Git Setup Instructions

Quick Setup for GitHub Repository

1. Initialize Git Repository (if not already done)

```
cd /home/ubuntu/ai_rfp_risk_scanner
git init
git branch -M main
```

2. Add All Files to Git

```
# Add all files (respecting .gitignore)
git add .

# Create initial commit
git commit -m "Initial commit: AI RFP Risk Scanner v2.0.0

- Comprehensive risk analysis with 40+ categories
- Regulatory compliance mapping (GDPR, NIS2, DORA, EU AI Act, AI RMF, OWASP)
- Industry best practices integration
- Enhanced LLM analysis engine
- Modern Next.js 14 application with TypeScript
- PostgreSQL database with Prisma ORM
- User authentication with NextAuth.js
- Production-ready deployment configuration"
```

3. Connect to GitHub Repository

Option A: Create New Repository on GitHub

1. Go to https://github.com/new
2. Create repository named `ai-rfp-risk-scanner`
3. Don't initialize with README (we already have one)

Option B: Use Existing Repository

Replace `YOUR_USERNAME` and `YOUR_REPO_NAME` with your actual values:

```
# Add remote origin
git remote add origin https://github.com/YOUR_USERNAME/YOUR_REPO_NAME.git

# Push to GitHub
git push -u origin main
```

4. Verify Upload

```
# Check remote status
git remote -v

# Check branch status
git branch -a

# View commit log
git log --oneline
```

Repository Structure

Your GitHub repository will contain:

```
ai-rfp-risk-scanner/
├── 📁 app/                      # Next.js application
├── 📄 README.md                # Comprehensive documentation
├── 📄 DEPLOYMENT_GUIDE.md      # Deployment instructions
├── 📄 CHANGELOG.md             # Version history
├── 📄 LICENSE                  # MIT License
├── 📄 .gitignore             # Git ignore rules
├── 📄 .env.example           # Environment template
├── 📄 package.json           # Root package configuration
└── 📄 GIT_SETUP_INSTRUCTIONS.md # This file
```

Environment Setup for New Team Members

Clone and Setup

```
# Clone repository
git clone https://github.com/YOUR_USERNAME/ai-rfp-risk-scanner.git
cd ai-rfp-risk-scanner

# Setup development environment
yarn setup

# Copy environment template
cp .env.example app/.env.local

# Edit environment variables (see README.md for details)
nano app/.env.local
```

Required Environment Variables

```
DATABASE_URL="postgresql://username:password@localhost:5432/ai_rfp_scanner"
NEXTAUTH_URL="http://localhost:3000"
NEXTAUTH_SECRET="generate-with-openssl-rand-base64-32"
ABACUSAI_API_KEY="your-abacus-ai-api-key"
```

Start Development

```
# Start development server
yarn dev

# Visit http://localhost:3000
```

Branch Strategy

Recommended Git Flow

```
# Create feature branch
git checkout -b feature/your-feature-name

# Make changes and commit
git add .
git commit -m "Add: your feature description"

# Push feature branch
git push origin feature/your-feature-name

# Create Pull Request on GitHub
# After review and merge, delete feature branch
git checkout main
git pull origin main
git branch -d feature/your-feature-name
```

Branch Naming Convention

- `feature/` - New features
- `bugfix/` - Bug fixes
- `hotfix/` - Critical production fixes
- `docs/` - Documentation updates
- `refactor/` - Code refactoring

Deployment Integration

Vercel (Recommended)

1. Connect GitHub repository to Vercel
2. Set environment variables in Vercel dashboard
3. Deploy automatically on push to main

Other Platforms

- **Netlify**: Connect repository and configure build settings
- **Railway**: Connect GitHub for automatic deployments
- **DigitalOcean App Platform**: Connect repository with build configuration

Security Notes

Never Commit

- ❌ `.env` files with real secrets

- ❌ `node_modules/` directories
- ❌ Build artifacts
- ❌ Database files
- ❌ API keys in code
- ❌ Personal uploads

Always Use

- ✅ Environment variables for secrets
- ✅ `.env.example` for templates
- ✅ `.gitignore` for sensitive files
- ✅ Secure database connections
- ✅ HTTPS in production

Collaboration Workflow

Code Review Process

1. Create feature branch
2. Implement changes with tests
3. Create Pull Request
4. Request review from team members
5. Address feedback
6. Merge after approval

Issue Tracking

- Use GitHub Issues for bug reports
- Create issue templates for consistency
- Label issues by priority and type
- Link PRs to related issues

Documentation Updates

- Update README.md for major changes
- Update CHANGELOG.md for all releases
- Update API documentation as needed
- Keep deployment guides current

Maintenance Commands

Update Dependencies

```
cd app
yarn upgrade-interactive
```

Database Maintenance

```
# Create migration
npx prisma migrate dev --name your_migration_name

# Reset database (development only)
npx prisma migrate reset

# Generate Prisma client
npx prisma generate
```

Code Quality

```
# Lint code
yarn lint

# Type check
yarn tsc --noEmit

# Format code
yarn prettier --write .
```

Troubleshooting

Common Git Issues

Large File Upload

If you encounter large file errors:

```
# Remove large files from history
git filter-branch --force --index-filter 'git rm --cached --ignore-unmatch
PATH_TO_LARGE_FILE' --prune-empty --tag-name-filter cat -- --all

# Push force (dangerous - use carefully)
git push origin --force --all
```

Authentication Issues

```
# Use personal access token instead of password
git config --global credential.helper store

# Or use SSH keys (recommended)
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
# Add public key to GitHub settings
```

Merge Conflicts

```
# Pull latest changes
git pull origin main

# Resolve conflicts in editor
# Then commit resolution
git add .
git commit -m "Resolve merge conflicts"
```

Support

For issues with Git setup or repository management:
1. Check GitHub documentation
2. Review this guide
3. Create issue in repository
4. Contact team lead

Happy coding! 🚀