VINEFOX Troubleshooting Guide

🔧 Troubleshooting & Debugging Overview

Current Version: vinefox (5f3b53c)
Target Audience: Developers, DevOps Engineers, Support Team
Issue Resolution Level: Production Support

🚨 Critical Issues & Quick Fixes

1. Application Won't Start

Issue: Development Server Fails to Start

```
# Error: Port 3000 already in use
Error: listen EADDRINUSE: address already in use :::3000
```

Quick Fix:

```
# Kill process using port 3000
lsof -ti:3000 | xargs kill -9

# Or use different port
npm run dev -- -p 3001
```

Prevention:

- Always check for running processes before starting dev server
- Use `npm run dev -- --help` to see available options

Issue: Build Failures

```
# Error: Module not found
Module not found: Can't resolve '@/components/ui/button'
```

Quick Fix:

```
# Clear Next.js cache
rm -rf .next
rm -rf node_modules/.cache

# Reinstall dependencies
npm install

# Restart dev server
npm run dev
```

Root Cause Analysis:

- Missing or incorrect import paths
- TypeScript path mapping issues
- Cached build artifacts

2. Database Connection Issues

Issue: Prisma Connection Failed

```
# Error: Can't reach database server
Error: P1001: Can't reach database server at `localhost:5432`
```

Quick Fix:

```
# Check if PostgreSQL is running
sudo systemctl status postgresql

# Start PostgreSQL if stopped
sudo systemctl start postgresql

# Verify connection
psql -h localhost -U username -d wine_store
```

Database Health Check:

```
# Test database connection
npx prisma db pull

# Check database status
npx prisma migrate status

# Verify schema
npx prisma validate
```

Issue: Migration Failures

```
# Error: Migration failed
Error: P3006: Migration `20241201120000_add_new_field` failed
```

Quick Fix:

```
# Reset database (WARNING: This will delete all data)
npx prisma migrate reset

# Or apply specific migration
npx prisma migrate resolve --applied 20241201120000_add_new_field
```

Prevention:

- Always backup database before migrations
- Test migrations on development environment first
- Use `--create-only` flag to review migration SQL

3. Authentication Issues

Issue: Clerk Authentication Fails

```
# Error: Clerk configuration error
Error: Clerk configuration error: Missing publishable key
```

Quick Fix:

```
# Verify environment variables
echo $NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY
echo $CLERK_SECRET_KEY

# Check .env.local file
cat .env.local | grep CLERK

# Restart development server
npm run dev
```

Environment Variable Checklist:

- [ ] `NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY` is set
- [ ] `CLERK_SECRET_KEY` is set
- [ ] Keys are valid and active
- [ ] Environment file is loaded correctly

Issue: User Session Expired

```
// Error: User not authenticated
Error: User not found or session expired
```

Quick Fix:

```
// Clear local storage and cookies
localStorage.clear();
document.cookie.split(";").forEach(function (c) {
  document.cookie = c
    .replace(/^ +/, "")
    .replace(/=.*/, "=;expires=" + new Date().toUTCString() + ";path=/");
});

// Redirect to sign-in page
window.location.href = "/sign-in";
```

Prevention:

- Implement proper session timeout handling
- Use Clerk's built-in session management
- Handle authentication errors gracefully

🔍 Debugging Procedures

1. Frontend Debugging

Component Debugging

```
// Add debug logging to components
const ProductCard = ({ product }: ProductCardProps) => {
  console.log("ProductCard render:", { product, timestamp: Date.now() });

  useEffect(() => {
    console.log("ProductCard mounted:", product.id);
    return () => console.log("ProductCard unmounted:", product.id);
  }, [product.id]);

  // Component implementation
};
```

Debug Tools:

- **React Developer Tools:** Component tree and state inspection
- **Redux DevTools:** State management debugging
- **Network Tab:** API request/response inspection
- **Console Logging:** Strategic log placement

Performance Debugging

```
// Performance monitoring in components
const usePerformanceMonitor = (componentName: string) => {
  useEffect(() => {
    const startTime = performance.now();

    return () => {
      const endTime = performance.now();
      console.log(`${componentName} render time: ${endTime - startTime}ms`);
    };
  });
};

// Usage
const ProductGrid = () => {
  usePerformanceMonitor("ProductGrid");
  // Component implementation
};
```

Performance Metrics:

- **Render Times:** Component render performance
- **Re-render Frequency:** Unnecessary re-renders
- **Bundle Size:** JavaScript bundle analysis
- **Image Loading:** Image optimization issues

2. Backend Debugging

API Route Debugging

```
// Enhanced error logging in API routes
export async function GET(request: NextRequest) {
  const startTime = Date.now();
  const requestId = crypto.randomUUID();

  console.log(`[${requestId}] API Request Started:`, {
    url: request.url,
    method: request.method,
    headers: Object.fromEntries(request.headers.entries()),
    timestamp: new Date().toISOString(),
  });
```

```
    try {
      const result = await fetchData();

      const responseTime = Date.now() - startTime;
      console.log(`[${requestId}] API Request Success:`, {
        responseTime,
        resultSize: JSON.stringify(result).length,
      });

      return NextResponse.json(result);
    } catch (error) {
      const responseTime = Date.now() - startTime;
      console.error(`[${requestId}] API Request Failed:`, {
        error: error.message,
        stack: error.stack,
        responseTime,
      });

      return NextResponse.json(
        { error: "Internal server error", requestId },
        { status: 500 }
      );
    }
  }
```

Debug Information:

- **Request ID:** Unique identifier for request tracking
- **Timing:** Request processing duration
- **Headers:** Request header information
- **Error Details:** Comprehensive error information

Database Query Debugging

```
// Prisma query logging
const prisma = new PrismaClient({
  log: [
    {
      emit: "event",
      level: "query",
    },
    {
      emit: "event",
      level: "error",
    },
    {
      emit: "event",
      level: "info",
    },
  ],
});

// Query performance monitoring
prisma.$on("query", (e) => {
  console.log("Database Query:", {
    query: e.query,
    params: e.params,
    duration: `${e.duration}ms`,
    timestamp: new Date().toISOString(),
  });

  // Alert on slow queries
  if (e.duration > 1000) {
    console.warn("Slow Query Detected:", e.query, `${e.duration}ms`);
  }
});

// Error monitoring
prisma.$on("error", (e) => {
  console.error("Database Error:", {
    target: e.target,
    timestamp: new Date().toISOString(),
  });
});
```

3. Environment Debugging

Environment Variable Debugging

```
// Environment variable validation
export function validateEnvironment() {
  const requiredVars = [
    "DATABASE_URL",
    "NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY",
    "CLERK_SECRET_KEY",
    "STRIPE_SECRET_KEY",
  ];

  const missingVars = requiredVars.filter((varName) => !process.env[varName]);

  if (missingVars.length > 0) {
    throw new Error(
      `Missing required environment variables: ${missingVars.join(", ")}`
    );
  }

  console.log("Environment validation passed");
}

// Call this in your app initialization
validateEnvironment();
```

Configuration Debugging

```
// Configuration debugging helper
export function debugConfiguration() {
  console.log("Application Configuration:", {
    nodeEnv: process.env.NODE_ENV,
    databaseUrl: process.env.DATABASE_URL ? "Set" : "Missing",
    clerkKey: process.env.NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY ? "Set" :
"Missing",
    stripeKey: process.env.STRIPE_SECRET_KEY ? "Set" : "Missing",
    appUrl: process.env.NEXT_PUBLIC_APP_URL,
    timestamp: new Date().toISOString(),
  });
}
```

🐛 Common Issues & Solutions

1. UI/UX Issues

Issue: Mobile Responsiveness Problems

Symptoms:

- Elements overlapping on mobile devices
- Touch targets too small
- Layout breaking on small screens

Solutions:

```css
/* Ensure proper mobile breakpoints */
@media (max-width: 640px) {
  .mobile-optimized {
    padding: 1rem;
    font-size: 0.875rem;
  }

  .touch-target {
    min-height: 44px;
    min-width: 44px;
  }
}

/* Use responsive design patterns */
.container {
  width: 100%;
  max-width: 1200px;
  margin: 0 auto;
  padding: 0 1rem;
}

@media (min-width: 640px) {
  .container {
    padding: 0 2rem;
  }
}
```

Testing Checklist:

- [ ] Test on multiple device sizes
- [ ] Verify touch interactions
- [ ] Check text readability
- [ ] Test navigation usability

Issue: Dark Mode Not Working

Symptoms:

- Theme toggle not functioning
- Inconsistent theme application
- Theme not persisting

Solutions:

```tsx
// Verify theme provider setup
import { ThemeProvider } from "next-themes";

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html lang="en" suppressHydrationWarning>
      <body>
        <ThemeProvider
          attribute="class"
          defaultTheme="system"
          enableSystem
          disableTransitionOnChange
        >
          {children}
        </ThemeProvider>
      </body>
    </html>
  );
}

// Check theme toggle implementation
const ThemeToggle = () => {
  const { theme, setTheme } = useTheme();

  console.log("Current theme:", theme); // Debug logging

  return (
    <button onClick={() => setTheme(theme === "dark" ? "light" : "dark")}>
      Toggle Theme
    </button>
  );
};
```

2. Performance Issues

Issue: Slow Page Load Times

Symptoms:

- Pages take > 3 seconds to load
- Images loading slowly
- Large JavaScript bundles

Solutions:

```
// Implement lazy loading
import dynamic from "next/dynamic";

const LazyComponent = dynamic(() => import("./HeavyComponent"), {
  loading: () => <div>Loading...</div>,
  ssr: false, // Disable server-side rendering if not needed
});

// Optimize images
import Image from "next/image";

<Image
  src={product.image}
  alt={product.name}
  width={300}
  height={400}
  priority={isAboveFold} // Prioritize above-fold images
  sizes="(max-width: 640px) 100vw, (max-width: 1024px) 50vw, 33vw"
/>;
```

Performance Checklist:

- [ ] Enable Next.js image optimization
- [ ] Implement code splitting
- [ ] Use lazy loading for non-critical components
- [ ] Optimize bundle size

Issue: Database Query Performance

Symptoms:

- API responses > 2 seconds
- Database connection timeouts
- High memory usage

Solutions:

```sql
-- Add database indexes
CREATE INDEX "Wine_search_idx" ON "Wine"("name", "type", "price");
CREATE INDEX "Wine_featured_idx" ON "Wine"("featured", "createdAt");

-- Optimize queries
SELECT w.*, r.name as region_name
FROM "Wine" w
JOIN "Region" r ON w."regionId" = r.id
WHERE w.featured = true
ORDER BY w."createdAt" DESC
LIMIT 10;

-- Monitor query performance
SELECT query, mean_time, calls
FROM pg_stat_statements
ORDER BY mean_time DESC
LIMIT 10;
```

3. Security Issues

Issue: Authentication Bypass

Symptoms:

- Users accessing protected routes without authentication
- Session management issues
- Unauthorized API access

Solutions:

```typescript
// Verify middleware configuration
import { clerkMiddleware } from "@clerk/nextjs/server";

export default clerkMiddleware();

export const config = {
  matcher: [
    "/((?!_next|[^?]*\\.(?:html?|css|js(?!on)|jpe?g|webp|png|gif|svg|ttf|woff2?
|ico|csv|docx?|xlsx?|zip|webmanifest)).*)",
    "/(api|trpc)(.*)",
  ],
};

// Verify API route protection
export async function GET(request: NextRequest) {
  const { userId } = await auth();
  if (!userId) {
    return NextResponse.json({ error: "Unauthorized" }, { status: 401 });
  }

  // Protected route logic
}
```

Security Checklist:

- [ ] All routes properly protected
- [ ] API endpoints require authentication
- [ ] Input validation implemented
- [ ] CORS properly configured

🔧 Maintenance & Prevention

1. Regular Health Checks

Application Health Monitoring

```
// Health check endpoint
export async function GET() {
  const healthChecks = {
    database: false,
    authentication: false,
    payment: false,
    timestamp: new Date().toISOString(),
  };

  try {
    // Database health check
    await prisma.$queryRaw`SELECT 1`;
    healthChecks.database = true;
  } catch (error) {
    console.error("Database health check failed:", error);
  }

  try {
    // Authentication health check
    const clerkHealth = await fetch("https://api.clerk.dev/health");
    healthChecks.authentication = clerkHealth.ok;
  } catch (error) {
    console.error("Authentication health check failed:", error);
  }

  try {
    // Payment health check
    const stripeHealth = await fetch("https://api.stripe.com/v1/health");
    healthChecks.payment = stripeHealth.ok;
  } catch (error) {
    console.error("Payment health check failed:", error);
  }

  const isHealthy = Object.values(healthChecks).every(
    (check) => check === true
  );

  return NextResponse.json(healthChecks, {
    status: isHealthy ? 200 : 503,
  });
}
```

Automated Monitoring Scripts

```bash
#!/bin/bash
# Health check script

APP_URL="https://your-app.vercel.app"
HEALTH_ENDPOINT="/api/health"
LOG_FILE="/var/log/app-health.log"

# Perform health check
response=$(curl -s -o /dev/null -w "%{http_code}" "$APP_URL$HEALTH_ENDPOINT")

if [ "$response" = "200" ]; then
  echo "$(date): Application is healthy" >> $LOG_FILE
else
  echo "$(date): Application health check failed (HTTP $response)" >> $LOG_FILE

  # Send alert
  curl -X POST -H "Content-Type: application/json" \
    -d '{"text":"Application health check failed"}' \
    $SLACK_WEBHOOK_URL
fi
```

2. Performance Monitoring

Core Web Vitals Tracking

```typescript
// Web Vitals monitoring
import { getCLS, getFID, getFCP, getLCP, getTTFB } from "web-vitals";

function sendToAnalytics(metric: any) {
  // Send to analytics service
  console.log("Web Vital:", metric);

  // Alert on poor performance
  if (metric.name === "LCP" && metric.value > 2500) {
    sendPerformanceAlert("Poor LCP performance", metric);
  }

  if (metric.name === "FID" && metric.value > 100) {
    sendPerformanceAlert("Poor FID performance", metric);
  }
}

// Monitor all web vitals
getCLS(sendToAnalytics);
getFID(sendToAnalytics);
getFCP(sendToAnalytics);
getLCP(sendToAnalytics);
getTTFB(sendToAnalytics);
```

Database Performance Monitoring

```typescript
// Database performance monitoring
export async function monitorDatabasePerformance() {
  try {
    // Monitor slow queries
    const slowQueries = await prisma.$queryRaw`
      SELECT query, mean_time, calls
      FROM pg_stat_statements
      WHERE mean_time > 100
      ORDER BY mean_time DESC
      LIMIT 10
    `;

    if (slowQueries.length > 0) {
      console.warn("Slow database queries detected:", slowQueries);
      sendDatabasePerformanceAlert(slowQueries);
    }

    // Monitor connection pool
    const connectionStats = await prisma.$queryRaw`
      SELECT count(*) as active_connections
      FROM pg_stat_activity
      WHERE state = 'active'
    `;

    if (connectionStats[0].active_connections > 80) {
      sendConnectionPoolAlert(connectionStats[0]);
    }
  } catch (error) {
    console.error("Database monitoring error:", error);
  }
}
```

3. Error Prevention

Code Quality Checks

```
# Pre-commit hooks
#!/bin/bash
# .git/hooks/pre-commit

echo "Running pre-commit checks..."

# Run linting
npm run lint
if [ $? -ne 0 ]; then
  echo "Linting failed. Please fix issues before committing."
  exit 1
fi

# Run tests
npm run test
if [ $? -ne 0 ]; then
  echo "Tests failed. Please fix issues before committing."
  exit 1
fi

# Check for console.log statements
if git diff --cached | grep -q "console.log"; then
  echo "Warning: console.log statements detected. Consider removing them."
fi

echo "Pre-commit checks passed!"
```

Automated Testing

```
// Automated test suite
describe("Critical User Flows", () => {
  test("User can complete purchase flow", async () => {
    // Test complete purchase flow
    const user = await createTestUser();
    const product = await createTestProduct();

    // Add to cart
    const cart = await addToCart(product.id, 1, user.id);
    expect(cart.numItemsInCart).toBe(1);

    // Checkout
    const order = await createOrder(user.id, cart.cartItems);
    expect(order.isPaid).toBe(false);

    // Payment
    const payment = await processPayment(order.id);
    expect(payment.status).toBe("succeeded");
  });

  test("Authentication flow works correctly", async () => {
    // Test authentication flow
    const user = await createTestUser();
    const session = await authenticateUser(user.email, "password");

    expect(session.userId).toBe(user.id);
    expect(session.isValid).toBe(true);
  });
});
```

📞 Support & Escalation

1. Issue Classification

Priority Levels

- **P1 (Critical):** Application completely down, data loss
- **P2 (High):** Major functionality broken, security issues
- **P3 (Medium):** Minor functionality issues, performance problems
- **P4 (Low):** UI/UX issues, documentation updates

Escalation Procedures

```
// Escalation matrix
export const escalationMatrix = {
  P1: {
    responseTime: "15 minutes",
    escalation: "Immediate to senior developer + DevOps",
    notification: "Slack + Email + Phone",
  },
  P2: {
    responseTime: "1 hour",
    escalation: "Senior developer within 1 hour",
    notification: "Slack + Email",
  },
  P3: {
    responseTime: "4 hours",
    escalation: "Developer within 4 hours",
    notification: "Slack",
  },
  P4: {
    responseTime: "24 hours",
    escalation: "Developer within 24 hours",
    notification: "GitHub issue",
    },
  };
```

2. Support Resources

Internal Documentation

- **Code Documentation:** Inline code comments and JSDoc
- **Architecture Documentation:** System design and component guides
- **API Documentation:** Endpoint specifications and examples
- **Deployment Guides:** Environment setup and deployment procedures

External Resources

- **Next.js Documentation:** https://nextjs.org/docs
- **Prisma Documentation:** https://www.prisma.io/docs
- **Clerk Documentation:** https://clerk.com/docs
- **Stripe Documentation:** https://stripe.com/docs
- **Tailwind CSS Documentation:** https://tailwindcss.com/docs

Community Support

- **GitHub Issues:** Project-specific issues and discussions
- **Stack Overflow:** General programming questions
- **Discord/Slack:** Real-time community support
- **Reddit:** Programming community discussions

📋 Troubleshooting Checklist

Pre-Troubleshooting Steps

- [ ] **Verify Environment:** Check environment variables and configuration
- [ ] **Check Logs:** Review application and error logs
- [ ] **Reproduce Issue:** Confirm issue is reproducible
- [ ] **Check Dependencies:** Verify all dependencies are up to date
- [ ] **Clear Cache:** Clear browser and application cache

Issue Resolution Steps

- [ ] **Identify Root Cause:** Determine the underlying issue
- [ ] **Implement Fix:** Apply the appropriate solution
- [ ] **Test Solution:** Verify the fix resolves the issue
- [ ] **Document Solution:** Record the solution for future reference
- [ ] **Monitor Results:** Watch for any side effects or new issues

Post-Resolution Steps

- [ ] **Update Documentation:** Reflect changes in relevant documentation
- [ ] **Communicate Changes:** Notify team of resolution
- [ ] **Prevent Recurrence:** Implement preventive measures
- [ ] **Review Process:** Evaluate troubleshooting effectiveness
- [ ] **Knowledge Sharing:** Share learnings with team

This troubleshooting guide provides comprehensive guidance for identifying, diagnosing, and resolving issues in the VINEFOX wine store application. Regular updates to this document ensure it remains current with the evolving codebase and common issues.