Before I go deploy. Please review entire codebase and tell me what secrets need to be defined in railway for the entire project

Edit

Scanned codebase to identify crucial Railway deployment secrets.

1.
2.
3.
4.
5.

I'll review the entire codebase to identify all required secrets for Railway deployment.

Analyzed data

View analysis

The repl doesn't have access to the fs module in the browser environment.

S

Search files

I need to use a different approach.

R

Read file

Strategized alternative method to locate environment variables.

Let me systematically review the codebase for all required environment variables by examining the key configuration files:

site:github.com railway environment variables secrets configuration best practices

10 results

Investigated project secrets for comprehensive Railway deployment.

Based on my review of your yoohoo.guru codebase and the documentation provided, here are all the **required secrets/environment variables** that need to be defined in Railway:

🔐 **CRITICAL SECRETS FOR RAILWAY DEPLOYMENT**

Core Application (Required)

bash

NODE_ENV=production

PORT=8000 *# Railway will override, but good to set*

JWT_SECRET=your_super_secret_jwt_key_min_32_chars_change_this_in_production

JWT_EXPIRES_IN=7d

Firebase Configuration (Required)

bash

FIREBASE_PROJECT_ID=your-firebase-project-id

FIREBASE_API_KEY=your_firebase_api_key

FIREBASE_AUTH_DOMAIN=your-project.firebaseapp.com

FIREBASE_DATABASE_URL=https://your-project-default-rtdb.firebaseio.com

FIREBASE_STORAGE_BUCKET=your-project.appspot.com

FIREBASE_MESSAGING_SENDER_ID=123456789012

FIREBASE_APP_ID=1:123456789012:web:abcdef123456

Stripe Payment Processing (Required)

bash

STRIPE_SECRET_KEY=sk_live_your_stripe_secret_key_for_production

STRIPE_WEBHOOK_SECRET=whsec_your_webhook_secret_from_stripe_dashboard

STRIPE_PUBLISHABLE_KEY=pk_live_your_stripe_publishable_key

OpenRouter AI Integration (Required)

bash

OPENROUTER_API_KEY=sk-or-v1-your_openrouter_api_key_here

OPENROUTER_API_URL=https://openrouter.ai/api/v1

Rate Limiting & Security (Recommended)

bash

RATE_LIMIT_WINDOW_MS=900000

RATE_LIMIT_MAX_REQUESTS=100

CORS & Domain Configuration (Production)

bash

APP_DOMAIN=yoohoo.guru

APP_SUPPORT_EMAIL=support@yoohoo.guru

🎯 **FRONTEND VARIABLES (if using React frontend)**

Only needed if you deploy the frontend separately or serve it from the backend:

bash

REACT_APP_API_URL=https://your-railway-backend-domain.railway.app/api

REACT_APP_ENVIRONMENT=production

REACT_APP_FIREBASE_API_KEY=your_firebase_api_key

REACT_APP_FIREBASE_AUTH_DOMAIN=your-project.firebaseapp.com

REACT_APP_FIREBASE_DATABASE_URL=https://your-project-default-rtdb.firebaseio.com

REACT_APP_FIREBASE_PROJECT_ID=your-firebase-project-id

REACT_APP_FIREBASE_STORAGE_BUCKET=your-project.appspot.com

REACT_APP_FIREBASE_MESSAGING_SENDER_ID=123456789012

REACT_APP_FIREBASE_APP_ID=1:123456789012:web:abcdef123456

📧 **EMAIL CONFIGURATION (Optional)**

If you plan to send emails from the backend:

bash

SMTP_HOST=smtp.gmail.com

SMTP_PORT=587

SMTP_USER=your_email@gmail.com

SMTP_PASS=your_app_specific_password

🗂️ **DATABASE (Optional)**

If using additional database beyond Firebase:

bash

DATABASE_URL=postgresql://username:password@host:port/database

---

📊 **LOGGING & MONITORING (Optional)**

bash

LOG_LEVEL=info

LOG_FILE_PATH=/tmp/app.log

⚠️ **PRIORITY ORDER FOR RAILWAY SETUP:**

IMMEDIATE (Deploy will fail without these):

1. NODE_ENV=production
2. JWT_SECRET (min 32 characters)
3. All Firebase variables
4. STRIPE_SECRET_KEY & STRIPE_WEBHOOK_SECRET
5. OPENROUTER_API_KEY

SOON (Features won't work without these):

6. Rate limiting variables
7. App domain configuration
8. Email configuration (if sending emails)

LATER (Nice to have):

9. Logging configuration
10. Additional database connections

🚀 **HOW TO SET IN RAILWAY:**

bash

 # Via Railway Dashboard:

 # 1. Go to your project → Variables tab

 # 2. Add each variable with its value

 # 3. Deploy

 # Via Railway CLI:

railway variables set NODE_ENV=production

railway variables set JWT_SECRET=your_32_char_secret_here

railway variables set FIREBASE_PROJECT_ID=your-project-id

 # ... etc

Total Essential Variables: ~15-20 depending on which features you're using.

The **absolute minimum** to get your app running is about 8-10 variables (Node.js + Firebase + JWT + Stripe basics). You can add the others as you enable more features!

Retry

MB

How does the real-time db need to be setup in firebase?

Edit

Investigating Firebase database setup for yoohoo.guru project

1.
2.
3.
4.
5.
6.
1.
2.
3.
4.
5.

I'll provide a complete Firebase Realtime Database setup guide based on your yoohoo.guru application architecture.

Firebase Realtime Database setup security rules Node.js skill sharing platform

10 results

Mapped out comprehensive Firebase Realtime Database setup strategy.

1.
2.
3.
4.

5.
6.
1.
2.
3.
4.
5.

🔥 **Firebase Realtime Database Setup for yoohoo.guru**

Based on your skill-sharing platform architecture, here's the complete Firebase setup guide:

1. Create the Database

Step 1: Firebase Console Setup

1. Go to Firebase Console
2. Select your project (or create one)
3. Navigate to **Realtime Database** in the left sidebar
4. Click **"Create Database"**
5. **Choose location**: us-central1 (recommended for best performance)
6. **Security Rules**: Select **"Start in test mode"** (we'll secure it properly below)

Step 2: Get Your Database URL

Your database URL will be: https://your-project-id-default-rtdb.firebaseio.com/

---

2. Database Structure for yoohoo.guru

json

```
{
  "users": {
    "user_id_123": {
      "profile": {
        "displayName": "John Doe",
        "email": "john@example.com",
        "avatar": "https://...",
        "location": "San Francisco, CA",
        "bio": "Experienced guitar teacher and cooking enthusiast",
        "verified": true,
        "createdAt": 1672531200000
```

```
    },
    "roles": {
      "coach": true,
      "learner": true
    },
    "stats": {
      "sessionsCompleted": 25,
      "rating": 4.8,
      "totalEarnings": 1250
    }
  }
},

"skills": {
  "skill_id_456": {
    "name": "Guitar Lessons",
    "category": "music",
    "description": "Beginner to intermediate guitar instruction",
    "ownerId": "user_id_123",
    "type": "coach", // or "angel" for Angel's List services
    "pricing": {
      "hourly": 50,
      "currency": "USD",
      "credits": 2
    },
    "availability": {
      "flexible": true,
      "preferredTimes": ["evenings", "weekends"]
    },
    "location": {
      "type": "both", // "online", "in-person", "both"
      "address": "San Francisco Bay Area"
    },
```

```
    "active": true,
    "createdAt": 1672531200000
  }
},

"requests": {
  "request_id_789": {
    "learnerId": "user_id_456",
    "skillCategory": "cooking",
    "title": "Learn Italian Cooking",
    "description": "Want to learn authentic pasta making",
    "budget": {
      "max": 40,
      "currency": "USD",
      "paymentType": "credits"
    },
    "schedule": {
      "flexible": true,
      "preferredTimes": ["weekends"]
    },
    "status": "active", // "active", "matched", "completed", "cancelled"
    "createdAt": 1672531200000
  }
},

"sessions": {
  "session_id_101": {
    "coachId": "user_id_123",
    "learnerId": "user_id_456",
    "skillId": "skill_id_456",
    "status": "confirmed", // "pending", "confirmed", "completed", "cancelled"
    "scheduling": {
      "startTime": 1672617600000,
```

```
      "endTime": 1672621200000,
      "timezone": "America/Los_Angeles"
    },
    "location": {
      "type": "online",
      "meetingLink": "https://meet.google.com/abc-def-ghi"
    },
    "payment": {
      "amount": 50,
      "currency": "USD",
      "stripePaymentIntentId": "pi_1234567890",
      "escrowReleased": false
    },
    "createdAt": 1672531200000,
    "updatedAt": 1672531200000
  }
},

"reviews": {
  "review_id_202": {
    "sessionId": "session_id_101",
    "reviewerId": "user_id_456",
    "revieweeId": "user_id_123",
    "rating": 5,
    "comment": "Excellent guitar lesson! Very patient teacher.",
    "tags": ["patient", "clear", "prepared"],
    "approved": true,
    "createdAt": 1672531200000
  }
},

"messages": {
  "conversation_id_303": {
```

```
    "participants": {
      "user_id_123": true,
      "user_id_456": true
    },
    "lastMessage": {
      "text": "Looking forward to our session tomorrow!",
      "senderId": "user_id_456",
      "timestamp": 1672531200000
    },
    "messages": {
      "msg_id_001": {
        "senderId": "user_id_456",
        "text": "Hi, interested in guitar lessons",
        "timestamp": 1672531200000,
        "read": true
      }
    }
  }
},

"matches": {
  "match_id_404": {
    "coachId": "user_id_123",
    "learnerId": "user_id_456",
    "skillId": "skill_id_456",
    "requestId": "request_id_789",
    "score": 0.92,
    "status": "pending", // "pending", "accepted", "declined"
    "aiRecommendation": {
      "reasons": ["skill match", "schedule compatibility", "location proximity"],
      "confidence": 0.92
    },
    "createdAt": 1672531200000
```

```
    }
  }
}
```

---

3. Security Rules

Replace the default rules with these **production-ready** security rules:

```
json
{
  "rules": {
    ".read": false,
    ".write": false,

    // Users can read/write their own profile data
    "users": {
      "$userId": {
        ".read": "auth != null && (auth.uid == $userId || auth.token.admin == true)",
        ".write": "auth != null && auth.uid == $userId",
        ".validate": "newData.hasChildren(['profile']) &&
newData.child('profile').hasChildren(['displayName', 'email'])"
      }
    },

    // Skills - coaches can manage their own, everyone can read active skills
    "skills": {
      ".read": "auth != null",
      "$skillId": {
        ".write": "auth != null && (auth.uid == newData.child('ownerId').val() || auth.token.admin ==
true)",
        ".validate": "newData.hasChildren(['name', 'ownerId', 'active']) &&
newData.child('ownerId').val() == auth.uid"
      }
    },

    // Requests - learners can manage their own, everyone can read active requests
```

```
  "requests": {
    ".read": "auth != null",
    "$requestId": {
      ".write": "auth != null && (auth.uid == newData.child('learnerId').val() || auth.token.admin == true)",
      ".validate": "newData.hasChildren(['learnerId', 'title']) && newData.child('learnerId').val() == auth.uid"
    }
  },

  // Sessions - participants can read/write their sessions
  "sessions": {
    "$sessionId": {
      ".read": "auth != null && (auth.uid == data.child('coachId').val() || auth.uid == data.child('learnerId').val() || auth.token.admin == true)",
      ".write": "auth != null && (auth.uid == data.child('coachId').val() || auth.uid == data.child('learnerId').val() || auth.token.admin == true)"
    }
  },

  // Reviews - reviewers can create, everyone can read approved reviews
  "reviews": {
    ".read": "auth != null",
    "$reviewId": {
      ".write": "auth != null && (auth.uid == newData.child('reviewerId').val() || auth.token.admin == true)",
      ".validate": "newData.hasChildren(['sessionId', 'reviewerId', 'revieweeId', 'rating']) && newData.child('rating').val() >= 1 && newData.child('rating').val() <= 5"
    }
  },

  // Messages - only conversation participants can access
  "messages": {
    "$conversationId": {
      ".read": "auth != null && data.child('participants').child(auth.uid).val() == true",
```

```
      ".write": "auth != null && data.child('participants').child(auth.uid).val() == true",
      "messages": {
        "$messageId": {
          ".validate": "newData.hasChildren(['senderId', 'text', 'timestamp']) &&
newData.child('senderId').val() == auth.uid"
        }
      }
    }
  },

  // AI Matches - users can read their own matches
  "matches": {
    "$matchId": {
      ".read": "auth != null && (auth.uid == data.child('coachId').val() || auth.uid ==
data.child('learnerId').val())",
      ".write": "auth != null && auth.token.admin == true" // Only backend/AI can create matches
    }
  },

  // Public data that everyone can read
  "public": {
    ".read": true,
    ".write": "auth != null && auth.token.admin == true",
    "categories": {
      ".read": true
    },
    "locations": {
      ".read": true
    }
  }
 }
}
```

---

4. Database Indexes

Add these indexes for optimal query performance. Go to **Database > Rules tab** and add:

```
json
{
  "rules": {
    // ... your security rules above ...
  },
  ".indexOn": [
    // Skills indexes
    "skills": {
      ".indexOn": ["category", "active", "ownerId", "createdAt"]
    },

    // Requests indexes
    "requests": {
      ".indexOn": ["skillCategory", "status", "learnerId", "createdAt"]
    },

    // Sessions indexes
    "sessions": {
      ".indexOn": ["coachId", "learnerId", "status", "createdAt"]
    },

    // Reviews indexes
    "reviews": {
      ".indexOn": ["revieweeId", "approved", "createdAt"]
    },

    // Matches indexes
    "matches": {
      ".indexOn": ["coachId", "learnerId", "status", "score", "createdAt"]
    }
  ]
}
```

---

5. Firebase Authentication Setup

Enable Authentication Methods:

1. Go to **Authentication > Sign-in method**
2. Enable:
    - ✅ **Email/Password**
    - ✅ **Google** (recommended)
    - ✅ **Apple** (for mobile apps)
    - ✅ **Anonymous** (for guest browsing)

Custom Claims for Admin Users:

You'll need to set up custom claims for admin functionality. In your backend:

javascript

```
// Set admin claim (run once for admin users)
admin.auth().setCustomUserClaims(userId, {
  admin: true
});
```

---

6. Test Your Setup

Rules Simulator:

1. Go to **Database > Rules tab**
2. Click **"Simulator"**
3. Test different scenarios:
    - Authenticated user reading their own data ✅
    - User trying to read another user's private data ❌
    - Anonymous user accessing public data ✅

---

7. Environment Variables for Railway

Add these to Railway (in addition to your previous list):

bash

```
# Firebase Realtime Database (add these to your existing Firebase vars)
FIREBASE_DATABASE_URL=https://your-project-id-default-rtdb.firebaseio.com/
FIREBASE_ADMIN_SDK_JSON={"type":"service_account","project_id":"..."}
```

---

8. Initialize in Your Backend Code

Your backend should connect like this:

javascript

```
// backend/src/config/firebase.js
const admin = require('firebase-admin');

const serviceAccount = JSON.parse(process.env.FIREBASE_ADMIN_SDK_JSON);

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount),
  databaseURL: process.env.FIREBASE_DATABASE_URL
});

const db = admin.database();
module.exports = { admin, db };
```

🚨 **Security Checklist:**

- ✅ **Never use test mode in production**
- ✅ **Always authenticate users before database access**
- ✅ **Use custom claims for role-based permissions**
- ✅ **Validate all data writes with .validate rules**
- ✅ **Index frequently queried fields**
- ✅ **Keep sensitive data (payment info) in secure backend functions**
- ✅ **Test rules thoroughly before deploying**

This setup provides a scalable, secure foundation for your skill-sharing platform with proper data isolation, role-based access, and performance optimizations! 🎯

Retry

MB

What is pricing scheme and flow of all money?

Edit

Probed platform's financial mechanics and revenue streams.

1.
2.

3.
4.
5.
6.
7.
-
-
-
-
-

Based on your yoohoo.guru platform documentation, here's the complete pricing scheme and money flow architecture:

💰 **PRICING SCHEME**

1. Dual Currency System

Credits (Internal Currency)

- **1 Credit = 1 Hour Baseline**
- **Earning Credits:**
    - Teaching skills on Coach Yeti
    - Completing jobs on Angel's List
    - Referral bonuses
    - Platform rewards/challenges
- **Spending Credits:**
    - Learning sessions with coaches
    - Skill swaps (mutual credit exchange)
    - Premium features

Cash (USD via Stripe)

- **Direct payments** for premium services
- **One-time session bookings**
- **Subscription fees** (Guru Pass)

2. Session Pricing Structure

Coach Yeti (Skill Teaching)

Base Rate: $25/hour average

Coach Sets: $15-$100/hour (or credit equivalent)

Credit Multiplier: 1.0x to 2.5x based on skill complexity

Examples:

- Basic Cooking: 1.0 credits/hour or $25
- Guitar Lessons: 1.5 credits/hour or $35
- Programming: 2.0 credits/hour or $50
- Professional Consulting: 2.5 credits/hour or $65

Angel's List (Services/Rentals)

Service-based pricing:

- Handyman: $30-80/hour
- Moving Help: $25-40/hour
- Pet Care: $15-30/hour
- Equipment Rental: $10-50/day
- Assembly: $20-60/project

🔄 MONEY FLOW ARCHITECTURE

Revenue Streams (Platform Takes)

1. Commission on Cash Transactions: 15%

User Pays: $50 for guitar lesson

Platform Keeps: $7.50 (15%)

Coach Receives: $42.50

2. Credit Transactions: 0% Commission

Community-friendly approach

Pure skill swap - no platform cut

Encourages local economy building

3. Premium Subscriptions

Guru Pass: $10/month

- Unlimited bookings
- Priority matching
- Analytics dashboard
- Lower cash transaction fees (10% vs 15%)

4. Value-Added Services

Skill Verification: $15 one-time

- AI assessment + peer review
- Certification badge
- Priority in search results

5. Business Services

Community Event Tools: $50/month

- For libraries, NGOs, organizations
- Advanced analytics and promotion
- Bulk event management

💳 PAYMENT PROCESSING FLOW

Stripe Integration Architecture

Step 1: Session Booking

1. Learner books session ($50)
2. Payment held in Stripe escrow
3. Platform fee calculated (15% = $7.50)
4. Coach notified of booking

Step 2: Session Completion

1. Session occurs
2. Both parties confirm completion
3. OR 24-hour auto-confirm if no disputes
4. Funds released from escrow

Step 3: Fund Distribution

From $50 session payment:

├── Platform Fee: $7.50 (15%)
├── Stripe Processing: ~$1.75 (2.9% + $0.30)
└── Coach Payout: $40.75

Net Platform Revenue: $5.75 per $50 session

🏦 ESCROW & PAYOUT SYSTEM

Escrow Timeline

Payment Held: Until session completion

Auto-Release: 24 hours post-session (no disputes)

Dispute Window: 48 hours for resolution

Payout Schedule: Weekly (Fridays)

Minimum Payout: $25

Dispute Resolution

1. Either party flags issue within 24 hours
2. Platform reviews evidence
3. Possible outcomes:
   - Full refund to learner
   - Full payment to coach
   - Partial split (e.g., 50/50)
   - Mediated resolution

📊 FINANCIAL PROJECTIONS

Monthly Revenue Breakdown (at 2,000 MAU)

Commission Revenue

Average session price: $25

Sessions per user/month: 0.4

Total monthly sessions: 800

Gross transaction volume: $20,000

Platform commission (15%): $3,000/month

Subscription Revenue

Premium users (15% of 2,000): 300 users

Guru Pass price: $10/month

Monthly revenue: $3,000/month

Additional Revenue

Skill verifications: $250/month

Event tools: $500/month

Sponsored challenges: $2,000/month

Data insights: $1,000/month

Other revenue: $5,250/month

Total Monthly Revenue: $15,000

Cost Structure

Monthly Costs: $8,500

├── Technology: $2,500

├── Personnel: $5,000

└── Operations: $1,000

Net Profit: $6,500/month (43% margin)

🎯 USER PRICING EXAMPLES

Coach Yeti Scenarios

Scenario 1: Credit Swap

Teacher: Offers 2 hours Python coding

Learner: Offers 2 hours Spanish lessons

Exchange: 2 credits each direction

Platform Fee: $0 (0% on credits)

Scenario 2: Cash Payment

Guitar lesson: $40/hour

Platform takes: $6 (15%)

Coach receives: $34

Learner pays: $40

Angel's List Scenarios

Scenario 3: Handyman Service

Furniture assembly: $60

Platform takes: $9 (15%)

Helper receives: $51

Customer pays: $60

Scenario 4: Equipment Rental

Power drill rental: $25/day

Platform takes: $3.75 (15%)

Owner receives: $21.25

Renter pays: $25

🔐 PAYMENT SECURITY & COMPLIANCE

PCI Compliance

- All payments processed through Stripe

- No credit card data stored on platform
- SSL/TLS encryption for all transactions

Anti-Fraud Measures

- Stripe Radar for fraud detection
- User verification requirements
- Dispute/chargeback protection
- Velocity limits on new accounts

📈 PRICING STRATEGY

Competitive Positioning

Uber/Lyft: 25-30% commission

Airbnb: 14-16% host fee + 3% guest fee

Upwork: 5-20% sliding scale

TaskRabbit: ~15-20% service fee

YooHoo Guru: 15% cash, 0% credits

= More coach-friendly than most platforms

Growth Strategy

Phase 1: 0% credits to build network effects

Phase 2: Keep low cash fees to compete

Phase 3: Add premium features for additional revenue

Phase 4: Potential dynamic pricing based on demand

This dual-currency model encourages community building (via credits) while generating sustainable revenue (via cash transactions and subscriptions). The 15% commission is competitive while the 0% credit model is unique in the marketplace! 🚀