Eswatini Events Ticketing System Documentation

Overview

Eswatini Events is a digital ticketing platform for event management in Eswatini, designed for festivals like MTN Bushfire (90% internet connectivity, 70% rural population, significant tourist attendance). Built by a solo non-coder on Windows/Android, it supports ticket purchases, refunds, and event management via a mobile app (buyers), organizer web portal, and super-admin web portal. The system prioritizes simplicity, 2G compatibility (<10-second page loads, <10KB pages), and inclusivity for locals (MoMo/M-Pesa) and tourists (international phones, email, Visa/MasterCard). Development spans Month 4 (Weeks 9-12, May-June 2025, mock data, 160 hours) and Month 5 (Weeks 13-14, August 10, 2025, MongoDB, 80 hours).

Objectives

- **User-Friendly**: English-only UI, simple for non-technical users, 2G-ready (<10-second load).
- **Inclusive**: Supports locals (Swazi numbers, +268, MoMo/M-Pesa) and tourists (international numbers, +44/+27, email, Visa/MasterCard).
- **Secure**: Role-based access control (RBAC), biometrics, encryption, audit logs (Month 4), OAuth 2.0 (Month 5).
- **Scalable**: Mock data (Month 4) transitions to MongoDB (Month 5), supports 100+ simultaneous users, 5-ticket scenario (e.g., 5 Bushfire tickets, SZL 290).

Scope

- **Mobile App**: Buyers purchase, view (offline QR codes), and manage tickets (Week 10, TicketsScreen.js).
- **Organizer Portal**: Event management, ticket search, refunds, analytics (Week 12, App.js).
- **Super-Admin Portal**: Full system control (events, tickets, users, refunds, analytics, Week 12, SuperAdmin.js).

- **Payments**: Visa, MasterCard, MoMo, M-Pesa, Cash (offline, Week 9).
- **Communication**: WhatsApp (primary, manual Month 4, API Month 5), SMS (secondary, Twilio, Month 5), in-app chat (Month 5, ChatScreen.js).

System Requirements

Functional Requirements

1. **Ticket Purchase (Week 9)**:
    a. Buyers purchase via app (one-click checkout: Visa, MasterCard, MoMo, M-Pesa, Cash offline).
    b. Example: 5 Bushfire tickets (SZL 290, SZL 58/ticket).
    c. Store in Payment model (phone, email, transaction ID, ticket IDs).
    d. Notify via WhatsApp/SMS/email (e.g., "TXN-789012, TKT-123 to TKT-127, SZL 290").
    e. Generate QR code per ticket (offline viewable, TicketsScreen.js).
    f. Error Handling: Handle failed payments (e.g., declined card → retry prompt, ~3 hours).
2. **Ticket Search (Week 12)**:
    a. Organizers/super-admin search by phone (+268, +44, +27), transaction ID (TXN-789012), email (john@email.com), ticket ID (TKT-123), or ticket type (Standard, VIP).
    b. Results: Ticket ID, type, price (SZL 58), status (valid/used/refunded).
    c. API: /api/tickets/search (ticketController.js, RBAC restricts organizers to their events).
    d. Response Format:

```
{
  "tickets": [
    { "id": "TKT-123", "eventId": "EVT-1", "type": "standard",
"price": 58, "status": "valid" }
  ]
}
```

    e. Error Handling: Invalid input (e.g., wrong phone format → "Invalid phone number"), no results (→ "No tickets found").
3. **Refunds (Week 12)**:

    a. Auto-amount refunds (SZL 58/ticket) from escrow to Visa/MasterCard/MoMo/M-Pesa (Cash offline, manual booth verification).
    b. Organizers refund their events' tickets; super-admin overrides all.
    c. Notify via WhatsApp (primary) or email/SMS (e.g., "Refunded SZL 116, TKT-123, TKT-124").
    d. Update sales (e.g., SZL 290 → SZL 174 after refunding 2 tickets).
    e. Error Handling: Insufficient escrow funds (→ "Contact super-admin"), duplicate refund (→ "Ticket already refunded").
4. **Account Creation (Week 13)**:
    a. Social Login: Google, Facebook, Apple (OAuth 2.0, Firebase, email auto-filled).
    b. Traditional: Email (mandatory), password (8+ characters), phone (optional, +268/+44/+27, tied to MoMo/M-Pesa), name (optional).
    c. Roles: Buyer (app), organizer (super-admin created), super-admin (admin@eswatini-events.com).
    d. Error Handling: Duplicate email (→ "Email already exists"), weak password (→ "Password must be 8+ characters").
5. **Login (Week 13)**:
    a. Social: Google, Facebook, Apple buttons (OAuth, redirect to app/portals).
    b. Traditional: Email/password, biometrics (fingerprint, Week 10, react-native-biometrics).
    c. Redirects: Buyers (app, My Tickets), organizers (localhost:3000), super-admin (/super-admin).
    d. Error Handling: Failed login (→ "Invalid credentials"), session timeout (JWT, ~5 hours).
6. **Communication**:
    a. WhatsApp: Primary for notifications (purchases, refunds) and refund requests (manual Month 4, Business API Month 5).
    b. SMS: Secondary for critical notifications (Twilio, Month 5, e.g., "TKT-123 purchased").
    c. In-App Chat: Buyer-to-organizer messaging (Month 5, ChatScreen.js, Firebase Realtime Database), email-based for no-phone users, offline queuing (AsyncStorage, Week 2).
    d. Example: Tourist chats "Refund TKT-123, john@email.com"; local uses WhatsApp (+26812345678, "Refund TKT-123").
    e. Error Handling: Failed WhatsApp delivery (→ fallback to email), offline chat sync failure (→ "Retry when online").
7. **Super-Admin Rights (Week 12)**:

    a. Full CRUD: Events, tickets, users, refunds, analytics (SuperAdmin.js).
    b. Override organizer actions (e.g., cancel refunds).
    c. Create/delete organizer accounts.
    d. Error Handling: Unauthorized access (→ "Super-admin only").
8. **Manual Mark-as-Used (Week 8)**:
    a. Organizers mark tickets as used (scanned QR or manual input, App.js).
    b. Updates status (valid → used), prevents reuse.
    c. Error Handling: Already used (→ "Ticket already scanned").
9. **Analytics (Week 12)**:
    a. Organizers: Sales, check-ins for their events (App.js).
    b. Super-Admin: System-wide analytics (SuperAdmin.js, Chart.js for real-time dashboards).
    c. Example: Sales trend (SZL 290/day), user demographics (locals vs. tourists).
    d. Error Handling: No data (→ "No analytics available").
10. **Promo Codes (Week 13)**:
    a. Buyers apply codes at checkout (e.g., PROMO-001, 10% off).
    b. Model: { id: "PROMO-001", discount: 10, eventId: "EVT-1" }.
    c. Error Handling: Invalid code (→ "Promo code invalid").

Non-Functional Requirements

- **Performance**: 2G-compatible (<10-second load, <10KB pages, minified JS/CSS via Webpack), offline ticket viewing (QR codes, AsyncStorage, Week 2).
- **Scalability**: Mock data (Month 4) to MongoDB (Month 5), supports 100+ simultaneous users (rate-limited APIs, Redis caching, Month 5).
- **Security**:
    - RBAC: Super-admin (full access), organizer (event-specific), buyer (tickets only).
    - Biometrics: Fingerprint login (app, Week 10).
    - Encryption: Email/phone in MongoDB (Month 5).
    - Audit Logs: Track actions (Month 4, Winston, file-based).
    - OAuth 2.0: Social login (Month 5, Firebase).
    - CAPTCHA: Prevent bot abuse (Month 5, ~3 hours).
- **Accessibility**: Large fonts, clear UI, English-only.
- **Usability**: Simple for non-technical users, tourist-friendly (email, social login, QR codes).

Constraints

- **Connectivity**: 30% internet, prioritize 2G, offline caching (AsyncStorage).
- **Cash Payments**: Manual refunds, no transaction/ticket IDs, offline booth verification.
- **Tourists**: No Swazi phones, use email/international numbers (+44, +27).
- **Budget**: Minimize SMS costs (use WhatsApp, Amazon SES for email), WhatsApp Business API (free, Month 5).
- **Effort**: 40 hours/week, Month 4 (160 hours), Month 5 (80 hours).

System Architecture

Tech Stack

- **Frontend**:
  - Mobile App: React Native, Tailwind CSS, react-native-biometrics (Week 10), AsyncStorage (Week 2).
  - Web Portals: React, Tailwind CSS, react-router-dom (Week 12).
  - CDN: cdn.jsdelivr.net for React, dependencies.
- **Backend**: Node.js, Express.js, mock data (Month 4), MongoDB (Month 5).
- **Database**: Mock arrays (Month 4), MongoDB (Month 5, schemas: users, tickets, payments, events, promos).
- **APIs**: RESTful, /api/tickets/*, /api/users/*, /api/payments/*, /api/promos/* (rate-limited, express-rate-limit).
- **Communication**: WhatsApp (Business API, Month 5), SMS (Amazon SES, Month 5), in-app chat (Firebase, Month 5).
- **Security**: OAuth 2.0 (Firebase, Month 5), bcrypt (passwords), JWT (web sessions), Winston (audit logs, Month 4).

Data Models (Mock Month 4, MongoDB Month 5)

1. **User**:

```
{
  "id": "user1",
  "email": "john@email.com",
  "password": "hashedpassword", // Optional for social login
  "phone": "+26812345678", // Optional, Swazi/International
```

```
  "name": "John Smith", // Optional
  "role": "buyer" // or "organizer", "super-admin"
}
```

2. **Ticket**:

```
{
  "id": "TKT-123",
  "eventId": "EVT-1",
  "paymentId": "TXN-789012",
  "ticketType": "standard",
  "price": 58,
  "status": "valid", // or "used", "refunded"
  "qrCode": "base64-encoded-qr"
}
```

3. **Payment**:

```
{
  "id": "TXN-789012",
  "phone": "+26812345678", // Optional
  "email": "john@email.com", // Optional
  "amount": 290,
  "method": "MoMo", // or Visa, MasterCard, M-Pesa, Cash
  "ticketIds": ["TKT-123", "TKT-124"],
  "createdAt": "2025-05-26T07:00:00Z"
}
```

4. **Event**:

```
{
  "id": "EVT-1",
  "name": "Bushfire 2025",
  "startTime": "2025-05-30T18:00:00Z",
  "organizerId": "org1"
}
```

5. **Promo**:

```
{
  "id": "PROMO-001",
```

```
  "discount": 10, // Percentage
  "eventId": "EVT-1",
  "createdAt": "2025-05-01T00:00:00Z"
}
```

System Flow

1. **Buyer**:
    a. Signs up/logs in (app, social: Google/Facebook/Apple, or email/password, biometrics).
    b. Purchases tickets (one-click, Visa/MoMo), receives QR code (TicketsScreen.js), WhatsApp/SMS/email (TKT-123, TXN-789012).
    c. Views tickets offline (QR codes, AsyncStorage).
    d. Requests refund via WhatsApp (+26812345678, "Refund TKT-123") or in-app chat (Month 5, "Refund TKT-123, john@email.com").
2. **Organizer**:
    a. Logs in (web, localhost:3000, social/email).
    b. Searches tickets (phone, transaction ID, email, ticket ID), refunds from escrow, views analytics (sales, check-ins), marks tickets used (Week 8).
3. **Super-Admin**:
    a. Logs in (web, /super-admin, social/email).
    b. Full control: searches/refunds all tickets, creates events, manages users, overrides refunds.
4. **Communication**:
    a. WhatsApp: Purchase/refund notifications, refund requests (manual Month 4, API Month 5).
    b. SMS: Critical notifications (Amazon SES, Month 5).
    c. In-App Chat: Email-based, offline queuing (Month 5).

UI/UX Design

Mobile App

- **LoginScreen.js**:
    - Buttons: "Log In with Google" (icon), "Log In with Facebook" (icon), "Sign In with Apple" (iOS logo).
    - Fields: Email, password, "Log In", "Sign Up", "Forgot Password" (Month 5).
    - Toggle: "Use Fingerprint" (biometrics, Week 10).

    - Design: Tailwind CSS, umhlanga-blue buttons, white background, large fonts, 2G-ready (<5-second load).
- **TicketsScreen.js**:
    - List: Ticket ID (TKT-123), transaction ID (TXN-789012), event (Bushfire), type (Standard), price (SZL 58), status (valid), QR code.
    - Popup: "Find Ticket ID/Transaction ID in app, WhatsApp, or email for refunds."
    - Offline: Cached in AsyncStorage (Week 2).
- **CheckoutScreen.js**:
    - Fields: Email (mandatory), phone (optional), payment method (Visa, MasterCard, MoMo, M-Pesa), promo code (Month 5).
    - Button: bushfire-red "Buy Now".
- **ChatScreen.js (Month 5)**:
    - WhatsApp-like interface, buyer-to-organizer messaging, email-based, offline queuing.

Web Portals (App.js, SuperAdmin.js)

- **Login**: Social buttons (Google, Facebook, Apple), email/password, bushfire-red "Log In".
- **Search/Refund**: Inputs for phone, transaction ID, email, ticket ID, ticket type filter, umhlanga-blue "Search".
- **Analytics**: Real-time dashboards (Chart.js, sales trends, demographics).
- **Design**: Tailwind CSS, 2G-ready (<10-second load).

Development Roadmap

Month 4 (Weeks 9-12, May-June 2025, 160 hours, Completed)

- **Week 9 (40 hours)**:
    - One-click checkout, payments (Visa, MoMo, M-Pesa, Cash).
    - QR code generation (TicketsScreen.js, ~5 hours).
    - Mock data setup (users, tickets, payments, events).
- **Week 10 (40 hours)**:
    - Biometrics (react-native-biometrics).
    - Offline ticket viewing (AsyncStorage, QR codes).
- **Week 12 (80 hours)**:

    - Search: /api/tickets/search, UI in App.js, SuperAdmin.js (~10 hours).
    - Refunds: Auto-amount, escrow, notifications (~15 hours).
    - Super-admin portal: Full CRUD, override refunds (~15 hours).
    - Audit logs: File-based (Winston, ~5 hours).
    - MongoDB local testing (Docker, ~10 hours).
    - Unit tests: Backend (Jest, ~10 hours), UI (React Native Testing Library, ~10 hours).

Month 5 (Weeks 13-14, August 10, 2025, 80 hours)

- **Week 13 (50 hours)**:
    - MongoDB: Replace mock data, schemas (users, tickets, payments, events, promos, ~25 hours).
    - Social Login: Google, Facebook, Apple (Firebase Auth, ~10 hours).
    - WhatsApp: Business API, automate notifications/refunds (~10 hours).
    - Promo codes: Checkout integration, Promo model (~5 hours).
- **Week 14 (30 hours)**:
    - In-App Chat: ChatScreen.js, Firebase, offline queuing (~10 hours).
    - Real-time analytics: Chart.js dashboards (~5 hours).
    - Auto mark-as-used: Scanner sync (~10 hours).
    - CAPTCHA: Social login protection (~3 hours).
    - Forgot password: Email link (Amazon SES, ~2 hours).

Future (Month 6, ~30 hours)

- Multi-language support: French, Portuguese for tourists (~10 hours).
- AI-based fraud detection: Refund anomalies (~15 hours).
- Feedback form: In-app/portal user feedback (~5 hours).

5-Ticket Scenario

- **Purchase**: UK tourist (john@email.com, +447123456789) or local (+26812345678) signs up via Google or email/password, buys 5 Bushfire tickets (SZL 290, Visa/MoMo, applies PROMO-001 for 10% off, SZL 261). App shows TKT-123 to TKT-127, TXN-789012, QR codes (offline); WhatsApp/email confirms.

- **Refund Request**: Tourist chats "Refund TKT-123, john@email.com"; local uses WhatsApp (+26812345678, "Refund TKT-123"). Organizer searches phone/email/ticket ID in App.js, refunds 2 tickets (SZL 116).
- **Result**: Escrow updates (SZL 145), buyer sees refunded tickets (app, QR disabled), notified via WhatsApp (locals) or email (tourists). Audit log records refund (Month 4).

Setup Instructions

1. **Clone Repository**:

```
git clone https://github.com/eswatini-events.git
cd eswatini-events
```

2. **Backend**:

```
cd backend
npm install
npm start
```

3. **Mobile App**:

```
cd mobile
npm install
npx react-native run-android
```

4. **Web Portals**:

```
cd web
npm install
npm start
```

5. **Dependencies**:
    a. Node.js, React Native, Express, Tailwind CSS, react-native-biometrics, AsyncStorage, axios, react-router-dom.
    b. Month 5: MongoDB, Firebase Auth, WhatsApp Business API, Amazon SES, Winston, Chart.js, express-rate-limit.
6. **Test**:
    a. Postman: /api/tickets/search, /api/users/login.
    b. Chrome: localhost:3000, /super-admin, 2G throttle.

c. Android Emulator: App, 2G, offline tickets.
d. Load Testing: Artillery, 100+ users (Month 4, ~5 hours).

Maintenance

- **Commits**: Daily, e.g., git commit -m "Week 12: Search feature".
- **Issues**: Report bugs (e.g., "Search fails for email") to GitHub Issues.
- **Updates**: Month 5 (MongoDB, chat, social login), versioned docs (e.g., v1.0-month4).

Notes

- **Eswatini Context**: 30% internet, 70% rural, prioritize WhatsApp (free, 2G), offline QR codes.
- **Tourists**: Email-based social login, international phones, no-phone chat.
- **Security**: RBAC, biometrics, encryption, audit logs (Month 4).
- **Cash**: Manual refunds via booth, QR code verification.
- **Contact**: admin@eswatini-events.com or in-app feedback form (Month 5).

This documentation provides a clear, actionable blueprint for building the Eswatini Events system, optimized for Eswatini's constraints and scalable for future growth.