Novi - Technical Documentation

Novi
Smart Commerce Suite

Generated on: 28/06/2025

Novi - Technical Documentation

Ø=Ý' System Architecture & Development Guide

Table of Contents

Ø<ß×þ System Overview

Technology Stack

<h4>Backend</h4>

- Runtime: Node.js (v18+)
- Framework: Express.js
- Database: PostgreSQL
- ORM: Knex.js
- Authentication: Session-based with bcrypt
- WhatsApp: whatsapp-web.js

<h4>Frontend</h4>

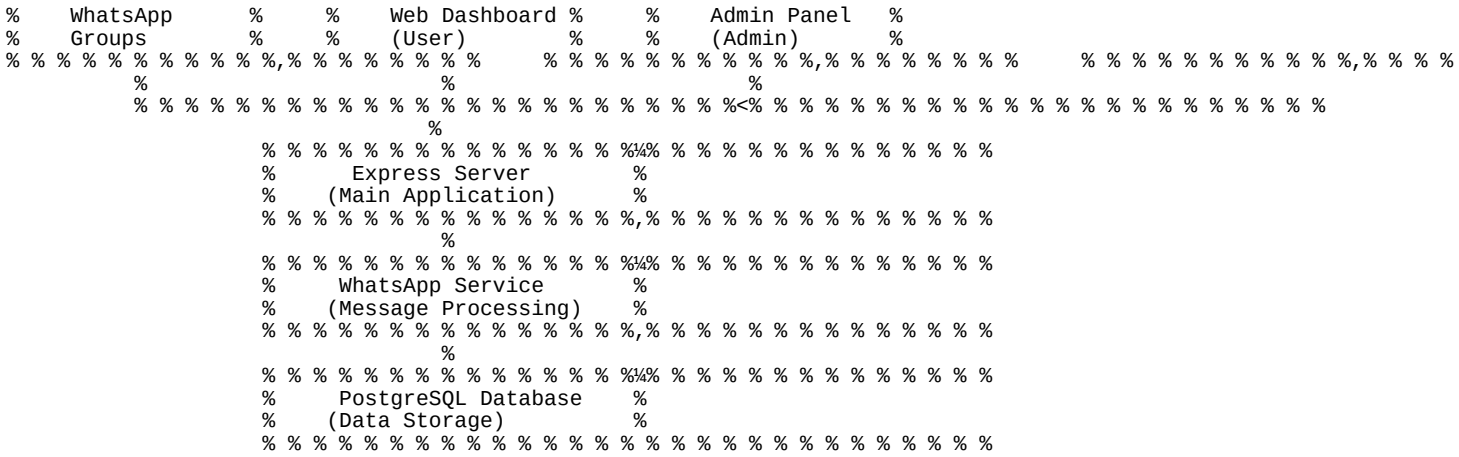
- Template Engine: EJS
- Styling: CSS3 with custom themes

- JavaScript: Vanilla JS with AJAX
- Charts: Chart.js
- UI Components: Custom components

<h4>Infrastructure</h4>

- Hosting: Railway/Heroku
- Process Manager: PM2
- Monitoring: Custom memory monitor
- Logging: Winston logger

System Components



Ø<ßÛþ Architecture

Core Services

<h4>1. WhatsAppService</h4>

- Purpose: Manages WhatsApp connection and message processing

Key Functions:

- WhatsApp Web.js client management
- Message parsing and order extraction
- Automatic responses and notifications
- Group management and monitoring

<h4>2. OrderService</h4>

- Purpose: Handles order lifecycle management

Key Functions:

- Order creation and validation
- Status updates and tracking
- Order history and analytics
- Export functionality

<h4>3. AdminService</h4>

- Purpose: Administrative functions and user management

Key Functions:

• User management (CRUD operations)

• Business management

• System analytics and reporting

• Admin dashboard data

<h4>4. MessageService</h4>

• Purpose: Message processing and order parsing

Key Functions:

• Natural language processing

• Order data extraction

• Message validation

• Response generation

<h4>5. CacheService</h4>

• Purpose: Performance optimization and data caching

Key Functions:

• Dashboard data caching

• Analytics caching

• Cache invalidation

• Memory management

Data Flow

<h4>Order Processing Flow</h4>

```
2. WhatsAppService receives message
3. MessageService parses and extracts order data
4. OrderService validates and creates order
5. Database stores order information
6. Dashboard updates in real-time
7. Automatic confirmation sent to customer
```

<h4>Dashboard Data Flow</h4>

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

Ø=ÝÄþ Database Schema

Core Tables

<h4>users</h4>

id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

username VARCHAR(255) UNIQUE NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL,

password_hash VARCHAR(255) NOT NULL,

phone VARCHAR(20),

```
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

<h4>businesses</h4>

```
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    address TEXT,
    phone VARCHAR(20),
    email VARCHAR(255),
    short_code VARCHAR(10) UNIQUE,
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

<h4>groups</h4>

```
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    business_id UUID REFERENCES businesses(id),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    group_type VARCHAR(50) NOT NULL, -- &#39;sales&#39;, &#39;delivery&#39;, &#39;admin&#39;
    whatsapp_id VARCHAR(255),
    is_active BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

<h4>orders</h4>

id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

business_id UUID REFERENCES businesses(id),

group_id UUID REFERENCES groups(id),

customer_name VARCHAR(255) NOT NULL,

customer_phone VARCHAR(20),

customer_address TEXT,

items TEXT NOT NULL,

total_amount DECIMAL(10,2),

status VARCHAR(50) DEFAULT 'pending',

delivery_notes TEXT,

response_times JSONB,

created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

updated_by UUID REFERENCES users(id)

);

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

<h4>admins</h4>

id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

username VARCHAR(255) UNIQUE NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL,

password_hash VARCHAR(255) NOT NULL,

role VARCHAR(50) DEFAULT 'admin',

is_active BOOLEAN DEFAULT true,

created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
```

```
6. Real-time updates via AJAX
```

<h4>bot_metrics</h4>

id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

business_id UUID REFERENCES businesses(id),

date DATE NOT NULL,

total_messages INTEGER DEFAULT 0,

orders_processed INTEGER DEFAULT 0,

response_time_avg DECIMAL(10,2),

uptime_percentage DECIMAL(5,2),

created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

Indexes

CREATE INDEX idx_orders_business_id ON orders(business_id);

CREATE INDEX idx_orders_status ON orders(status);

CREATE INDEX idx_orders_created_at ON orders(created_at);

CREATE INDEX idx_groups_business_id ON groups(business_id);

CREATE INDEX idx_businesses_user_id ON businesses(user_id);

CREATE INDEX idx_bot_metrics_business_date ON bot_metrics(business_id, date);


```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

Ø=Ý API Documentation

Authentication Endpoints

<h4>POST /auth/login</h4>

- Purpose: User login
- Body: <code>{ username, password }</code>
- Response: <code>{ success, user, session }</code>

<h4>POST /auth/logout</h4>

- Purpose: User logout
- Response: <code>{ success }</code>

<h4>POST /auth/register</h4>

- Purpose: User registration
- Body: <code>{ username, email, password, phone }</code>
- Response: <code>{ success, user }</code>

Dashboard Endpoints

<h4>GET /dashboard</h4>

- Purpose: Get dashboard data
- Query: <code>userId</code> (string)
- Response: <code>{ kpis, recentOrders, charts }</code>

<h4>GET /api/dashboard/analytics</h4>

- Purpose: Get analytics data
- Query: <code>userId</code> (string)
- Response: <code>{ orderTrends, revenueChart, statusDistribution }</code>

Order Management

<h4>GET /orders</h4>

- Purpose: Get orders list
- Query: <code>page, limit, status, businessId</code>
- Response: <code>{ orders, pagination }</code>

<h4>GET /orders/:id</h4>

- Purpose: Get order details
- Response: <code>{ order }</code>

<h4>PUT /orders/:id</h4>

- Purpose: Update order
- Body: <code>{ status, items, customerInfo }</code>
- Response: <code>{ success, order }</code>

<h4>POST /orders/export</h4>

- Purpose: Export orders
- Body: <code>{ format, dateRange, filters }</code>
- Response: File download

Business Management

<h4>GET /business</h4>

- Purpose: Get business list
- Response: <code>{ businesses }</code>

<h4>POST /business</h4>

- Purpose: Create business
- Body: <code>{ name, description, address, phone, email }</code>
- Response: <code>{ success, business }</code>

<h4>PUT /business/:id</h4>

- Purpose: Update business
- Body: <code>{ name, description, address, phone, email }</code>
- Response: <code>{ success, business }</code>

Group Management

<h4>GET /groups</h4>

- Purpose: Get groups list
- Query: <code>businessId</code>
- Response: <code>{ groups }</code>

<h4>POST /groups</h4>

- Purpose: Create group
- Body: <code>{ name, description, groupType, businessId }</code>
- Response: <code>{ success, group }</code>

Admin Endpoints

<h4>GET /admin/dashboard</h4>

- Purpose: Admin dashboard
- Response: <code>{ systemMetrics, recentActivity, botStatus }</code>

<h4>GET /admin/users</h4>

- Purpose: Get users list
- Response: <code>{ users }</code>

<h4>PUT /admin/users/:id/toggle</h4>

- Purpose: Toggle user active status
- Response: <code>{ success }</code>

<h4>GET /admin/orders</h4>

- Purpose: Get all orders (admin view)
- Response: <code>{ orders }</code>

Error Responses

"error": true,

"message": "Error description",

"code": "ERROR_CODE"

}

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

Ø=Üñ WhatsApp Integration

WhatsApp Web.js Setup

<h4>Client Configuration</h4>

authStrategy: new LocalAuth(),

puppeteer: {

headless: true,

args: [

'--no-sandbox',

'--disable-setuid-sandbox',

'--disable-dev-shm-usage',

'--disable-accelerated-2d-canvas',

'--no-first-run',

'--no-zygote',

'--disable-gpu'

]

}

});

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

<h4>Event Handlers</h4>

client.on('qr', (qr) => { /* Handle QR code */ });

client.on('ready', () => { /* Platform ready */ });

client.on('disconnected', () => { /* Handle disconnect */ });

// Message events

client.on('message', async (message) => {

// Process incoming messages

await processMessage(message);

});

// Group events

client.on('group_join', (notification) => { /* Handle joins */ });

client.on('group_leave', (notification) => { /* Handle leaves */ });

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
```

```
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

Message Processing

<h4>Order Parsing Algorithm</h4>

```
    // 1. Extract customer information
    const customerInfo = extractCustomerInfo(message);
    // 2. Extract order items
    const items = extractOrderItems(message);
    // 3. Extract delivery information
    const deliveryInfo = extractDeliveryInfo(message);
    // 4. Validate order completeness
    const validation = validateOrder(customerInfo, items, deliveryInfo);
    return {
        customerInfo,
        items,
        deliveryInfo,
        validation
    };
}
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

<h4>Response Generation</h4>

```
    const templates = {
        confirmation: `Order confirmed! Order #${order.id}`,
        processing: `Your order is being prepared...`,
        delivered: `Order delivered! Thank you for your business.`
    };
    return templates[status] || &#39;Thank you for your order.&#39;;
}
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

Group Management

<h4>Group Types</h4>

- Sales Groups: Receive customer orders
- Delivery Groups: Coordinate deliveries
- Admin Groups: Management communication

<h4>Platform Permissions</h4>

- Read messages
- Send messages
- Send media
- Manage group settings (admin groups)

Ø=Ý Security

Authentication & Authorization

<h4>Session Management</h4>

app.use(session({

    secret: process.env.SESSION_SECRET,

    resave: false,

    saveUninitialized: false,

    cookie: {

        secure: process.env.NODE_ENV === 'production',

        httpOnly: true,

        maxAge: 24 * 60 * 60 * 1000 // 24 hours

    }

}));

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

<h4>Password Security</h4>

const bcrypt = require('bcrypt');

const saltRounds = 12;

async function hashPassword(password) {

    return await bcrypt.hash(password, saltRounds);

}

async function verifyPassword(password, hash) {

    return await bcrypt.compare(password, hash);

```
}
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

<h4>Route Protection</h4>

```
function requireAuth(req, res, next) {
    if (req.session.userId) {
        next();
    } else {
        res.redirect(&amp;#39;/login&amp;#39;);
    }
}
// Admin authorization
function requireAdmin(req, res, next) {
    if (req.session.userRole === &amp;#39;admin&amp;#39;) {
        next();
    } else {
        res.status(403).json({ error: &amp;#39;Admin access required&amp;#39; });
    }
}
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

Data Protection

<h4>Input Validation</h4>

```
const orderSchema = Joi.object({
    customerName: Joi.string().required().max(255),
    customerPhone: Joi.string().pattern(/^\+?[\d\s-]+$/),
    items: Joi.string().required(),
    totalAmount: Joi.number().positive().optional()
});
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

<h4>SQL Injection Prevention</h4>

- Use parameterized queries with Knex.js
- Input sanitization
- Prepared statements

<h4>XSS Prevention</h4>

- Output encoding
- Content Security Policy
- Input sanitization

WhatsApp Security

<h4>Message Encryption</h4>

- End-to-end encryption (WhatsApp's responsibility)
- Secure message storage
- Access control

<h4>Session Security</h4>

- LocalAuth strategy
- Secure session storage
- Regular session cleanup

&¡ Performance & Optimization

Memory Management

<h4>Memory Monitor</h4>

```
  constructor() {
    this.threshold = 500 * 1024 * 1024; // 500MB
    this.checkInterval = 5 * 60 * 1000; // 5 minutes
  }
  start() {
    setInterval(() =&amp;gt; {
      const usage = process.memoryUsage();
      if (usage.heapUsed &amp;gt; this.threshold) {
        this.cleanup();
      }
    }, this.checkInterval);
  }
  cleanup() {
```

```
        // Clear caches

        // Restart WhatsApp client if needed

        // Garbage collection

    }

}
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

<h4>WhatsApp Client Optimization</h4>

```
const puppeteerOptions = {

    headless: true,

    args: [

        &#39;--no-sandbox&#39;,

        &#39;--disable-setuid-sandbox&#39;,

        &#39;--disable-dev-shm-usage&#39;,

        &#39;--disable-accelerated-2d-canvas&#39;,

        &#39;--no-first-run&#39;,

        &#39;--no-zygote&#39;,

        &#39;--disable-gpu&#39;,

        &#39;--disable-background-timer-throttling&#39;,

        &#39;--disable-backgrounding-occluded-windows&#39;,

        &#39;--disable-renderer-backgrounding&#39;

    ]

};
```


```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```


Database Optimization

<h4>Query Optimization</h4>

```
const orders = await knex(&#39;orders&#39;)

    .select(&#39;*&#39;)

    .where(&#39;business_id&#39;, businessId)

    .where(&#39;created_at&#39;, &#39;&gt;=&#39;, startDate)
```

```
    .orderBy(&#39;created_at&#39;, &#39;desc&#39;)
    .limit(limit);
// Use joins sparingly
const ordersWithDetails = await knex(&#39;orders as o&#39;)
    .select(&#39;o.*&#39;, &#39;b.name as business_name&#39;)
    .leftJoin(&#39;businesses as b&#39;, &#39;o.business_id&#39;, &#39;b.id&#39;)
    .where(&#39;o.business_id&#39;, businessId);
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
<h4>Caching Strategy</h4>
    constructor() {
        this.cache = new Map();
        this.ttl = 5 * 60 * 1000; // 5 minutes
    }
    set(key, value) {
        this.cache.set(key, {
            value,
            timestamp: Date.now()
        });
    }
    get(key) {
        const item = this.cache.get(key);
        if (item &amp;&amp; Date.now() - item.timestamp &lt; this.ttl) {
            return item.value;
        }
        this.cache.delete(key);
        return null;
    }
}
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

Frontend Optimization

<h4>AJAX Loading</h4>

```
async function loadDashboardData() {
    try {
        const response = await fetch(&amp;#39;/api/dashboard/analytics&amp;#39;);
        const data = await response.json();
        updateDashboard(data);
    } catch (error) {
        console.error(&amp;#39;Failed to load dashboard data:&amp;#39;, error);
    }
}
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

<h4>Lazy Loading</h4>

```
async function loadOrders(page = 1) {
    const response = await fetch(`/orders?page=${page}&amp;amp;limit=20`);
    const data = await response.json();
    renderOrders(data.orders);
    updatePagination(data.pagination);
}
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

Ø=Þ€ Deployment

Environment Setup

<h4>Environment Variables</h4>

```
DATABASE_URL=postgresql://user:password@host:port/database

# WhatsApp
WHATSAPP_SESSION_PATH=./sessions

# Security
SESSION_SECRET=your-secret-key
```

NODE_ENV=production

Server

PORT=3000

HOST=0.0.0.0

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

<h4>Database Migration</h4>

npx knex migrate:latest

Seed database

npx knex seed:run


```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```


Production Deployment

<h4>Railway Deployment</h4>

[build]

builder = "nixpacks"

[deploy]

startCommand = "npm start"

healthcheckPath = "/health"

healthcheckTimeout = 300


```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```


<h4>PM2 Configuration</h4>

module.exports = {

  apps: [{

    name: 'novi-platform',

    script: 'src/server.js',

    instances: 1,

    autorestart: true,

    watch: false,

    max_memory_restart: '1G',

```
    env: {
      NODE_ENV: &#39;production&#39;
    }
  }]
};
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

Monitoring & Logging

<h4>Winston Logger</h4>

```
const logger = winston.createLogger({
  level: &#39;info&#39;,
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  transports: [
    new winston.transports.File({ filename: &#39;error.log&#39;, level: &#39;error&#39; }),
    new winston.transports.File({ filename: &#39;combined.log&#39; })
  ]
});
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

<h4>Health Checks</h4>

```
  res.json({
    status: &#39;healthy&#39;,
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    memory: process.memoryUsage()
  });
});
```

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
```

```
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

Ø=Üh Ø=Ü» Development Guide

Setup Development Environment

<h4>Prerequisites</h4>

- Node.js v18+
- PostgreSQL 12+
- Git

<h4>Installation</h4>

git clone <repository-url>

cd novi-platform

Install dependencies

npm install

Setup database

cp .env.example .env

Edit .env with your database credentials

Run migrations

npx knex migrate:latest

Seed database

npx knex seed:run

Start development server

npm run dev

```
2. CacheService checks for cached data
3. If cache miss: AdminService fetches data
4. Data formatted and cached
5. Dashboard rendered with data
6. Real-time updates via AJAX
```

Code Structure

```
% % %  config/           # Configuration files
% % %  database/         # Database migrations and seeds
% % %  services/         # Business logic services
% % %  utils/           # Utility functions
% % %  views/           # EJS templates
% % %  public/          # Static assets
% % %  server.js        # Main server file
% % %  index.js         # Application entry point
```

Development Workflow

<h4>Adding New Features</h4>

- Create feature branch

- Implement feature
- Add tests
- Update documentation
- Create pull request

<h4>Database Changes</h4>

- Create migration file
- Update models/services
- Test migration
- Update documentation

<h4>API Changes</h4>

- Update API documentation
- Implement endpoint
- Add validation
- Test with Postman/curl

Testing

<h4>Unit Tests</h4>

describe('OrderService', () => {

test('should create order', async () => {

const orderData = {

customerName: 'John Doe',

items: 'Pizza, Coke',

totalAmount: 25.00

};

const order = await OrderService.createOrder(orderData);

expect(order.customerName).toBe('John Doe');

});

});

```
% % %  config/           # Configuration files
% % %  database/         # Database migrations and seeds
% % %  services/         # Business logic services
% % %  utils/           # Utility functions
% % %  views/           # EJS templates
% % %  public/          # Static assets
% % %  server.js        # Main server file
% % %  index.js         # Application entry point
```

<h4>Integration Tests</h4>

describe('Orders API', () => {

test('GET /orders should return orders', async () => {

const response = await request(app)

.get('/orders')

.expect(200);

expect(response.body.orders).toBeDefined();

});

});

```
% % %  config/           # Configuration files
% % %  database/         # Database migrations and seeds
% % %  services/         # Business logic services
% % %  utils/           # Utility functions
% % %  views/           # EJS templates
% % %  public/          # Static assets
% % %  server.js        # Main server file
% % %  index.js         # Application entry point
```

Code Standards

<h4>JavaScript Style</h4>

- Use ES6+ features
- Prefer async/await over callbacks
- Use meaningful variable names
- Add JSDoc comments for functions

<h4>Error Handling</h4>

try {

const result = await someAsyncOperation();

return result;

} catch (error) {

logger.error('Operation failed:', error);

throw new Error('Operation failed');

}


```
% % %  config/           # Configuration files
% % %  database/         # Database migrations and seeds
% % %  services/         # Business logic services
% % %  utils/           # Utility functions
% % %  views/           # EJS templates
% % %  public/          # Static assets
% % %  server.js        # Main server file
% % %  index.js         # Application entry point
```


<h4>Logging</h4>

logger.info('Order created', {

orderId: order.id,

businessId: order.business_id,

customerName: order.customer_name

});


```
% % %  config/           # Configuration files
% % %  database/         # Database migrations and seeds
```

```
% % %  services/         # Business logic services
% % %  utils/           # Utility functions
% % %  views/           # EJS templates
% % %  public/          # Static assets
% % %  server.js        # Main server file
% % %  index.js         # Application entry point
```

Ø=ÜÚ Additional Resources

Documentation

- Express.js Documentation
- Knex.js Documentation
- WhatsApp Web.js Documentation
- PostgreSQL Documentation

Tools

- API Testing: Postman, Insomnia
- Database Management: pgAdmin, DBeaver
- Monitoring: PM2, Winston
- Development: VS Code, WebStorm

Best Practices

- Regular security updates
- Database backups
- Performance monitoring
- Code reviews
- Documentation updates

Technical Support
For technical questions or issues, contact:

- Email: tech@novi.com
- GitHub: Create an issue in the repository
- Documentation: Check the docs folder

Novi
Smart Commerce Suite - Built with modern technologies for optimal performance and reliability