flow\*\*

\*\*What happens:\*\* Existing users sign into their account.

\*\*Real-life example:\*\*

- John opens the app

- He enters his phone number and password

- The app checks if the credentials are correct

- If correct, he's taken to the main dashboard

\*\*Technical Flow:\*\*

```plaintext

Enter credentials → Submit to server → Server validates → Generate access token → Store token locally → Redirect to dashboard

```

\*\*Endpoints needed:\*\*

- `POST /api/auth/login` - Authenticate user credentials

- `POST /api/auth/refresh-token` - Refresh expired tokens

- `GET /api/user/profile` - Get user profile data after login

---

### 4. \*\*Main Dashboard Flow\*\*

\*\*What happens:\*\* The home screen that shows everything available.

\*\*Real-life example:\*\*

- After logging in, Maria sees a dashboard with big buttons for:

- Emergency Services (red button)

- Nearby Help (blue button)

- My Reports (green button)

- Digital Services (purple button)

- Notifications (bell icon with a red dot if there are new ones)

\*\*Technical Flow:\*\*

```plaintext

User logs in → Load dashboard → Fetch user's recent activities → Fetch notifications count → Fetch nearby services → Display everything

```

\*\*Endpoints needed:\*\*

- `GET /api/dashboard/summary` - Get dashboard overview data

- `GET /api/notifications/unread-count` - Count of unread notifications

- `GET /api/services/nearby` - Get nearby services based on user location

- `GET /api/reports/recent` - Get user's recent reports

---

### 5. \*\*Emergency Services Flow\*\*

\*\*What happens:\*\* When someone needs immediate help.

\*\*Real-life example:\*\*

- Ahmed sees a car accident

- He taps the red "Emergency" button

- The app shows options: Police, Fire Department, Ambulance

- He selects "Ambulance"

- The app automatically gets his location and calls emergency services

- It also sends his location to the emergency dispatcher

\*\*Technical Flow:\*\*

```plaintext

Tap Emergency → Show emergency types → Select type → Get user location → Create emergency report → Send to emergency services → Show confirmation → Track response

```

\*\*Endpoints needed:\*\*

- `GET /api/emergency/types` - Get list of emergency service types

- `POST /api/emergency/report` - Create new emergency report

- `POST /api/emergency/location` - Send current location

- `GET /api/emergency/status/{reportId}` - Track emergency response status

- `POST /api/emergency/call` - Initiate emergency call

\*\*Database operations:\*\*

- Create record in `EmergencyReport` table

- Create record in `EmergencyContact` table

- Update `UserLocation` table

- Create record in `EmergencyResponse` table

---

### 6. \*\*Nearby Services Flow\*\*

\*\*What happens:\*\* Finding helpful places around you.

\*\*Real-life example:\*\*

- Lisa needs to find a pharmacy at 10 PM

- She taps "Nearby Help"

- She sees categories: Hospitals, Pharmacies, Police Stations, Fire Stations

- She taps "Pharmacies"

- The app shows a map with all nearby pharmacies

- It shows which ones are open now

- She can tap on one to get directions or call them

\*\*Technical Flow:\*\*

```plaintext

Tap Nearby Help → Get user location → Show service categories → Select category → Fetch nearby services → Show on map → Allow actions (call, directions)

```

\*\*Endpoints needed:\*\*

- `GET /api/services/categories` - Get all service categories

- `GET /api/services/nearby/{category}` - Get nearby services by category

- `GET /api/services/{serviceId}/details` - Get detailed info about a service

- `POST /api/services/directions` - Get directions to a service

- `GET /api/services/operating-hours` - Check if service is currently open

\*\*Database operations:\*\*

- Query `ServiceProvider` table with location filters

- Query `ServiceCategory` table

- Query `ServiceHours` table for operating times

- Create record in `UserServiceInteraction` table

---

### 7. \*\*Report Incident Flow\*\*

\*\*What happens:\*\* Reporting problems in the community.

\*\*Real-life example:\*\*

- Carlos sees a broken streetlight on his street

- He taps "Reports" then "New Report"

- He selects "Infrastructure" then "Street Lighting"

- He takes a photo of the broken light

- He adds a description: "Streetlight on Main St is not working"

- The app gets his location automatically

- He submits the report

- The city receives the report and assigns it to the maintenance team

\*\*Technical Flow:\*\*

```plaintext

Tap New Report → Select category → Select subcategory → Take photo → Add description → Get location → Submit report → Generate report ID → Send to relevant department → Show confirmation

```

\*\*Endpoints needed:\*\*

- `GET /api/reports/categories` - Get report categories

- `GET /api/reports/subcategories/{categoryId}` - Get subcategories

- `POST /api/reports/upload-image` - Upload report photos

- `POST /api/reports/create` - Create new report

- `GET /api/reports/my-reports` - Get user's submitted reports

- `GET /api/reports/{reportId}/status` - Track report status

\*\*Database operations:\*\*

- Create record in `Report` table

- Create record in `ReportMedia` table (for photos)

- Create record in `ReportLocation` table

- Create record in `ReportAssignment` table

---

### 8. \*\*Digital Address System Flow\*\*

\*\*What happens:\*\* Getting a digital address for your location.

\*\*Real-life example:\*\*

- Fatima lives in a new area without street addresses

- She taps "Digital Services" then "Digital Address"

- The app uses GPS to find her exact location

- It generates a unique code like "GH-123-4567"

- This code can be used for deliveries, emergency services, etc.

- She can share this code with others to find her location

\*\*Technical Flow:\*\*

```plaintext

Request digital address → Get precise GPS coordinates → Generate unique address code → Validate location → Save to database → Display QR code → Allow sharing

```

\*\*Endpoints needed:\*\*

- `POST /api/digital-address/generate` - Generate new digital address

- `GET /api/digital-address/validate/{code}` - Validate address code

- `GET /api/digital-address/my-addresses` - Get user's digital addresses

- `POST /api/digital-address/share` - Share address with others

- `GET /api/digital-address/{code}/location` - Get location from address code

\*\*Database operations:\*\*

- Create record in `DigitalAddress` table

- Create record in `AddressLocation` table

- Update `UserProfile` table with primary address

---

### 9. \*\*Licensing System Flow\*\*

\*\*What happens:\*\* Applying for or renewing government licenses.

\*\*Real-life example:\*\*

- David needs to renew his driver's license

- He taps "Digital Services" then "Licenses"

- He sees his current licenses and their expiry dates

- He taps "Renew" on his driver's license

- He fills out a renewal form

- He uploads required documents (photo, medical certificate)

- He pays the renewal fee through the app

- He gets a temporary digital license while waiting for the physical one

\*\*Technical Flow:\*\*

```plaintext

View licenses → Select license to renew → Fill renewal form → Upload documents → Calculate fees → Process payment → Generate temporary license → Submit to licensing authority → Track application status

```

\*\*Endpoints needed:\*\*

- `GET /api/licenses/my-licenses` - Get user's current licenses

- `GET /api/licenses/types` - Get available license types

- `POST /api/licenses/apply` - Apply for new license

- `POST /api/licenses/renew/{licenseId}` - Renew existing license

- `POST /api/licenses/upload-documents` - Upload required documents

- `POST /api/licenses/payment` - Process license fees

- `GET /api/licenses/{applicationId}/status` - Track application status

\*\*Database operations:\*\*

- Query `License` table for user's licenses

- Create record in `LicenseApplication` table

- Create records in `LicenseDocument` table

- Create record in `Payment` table

- Update `License` table with renewal info

---

### 10. \*\*Notifications Flow\*\*

\*\*What happens:\*\* Receiving and managing notifications.

\*\*Real-life example:\*\*

- The city sends a notification: "Water supply will be interrupted tomorrow 9 AM - 3 PM in your area"

- Elena sees a red dot on the notification bell

- She taps it and sees all her notifications

- She can mark them as read or delete them

- She can also set preferences for what types of notifications she wants

\*\*Technical Flow:\*\*

```plaintext

System generates notification → Send to user's device → Store in database → Show notification badge → User opens notifications → Mark as read → Update database

```

\*\*Endpoints needed:\*\*

- `GET /api/notifications/list` - Get user's notifications

- `POST /api/notifications/mark-read/{notificationId}` - Mark notification as read

- `DELETE /api/notifications/{notificationId}` - Delete notification

- `GET /api/notifications/preferences` - Get notification preferences

- `PUT /api/notifications/preferences` - Update notification preferences

- `POST /api/notifications/send` - Send notification (admin only)

\*\*Database operations:\*\*

- Query `Notification` table

- Update `NotificationStatus` table

- Query `NotificationPreference` table

---

### 11. \*\*Q&A System Flow\*\*

\*\*What happens:\*\* Getting answers to common questions.

\*\*Real-life example:\*\*

- Roberto has a question: "How do I apply for a birth certificate?"

- He taps "Help" then "Q&A"

- He can either browse categories or search

- He types "birth certificate" in the search

- The app shows relevant questions and answers

- If he doesn't find his answer, he can submit a new question

\*\*Technical Flow:\*\*

```plaintext

Open Q&A → Browse categories or search → Display relevant Q&As → If not found, allow new question submission → Admin reviews and answers → User gets notification when answered

```

\*\*Endpoints needed:\*\*

- `GET /api/qa/categories` - Get Q&A categories

- `GET /api/qa/questions/{categoryId}` - Get questions by category

- `GET /api/qa/search` - Search questions and answers

- `POST /api/qa/ask-question` - Submit new question

- `GET /api/qa/my-questions` - Get user's submitted questions

- `POST /api/qa/answer` - Answer question (admin only)

\*\*Database operations:\*\*

- Query `QACategory` table

- Query `Question` and `Answer` tables

- Create record in `Question` table for new questions

- Create record in `Answer` table for responses

---

### 12. \*\*Emergency Contacts Flow\*\*

\*\*What happens:\*\* Managing personal emergency contacts.

\*\*Real-life example:\*\*

- Sofia wants to add her family members as emergency contacts

- She taps "Emergency Contacts"

- She adds her husband: Name "Miguel", Relationship "Spouse", Phone "+1234567890"

- She adds her doctor: Name "Dr. Smith", Relationship "Doctor", Phone "+0987654321"

- In case of emergency, these contacts can be notified automatically

\*\*Technical Flow:\*\*

```plaintext

Open emergency contacts → Add new contact → Fill contact details → Save to database → In emergency, system can auto-notify these contacts

```

\*\*Endpoints needed:\*\*

- `GET /api/emergency-contacts/list` - Get user's emergency contacts

- `POST /api/emergency-contacts/add` - Add new emergency contact

- `PUT /api/emergency-contacts/{contactId}` - Update emergency contact

- `DELETE /api/emergency-contacts/{contactId}` - Delete emergency contact

- `POST /api/emergency-contacts/notify` - Notify emergency contacts (during emergency)

\*\*Database operations:\*\*

- Query `EmergencyContact` table

- Create/Update/Delete records in `EmergencyContact` table

- Create records in `EmergencyNotification` table when contacts are notified

---

### 13. \*\*User Profile Management Flow\*\*

\*\*What happens:\*\* Managing personal information and settings.

\*\*Real-life example:\*\*

- Marcus wants to update his phone number

- He taps "Profile"

- He sees his current information: name, phone, email, address

- He taps "Edit" next to phone number

- He enters his new number

- The app sends a verification code to the new number

- He enters the code and his phone number is updated

\*\*Technical Flow:\*\*

```plaintext

Open profile → View current info → Tap edit → Update information → If phone/email changed, verify → Save changes → Update database

```

\*\*Endpoints needed:\*\*

- `GET /api/user/profile` - Get user profile information

- `PUT /api/user/profile` - Update user profile

- `POST /api/user/change-phone` - Change phone number (requires verification)

- `POST /api/user/change-email` - Change email (requires verification)

- `POST /api/user/change-password` - Change password

- `POST /api/user/upload-avatar` - Upload profile picture

\*\*Database operations:\*\*

- Query `User` and `UserProfile` tables

- Update records in `User` and `UserProfile` tables

- Create records in `VerificationCode` table for phone/email changes

---

### 14. \*\*Tracking/History Flow\*\*

\*\*What happens:\*\* Viewing history of activities and tracking ongoing requests.

\*\*Real-life example:\*\*

- Anna wants to see the status of her license application from last week

- She taps "Tracking"

- She sees all her activities: reports submitted, license applications, emergency calls

- She taps on her license application

- She sees: "Application received → Documents verified → Payment processed → Under review"

- She knows her license should be ready soon

\*\*Technical Flow:\*\*

```plaintext

Open tracking → Fetch user's activities → Display with status → Allow detailed view → Show progress timeline → Update status in real-time

```

\*\*Endpoints needed:\*\*

- `GET /api/tracking/activities` - Get user's all activities

- `GET /api/tracking/reports` - Get user's report statuses

- `GET /api/tracking/licenses` - Get license application statuses

- `GET /api/tracking/emergency` - Get emergency report statuses

- `GET /api/tracking/{activityId}/details` - Get detailed tracking info

\*\*Database operations:\*\*

- Query multiple tables: `Report`, `License`, `EmergencyReport`, etc.

- Query `ActivityLog` table for status updates

- Query `TrackingStatus` table for current statuses

---

### 15. \*\*Forgot Password Flow\*\*

\*\*What happens:\*\* When users forget their password.

\*\*Real-life example:\*\*

- James forgot his password

- On the login screen, he taps "Forgot Password"

- He enters his phone number

- The app sends him a reset code via SMS

- He enters the code

- He creates a new password

- He can now login with the new password

\*\*Technical Flow:\*\*

```plaintext

Tap forgot password → Enter phone number → Send reset code → Enter code → Verify code → Set new password → Password updated → Auto login

```

\*\*Endpoints needed:\*\*

- `POST /api/auth/forgot-password` - Initiate password reset

- `POST /api/auth/verify-reset-code` - Verify the reset code

- `POST /api/auth/reset-password` - Set new password

- `POST /api/auth/login` - Auto login after password reset

\*\*Database operations:\*\*

- Create record in `PasswordResetCode` table

- Update `User` table with new password hash

- Delete used reset code from `PasswordResetCode` table

---

## 🔄 How All These Flows Connect

Think of the app like a big building with many rooms (features). Users can move between rooms freely:

1. \*\*Main Lobby (Dashboard)\*\* - This is where everyone starts and can go to any other room

2. \*\*Emergency Room\*\* - For urgent situations, accessible from anywhere in the app

3. \*\*Service Desk (Nearby Help)\*\* - To find help around you

4. \*\*Complaint Office (Reports)\*\* - To report problems

5. \*\*Government Office (Digital Services)\*\* - For official business

6. \*\*Information Desk (Q&A)\*\* - To get answers

7. \*\*Personal Office (Profile)\*\* - To manage your information

## 📊 Database Relationships

All the data is connected like a family tree:

- \*\*User\*\* is the parent - everything belongs to a user

- \*\*Reports\*\* are children of User - each report belongs to one user

- \*\*Emergency Contacts\*\* are children of User - each contact belongs to one user

- \*\*Licenses\*\* are children of User - each license belongs to one user

- \*\*Notifications\*\* can be sent to many users (like a teacher sending a message to all students)

## 🔐 Security Flow

Every request (like asking for information) goes through security checks:

1. \*\*Authentication\*\* - "Are you who you say you are?" (like showing ID)

2. \*\*Authorization\*\* - "Are you allowed to do this?" (like checking if you have permission)

3. \*\*Data Validation\*\* - "Is the information you're sending correct?" (like checking if an email address looks right)

This ensures that only the right people can access the right information, just like how only you can open your own locker at school with your combination.

---

This emergency citizen app is like having a helpful government assistant in your pocket, ready to help 24/7 with anything from emergencies to everyday government services!