Simple Quasar (Vue) ⇄ Django REST Framework (DRF)

Goal: Build a *very simple* app so you can clearly see how the Quasar (Vue) frontend talks to a DRF backend. We'll create one backend endpoint (`GET /api/hello/`) and a Quasar page with a button that calls it and shows the response.

This guide is written for **complete beginners**. Every code block has comments that explain *why* each line is needed.

Architecture diagram

Mermaid (if your viewer supports it):

```
graph LR
  Browser[User's browser]
  QuasarDev[Quasar Dev Server (localhost:9000)]
  QuasarApp[Quasar App - Axios boot]
  Proxy[/api → proxy]
  Django[DRF Backend (localhost:8000)]
  DB[(SQLite / DB)]

  Browser --> QuasarDev
  QuasarDev --> QuasarApp
  QuasarApp --> Proxy
  Proxy --> Django
  Django --> DB
```

ASCII fallback:

```
Browser (http://localhost:9000)
     |
     |  (click button -> axios GET /api/hello/)
     V
Quasar Dev Server (dev proxy)
     |
     |  (proxies /api/* requests to)
     V
DRF Backend (http://127.0.0.1:8000)
     |
    V
Database (SQLite during dev)
```

Short explanation of the flow: - In development, Quasar's dev server acts as a *proxy* for `/api` requests so the browser does not hit CORS issues. When front-end code `GET /api/hello/` the dev server forwards it to `http://127.0.0.1:8000/api/hello/` (the Django server). - Axios (running inside the browser) makes the call; Quasar dev server proxies it; Django responds.

Prerequisites

- Node.js + npm (or yarn)
- Python 3.8+ and `pip`
- (Optional) Quasar CLI — we will use `npm create quasar@latest` which works without globally installing the CLI.

STEP 1 — Backend: minimal DRF app

Folder: `backend`

1. Create folder and virtual environment, install packages:

```
mkdir backend && cd backend
python -m venv venv
# macOS / Linux
source venv/bin/activate
# Windows (PowerShell)
# .\venv\Scripts\Activate.ps1

pip install django djangorestframework django-cors-headers
```

> **Why?** `venv` isolates Python packages. `djangorestframework` provides easy API tools. `django-cors-headers` helps if you decide to enable CORS instead of using a proxy.

1. Start project + app:

```
django-admin startproject config .
python manage.py startapp api
python manage.py migrate
```

1. Edit `config/settings.py` — add installed apps and middleware (explain in-file comments):

```python
# config/settings.py (only the parts you need to add/change)

INSTALLED_APPS += [
    'rest_framework',       # enables DRF features
    'corsheaders',          # optional; helpful if you want to enable CORS
    'api',                  # our simple api app
]

MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',  # must be near the top
    'django.middleware.common.CommonMiddleware',
    # ... keep the default middleware below
]

# For learning/demo: allow all origins (WARNING: don't use in production)
CORS_ALLOW_ALL_ORIGINS = True

# Optional: rest framework defaults (not strictly required for this demo)
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny',
    ]
}
```

> **Why CORS middleware?** If you choose to call Django directly from the browser (`http://127.0.0.1:8000`), the browser will block cross-origin requests unless the server allows them. We will use Quasar's proxy (recommended) but adding the CORS middleware is useful to know.

1. Create a very small API view to return a JSON message.

`api/views.py`

```python
from rest_framework.decorators import api_view
from rest_framework.response import Response

@api_view(['GET'])
def hello(request):
    # This view returns a small JSON object. Great for testing connectivity.
    return Response({'message': 'Hello from Django!'})
```

`api/urls.py`

```
from django.urls import path
from .views import hello

urlpatterns = [
    path('hello/', hello),  # reachable at /api/hello/
]
```

Add the `api` urlpatterns to the project `config/urls.py`:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('api.urls')),  # all api/ urls go to our app
]
```

1. Run the backend server:

```
python manage.py runserver 8000
```

Open `http://127.0.0.1:8000/api/hello/` in the browser — you should see:

```
{"message": "Hello from Django!"}
```

STEP 2 — Frontend: Quasar project (minimal)

Folder: `quasar-frontend` (or any name you like)

1. Create a Quasar project (quick):

```
# from parent folder
npm create quasar@latest quasar-frontend
cd quasar-frontend
npm install
```

Choose the default options (Vite, Vue 3) when prompted. These choices are a good modern default.

1. We will use Quasar's dev proxy so the browser never sees a different origin and you won't get CORS errors.

Create an `.env` **file** in your `quasar-frontend` root (this is optional but recommended to store backend url for later):

```
# quasar-frontend/.env
DEV_API=http://127.0.0.1:8000
```

Why `.env`**?** It keeps environment-specific values (like backend URL) separate from source code. Later, for production you change the URL without editing code.

1. Edit `quasar.config.js` to add a proxy in the `devServer` section. This file runs in Node when the dev server starts, so we can `require('dotenv')` to load our `.env` value.

```
/* quasar.config.js */
/* eslint-env node */
require('dotenv').config() // loads .env into process.env
const { configure } = require('quasar/wrappers')

module.exports = configure(function () {
  return {
    // ... other quasar config ...

    devServer: {
      proxy: {
        // any request that starts with /api will be forwarded
        '/api': {
          // use the DEV_API from .env if present, otherwise fallback
          target: process.env.DEV_API || 'http://127.0.0.1:8000',
          changeOrigin: true,
          secure: false
        }
      }
    },

    // register boot files below (we will add axios boot next)
    boot: [
      'axios'
    ],
  }
})
```

Why proxy here? Browsers enforce same-origin policy. When you call `fetch('/api/hello/')` from `http://localhost:9000`, the dev server forwards that request to your Django server (so the browser thinks it's the same origin). This is easiest for development.

1. Create an Axios boot file so the `axios` instance is available app-wide.

Create `src/boot/axios.js`:

```
// src/boot/axios.js
import { boot } from 'quasar/wrappers'
import axios from 'axios'

// baseURL '/api' -> the Quasar dev server will proxy this to Django
const api = axios.create({ baseURL: '/api', timeout: 10000 })

export default boot(({ app }) => {
  // make it available in Vue components as this.$api (Options API)
  app.config.globalProperties.$api = api
})

export { api }
```

Why a boot file? Quasar boots allow you to register app-wide plugins or helpers (like axios). It keeps your code DRY and makes `api` available everywhere.

1. Add a small page that calls the backend. Replace or edit `src/pages/IndexPage.vue` with this:

```
<template>
  <q-page padding>
    <q-btn label="Call Backend" @click="callApi" />

    <div style="margin-top: 12px;">
      <strong v-if="loading">Loading...</strong>
      <div v-if="message">Response: {{ message }}</div>
      <div v-if="error" style="color: red">Error: {{ error }}</div>
    </div>
  </q-page>
</template>

<script setup>
import { ref } from 'vue'
import { api } from 'boot/axios' // our axios instance

const message = ref('')
const error = ref(null)
const loading = ref(false)

async function callApi () {
  loading.value = true
  error.value = null
  try {
    // Because baseURL is '/api', this will call '/api/hello/'
    const res = await api.get('/hello/')
    message.value = res.data.message
  }
  catch (err) {
    console.error(err)
    error.value = err?.response?.data || err.message
  }
  finally {
    loading.value = false
  }
}
</script>
```

Why `api.get('/hello/')`? The axios instance already points at `/api`. With the dev proxy configured, Quasar will forward `/api/hello/` to `http://127.0.0.1:8000/api/hello/`.

1. Start both servers and test:

Backend (in its folder):

```
# activate venv then
python manage.py runserver 8000
```

Frontend (in quasar-frontend folder):

```
npm run dev
```

Open `http://localhost:9000` (Quasar's default dev address). Click **Call Backend** — you should see `Response: Hello from Django!`.

Quick commands cheat-sheet (copy/paste)

Backend

```
# backend folder
python -m venv venv
source venv/bin/activate   # or Windows activate script
pip install django djangorestframework django-cors-headers
django-admin startproject config .
python manage.py startapp api
# add views/urls as shown earlier
```

```
python manage.py migrate
python manage.py runserver 8000
```

Frontend

```
# parent folder
npm create quasar@latest quasar-frontend
cd quasar-frontend
npm install
# add boot file and quasar.config.js proxy edits
npm run dev
```

Common beginner problems & fixes

- **Blank / CORS blocked in browser console**: If you see a CORS error, either:
- Make sure Quasar dev server proxy is running (and you called `/api/...` not the full backend URL), or

- Enable CORS on Django temporarily by `CORS_ALLOW_ALL_ORIGINS = True` (in `settings.py`) and restart Django.

- **404 for `/api/hello/`**: Confirm Django is running on port 8000 and that `api/urls.py` is included in `config/urls.py`.

- **Dev proxy not forwarding**: Re-start Quasar dev server after editing `quasar.config.js` (changes require restart).

- **Using a real backend URL**: In production you won't use the dev proxy. Instead build the Quasar app (`quasar build`) and serve it from a server; set `axios` baseURL to the real API (via `import.meta.env.VITE_API_BASE` or similar runtime env variable).

Small next steps (if you want to continue)

- Replace the simple `/hello/` view with a `POST /api/echo/` to send data in the request body and show it in Quasar.
- Add authentication (SimpleJWT) and a Pinia store for tokens.
- Show how to set `httpOnly` cookies from Django for secure auth flows.

If you'd like, I can now: - generate the exact files (`api/views.py`, `api/urls.py`, `config/urls.py`, `src/boot/axios.js`, `src/pages/IndexPage.vue`, `quasar.config.js`) so you can copy/paste them, **or** - walk you through running the commands step-by-step while you do them and I help debug.

Tell me which one you prefer and I'll produce the files (ready-to-copy) right away.