

PROMPT

Jan 19, 2026, 10:52 AM

Write a step-by-step tutorial for Nuxt aimed at beginner developers for connecting their frontend to Neon. Include prerequisites, clear learning objectives, hands-on exercises, and troubleshooting tips.

Response from Claude Sonnet 4

Complete Guide: Connecting Nuxt to Neon Database for Beginners

Learning Objectives

By the end of this tutorial, you will be able to:

- Set up a new Nuxt 3 project from scratch
- Create and configure a Neon PostgreSQL database
- Connect your Nuxt frontend to Neon database
- Perform basic CRUD operations (Create, Read, Update, Delete)
- Handle database queries using Prisma ORM
- Implement error handling and troubleshoot common issues

Prerequisites

Before starting this tutorial, ensure you have:

- Node.js (version 16.10 or later) installed on your machine
- npm or yarn package manager
- Basic knowledge of JavaScript and Vue.js
- A GitHub account (recommended for deployment)
- A Neon account (free tier available)

Checking Your Prerequisites

bash

```
# Check Node.js version
node --version
# Should output: v16.10.0 or higher

# Check npm version
npm --version
# Should output: 7.0.0 or higher
```

Step 1: Setting Up Your Nuxt Project

Create a New Nuxt Project

bash

```
# Create a new Nuxt 3 project
npx nuxi@latest init nuxt-neon-tutorial
cd nuxt-neon-tutorial

# Install dependencies
npm install
```

Verify Installation

bash

```
# Start the development server
npm run dev
```

Expected Output:

```
Nuxt 3.8.0 with Nitro 2.7.0
Local:   http://localhost:3000/
Network: use --host to expose
```

Visit <http://localhost:3000> in your browser. You should see the default Nuxt welcome page.

Step 2: Setting Up Neon Database

Create Your Neon Account and Database

1. Go to neon.tech and sign up for a free account
2. Click "Create a project"
3. Choose your region (select the one closest to your users)
4. Give your project a name: `nuxt-tutorial-db`
5. Click "Create project"

Get Your Connection String

After creating your project, you'll see a connection string like:

```
postgresql://username:password@host/dbname?sslmode=require
```

Important: Copy this connection string - you'll need it in the next step.

Step 3: Installing Required Dependencies

Install Database and ORM Dependencies

bash

```
# Install Prisma (our ORM)
npm install prisma @prisma/client

# Install additional dependencies for environment variables
npm install @nuxtjs/google-fonts dotenv
```

Initialize Prisma

```
# Initialize Prisma in your project
npx prisma init
```

bash

Expected Output:

```
Your Prisma schema was created at prisma/schema.prisma
You can now open it in your favorite editor.

Next steps:
1. Set the DATABASE_URL in the .env file to point to your existing database
2. Set the provider of the "db" datasource in your schema.prisma file to match your database
3. Run prisma db pull to turn your database schema into a Prisma schema
4. Run prisma generate to generate the Prisma Client
```

Step 4: Configure Environment Variables

Create Environment File

Create a .env file in your project root:

```
# Create .env file
touch .env
```

bash

Add Your Neon Connection String

Add the following to your .env file:

```
# Replace with your actual Neon connection string
DATABASE_URL="postgresql://username:password@host/dbname?sslmode=require"
```

env

Important: Never commit your .env file to version control. It should already be in .gitignore.

Step 5: Define Your Database Schema

Update Prisma Schema

Edit prisma/schema.prisma:

prisma

```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id          Int      @id @default(autoincrement())
  email       String   @unique
  name        String?
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  posts       Post[]
}

model Post {
  id          Int      @id @default(autoincrement())
  title       String
  content     String?
  published   Boolean  @default(false)
  author      User     @relation(fields: [authorId], references: [id])
  authorId    Int
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
}
```

Push Schema to Database

bash

```
# Push the schema to your Neon database
npx prisma db push
```

Expected Output:

```
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Datasource "db": PostgreSQL database "neondb", schema "public" at "xxx.neon.tech:5432"

=€Your database is now in sync with your Prisma schema. Done in 2.1s
```

Generate Prisma Client

bash

```
# Generate the Prisma client
npx prisma generate
```

Expected Output:

```
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma

Generated Prisma Client (4.6.1 | library) to ./node_modules/@prisma/client in 89ms
You can now start using Prisma Client in your code. Reference: https://pris.ly/d/client
```

Step 6: Create Database Service

Create Prisma Service

Create lib/prisma.js:

```
# Create lib directory
mkdir lib
touch lib/prisma.js
```

bash

Add the following code to lib/prisma.js:

```
import { PrismaClient } from '@prisma/client'

let prisma

if (process.env.NODE_ENV === 'production') {
  prisma = new PrismaClient()
} else {
  if (!global.__prisma) {
    global.__prisma = new PrismaClient()
  }
  prisma = global.__prisma
}

export default prisma
```

javascript

Step 7: Create API Routes

Create Server API Directory

```
# Create server API directories
mkdir -p server/api/users
mkdir -p server/api/posts
```

bash

Create User API Routes

Create server/api/users/index.get.js:

javascript

```
import prisma from '~/lib/prisma'

export default defineEventHandler(async (event) => {
  try {
    const users = await prisma.user.findMany({
      include: {
        posts: true
      }
    })
    return users
  } catch (error) {
    throw createError({
      statusCode: 500,
      statusMessage: 'Failed to fetch users'
    })
  }
})
```

Create server/api/users/index.post.js:

javascript

```
import prisma from '~/lib/prisma'

export default defineEventHandler(async (event) => {
  try {
    const body = await readBody(event)

    const user = await prisma.user.create({
      data: {
        name: body.name,
        email: body.email
      }
    })

    return user
  } catch (error) {
    if (error.code === 'P2002') {
      throw createError({
        statusCode: 400,
        statusMessage: 'Email already exists'
      })
    }

    throw createError({
      statusCode: 500,
      statusMessage: 'Failed to create user'
    })
  }
})
```

Create Post API Routes

Create server/api/posts/index.get.js:

javascript

```
import prisma from '~/lib/prisma'

export default defineEventHandler(async (event) => {
  try {
    const posts = await prisma.post.findMany({
      include: {
        author: {
          select: {
            id: true,
            name: true,
            email: true
          }
        }
      },
      orderBy: {
        createdAt: 'desc'
      }
    })
    return posts
  } catch (error) {
    throw createError({
      statusCode: 500,
      statusMessage: 'Failed to fetch posts'
    })
  }
})
```

Create server/api/posts/index.post.js:

javascript

```
import prisma from '~/lib/prisma'

export default defineEventHandler(async (event) => {
  try {
    const body = await readBody(event)

    const post = await prisma.post.create({
      data: {
        title: body.title,
        content: body.content,
        authorId: parseInt(body.authorId)
      },
      include: {
        author: true
      }
    })

    return post
  } catch (error) {
    throw createError({
      statusCode: 500,
      statusMessage: 'Failed to create post'
    })
  }
})
```

Step 8: Create Frontend Components

Update Main Layout

Replace the content of app.vue:

```
<template>
<div>
<header class="bg-blue-600 text-white p-4">
<div class="container mx-auto">
<h1 class="text-2xl font-bold">Nuxt + Neon Tutorial</h1>
</div>
</header>

<main class="container mx-auto p-4">
<NuxtPage />
</main>
</div>
</template>

<style>
body {
  font-family: 'Inter', sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f5f5f5;
}

.container {
  max-width: 1200px;
}
</style>
```

Create User Management Page

Create pages/index.vue:

```
<template>
<div class="space-y-8">
<div class="bg-white p-6 rounded-lg shadow">
<h2 class="text-xl font-semibold mb-4">Create New User</h2>

<form @submit.prevent="createUser" class="space-y-4">
<div>
<label class="block text-sm font-medium mb-1">Name:</label>
<input
  v-model="newUser.name"
  type="text"
  required
  class="w-full p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:border-blue-500"
  placeholder="Enter user name"
>
</div>

<div>
<label class="block text-sm font-medium mb-1">Email:</label>
<input
  v-model="newUser.email"
  type="email"
  required
  class="w-full p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:border-blue-500"
  placeholder="Enter email address"
>
</div>
</form>
</div>
</div>
```



```
const user = await $fetch('/api/users', {
  method: 'POST',
  body: newUser.value
})

// Add new user to the list
users.value.push(user)

// Reset form
newUser.value = { name: '', email: '' }

// Success message
message.value = 'User created successfully!'
messageClass.value = 'bg-green-100 text-green-800 border border-green-200'

// Clear message after 3 seconds
setTimeout(() => {
  message.value = ''
}, 3000)

} catch (error) {
  message.value = error.data?.message || 'Failed to create user'
  messageClass.value = 'bg-red-100 text-red-800 border border-red-200'
}

loading.value = false
}

// Set page metadata
useHead({
  title: 'Nuxt + Neon Tutorial - Users'
})
</script>
```

Create Posts Page

Create pages/posts.vue:

vue

```
<template>
<div class="space-y-8">
<div class="bg-white p-6 rounded-lg shadow">
<h2 class="text-xl font-semibold mb-4">Create New Post</h2>

<form @submit.prevent="createPost" class="space-y-4">
<div>
<label class="block text-sm font-medium mb-1">Author:</label>
<select
  v-model="newPost.authorId"
  required
  class="w-full p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:border-blue-500"
>
<option value="">Select an author</option>
<option v-for="user in users" :key="user.id" :value="user.id">
  {{ user.name }} ({{ user.email }})
</option>
</select>
</div>

<div>
<label class="block text-sm font-medium mb-1">Title:</label>
<input
```

```
v-model="newPost.title"
type="text"
required
class="w-full p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:border-blue-500"
placeholder="Enter post title"
>
</div>

<div>
<label class="block text-sm font-medium mb-1">Content:</label>
<textarea
  v-model="newPost.content"
  rows="4"
  class="w-full p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:border-blue-500"
  placeholder="Enter post content"
></textarea>
</div>

<button
  type="submit"
  :disabled="loading"
  class="bg-blue-600 text-white px-4 py-2 rounded hover:bg-blue-700 disabled:opacity-50"
  >
  {{ loading ? 'Creating...' : 'Create Post' }}
</button>
</form>

<div v-if="message" :class="messageClass" class="mt-4 p-3 rounded">
  {{ message }}
</div>
</div>

<div class="bg-white p-6 rounded-lg shadow">
<h2 class="text-xl font-semibold mb-4">Posts</h2>

<div v-if="posts.length === 0" class="text-gray-500">
  No posts found. Create your first post above!
</div>

<div v-else class="space-y-4">
<article
  v-for="post in posts"
  :key="post.id"
  class="border p-6 rounded hover:bg-gray-50"
  >
<h3 class="text-lg font-semibold mb-2">{{ post.title }}</h3>
<p class="text-gray-700 mb-3">{{ post.content }}</p>
<div class="text-sm text-gray-500">
<p>By: {{ post.author.name }} ({{ post.author.email }})</p>
<p>Created: {{ new Date(post.createdAt).toLocaleDateString() }}</p>
</div>
</article>
</div>
</div>
</div>
</div>
</template>

<script setup>
// Reactive data
const posts = ref([])
const users = ref([])
const newPost = ref({
  title: '',
  content: '',
  authorId: ''
```

```
})
const loading = ref(false)
const message = ref('')
const messageClass = ref('')

// Fetch initial data
const [postsData, usersData] = await Promise.all([
  $fetch('/api/posts'),
  $fetch('/api/users')
])

posts.value = postsData || []
users.value = usersData || []

// Create new post
const createPost = async () => {
  loading.value = true
  message.value = ''

  try {
    const post = await $fetch('/api/posts', {
      method: 'POST',
      body: newPost.value
    })
  }

  // Add new post to the beginning of the list
  posts.value.unshift(post)

  // Reset form
  newPost.value = { title: '', content: '', authorId: '' }

  // Success message
  message.value = 'Post created successfully!'
  messageClass.value = 'bg-green-100 text-green-800 border border-green-200'

  // Clear message after 3 seconds
  setTimeout(() => {
    message.value = ''
  }, 3000)

  } catch (error) {
    message.value = error.data?.message || 'Failed to create post'
    messageClass.value = 'bg-red-100 text-red-800 border border-red-200'
  }

  loading.value = false
}

// Set page metadata
useHead({
  title: 'Nuxt + Neon Tutorial - Posts'
})
</script>
```

Add Navigation

Update app.vue to include navigation:

vue

```
<template>
<div>
<header class="bg-blue-600 text-white p-4">
```

```
<div class="container mx-auto flex justify-between items-center">
<h1 class="text-2xl font-bold">Nuxt + Neon Tutorial</h1>
<nav class="space-x-4">
<NuxtLink to="/" class="hover:underline">Users</NuxtLink>
<NuxtLink to="/posts" class="hover:underline">Posts</NuxtLink>
</nav>
</div>
</header>

<main class="container mx-auto p-4">
<NuxtPage />
</main>
</div>
</template>

<style>
body {
  font-family: 'Inter', sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f5f5f5;
}

.container {
  max-width: 1200px;
}
</style>
```

Step 9: Add CSS Framework (Optional)

Install Tailwind CSS

```
# Install Tailwind CSS
npm install --save-dev @nuxtjs/tailwindcss

# Update nuxt.config.ts
```

bash

Update nuxt.config.ts:

```
// https://nuxt.com/docs/api/configuration/nuxt-config
export default defineNuxtConfig({
  devtools: { enabled: true },
  modules: ['@nuxtjs/tailwindcss']
})
```

typescript

Step 10: Testing Your Application

Start the Development Server

```
npm run dev
```

bash

Test User Creation

1. Navigate to `http://localhost:3000`
2. Fill out the user creation form
3. Click "Create User"

Expected Result:

```
User created successfully!
```

The new user should appear in the users list below.

Test Post Creation

1. Navigate to `http://localhost:3000/posts`
2. Select a user from the dropdown
3. Enter a title and content
4. Click "Create Post"

Expected Result:

```
Post created successfully!
```

The new post should appear in the posts list below.

Troubleshooting

Common Issues and Solutions

Issue 1: "Environment variable not loaded"

Error Message:

```
Error: Environment variable DATABASE_URL is not set
```

Solution:

1. Ensure `.env` file exists in project root
2. Check that `DATABASE_URL` is correctly set
3. Restart the development server

```
# Stop the server (Ctrl+C) and restart  
npm run dev
```

bash

Issue 2: "Database connection failed"

Error Message:

```
Can't reach database server at `xxx.neon.tech`:`5432`
```

Solution:

1. Check your internet connection
2. Verify the connection string is correct
3. Ensure your Neon database is active

```
# Test the connection  
npx prisma db push
```

bash

Issue 3: "Prisma Client not generated"

Error Message:

```
Cannot find module '@prisma/client'
```

Solution:

```
# Reinstall and regenerate Prisma client  
npm install @prisma/client  
npx prisma generate
```

bash

Issue 4: "Table doesn't exist"

Error Message:

```
Table 'public.User' doesn't exist in the current database
```

Solution:

```
# Push your schema to the database  
npx prisma db push  
  
# If that doesn't work, reset the database  
npx prisma db push --force-reset
```

bash

Issue 5: "CORS or API route not found"

Error Message:

```
404 - Page not found
```

Solution:

1. Ensure API files are in the correct directory structure:

...

server/

 api/

 users/

 index.get.js

 index.post.js

 posts/

 index.get.js

 index.post.js

...

2. Restart the development server

Issue 6: "Hydration mismatch"

Error Message:

```
Hydration completed but contains mismatches
```

Solution:

This usually happens with SSR. Wrap client-only code:

```
<template>
  <ClientOnly>
  <YourComponent />
</ClientOnly>
</template>
```

vue

Debug Mode

Enable debug mode to get more detailed error messages:

```
# Run with debug mode
DEBUG=* npm run dev
```

bash

Database Inspection

Use Prisma Studio to inspect your database:

bash

```
# Open Prisma Studio  
npx prisma studio
```

This opens a web interface at <http://localhost:5555> where you can view and edit your data.

Next Steps and Best Practices

Security Best Practices

1. Environment Variables: Never commit `.env` files
2. Input Validation: Add proper validation to your API routes
3. Rate Limiting: Implement rate limiting for production

Performance Optimization

1. Connection Pooling: Configure Prisma connection pooling
2. Caching: Implement caching strategies
3. Indexing: Add database indexes for frequently queried fields

Additional Features to Explore

- Authentication: Add user authentication with Auth0 or NextAuth
- Real-time Updates: Implement WebSocket connections
- File Uploads: Add image upload functionality
- Search: Implement full-text search
- Pagination: Add pagination for large datasets

Production Deployment

When ready to deploy:

1. Environment Setup: Configure production environment variables
2. Database Migration: Use `prisma migrate` instead of `db push`
3. Error Handling: Implement comprehensive error logging
4. Monitoring: Add application monitoring

Conclusion

Congratulations! You've successfully:

- Created a Nuxt 3 application
- Set up a Neon PostgreSQL database
- Connected your frontend to the database using Prisma
- Implemented CRUD operations
- Built a functional user interface
- Learned troubleshooting techniques

Your application now has a solid foundation for building more complex features. The combination of Nuxt 3's powerful frontend capabilities and Neon's serverless PostgreSQL provides an excellent foundation for scalable web applications.

Project Structure Summary:

```
nuxt-neon-tutorial/  
  lib/  
    prisma.js          # Database connection  
  pages/  
    index.vue         # Users management  
    posts.vue        # Posts management  
  prisma/  
    schema.prisma     # Database schema  
  server/  
    api/              # API routes  
  .env                # Environment variables  
  app.vue             # Main application layout
```

Happy coding! =€