**Developer Freelance Marketplace**
Stack:

- **Next.js 14 (App Router)**
- **Better Auth (inside Next.js)**
- **JWT (HttpOnly cookie-based, self-contained)**
- **FastAPI (backend)**
- **SQLModel (ORM)**
- **Neon Serverless PostgreSQL**
- **WebSockets (FastAPI)**
- **Single global marketplace**
- **Manual wallet/bank payment confirmation system**
- **No AI in v1**

This PRD will be structured like a real startup internal document.

---

## 📘 PRODUCT REQUIREMENTS DOCUMENT (PRD)

---

## 1️⃣ PRODUCT OVERVIEW

### 1.1 Product Name

DevMarket (working name)

### 1.2 Product Type

Single-tenant global freelance marketplace.

### 1.3 Target Users

- Freelancers (developers/designers)
- Clients
- Admin

### 1.4 Core Capabilities

- Account system
- Gig marketplace
- Order lifecycle management
- Messaging system
- Manual payment confirmation system

- Review system
- Admin moderation
- Real-time notifications (minimal)

---

## 2️⃣ HIGH-LEVEL ARCHITECTURE

Client (Browser)

↓

Next.js 14 (Frontend + Auth API)

↓ (JWT via HttpOnly cookie)

FastAPI (Business Logic API)

↓

Neon PostgreSQL (SQLModel)

Realtime:

Client ↔ FastAPI WebSocket

---

## 3️⃣ AUTHENTICATION SYSTEM (CRITICAL)

### 3.1 Better Auth Setup (Inside Next.js)

- Configure Better Auth in /app/api/auth/[...route]/route.ts
- Use shared JWT_SECRET
- Token algorithm: HS256
- Expiration: 30 minutes
- HttpOnly cookie
- Secure flag (production)
- SameSite=Lax

---

### 3.2 JWT Payload Structure

Self-contained token must include:

```
{
  "sub": "user_id",
  "email": "user@email.com",
  "role": "user",
```

"is_admin": false,

"iat": 1710000000,

"exp": 1710001800

}

FastAPI verifies token using same secret.

---

### 3.3 FastAPI JWT Verification

Every protected endpoint must:

- Extract JWT from cookie
- Verify signature
- Decode payload
- Attach user to request context

No session storage allowed. Fully stateless.

---

## 4️⃣ ROLE SYSTEM

Single account can:

- Act as Client
- Act as Freelancer

Admin:

- Flag is_admin = true

---

## 5️⃣ DATABASE DESIGN (FINAL SCHEMA)

Using SQLModel.

---

### 5.1 USERS TABLE

| Field | Type |
|---|---|
| id | UUID (PK) |
| email | string (unique) |
| hashed_password | string |
| full_name | string |

| Field | Type |
| --- | --- |
| bio | text |
| avatar_url | string (nullable) |
| is_admin | bool |
| created_at | datetime |

---

**5.2 GIGS TABLE**

| Field | Type |
| --- | --- |
| id | UUID |
| freelancer_id | FK → users |
| title | string |
| description | text |
| price | decimal |
| delivery_days | int |
| is_active | bool |
| created_at | datetime |

---

**5.3 ORDERS TABLE**

| Field | Type |
| --- | --- |
| id | UUID |
| gig_id | FK |
| client_id | FK |
| freelancer_id | FK |
| status | string |
| payment_status | string |
| created_at | datetime |

---

**5.4 ORDER STATUS ENUM**

States:

- PENDING_PAYMENT
- PAYMENT_SUBMITTED
- PAYMENT_CONFIRMED
- IN_PROGRESS
- SUBMITTED
- REVISION_REQUESTED
- COMPLETED
- CANCELLED

Strict state transitions required.

---

**5.5 PAYMENT_PROOFS TABLE**

| Field | Type |
|---|---|
| id | UUID |
| order_id | FK |
| proof_reference | string (transaction ID) |
| payer_name | string |
| amount | decimal |
| submitted_at | datetime |
| verified | bool |

---

**5.6 MESSAGES TABLE**

| Field | Type |
|---|---|
| id | UUID |
| sender_id | FK |
| receiver_id | FK |
| order_id | FK (nullable) |
| content | text |

| Field | Type |
| --- | --- |
| created_at | datetime |

---

**5.7 REVIEWS TABLE**

| Field | Type |
| --- | --- |
| id | UUID |
| order_id | FK |
| reviewer_id | FK |
| reviewee_id | FK |
| rating | int (1–5) |
| comment | text |
| created_at | datetime |

Constraint:

- Only after COMPLETED
- One review per party per order

---

**6️ ORDER STATE MACHINE (STRICT)**

PENDING_PAYMENT

↓ (client submits proof)

PAYMENT_SUBMITTED

↓ (freelancer confirms)

PAYMENT_CONFIRMED

↓

IN_PROGRESS

↓ (freelancer submits work)

SUBMITTED

↓ (client approves)

COMPLETED

If dispute:
→ Admin can force CANCELLED or COMPLETED

No skipping states allowed.

---

## 7️⃣ PAYMENT SYSTEM (MANUAL WALLET)

Flow:

1. Client clicks “Order Gig”
2. System creates order → PENDING_PAYMENT
3. Show wallet/bank details
4. Client transfers money
5. Client submits:
    - Transaction ID
    - Amount
    - Payer name
6. Order → PAYMENT_SUBMITTED
7. Freelancer clicks “Confirm Received”
8. Order → PAYMENT_CONFIRMED

No automatic verification.

---

## 8️⃣ API DESIGN (FastAPI)

### 8.1 Auth Protected Routes

Middleware required for all except public routes.

---

### 8.2 Core API Groups

**USERS**

- GET /users/me
- GET /users/{id}

---

**GIGS**

- POST /gigs
- GET /gigs
- GET /gigs/{id}
- PATCH /gigs/{id}

- DELETE /gigs/{id}

---

**ORDERS**

- POST /orders
- GET /orders/my
- PATCH /orders/{id}/submit-payment
- PATCH /orders/{id}/confirm-payment
- PATCH /orders/{id}/submit-work
- PATCH /orders/{id}/approve
- PATCH /orders/{id}/request-revision

---

**MESSAGES**

- GET /messages/{user_id}
- POST /messages

---

**REVIEWS**

- POST /reviews
- GET /users/{id}/reviews

---

### 9️⃣ REAL-TIME SYSTEM (MINIMAL IMPLEMENTATION)

Use WebSockets for:

- New message notification
- Order status update notification

Connection manager:

- In-memory dictionary: {user_id: websocket}

Limitation:

- Works only on single instance
- Free tier compatible

---

### 🔟 ADMIN PANEL

Admin abilities:

- View all users
- Ban user
- Deactivate gig
- Override order status
- View disputes

Separate admin routes required.

---

## 1️⃣1️⃣ FRONTEND STRUCTURE (Next.js 14)

/app

/auth

/dashboard

/gigs

/orders

/messages

/admin

Use:

- Server Components where possible
- Client Components only when needed

Use:

- React Query for data fetching
- Tailwind for UI

---

## 1️⃣2️⃣ SECURITY REQUIREMENTS

- JWT verified in FastAPI
- Role-based route guards
- Order ownership validation
- Review duplication prevention
- Input validation with Pydantic
- CORS restricted to frontend domain

---

## 1️⃣3️⃣ DEPLOYMENT PLAN (100% FREE)

Frontend:
→ Vercel Free

Backend:
→ Render Free

Database:
→ Neon Free

Limitations:

- Cold starts
- Single instance WebSocket

---

## 1️⃣4️⃣ DEVELOPMENT MILESTONES

**Phase 1 (Week 1–2)**

- DB schema
- FastAPI setup
- JWT verification
- Basic auth flow

**Phase 2 (Week 3)**

- Gig CRUD
- Public listing

**Phase 3 (Week 4–5)**

- Order lifecycle
- Payment proof system

**Phase 4 (Week 6)**

- Messaging system (REST)

**Phase 5 (Week 7)**

- WebSockets integration

**Phase 6 (Week 8)**

- Reviews
- Admin panel
- Deployment

---

## 1️⃣5️⃣ EXPECTED DIFFICULTIES

You will struggle with:

1. JWT cookie verification in FastAPI
2. WebSocket authentication
3. Order state validation
4. Cross-service secret sharing
5. CORS debugging
6. Cold starts in free backend

These are normal.

---

## 1️⃣6️⃣ FUTURE V2 EXTENSIONS

- Stripe
- Escrow system
- Dispute resolution
- AI gig optimization
- Multi-language
- Foreign payment gateways