

# Proxying API Requests in Development

Note: this feature is available with `react-scripts@0.2.3` and higher.

People often serve the front-end React app from the same host and port as their backend implementation.

For example, a production setup might look like this after the app is deployed:

|                         |  |
|-------------------------|--|
| <code>/</code>          | - static server returns <code>index.html</code> with React app                     |
| <code>/todos</code>     | - static server returns <code>index.html</code> with React app                     |
| <code>/api/todos</code> | - server handles any <code>/api/*</code> requests using the backend implementation |

Such setup is **not** required. However, if you **do** have a setup like this, it is convenient to write requests like `fetch('/api/todos')` without worrying about redirecting them to another host or port during development.

To tell the development server to proxy any unknown requests to your API server in development, add a `proxy` field to your `package.json`, for example:

```
"proxy": "http://localhost:4000",
```

This way, when you `fetch('/api/todos')` in development, the development server will recognize that it's not a static asset, and will proxy your request to

`http://localhost:4000/api/todos` as a fallback. The development server will **only** attempt to send requests without `text/html` in its `Accept` header to the proxy.

Keep in mind that `proxy` only has effect in development (with `npm start`), and it is up to you to ensure that URLs like `/api/todos` point to the right thing in production. You don't have to use the `/api` prefix. Any unrecognized request without a `text/html` accept header will be redirected to the specified `proxy`.

The `proxy` option supports HTTP, HTTPS and WebSocket connections.

If the `proxy` option is **not** flexible enough for you, alternatively you can:

- [Configure the proxy yourself](#)
- Enable CORS on your server ([here's how to do it for Express](#)).
- Use [environment variables](#) to inject the right server host and port into your app.

## "Invalid Host Header" Errors After Configuring Proxy

When you enable the `proxy` option, you opt into a more strict set of host checks. This is necessary because leaving the backend open to remote hosts makes your computer vulnerable to DNS rebinding attacks. The issue is explained in [this article](#) and [this issue](#).

This shouldn't affect you when developing on `localhost`, but if you develop remotely like [described here](#), you will see this error in the browser after enabling the `proxy` option:

Invalid Host header

To work around it, you can specify your public development host in a file called `.env.development` in the root of your project:

```
HOST=mypublicdevhost.com
```

If you restart the development server now and load the app from the specified host, it should work.

If you are still having issues or if you're using a more exotic environment like a cloud editor, you can bypass the host check completely by adding a line to `.env.development.local`.

**Note that this is dangerous and exposes your machine to remote code execution if you visit malicious websites:**



```
# NOTE: THIS IS DANGEROUS!  
# It exposes your machine to attacks from the websites you visit.  
DANGEROUSLY_DISABLE_HOST_CHECK=true
```

We don't recommend this approach.

## Configuring the Proxy Manually

Note: this feature is available with `react-scripts@2.0.0` and higher.

If the `proxy` option is **not** flexible enough for you, you can get direct access to the Express app instance and hook up your own proxy middleware.

You can use this feature in conjunction with the `proxy` property in `package.json`, but it is recommended you consolidate all of your logic into `src/setupProxy.js`.

First, install `http-proxy-middleware` using npm or Yarn:

```
$ npm install http-proxy-middleware --save  
$ # or  
$ yarn add http-proxy-middleware
```

Next, create `src/setupProxy.js` and place the following contents in it:

```
const { createProxyMiddleware } = require('http-proxy-middleware');  
  
module.exports = function(app) {  
  // ...  
};
```

You can now register proxies as you wish! Here's an example using the above `http-proxy-middleware`:

```
const { createProxyMiddleware } = require('http-proxy-middleware');  
  
module.exports = function(app) {  
  app.use(  
    '/api',  
    createProxyMiddleware({  
      target: 'http://localhost:5000',
```




```
    changeOrigin: true,  
  })  
);  
};
```

**Note:** You do not need to import this file anywhere. It is automatically registered when you start the development server.

**Note:** This file only supports Node's JavaScript syntax. Be sure to only use supported language features (i.e. no support for Flow, ES Modules, etc).

**Note:** Passing the path to the proxy function allows you to use globbing and/or pattern matching on the path, which is more flexible than the express route matching.

 [Edit this page](#)

Last updated on **3/4/2020** by **Herman J. Radtke III**

Previous

« [Debugging Tests](#)

Next

[Fetching Data with AJAX Requests](#) »

"Invalid Host Header" Errors After Configuring Proxy

Configuring the Proxy Manually

## Docs

[Get Started](#)

[Learn React](#)

**Community**



- Stack Overflow
- GitHub Discussions
- Twitter
- Contributor Covenant

**Social**

GitHub

Copyright © 2021 Facebook, Inc.

