

Server Setup

- Servers will serve web pages
- Servers will serve services
- Different Teams will use different server setups

Option: Different Domains

- Web Page Servers on one domain
 - **http://example.com**
- Services Servers on different domain
 - **http://api.example.com**

Most common when many clients will use the services

- Allows new domains for clients to be created
- Frontend and service teams very distinct
 - Deployments and configs also distinct

Option: Same Domain (co-hosted)

- Web Page Servers on same domain as services
 - <http://example.com/>
 - <http://example.com/api/some-service>

Most common when same team is developing both

Option: Forwarding Proxy

- Internally, different domains
 - <http://pages.example.com/>
 - <http://services.example.com/>
- Externally, one domain
 - <http://example.com/>
 - <http://example.com/api/some-service>

Most common when clients are outside company

- Allows MANY internal servers
- Keeps things simple for outside clients
 - Even when things change internally

What is a "proxy"?

Proxy - A server that forwards requests/responses

If you send requests to a proxy

- You will get responses back
- Just as if you are talking to the actual server
- Proxy passes requests/responses between you

How are Proxies used in WebDev?

Proxies are used in MANY places

- **Load balancing proxies** spread server load
- **Forwarding proxy** for servers hides internal config
- **Debugging proxies** let you inspect/alter traffic
- Concept is used many places

We will use a development proxy to mix our Vite Dev server with our express services server

Client Code / Services Code

Many (Most) teams will have distinct code packages

- Client Apps (possibly many)
- Services Apps (possibly many)

This makes sense

- Often developed by different teams
- Usually distinct deployment times
- Decoupled

But we will be different

We will deploy in a single app

- Both services and client in a single package
 - One `package.json`
 - One server

Why?

- Easier to test/grade
- Complexity tests your understanding
- Forces practice with proxies
 - Often used in development
- Can always revert to separate packages

Is a single app a good idea?

- Not normally
- Is fine for a single team deploying a coupled app
 - But that's usually not the desire

You can easily separate outside of course

- Doing so is a good discussion topic in interviews
- You can discuss why and how!

Our Development

In Production: Single Express Server

- **<http://example.com/>**
- **<http://example.com/api/some-service>**

In Development: Different Servers!

- Vite Dev Server
 - **<http://localhost:5173/>**
- Express Services Server
 - **<http://localhost:3000/api/some-service>**

Setup

- Create new package using Vite
 - `npm create vite MY-APP -- --template react`
 - **Please** don't call your app "MY-APP"
- Enter new folder
 - `cd MY-APP`
- Install any server-side libraries
 - `npm install express`
 - May also involve `cookie-parser`, `uuid`, etc

Reminder: We are doing a single app

In most situations there would be separate folders and separate `package.json` files for the client app and the services server

- Often there would be multiple client packages
- And/or multiple services packages
- You should already understand how to set them up separately

Very Basic Server

server.js

```
const express = require('express');
const app = express();
const PORT = 3000;

app.get('/api/simple', (req, res) => {
  app.json({test: 'successful'});
});

app.listen(
  PORT,
  () => console.log(`http://localhost:${PORT}`)
);
```

Linting Errors/Warnings!

Your editor is likely reporting problems!

- Ex: "require is not defined"

Vite installed a `.eslintrc.cjs` file

- We previously altered this for "prop-types" errors

Vite assumes

- Code for browsers (not for node)
- Code using import/export (not require)

Cleaning up Linting

Linting is valuable info!

- Better to properly configure than silence
- Not all linting is equal!
 - Linting detects code that doesn't match rules
 - "rules" are be subjective
 - Some are near globally agreed
 - Others are team opinion
 - Some options are contradictory
 - Team decides what is "best"
- These changes are for doing both client/server

Changes to Cleanup Warnings

package.json

- remove the line `"type": "module",`

.eslintrc.cjs

```
module.exports = {  
  env: { browser: true, es2020: true, node: true },  
  /* leave other parts as they are */  
  parserOptions: {  
    ecmaVersion: 'latest',  
    allowImportExportEverywhere: true,  
    sourceType: 'module',  
  },  
}
```


Very Basic Client

App.jsx

```
function App() {  
  return (  
    <>  
      <button  
        onClick={ () => {  
          fetch('/api/simple');  
        }}  
      >Test</button>  
    </>  
  )  
}  
  
export default App;
```

You will see more warnings if you didn't change `.eslintrc.cjs`!

React doesn't change service calls!

- `fetch()` still works
- Our service call wrapper functions will still work
 - We can copy our `services.js` unchanged!
- More on this later

Here we are just calling `fetch()` on click

- To test that it works

First Test

- Start both servers
 - Different terminals
- Load Dev Server Page and Click Button
 - Check console to see service call
 - You did set **Log XMLHttpRequests** at start of course, right?

Call succeeded; everything is fine, right?

No error messages, everything seems fine

- But that can't be right?!

Request was for `http://localhost:5173/api/simple`

- The Vite Dev Server
- Not our express services server
 - Should be `http://localhost:3000/api/simple`

More Confusion

Use the Network tab in DevTools

- Examine the Response of the call

This is the HTML for the web page!

- Compare to what you get from manual visit
- **<http://localhost:3000/api/simple>**

Dev Server Configured to Always Give Home Page

- Dev Server is for Single Page Applications
- Most requests will return same HTML/CSS/JS!
 - This empowers **routing libraries**
 - Will discuss later
 - Important Note: Our express server would require additional config to do the same
- This explains lack of error
 - But how do we request the actual service?

We want to proxy the request

- We COULD change to
 - `fetch('http://localhost:3000/api/simple')`
 - But that would break on actual deploy
 - When port would not be 3000
 - And domain not "localhost"
 - Bad to require untested changes on deploy
- Instead we will configure Vite Dev Server
 - to proxy requests to express server

Flow of our Proxy

- Browser requests `/api/simple` from Dev Server
- Dev Server sends request to Express Server
- Express Server responds to Dev Server
- Dev Server responds to Browser

Browser does not know Express Server exists

- Browser only talks to Dev Server

Express Server does not know about Browser

- Express Server only talks to Dev Server

Configuring the Proxy

vite.config.js

```
export default defineConfig({
  plugins: [react()],
  server: {
    proxy: {
      '/api': {
        target: 'http://localhost:3000',
      },
    },
  },
});
```

Test Two

Vite Server automatically reloads after vite.config.js

Click Button and confirm Response in DevTools

- Shows Service result!
- Browser talked to port 5173!
- Browser does not know about Express Server!
 - But got results from Express Server anyway
- Requests starting with `/api` are **proxied**

What about Production?

- Proxying works for us in Development
- What about our final production build?
- Use `npm run build`
 - Creates front end in `dist/`
- We want server to serve those files
 - Add a static document root to `server.js`
 - `app.use(express.static('./dist'));`
 - NOT `./public!`

Visit Production Page

- <http://localhost:3000/>
 - Not yet a real domain and port
 - But these are same files
- Page displays correctly
- Service call functions
- No Vite involved!
- No Proxy involved!
 - Static pages on same origin as services

Final Tweak

Let's add a script to `scripts` in `package.json`

- So that `npm start` will run `node server.js`

Development Flow: (visit **<http://localhost:5173>**)

- `npm start`
- `npm run dev` (separate terminal)

Production Flow: (visit **<http://localhost:3000>**)

- `npm run build`
- `npm start`

Summary - Server Setup

Teams will have different server setups

- Services/Clients on different domains
- Services/Clients on same domain
 - May LOOK like this from outside

We will do same domain for course

- But we still solve different during dev

Summary - Proxies

proxy - A server that forwards requests/responses

- Used for many purposes in web dev
 - Load balancing
 - Disguising complexity
 - Debugging
 - Allowing dev servers during development

Summary - Setup Process for this course

- Use Vite to create package
- Configure `.eslistrc.cjs`
 - To allow node and browser code
- Configure `vite.config.js`
 - To add proxy settings
- Adjust scripts in `package.json`
 - Such as `start` to run `server.js`