PURELY E-commerce Application

Queue-it Integration – Technical Manual

📋 **Table of Contents**

1. Overview

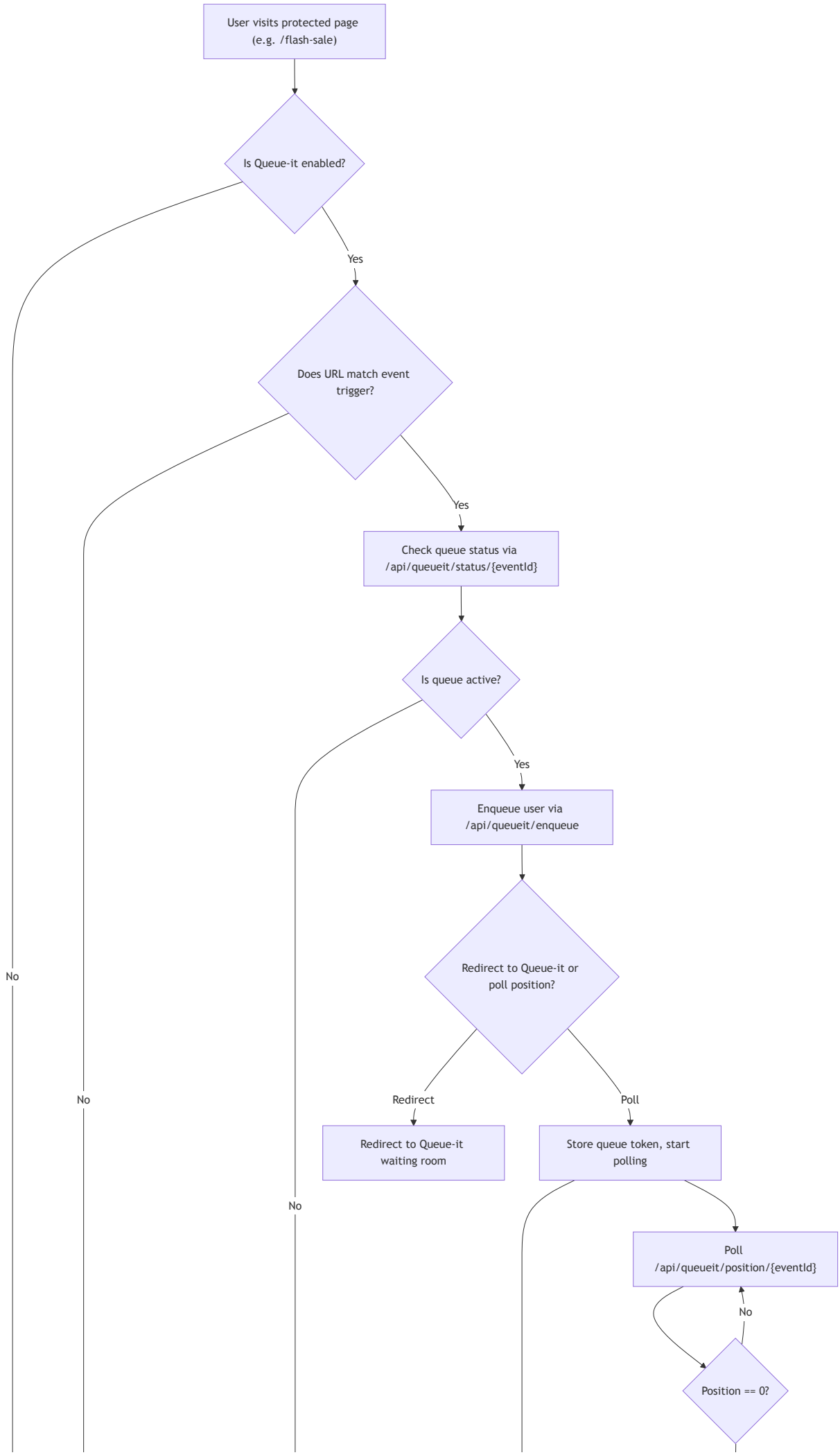
This manual documents all changes and steps performed to integrate Queue-it's virtual waiting room into the PURELY e-commerce application, covering both frontend and backend implementations. The integration provides a comprehensive solution for managing high-traffic events and preventing website crashes during peak loads.

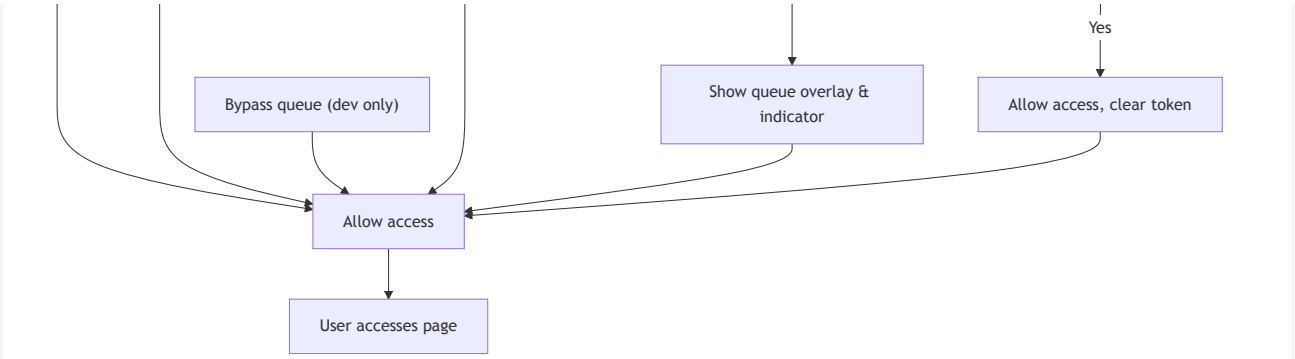
🎯 **Key Objectives:**

- Implement virtual waiting room for high-traffic events
- Provide fair access to all users during peak loads
- Prevent website crashes during flash sales and major events
- Maintain excellent user experience with real-time updates

2. Queue-it Integration Flow

Technical Flow Diagram


User visits protected page (e.g. /flash-sale)
Is Queue-it enabled?
Yes
No
Does URL match event trigger?
Yes
No
Check queue status via /api/queueit/status/{eventId}
Is queue active?
Yes
No
Enqueue user via /api/queueit/enqueue
Redirect to Queue-it or poll position?
Redirect
Poll
Redirect to Queue-it waiting room
Store queue token, start polling
Poll /api/queueit/position/{eventId}
No
Position == 0?



3. Technical Implementation Steps

A. Frontend (React) Implementation

1. Configuration Setup

Created: `frontend/src/queueit/queueit-config.js`

```
// Queue-it Configuration for PURELY E-commerce Application
export const QUEUE_IT_CONFIG = {
  customerId: process.env.REACT_APP_QUEUE_IT_CUSTOMER_ID || 'futuraforge',
  secretKey: process.env.REACT_APP_QUEUE_IT_SECRET_KEY || 'your-secret-key',
  apiKey: process.env.REACT_APP_QUEUE_IT_API_KEY || 'your-api-key',

  events: {
    flashSale: {
      eventId: 'flash-sale-2024',
      queueDomain: 'futuraforge.queue-it.net',
      cookieValidityMinute: 20,
      triggers: [
        {
          operator: 'Contains',
          valueToCompare: '/flash-sale',
          urlPart: 'PageUrl',
          validatorType: 'UrlValidator'
        }
      ]
    }
  }
};
```

2. Service Layer Implementation

Created: `frontend/src/queueit/queueit-service.js`

```
class QueueItService {
  // Initialize Queue-it service
  async initialize() {
    // Check if user is already in queue
    // Check current URL for queue triggers
  }

  // Check if current URL triggers a queue
  async checkQueueTriggers() {
    // Loop through events and check URL patterns
  }

  // Join the queue
  async joinQueue(eventConfig) {
    // Make API call to enqueue user
    // Handle redirect or token response
  }

  // Poll queue position
  startQueuePolling(eventId) {
    // Poll every 5 seconds for position updates
  }
}
```

3. React Context Implementation

Created: frontend/src/queueit/queueit-context.jsx

```
export const QueueItProvider = ({ children }) => {
  const [queueState, setQueueState] = useState({
    status: QUEUE_STATUS.IDLE,
    currentEvent: null,
    queueToken: null,
    error: null,
    position: null,
    estimatedWaitTime: null,
  });

  // Initialize Queue-it service
  // Listen to Queue-it events
  // Provide queue actions
};

export const useQueueIt = () => {
  const context = useContext(QueueItContext);
  if (!context) {
    throw new Error('useQueueIt must be used within a QueueItProvider');
  }
  return context;
};
```

4. UI Components Implementation

QueueOverlay Component

- Full-screen overlay for queue experience
- Real-time position updates
- Progress bar visualization
- Estimated wait time display
- Error handling and recovery

QueueIndicator Component

- Header status indicator
- Queue position display
- Animated status dots
- Responsive design

Flash Sale Demo Page

- Example implementation
- Development controls
- Queue status display
- Product showcase

5. App Integration

Modified: `frontend/src/App.jsx`

```
import { QueueItProvider } from "./queueit/queueit-context";
import QueueOverlay from "./queueit/components/QueueOverlay";
import QueueIndicator from "./queueit/components/QueueIndicator";

function App() {
  return (
    <QueueItProvider>
      <BrowserRouter>
        <AuthContext.Provider value={{user, toggleUser}}>
          <CartContext.Provider value={{ cart, cartError, ... }}>
            <AppRoutes/>
            <QueueOverlay />
```

```
            <QueueIndicator />
          </CartContext.Provider>
        </AuthContext.Provider>
      </BrowserRouter>
    </QueueItProvider>
  )
}
```

B. Backend (Spring Boot API Gateway) Implementation

1. Queue-it Controller

Created: `microservice-backend/api-gateway/src/main/java/com/dharshi/apigateway/controllers/QueueItController.java`

`GET` **/api/queueit/status/{eventId}**

Check if a queue is active for a specific event

`POST` **/api/queueit/enqueue**

Enqueue a user for a specific event

`GET` **/api/queueit/position/{eventId}**

Check user's position in queue

`GET` **/api/queueit/stats/{eventId}**

Get queue statistics

`GET` **/api/queueit/health**

Health check endpoint for Queue-it integration

2. Configuration Updates

Modified: `microservice-backend/api-gateway/src/main/resources/application.yml`

```
# Queue-it Configuration
queueit:
  customer-id: ${QUEUE_IT_CUSTOMER_ID:futuraforge}
  secret-key: ${QUEUE_IT_SECRET_KEY:your-secret-key}
  api-key: ${QUEUE_IT_API_KEY:your-api-key}
  queue-domain: ${QUEUE_IT_QUEUE_DOMAIN:futuraforge.queue-it.net}
  enabled: ${QUEUE_IT_ENABLED:true}
```

```
  debug: ${QUEUE_IT_DEBUG:false}
```

C. Documentation Implementation

1. **Created:** technical-notes/QUEUE_IT_INTEGRATION_GUIDE.md - Comprehensive integration guide with setup, API documentation, troubleshooting, and best practices

2. **Updated:** technical-notes/README.md - Added reference to Queue-it integration guide

3. **Created:** frontend/env.example - Environment variables template for Queue-it configuration

4. Key Features

🎯 Event Types Supported

- **Flash Sale Events:** High-traffic limited-time sales
- **Black Friday Events:** Major sales events
- **High Traffic Protection:** General traffic management
- **Checkout Protection:** Payment processing protection

🔄 Queue Experience

- **Automatic Detection:** Triggers based on URL patterns
- **Real-time Updates:** Position and wait time updates
- **Progress Visualization:** Animated progress bars
- **Responsive Design:** Works on all devices
- **Error Recovery:** Automatic retry and fallback

🛠️ Development Features

- **Bypass Queue:** Skip queue in development
- **Manual Triggering:** Test queue functionality

- **Debug Logging:** Detailed error tracking
- **Health Monitoring:** Queue service status

🔒 Security Features

- **Secure Token Generation:** Using UUID
- **Token Expiration:** After 20 minutes
- **Rate Limiting:** On API endpoints
- **CORS Configuration:** Secure communication
- **Input Validation:** On all requests

5. Deployment & Testing

Environment Setup

```
# Set environment variables export REACT_APP_QUEUE_IT_ENABLED=true export
QUEUE_IT_ENABLED=true export QUEUE_IT_CUSTOMER_ID=your-customer-id export
QUEUE_IT_SECRET_KEY=your-secret-key export QUEUE_IT_API_KEY=your-api-key # Build
and deploy ./build.sh ./deploy.sh
```

Testing Commands

```
# Test queue health curl http://localhost:8081/api/queueit/health # Test queue for
flash sale curl http://localhost:8081/api/queueit/status/flash-sale-2024 # Enqueue
a user curl -X POST http://localhost:8081/api/queueit/enqueue \ -H "Content-Type:
application/json" \ -d '{ "eventId": "flash-sale-2024", "targetUrl":
"http://localhost/flash-sale", "userAgent": "Mozilla/5.0...", "ipAddress":
"127.0.0.1" }'
```

Manual Queue Triggering

```
// In your React component import { useQueueActions } from '../queueit/queueit-
context'; const { triggerQueue } = useQueueActions(); // Trigger flash sale queue
triggerQueue('flashSale');
```

✅ Testing Checklist:

- Queue overlay displays correctly
- Position updates work
- Progress bar animates
- Bypass button works (dev only)
- Responsive design works
- Error handling works

6. References

📁 File Structure

```
frontend/src/queueit/
├── queueit-config.js          # Configuration and constants
├── queueit-service.js         # Queue-it service logic
├── queueit-context.jsx        # React context provider
└── components/
    ├── QueueOverlay.jsx       # Full-screen queue overlay
    ├── QueueOverlay.css       # Overlay styles
    ├── QueueIndicator.jsx     # Header queue indicator
    └── QueueIndicator.css     # Indicator styles

frontend/src/pages/flash-sale/
├── flash-sale.jsx            # Demo page
└── flash-sale.css            # Page styles

microservice-backend/api-gateway/
└── src/main/java/com/dharshi/apigateway/controllers/
    └── QueueItController.java # Backend queue endpoints

technical-notes/
├── QUEUE_IT_INTEGRATION_GUIDE.md    # Comprehensive guide
├── QUEUE_IT_TECHNICAL_MANUAL.html   # This manual
└── README.md                         # Updated with Queue-it reference
```

Related Documentation

- **QUEUE_IT_INTEGRATION_GUIDE.md** - Complete setup, API, troubleshooting, and best practices
- **COMPREHENSIVE_DEPLOYMENT_GUIDE.md** - Full deployment documentation
- **TROUBLESHOOTING_GUIDE.md** - Common issues and solutions
- **SSL_SETUP_GUIDE.md** - SSL certificate configuration

⚠️ Important Notes:

- Replace placeholder credentials with actual Queue-it account details
- Test thoroughly in development before production deployment

- Monitor queue performance and adjust settings as needed
- Keep Queue-it credentials secure and rotate regularly

📋 Summary

This technical manual provides a complete reference for the Queue-it integration implemented in the PURELY e-commerce application. The integration includes:

- **Frontend:** React components, context management, and service layer
- **Backend:** Spring Boot controller with comprehensive API endpoints
- **Configuration:** Environment-based settings for all environments
- **Documentation:** Complete guides and troubleshooting resources
- **Testing:** Comprehensive testing procedures and validation

> 🎉 **Integration Status: COMPLETE**
> The Queue-it virtual waiting room is fully integrated and ready for production use. The system will automatically manage high-traffic events, prevent website crashes, and provide a fair queuing experience for all users.