**b**

Deploying app to internet

Next let's connect the frontend we made in part 2 to our own backend.

In the previous part, the frontend could ask for the list of notes from the json-server we had as a backend at from the address <http://localhost:3001/notes>. Our backend has a bit different url structure, and the notes can be found from <http://localhost:3001/api/notes>. Let's change the attribute **baseUrl** in the *src/services/notes.js* like so:

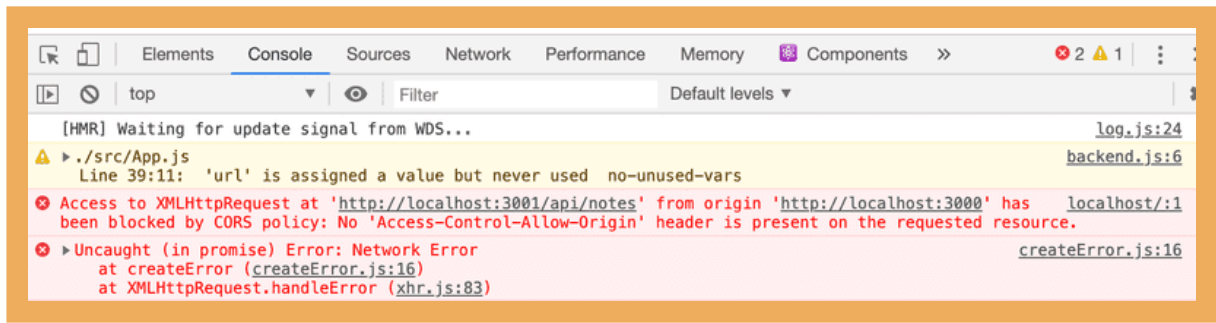
```
import axios from 'axios'
const baseUrl = 'http://localhost:3001/api/notes'

const getAll = () => {
  const request = axios.get(baseUrl)
  return request.then(response => response.data)
}

// ...

export default { getAll, create, update }
```

Now frontend's GET request to <http://localhost:3001/api/notes> does not work for some reason:



What's going on here? We can access the backend from a browser and from postman without any problems.

Same origin policy and CORS

The issue lies with a thing called CORS, or Cross-Origin Resource Sharing.

According to [Wikipedia](#):

Cross-origin resource sharing (CORS) is a mechanism that allows restricted resources (e.g. fonts) on a web page to be requested from another domain outside the domain from which the first resource was served. A web page may freely embed cross-origin images, stylesheets, scripts, iframes, and videos. Certain "cross-domain" requests, notably Ajax requests, are forbidden by default by the same-origin security policy.

In our context the problem is that, by default, the JavaScript code of an application that runs in a browser can only communicate with a server in the same [origin](#). Because our server is in localhost port 3001, and our frontend in localhost port 3000, they do not have the same origin.

Keep in mind, that [same origin policy](#) and CORS are not specific to React or Node. They are in fact universal principles of the operation of web applications.

We can allow requests from other *origins* by using Node's [cors](#) middleware.

Install *cors* with the command

```
npm install cors --save
```

take the middleware to use and allow for requests from all origins:

```
const cors = require('cors')  
  
app.use(cors())
```

And the frontend works! However, the functionality for changing the importance of notes has not yet been implemented to the backend.

You can read more about CORS from [Mozillas page](#).

Application to the Internet

Now that the whole stack is ready, let's move our application to the internet. We'll use good old Heroku for this.

If you have never used Heroku before, you can find instructions from Heroku documentation or by Googling.

Add a file called *Procfile* to the project's root to tell Heroku how to start the application.

```
web: node index.js
```

Change the definition of the port our application uses at the bottom of the *index.js* file like so:

```
const PORT = process.env.PORT || 3001
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`)
})
```

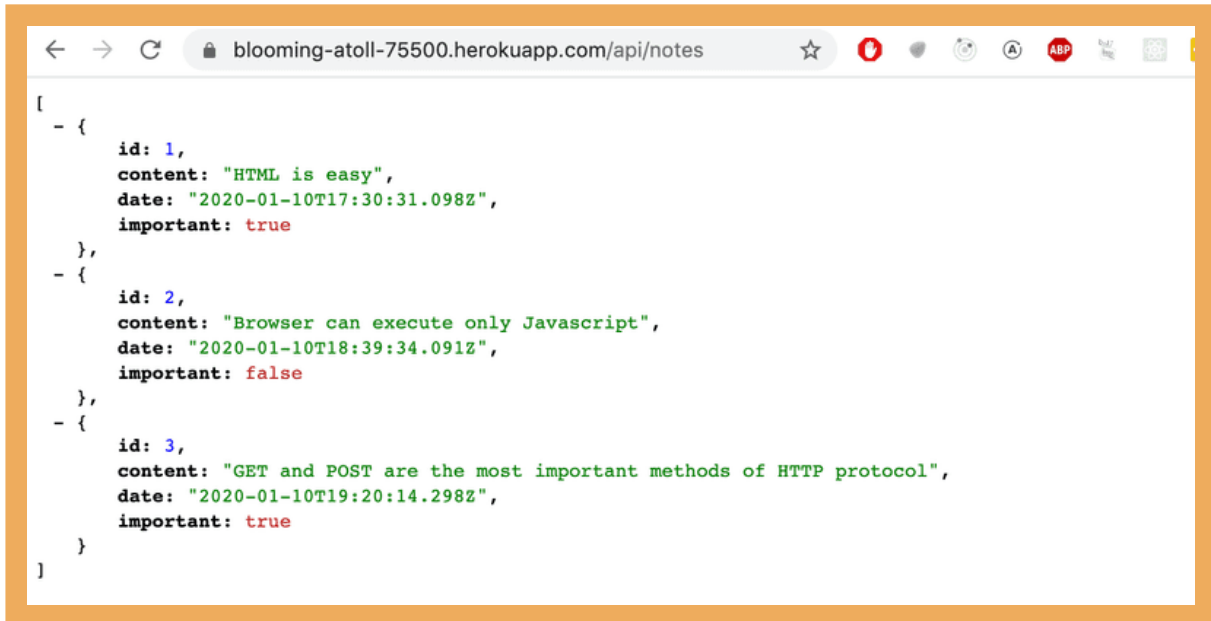
Now we are using the port defined in environment variable `PORT` or port 3001 if the environment variable `PORT` is undefined. Heroku configures application port based on the environment variable.

Create a Git repository in the project directory, and add *.gitignore* with the following contents

```
node_modules
```

Create a Heroku application with the command *heroku create*, commit your code to the repository and move it to Heroku with command *git push heroku master*.

If everything went well, the application works:

A screenshot of a web browser window. The address bar shows the URL 'blooming-atoll-75500.herokuapp.com/api/notes'. The browser's developer tools are open, displaying a JSON array of three note objects. The first note has id 1, content 'HTML is easy', date '2020-01-10T17:30:31.098Z', and important: true. The second note has id 2, content 'Browser can execute only Javascript', date '2020-01-10T18:39:34.091Z', and important: false. The third note has id 3, content 'GET and POST are the most important methods of HTTP protocol', date '2020-01-10T19:20:14.298Z', and important: true.

```
{
  - {
    id: 1,
    content: "HTML is easy",
    date: "2020-01-10T17:30:31.098Z",
    important: true
  },
  - {
    id: 2,
    content: "Browser can execute only Javascript",
    date: "2020-01-10T18:39:34.091Z",
    important: false
  },
  - {
    id: 3,
    content: "GET and POST are the most important methods of HTTP protocol",
    date: "2020-01-10T19:20:14.298Z",
    important: true
  }
}
```

If not, the issue can be found by reading heroku logs with command `heroku logs`.

NB At least in the beginning it's good to keep an eye on the heroku logs at all times. The best way to do this is with command `heroku logs -t` which prints the logs to console whenever something happens on the server.

NB If you are deploying from a git repository where your code is not on the master branch (i.e. if you are altering the notes repo from the last lesson) you will need to run `git push heroku HEAD:master`. If you have already done a push to heroku, you may need to run `git push heroku HEAD:master --force`.

The frontend also works with the backend on Heroku. You can check this by changing the backend's address on the frontend to be the backend's address in Heroku instead of `http://localhost:3001`.

The next question is, how do we deploy the frontend to the Internet? We have multiple options. Let's go through one of them next.

Frontend production build

So far we have been running React code in *development mode*. In development mode the application is configured to give clear error messages, immediately render code changes to the browser, and so on.

When the application is deployed, we must create a production build or a version of the application which is optimized for production.

A production build of applications created with *create-react-app* can be created with command `npm run build`.

Let's run this command from the *root of the frontend project*.

This creates a directory called *build* (which contains the only HTML file of our application, *index.html*) which contains the directory *static*. Minified version of our application's JavaScript code will be generated to the *static* directory. Even though the application code is in multiple

files, all of the JavaScript will be minified into one file. Actually all of the code from all of the application's dependencies will also be minified into this single file.

The minified code is not very readable. The beginning of the code looks like this:

```
!function(e){function r(r){for(var n,f,i=r[0],l=r[1],a=r[2],c=0,s=[];c<i.length;c++)f=i[c]
```

Serving static files from the backend

One option for deploying the frontend is to copy the production build (the *build* directory) to the root of the backend repository and configure the backend to show the frontend's *main page* (the file *build/index.html*) as its main page.

We begin by copying the production build of the frontend to the root of the backend. With my computer the copying can be done from the frontend directory with the command

```
cp -r build ../../../osa3/notes-backend
```

The backend directory should now look as follows:

```
→ notes-backend git:(master) x ls
Procfile          build              node_modules      package.json
README.md         index.js          package-lock.json
→ notes-backend git:(master) x
```

To make express show *static content*, the page *index.html* and the JavaScript etc. it fetches, we need a built-in middleware from express called static.

When we add the following amidst the declarations of middlewares

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

whenever express gets an HTTP GET request it will first check if the *build* directory contains a file corresponding to the request's address. If a correct file is found, express will return it.

Now HTTP GET requests to the address *www.serversaddress.com/index.html* or *www.serversaddress.com* will show the React frontend. GET requests to the address *www.serversaddress.com/api/notes* will be handled by the backend's code.

Because on our situation, both the frontend and the backend are at the same address, we can declare `baseUrl` as a relative URL. This means we can leave out the part declaring the server.

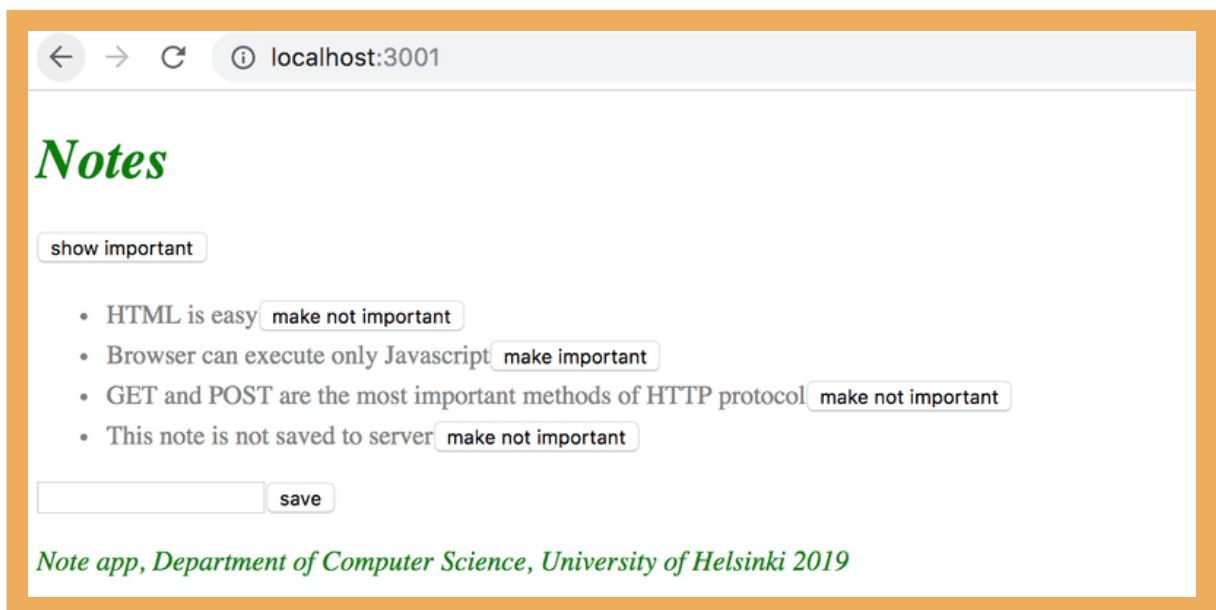
```
import axios from 'axios'
const baseUrl = '/api/notes'

const getAll = () => {
  const request = axios.get(baseUrl)
  return request.then(response => response.data)
}

// ...
```

After the change, we have to create a new production build and copy it to the root of the backend repository.

The application can now be used from the *backend* address <http://localhost:3001>:



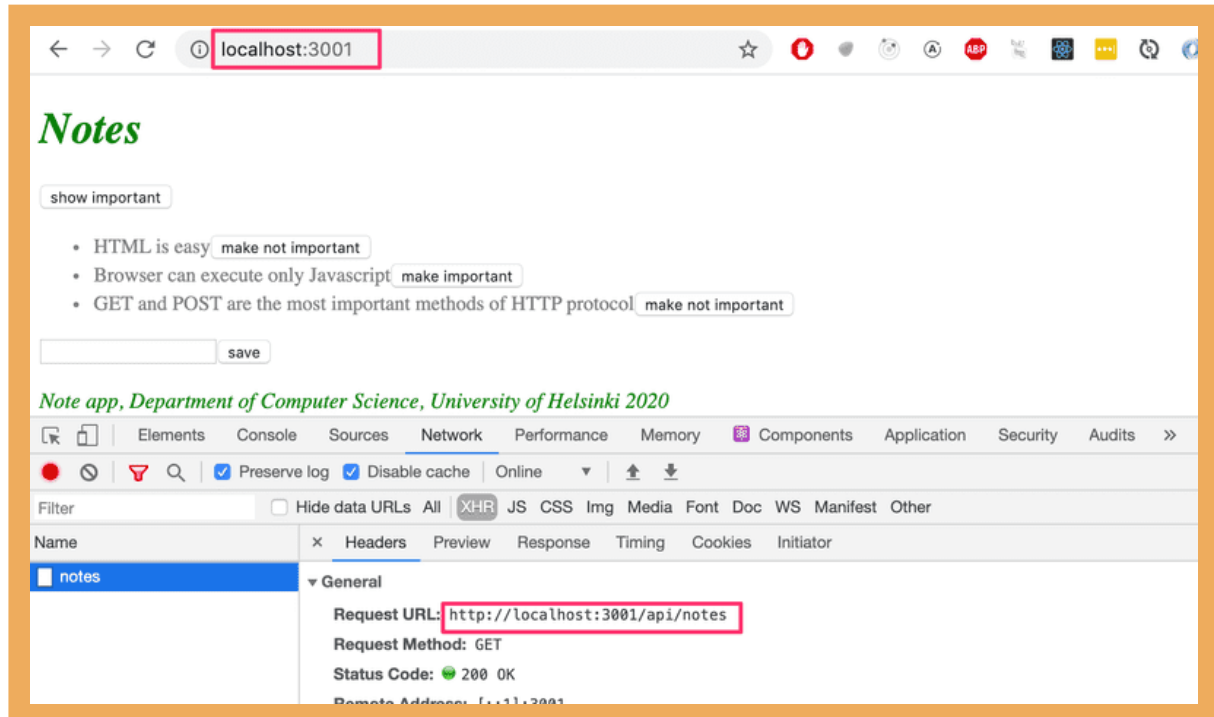
Our application now works exactly like the single-page app example application we studied in part 0.

When we use a browser to go to the address <http://localhost:3001>, the server returns the *index.html* file from the *build* repository. Summarized contents of the file are as follows:

```
<head>
  <meta charset="utf-8"/>
  <title>React App</title>
  <link href="/static/css/main.f9a47af2.chunk.css" rel="stylesheet">
</head>
<body>
  <div id="root"></div>
  <script src="/static/js/1.578f4ea1.chunk.js"></script>
  <script src="/static/js/main.104ca08d.chunk.js"></script>
</body>
</html>
```

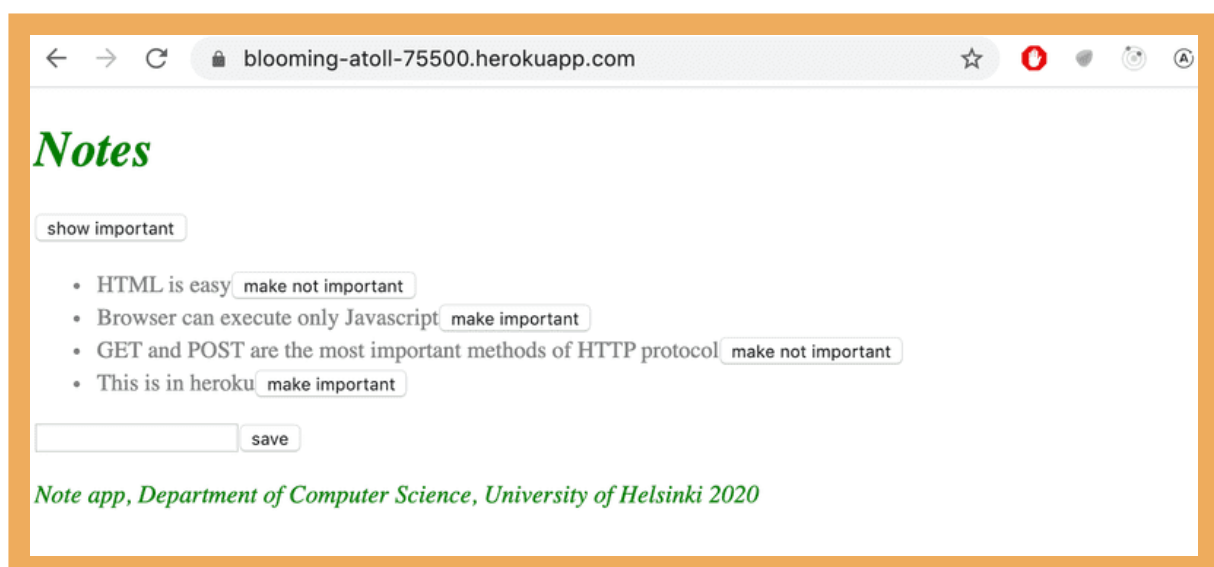
The file contains instructions to fetch a CSS stylesheet defining the styles of the application, and two *script* tags which instruct the browser to fetch the JavaScript code of the application - the actual React application.

The React code fetches notes from the server address <http://localhost:3001/api/notes> and renders them to the screen. The communications between the server and the browser can be seen in the *Network* tab of the developer console:



After ensuring that the production version of the application works locally, commit the production build of the frontend to the backend repository, and push the code to Heroku again.

The application works perfectly, except we haven't added the functionality for changing the importance of a note to the backend yet.



Our application saves the notes to a variable. If the application crashes or is restarted, all of the data will disappear.

The application needs a database. Before we introduce one, let's go through a few things.

Streamlining deploying of the frontend

To create a new production build of the frontend without extra manual work, let's add some npm-scripts to the *package.json* of the backend repository:

```
{
  "scripts": {
    //...
    "build:ui": "rm -rf build && cd ../../osa2/materiaali/notes-new && npm run build --prod",
    "deploy": "git push heroku master",
    "deploy:full": "npm run build:ui && git add . && git commit -m uibuild && npm run deploy",
    "logs:prod": "heroku logs --tail"
  }
}
```

The script `npm run build:ui` builds the frontend and copies the production version under the backend repository. `npm run deploy` releases the current backend to heroku.

`npm run deploy:full` combines these two and contains the necessary *git* commands to update the backend repository.

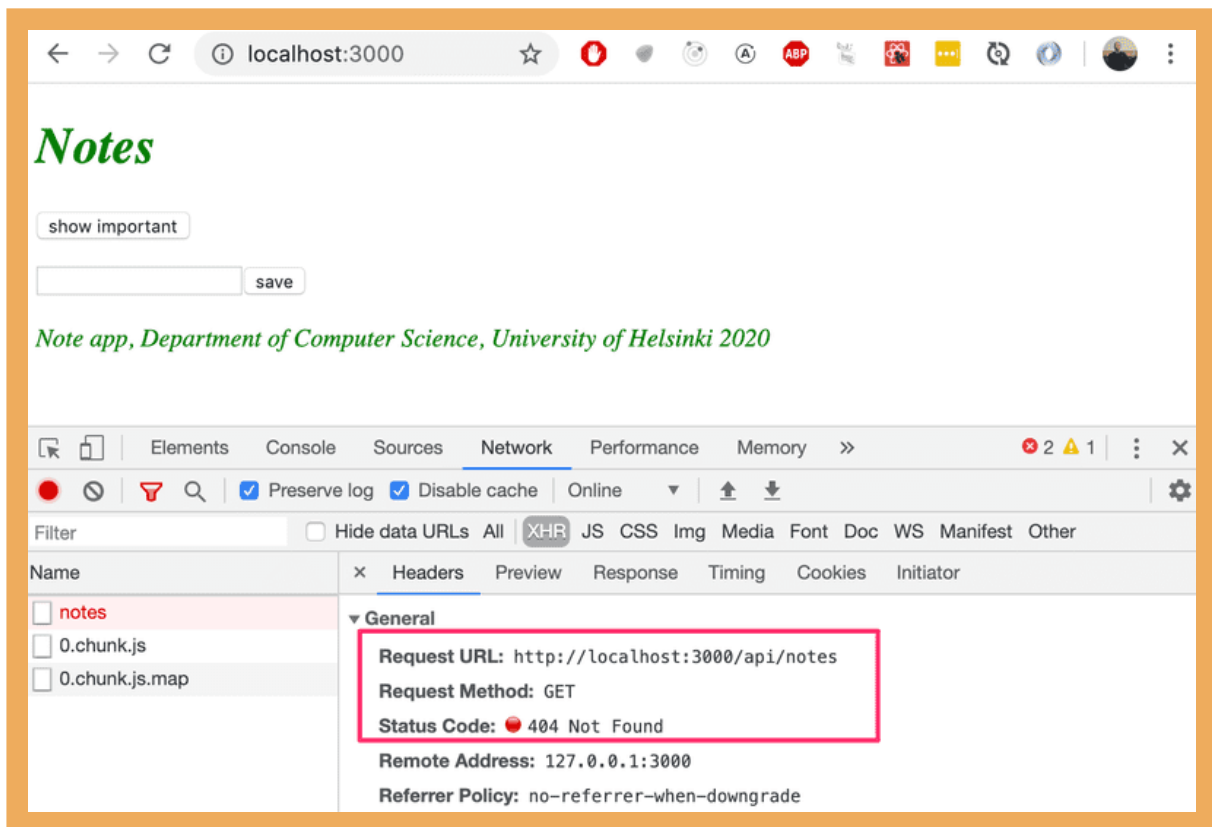
There is also a script `npm run logs:prod` to show the heroku logs.

Note that the directory paths in the script *build:ui* depend on the location of repositories in the file system.

NB *build:ui* does not work on Windows, go to [Solution](#)

Proxy

Changes on the frontend have caused it to no longer work in development mode (when started with command `npm start`), as the connection to the backend does not work.



This is due to changing the backend address to a relative URL:

```
const baseUrl = '/api/notes'
```

Because in development mode the frontend is at the address *localhost:3000*, the requests to the backend go to the wrong address *localhost:3000/api/notes*. The backend is at *localhost:3001*.

If the project was created with create-react-app, this problem is easy to solve. It is enough to add the following declaration to the *package.json* file of the frontend repository.

```
{
  "dependencies": {
    // ...
  },
  "scripts": {
    // ...
  },
  "proxy": "http://localhost:3001"
}
```

After a restart, the React development environment will work as a proxy. If the React code does an HTTP request to a server address at *http://localhost:3000* not managed by the React application itself (i.e. when requests are not about fetching the CSS or JavaScript of the application), the request will be redirected to the server at *http://localhost:3001*.

Now the frontend is also fine, working with the server both in development- and production mode.

A negative aspect of our approach is how complicated it is to deploy the frontend. Deploying a new version requires generating new production build of the frontend and copying it to the backend repository. This makes creating an automated deployment pipeline more difficult. Deployment pipeline means an automated and controlled way to move the code from the computer of the developer through different tests and quality checks to the production environment.

There are multiple ways to achieve this (for example placing both backend and frontend code to the same repository) but we will not go into those now.

In some situations it may be sensible to deploy the frontend code as its own application. With apps created with create-react-app it is straightforward.

Current code of the backend can be found on Github, in the branch *part3-3*. The changes in frontend code are in *part3-1* branch of the frontend repository.

Exercises 3.9.-3.11.

The following exercises don't require many lines of code. They can however be challenging, because you must understand exactly what is happening and where, and the configurations must be just right.

3.9 phonebook backend step9

Make the backend work with the frontend from the previous part. Do not implement the functionality for making changes to the phone numbers yet, that will be implemented in exercise 3.17.

You will probably have to do some small changes to the frontend, at least to the URLs for the backend. Remember to keep the developer console open in your browser. If some HTTP requests fail, you should check from the *Network*-tab what is going on. Keep an eye on the backend's console as well. If you did not do the previous exercise, it is worth it to print the request data or *request.body* to the console in the event handler responsible for POST requests.

3.10 phonebook backend step10

Deploy the backend to the internet, for example to Heroku.

NB the command `heroku` works on the department's computers and the freshman laptops. If for some reason you cannot install Heroku to your computer, you can use the command `npmx heroku-cli`.

Test the deployed backend with a browser and Postman or VS Code REST client to ensure it works.

PRO TIP: When you deploy your application to Heroku, it is worth it to at least in the beginning keep an eye on the logs of the heroku application **AT ALL TIMES** with the command `heroku logs -t`.

The following is a log about one typical problem. Heroku cannot find application dependency *express*:

```
2019-01-17T18:56:56.647740+00:00 heroku[web.1]: State changed from crashed to starting
2019-01-17T18:56:56.333986+00:00 app[api]: Release v4 created by user mluukkai@iki.fi
2019-01-17T18:56:56.333986+00:00 app[api]: Deploy 1bf3a652 by user mluukkai@iki.fi
2019-01-17T18:56:56.000000+00:00 app[api]: Build succeeded
2019-01-17T18:56:59.068722+00:00 heroku[web.1]: Starting process with command `node index.js`
2019-01-17T18:57:01.338939+00:00 heroku[web.1]: State changed from starting to crashed
2019-01-17T18:57:01.286454+00:00 heroku[web.1]: Process exited with status 1
2019-01-17T18:57:01.222299+00:00 app[web.1]: internal/modules/cjs/loader.js:583
2019-01-17T18:57:01.222323+00:00 app[web.1]: throw err;
2019-01-17T18:57:01.222324+00:00 app[web.1]: ^
2019-01-17T18:57:01.222326+00:00 app[web.1]:
2019-01-17T18:57:01.222327+00:00 app[web.1]: Error: Cannot find module 'express'
2019-01-17T18:57:01.222329+00:00 app[web.1]: at Function.Module._resolveFilename (internal/modules/cjs/loader.js:581:15)
2019-01-17T18:57:01.222331+00:00 app[web.1]: at Function.Module._load (internal/modules/cjs/loader.js:507:25)
2019-01-17T18:57:01.222333+00:00 app[web.1]: at Module.require (internal/modules/cjs/loader.js:637:17)
2019-01-17T18:57:01.222334+00:00 app[web.1]: at require (internal/modules/cjs/helpers.js:22:18)
2019-01-17T18:57:01.222336+00:00 app[web.1]: at Object.<anonymous> (/app/index.js:1:79)
2019-01-17T18:57:01.222338+00:00 app[web.1]: at Module._compile (internal/modules/cjs/loader.js:689:30)
2019-01-17T18:57:01.222339+00:00 app[web.1]: at Object.Module._extensions..js (internal/modules/cjs/loader.js:700:10)
2019-01-17T18:57:01.222341+00:00 app[web.1]: at Module.load (internal/modules/cjs/loader.js:599:32)
2019-01-17T18:57:01.222343+00:00 app[web.1]: at tryModuleLoad (internal/modules/cjs/loader.js:538:12)
2019-01-17T18:57:01.222345+00:00 app[web.1]: at Function.Module._load (internal/modules/cjs/loader.js:530:3)
```

The reason is that the option `--save` was forgotten when *express* was installed, so information about the dependency was not saved to the file *package.json*.

Another typical problem is that the application is not configured to use the port set to environment variable `PORT`:

```
2019-01-17T18:59:55.116663+00:00 heroku[web.1]: State changed from crashed to starting
2019-01-17T18:59:54.000000+00:00 app[api]: Build succeeded
2019-01-17T19:00:00.190003+00:00 heroku[web.1]: Starting process with command `node index.js`
2019-01-17T19:00:03.226796+00:00 app[web.1]: Server running on port 3001
2019-01-17T19:01:00.860055+00:00 heroku[web.1]: State changed from starting to crashed
2019-01-17T19:01:00.700724+00:00 heroku[web.1]: Error R10 (Boot timeout) -> Web process failed to bind to $PORT within 60 seconds of launch
2019-01-17T19:01:00.700724+00:00 heroku[web.1]: Stopping process with SIGKILL
2019-01-17T19:01:00.829188+00:00 heroku[web.1]: Process exited with status 137
```

Create a `README.md` at the root of your repository, and add a link to your online application to it.

3.11 phonebook full stack

Generate a production build of your frontend, and add it to the internet application using the method introduced in this part.

NB Make sure the directory *build* is not gitignored

Also make sure that the frontend still works locally.

[Propose changes to material](#)

[< Part 3a](#)
[Previous part](#)

[Part 3c >](#)
[Next part](#)

[About course](#)

[Course contents](#)

[FAQ](#)

[Partners](#)

[Challenge](#)



UNIVERSITY OF HELSINKI

HOUSTON