🚂 Railway Deployment Setup Guide

✅ Issues Fixed

This guide documents the fixes applied to resolve Railway deployment issues.

🔧 Changes Made:

1. **Fixed PORT Configuration**
   - Removed hardcoded `PORT=3000` from `.env` file
   - Railway automatically sets the `PORT` environment variable
   - Application correctly uses `process.env.PORT || 3001`

2. **Enhanced Health Check Endpoint**
   - Added database connectivity check to `/api/health`
   - Returns 200 OK even if database is temporarily unavailable
   - Prevents health check failures due to database migration delays

3. **Updated Dockerfile**
   - Health check now uses dynamic PORT from environment
   - Added clear comments about PORT handling

4. **Updated railway.toml**
   - Added documentation about automatic PORT assignment
   - Clarified Railway's PORT environment variable behavior

🔑 Required Environment Variables in Railway

You **MUST** set these environment variables in your Railway project:

1. Database Configuration

```
DATABASE_URL="postgresql://username:password@hostname:port/database?schema=public"
```

⚠️ **CRITICAL**: This should point to your Railway PostgreSQL database.

To get your DATABASE_URL:
1. Go to your Railway project
2. Click on your PostgreSQL service
3. Go to "Variables" tab
4. Copy the `DATABASE_URL` value
5. Add it to your backend service's environment variables

2. JWT Secrets

```
JWT_SECRET=your_super_secret_jwt_key_here_min_32_chars
JWT_REFRESH_SECRET=your_super_secret_refresh_jwt_key_here
JWT_EXPIRES_IN=7d
JWT_REFRESH_EXPIRES_IN=30d
```

Security Note: Generate strong random secrets for production!

```
# Generate secrets using Node.js:
node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"
```

3. Node Environment

```
NODE_ENV=production
```

4. CORS Configuration

```
CORS_ORIGIN=https://your-frontend-domain.com
```

Replace with your actual frontend URL (Railway will provide this).

5. File Upload Configuration (Optional)

```
MAX_FILE_SIZE=5242880
ALLOWED_FILE_TYPES=image/jpeg,image/png,image/jpg,application/pdf
```

6. Rate Limiting (Optional)

```
RATE_LIMIT_WINDOW_MS=900000
RATE_LIMIT_MAX_REQUESTS=100
```

📝 Step-by-Step Railway Setup

Step 1: Connect GitHub Repository

1. Go to Railway.app (https://railway.app)
2. Click “New Project”
3. Select “Deploy from GitHub repo”
4. Choose `volkanakbulut73/workigom`
5. Railway will detect the Dockerfile automatically

Step 2: Add PostgreSQL Database

1. In your project, click “+ New”
2. Select “Database” → “PostgreSQL”
3. Wait for the database to provision
4. Copy the `DATABASE_URL` from the Variables tab

Step 3: Configure Environment Variables

1. Click on your backend service (workigom)
2. Go to “Variables” tab
3. Click “+ New Variable”
4. Add all required variables from the section above

Quick Setup - Copy & Paste Template:

```
# Database - GET THIS FROM YOUR RAILWAY POSTGRESQL SERVICE
DATABASE_URL=postgresql://postgres:password@hostname:5432/railway

# JWT Configuration - GENERATE NEW SECRETS FOR PRODUCTION!
JWT_SECRET=changeme_generate_random_32_char_secret
JWT_REFRESH_SECRET=changeme_generate_random_32_char_refresh_secret
JWT_EXPIRES_IN=7d
JWT_REFRESH_EXPIRES_IN=30d

# Server Configuration
NODE_ENV=production

# CORS - UPDATE WITH YOUR FRONTEND URL
CORS_ORIGIN=https://your-frontend.railway.app

# File Upload
MAX_FILE_SIZE=5242880
ALLOWED_FILE_TYPES=image/jpeg,image/png,image/jpg,application/pdf

# Rate Limiting
RATE_LIMIT_WINDOW_MS=900000
RATE_LIMIT_MAX_REQUESTS=100
```

Step 4: Deploy

1. Railway will automatically deploy when you push to GitHub
2. Monitor the deployment logs in Railway dashboard
3. Check the deployment status

Step 5: Verify Deployment

1. Once deployed, Railway will provide a public URL
2. Test the health endpoint: `https://your-app.railway.app/api/health`
3. You should see:

```
{
  "success": true,
  "message": "Workigom API is running",
  "timestamp": "2025-10-24T...",
  "database": "connected"
}
```

🐛 Troubleshooting

Issue: “Application failed to respond”

Cause: The app isn’t starting properly.

Solutions:
1. Check Railway logs for error messages
2. Verify `DATABASE_URL` is set correctly
3. Ensure database migrations ran successfully
4. Check that all required environment variables are set

Issue: Health check failing

Cause: Health check endpoint not responding.

Solutions:
1. Verify the app is listening on `process.env.PORT`
2. Check that `/api/health` endpoint is accessible
3. Review application logs for startup errors
4. Increase `healthcheckTimeout` in railway.toml if needed

Issue: Database connection errors

Cause: Invalid `DATABASE_URL` or database not ready.

Solutions:
1. Verify `DATABASE_URL` format is correct
2. Ensure PostgreSQL service is running in Railway
3. Check database service logs
4. Wait for database to finish provisioning

Issue: “Port already in use”

Cause: This shouldn’t happen with Railway, but if testing locally:

Solutions:
1. In Railway: No action needed - Railway handles ports automatically
2. Locally: Stop other processes using the port
3. Use a different port in your `.env` file for local development

---

🎯 Port Configuration Explained

How It Works:

1. **Railway sets `PORT` automatically** - You don’t specify it
2. **Application code**: `const PORT = process.env.PORT || 3001`
3. **Local development**: Uses port 3001 (fallback)
4. **Railway deployment**: Uses Railway’s assigned port

What NOT to do:

❌ Don't hardcode `PORT` in `.env` file
❌ Don't specify port in railway.toml
❌ Don't use fixed port in your code

What TO do:

✅ Use `process.env.PORT` with a fallback
✅ Let Railway manage the port
✅ Test locally with the fallback port (3001)

📊 Deployment Checklist

Before deploying, ensure:

- [ ] All environment variables are set in Railway
- [ ] `DATABASE_URL` points to Railway PostgreSQL
- [ ] JWT secrets are strong and random (production)
- [ ] `NODE_ENV=production` is set
- [ ] `CORS_ORIGIN` is set to your frontend URL
- [ ] GitHub repository is connected to Railway
- [ ] Latest code is pushed to GitHub
- [ ] Prisma schema is up to date
- [ ] Migrations are ready to run

🚀 Deployment Commands

Railway runs these commands automatically:

1. **Build**: `npm run build` (compiles TypeScript)
2. **Migrate**: `npx prisma migrate deploy` (runs in start script)
3. **Start**: `node dist/server.js` (via start.sh)

📞 Need Help?

If you're still experiencing issues:

1. Check Railway logs: Project → Service → Deployments → View Logs
2. Verify environment variables: Service → Variables tab
3. Test health endpoint: `curl https://your-app.railway.app/api/health`
4. Review this guide: `/RAILWAY_SETUP_GUIDE.md`

🎉 Success Indicators

Your deployment is successful when you see:

✅ Green checkmark on Railway deployment
✅ Health endpoint returns 200 OK
✅ Logs show “Server running on http://localhost:[PORT]”
✅ Logs show “Database connection successful”
✅ No error messages in Railway logs

📝 Notes

- Railway automatically assigns a unique PORT for each service
- The health check endpoint is at `/api/health`
- Database migrations run automatically on startup
- The app uses the Dockerfile in the root directory
- Deployments trigger automatically on GitHub push

Last Updated: October 24, 2025
Status: Ready for Deployment ✅