

<WA1/>
<AW1/>
2022

Client-Server Interaction in React

Connecting React to HTTP APIs

Fulvio Corno
Luigi De Russis
Enrico Masala



Outline

- The “two servers” problem
 - Two servers + CORS → we will use this, in the course
 - React Development Server’s Proxy
 - Build + Express (single server)
 - Also: Understanding Build (webpack, imports, ...)



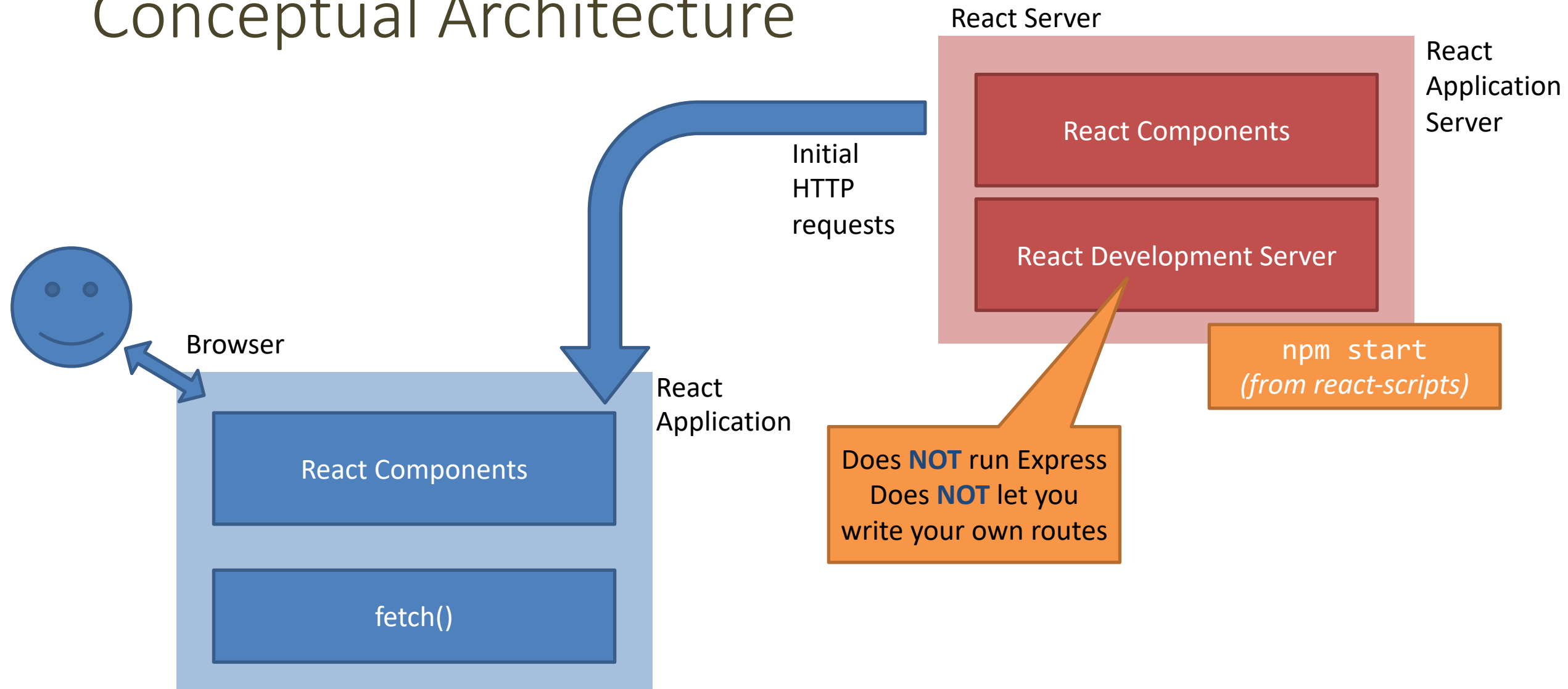
<https://www.robinwieruch.de/react-fetching-data>

Full Stack React, Chapter “Using Webpack with Create React App”

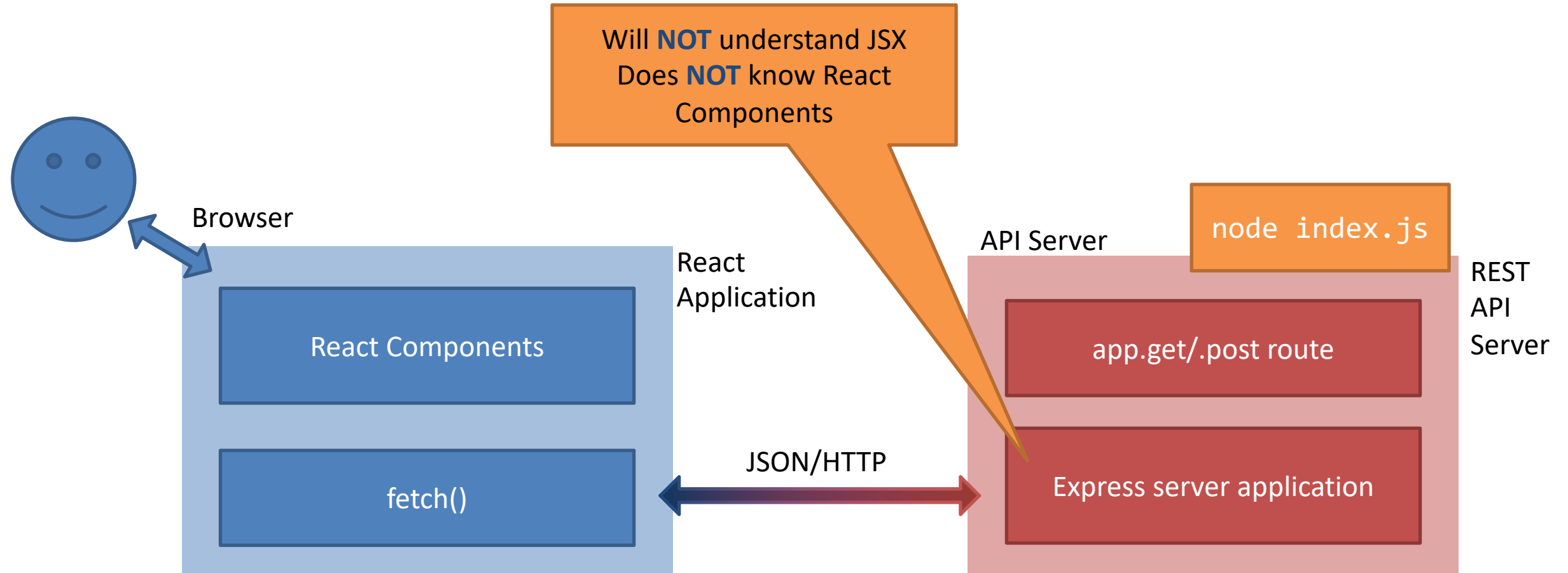
A Client and a Server walk into a bar...

THE “TWO SERVERS” PROBLEM

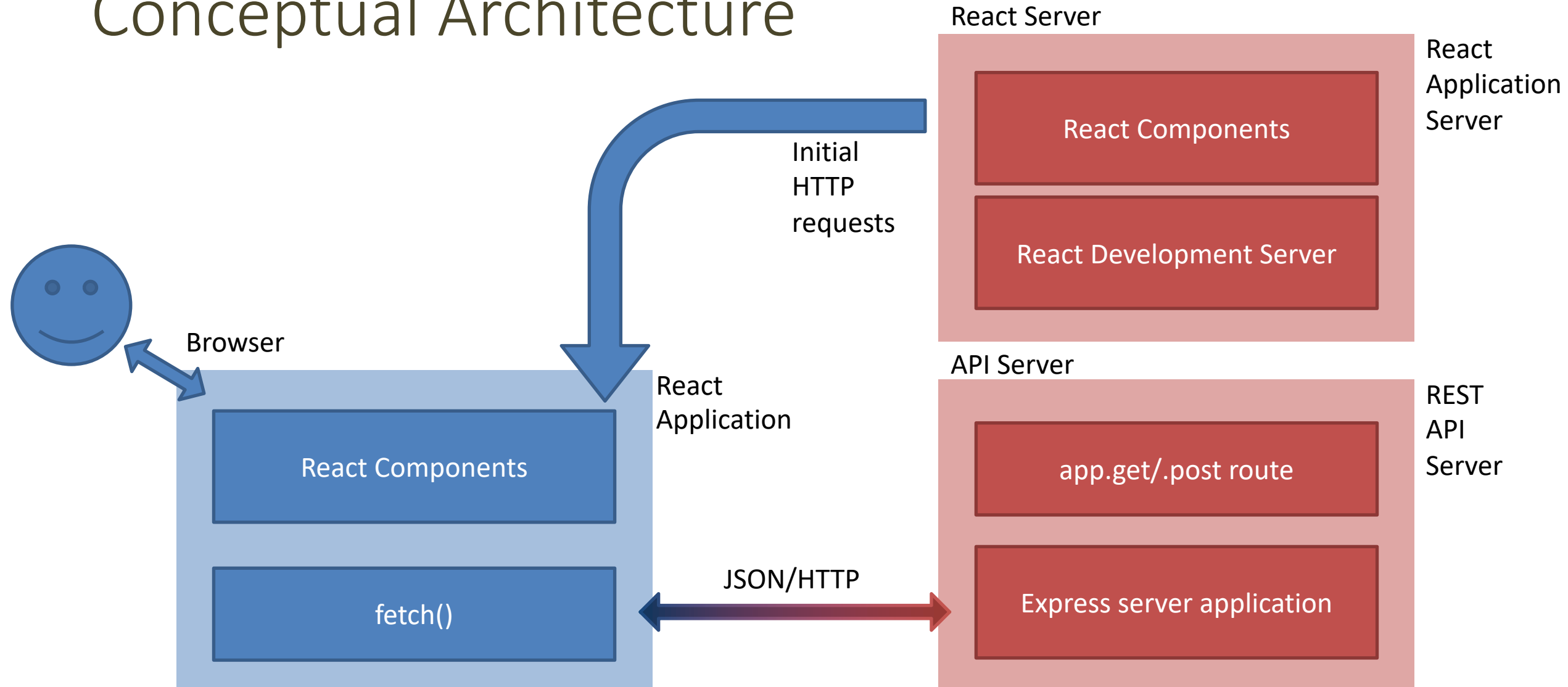
Conceptual Architecture



Conceptual Architecture



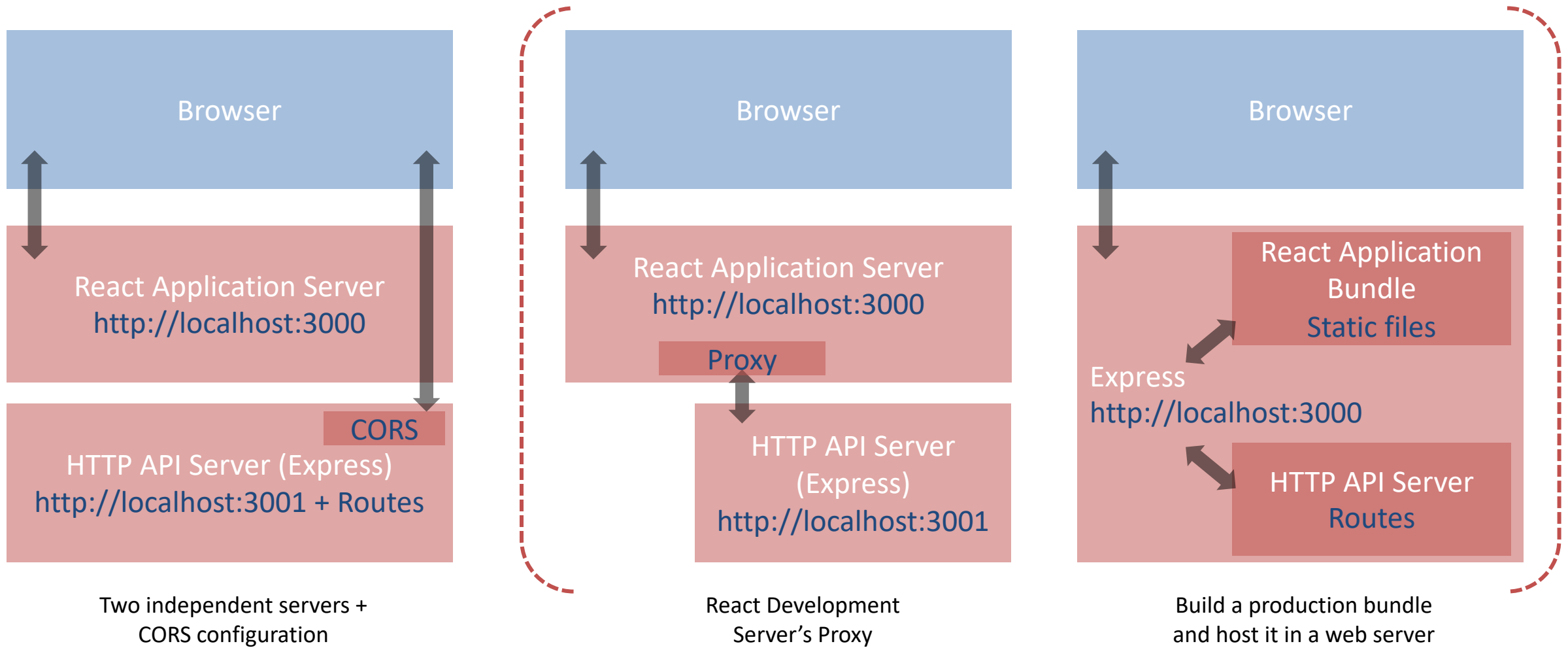
Conceptual Architecture



Issues

- Deployment
 - One-server-does-all or two-separate-servers?
 - Development vs. Production trade-off
 - convenience/debug/turnaround time vs performance/security
 - Cross-Origin security limitations
- Opportunities
 - Separate the load
 - Use any API Server (even 3rd party ones)

Three Possible Solutions



We will use this, in
the course



<https://www.newline.co/fullstack-react/articles/using-create-react-app-with-a-server/>

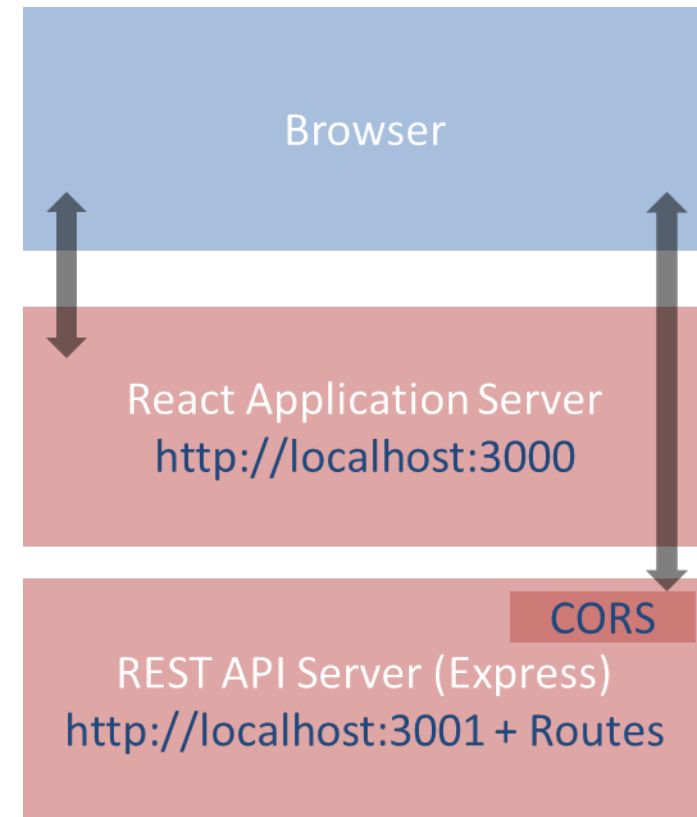
Full Stack React, Chapter “Using Webpack with Create React App / Using Create React App with an API server”

Side-by-side deployment

RUNNING TWO SEPARATE SERVERS

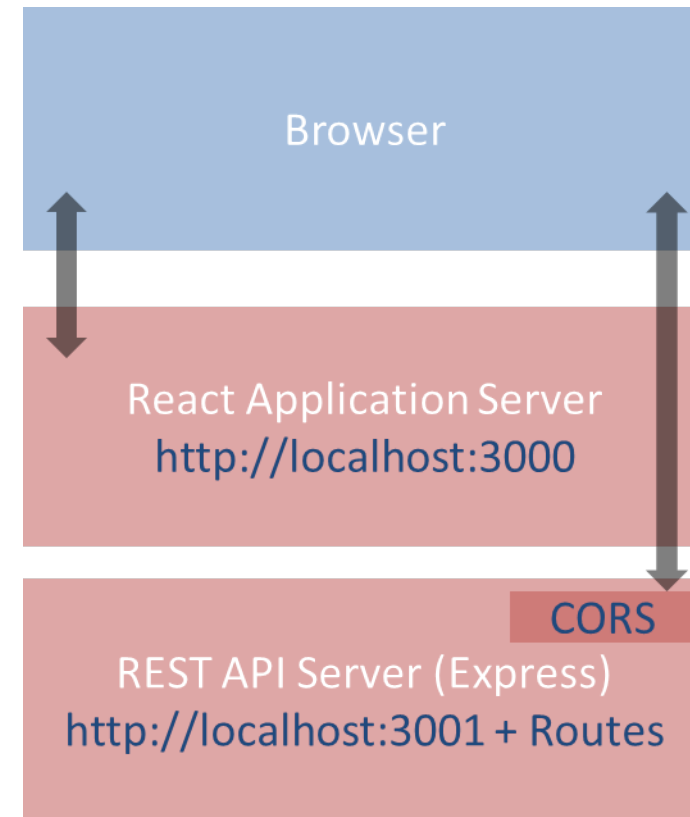
Double-Server Setup

- React Web Server and HTTP API server are hosted separately
 - Different hosts, and/or
 - Different ports
- The browser:
 - Receives the React application
 - Directs the API requests to the API server



Double-Server Setup

- Must run two web servers
 - React project: `npm start`
 - Express project: `node index.js`
 - Two projects, in two different directories (or different servers)
- Problem: handle CORS
 - Default security policy prevents loading data from other servers
 - Not discussed here



Advantages and Disadvantages

- Servers are easy to deploy
- Scalable solution: requests are sent to the appropriate server
- Only possible configuration if the HTTP API is provided by a third party
 - Public APIs
- Need to configure cross-origin resource sharing (CORS) on API server
- Requires using absolute URLs to access APIs
- Wrongly configured CORS might be a security risk (undesired access to APIs from e.g., mock websites)

How To Configure

- Configure CORS on API server for development

```
// index.js (node express server)  
  
//Enable All CORS Requests (for this server)  
app.use(cors());  
//Use ONLY for development, otherwise restrict domain
```

- In production mode, use different domains for React and API servers, NEVER allow CORS requests from any origin, always specify origin

Example

API.js in the React Application

```
const APIURL=new URL('http://localhost:3001');

async function getCourses() {
  return fetch(new URL('/courses', APIURL))
    .then((response)=>{
      if(response.ok) {
        return response.json() ;
      } else {
        throw response.statusText;
      }
    })
    .catch((error)=>{
      throw error;
    });
}
```

Called in useEffect()

index.js for the API Server

```
const express = require('express');
const port = 3001;
const cors = require('cors');
const app = express();
app.use(cors());

app.get('/courses', (req, res) => {
  dao.listCourses()
    .then((courses) => res.json(courses))
    .catch((err)=>
      res.status(503)
        .json(dbErrorObj));
});

app.listen(port, () => console.log(`Example app
listening at http://localhost:${port}`));
```

Calls DAO.js



<https://create-react-app.dev/docs/proxying-api-requests-in-development/>

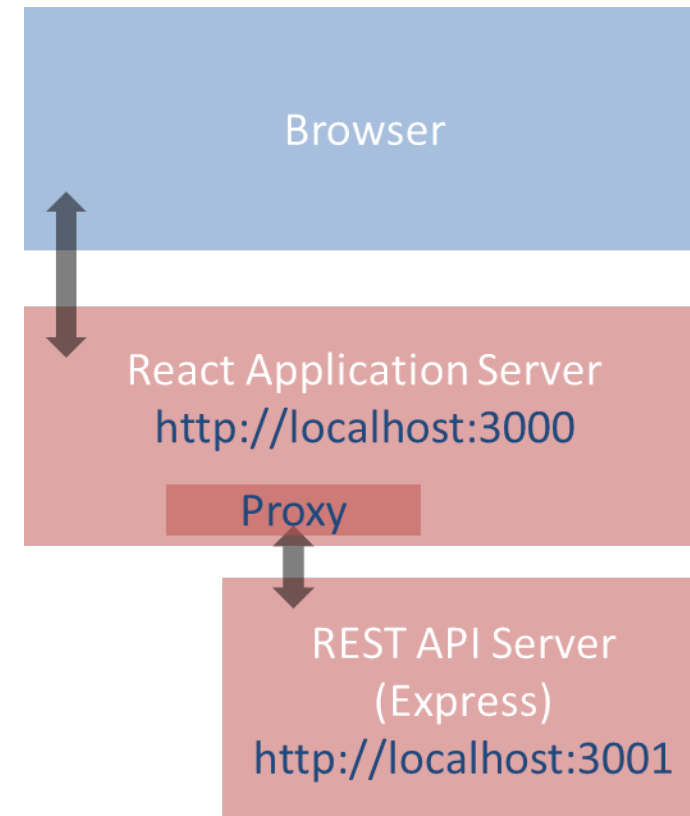
Full Stack React, Chapter “Using Webpack with Create React App / Using Create React App with an API server”

Double-Server made Easier

USING THE REACT DEVELOPMENT PROXY

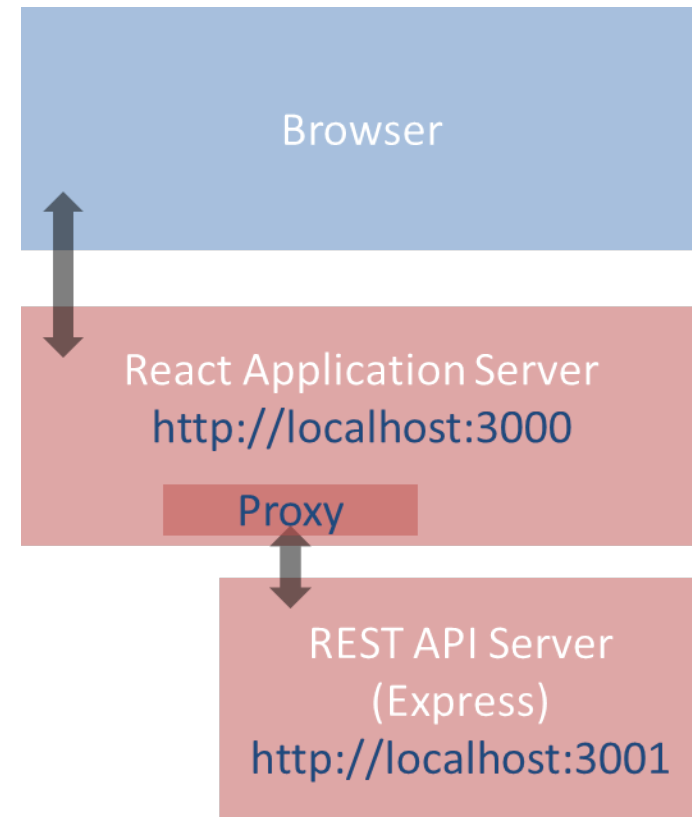
API Server Behind Application Server

- A feature provided by the React Development Server
 - uses react-scripts development modules
- Avoids the need to set-up CORS
- The browser thinks there is only one server



API Server Behind Application Server

- Browsers access only one server: the React application server
- The React web server is configured to act as a *proxy* for certain requests
- Those requests are sent to another web server via the proxy mechanism
- The proxy returns the response unaltered as its own response



How To Configure

- Just add one line in `package.json` originally written by `create-react-app`

```
// package.json
{
  ...
  ...,
  "proxy": "http://localhost:3001",
}
```

Address of the HTTP
API server

- N.B.: Works **only** in **development mode** while using the infrastructure of the `create-react-app` package

Proxy Rules

- The React development server will serve requests **directly** if:
 - It is a recognized static asset (e.g., image, stylesheet, ...)
 - The HTTP Accept header is set to `text/html`
- Otherwise, it will *attempt* to send the request to the **proxy**
 - The proxy response is returned
- If the resource is not found, it will serve the default HTML page

- Browsers use `text/html` only when expecting HTML content (e.g., first page)
- Best practice: avoid conflicting paths in URLs, if the path is found in React folders, it is served, otherwise it is passed to the proxy
 - Use unique path **prefix** for HTTP API requests, e.g., `/api`

Use In Production Mode

- The approach may be useful in production mode if the HTTP API server should not / cannot be accessed directly from the Internet
 - For instance, application server with private IPs or other network/security configuration reasons
- The main web server (Apache, nginx, etc.) should be able to determine which requests must be redirected to the other web server
 - For instance, depending on URLs (e.g., /api/... requests)

```
# nginx web server
location /api/ {
    proxy_pass http://backend-server;
}
```

```
# Apache web server

ProxyPass /api/ http://backend-server
```

Common Errors

- You are still running two web servers, on different ports
 - Remember to **start** the HTTP API server **before** launching the React application
 - May automate it by tweaking the startup scripts in `package.json`
- Production will be different
 - Need to configure the “real” proxy in production to be compatible with the same application path and API prefix



<https://create-react-app.dev/docs/deployment/#static-server>

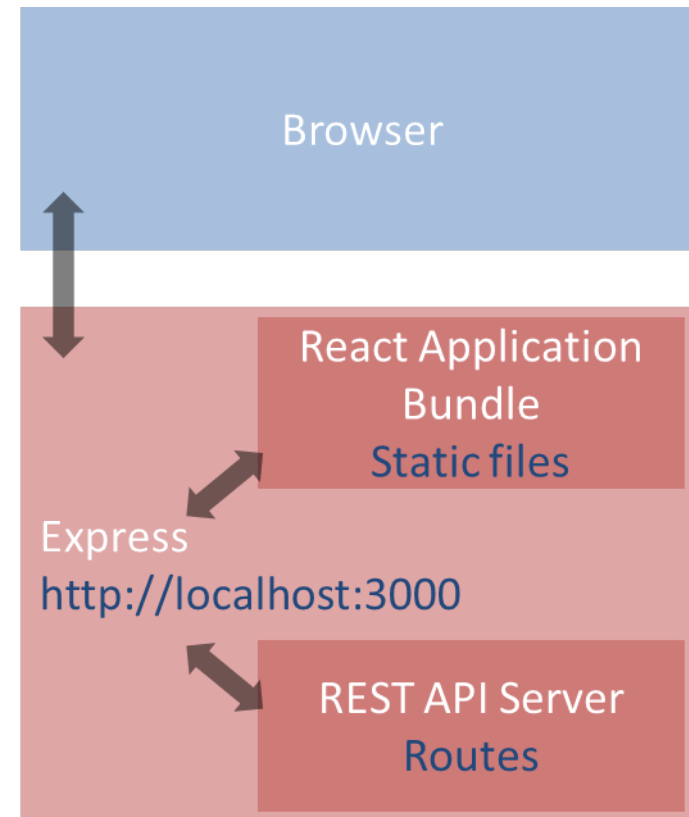
Full Stack React, Chapter “Using Webpack with Create React App / Creating a production build”

Packing and moving the React application into any web server

DEPLOYING A BUILD INSIDE A SERVER

Deploying the React Bundle

- React does not need to run in the Development Server
- `npm run build` will create a “production bundle” with all the contents needed to run the application
- This bundle is composed of static files (html, js, assets) and may be served by *any webserver* (including Apache, nginx, express, php, ...)



Build Command

npm run build

```
forno@Alieno: ~/src/react-scores
<1> forno@Alieno: ...
forno@Alieno:~/src/react-scores$ npm run build
> scores-r@0.1.0 build /home/forno/src/react-scores
> react-scripts build

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

 42.55 KB (-2 B)  build/static/js/2.1c7c2133.chunk.js
  2.96 KB (-17 B) build/static/js/main.f6993511.chunk.js
    778 B         build/static/js/runtime-main.d8864cb9.js

The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.
You may serve it with a static server:

  npm install -g serve
  serve -s build

Find out more about deployment here:

  bit.ly/CRA-deploy

forno@Alieno:~/src/react-scores$ |
wsl.exe[64]:14184  < 191012[32] 1/1 [+] NUM InpGrp PRI: 107x28 (35,274) 25V 14788
```

<https://create-react-app.dev/docs/deployment/>

Creates everything under ./build

Publish from / or from 'homepage' property

What Does “build” Do?

- Most of the work in “building” the static application is done by Babel and Webpack
 - Babel translates all JSX (and new JS syntax) into basic JS (according to the ‘production’ property in `package.json`)
 - Webpack packs and minimizes all JS code into a single file
 - Prepares an `index.html` that loads all the JS code
- The content of the “build” folder is self-contained and may be moved to the deployment server
- All debugging capabilities are removed

Hosting The Build in Express

- `cd express-api-server`
- `cp -r ../react-app/build .`
- Define a static route in `server.js`

```
app.use(express.static('./build'));
```

```
app.get('/', (req, res)=> {res.redirect('/index.html')} );
```

- In the application, you may call APIs locally
 - `fetch('/api/courses')...`

Pros and Cons

- Simple to deploy the final application (anywhere)
- May include the application inside the API server (in production, too)
- The JS code runs on every browser (thanks to polyfills and transpiling)

- The build cannot be directly modified
- Need a save/build/copy/reload cycle for every modification

Other “Magic” By Webpack

- Packing of all imported modules
- Bundling of Assets
 - Images
 - CSS files
- CSS Modules

In Development Mode...

- `npm start` runs the “Webpack development server” (WDS)
- All our code is transpiled and packed into a `bundle.js` that is automatically inserted into `index.html`
 - Contains all our code, plus React, plus imported modules
 - Also handles imports of non-JS files
- `bundle.js` does not exist – it’s kept in-memory by the WDS
- Sets up hot-reloading and synchronized error messages (via websockets)

Imports in Webpack

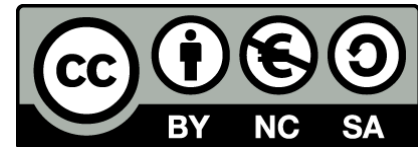
- `import logo from './logo.svg';`
- `import logo from './logo.png';`
 - Will include the image reference inside the bundle (placed under static/media)
 - Small files are rendered inline
- `import './Button.css';`
 - This component will use these CSS declarations
 - All CSS will be concatenated into a single file, but here we are stating the dependency
- `import styles from './Button.module.css';`
 - Files ending with `.module.css` are CSS modules
 - Styles may be applied with `className={styles.primary}`
 - Class names are *renamed to be unique*: no conflict with other Components' styles

Why Use Imports

- Scripts and stylesheets get minified and bundled together to avoid extra network requests.
- Missing files cause compilation errors instead of 404 errors for your users.
- Result filenames include content hashes, so you do not need to worry about browsers caching their old versions.
- They are an optional mechanism. “Traditional” loading (with link) still works, if you save your files in the public directory

References

- Taming the State in React, Robin Wieruch (2017)
<http://leanpub.com/taming-the-state-in-react>
- The Road to learn React, Robin Wieruch (2019)
<http://leanpub.com/the-road-to-learn-react>



License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
 - **Share** — copy and redistribute the material in any medium or format
 - **Adapt** — remix, transform, and build upon the material
 - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
 - **Attribution** — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
 - **NonCommercial** — You may not use the material for [commercial purposes](#).
 - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.
 - **No additional restrictions** — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>

