Metodologie per la Programmazione per il Web - MF0437 *Client-side routing*

Docente

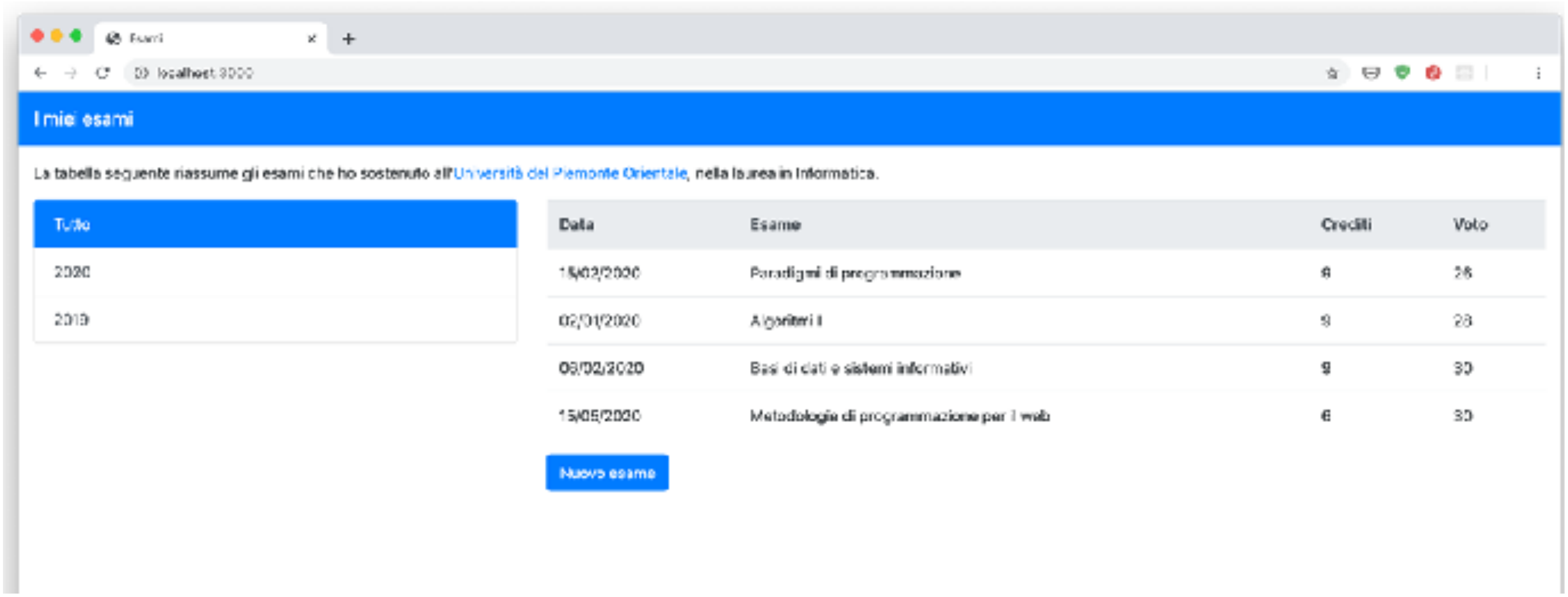
Giancarlo **Ruffo**[giancarlo.ruffo@uniupo.it]

Informazioni, materiale e risorse su:

moodle [https://www.dir.uniupo.it/course/view.php?id=16455]

Slide adattate di versioni precedenti a cura dei

Proff. Luigi De Russis ed Alessio Bottrighi



Esami

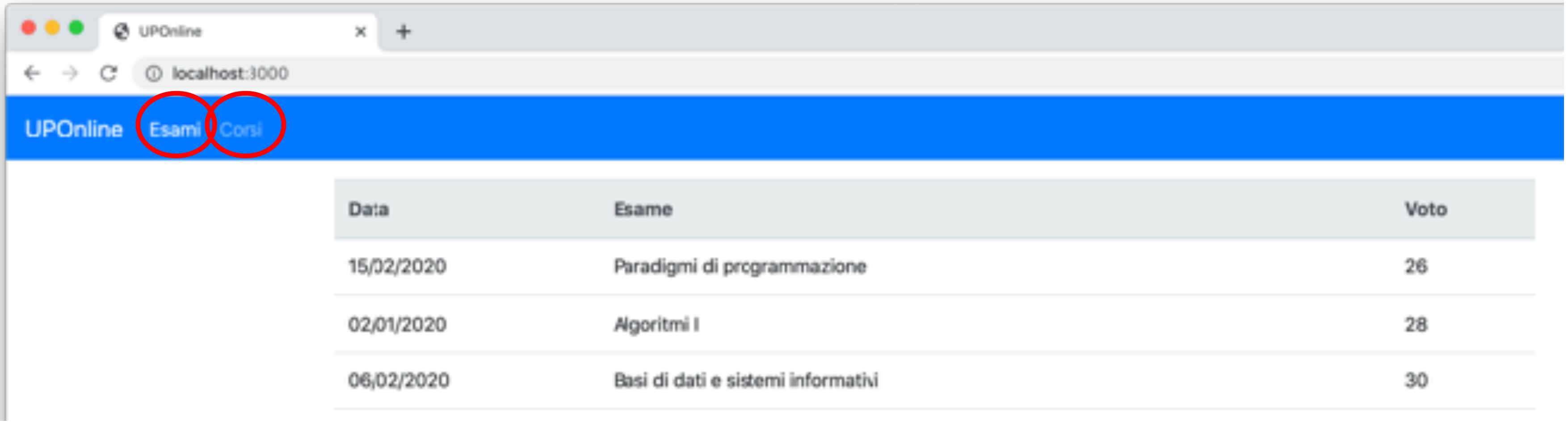
localhost:3000

I miei esami

La tabella seguente riassume gli esami che ho sostenuto all'Università del Piemonte Orientale, nella laurea in Informatica.

Data	Esame	Crediti	Voto
15/02/2020	Paradigmi di programmazione	6	26
02/01/2020	Algoritmi I	9	28
06/02/2020	Basi di dati e sistemi informativi	9	30
15/05/2020	Metodologie di programmazione per il web	6	30

Web apps have more than one page…



Data	Esame	Voto
15/02/2020	Paradigmi di programmazione	26
02/01/2020	Algoritmi I	28
06/02/2020	Basi di dati e sistemi informativi	30

Supporting Complex Web Applications

- Switching between many different page layouts
- Managing the flow of navigation across a set of "pages"
- Maintaining the default web navigation conventions
  - back, forward, bookmarks, …
- Allowing URLs to convey information
- Allowing re-loading KBs of JavaScript at every page change
- Keeping the state across page changes
- …

Using URLs for Navigation

- URLs determine the ***type*** of the page or the ***section*** of the website
  - Changing page ⇆ Changing the URL
- URLs also ***embed information*** about the item IDs, referrers, categories, filters, etc.
- URLs can be shared/saved/bookmarked, and they are sufficient for rebuilding the whole exact page
  - Deep Linking
- Back and forward buttons navigate the URL history

Example URLs on facebook.com:

```
/

/profile.name

/profile.name /posts/
1234123212422123

/pagename

/pages/?
category=your_pages
```

Approaches

Server-side Navigation

- <a> links in the page with href pointing to an actual HTML page
- The browser requests a new URL to the server
- The server returns a new page
  - rendered on the server
  - with a copy of the same JS application if the same page
- A different way to structure and create a web application
  - e.g., limited usage of fetch, few to no JSON, …

Approaches

Server-side Navigation

- <a> links in the page with href pointing to an actual HTML page
- The browser requests a new URL to the server
- The server returns a new page
  - rendered on the server
  - with a copy of the same JS application if the same page
- A different way to structure and create a web application
  - e.g., limited usage of fetch, few to no JSON, …

Traditional approach

- Pros
  - Simple
  - Perfectly integrates in browser navigation
- Cons
  - A lot of the same content is passed multiple time, *if* the web app heavily relies on JavaScript
  - Click on a link -> page load (or reload)

Approaches

Client-side Navigation

- Modify the location of the app (the URL)
- Interact with the HTML5 History API to enable "back", "forward", … buttons and features
- Determine which elements to render at a given location
- In principle, whenever the user clicks on a new URL
  - we prevent the browser from fetching the next page
  - we instruct the JS app to switch in&out elements

Approaches

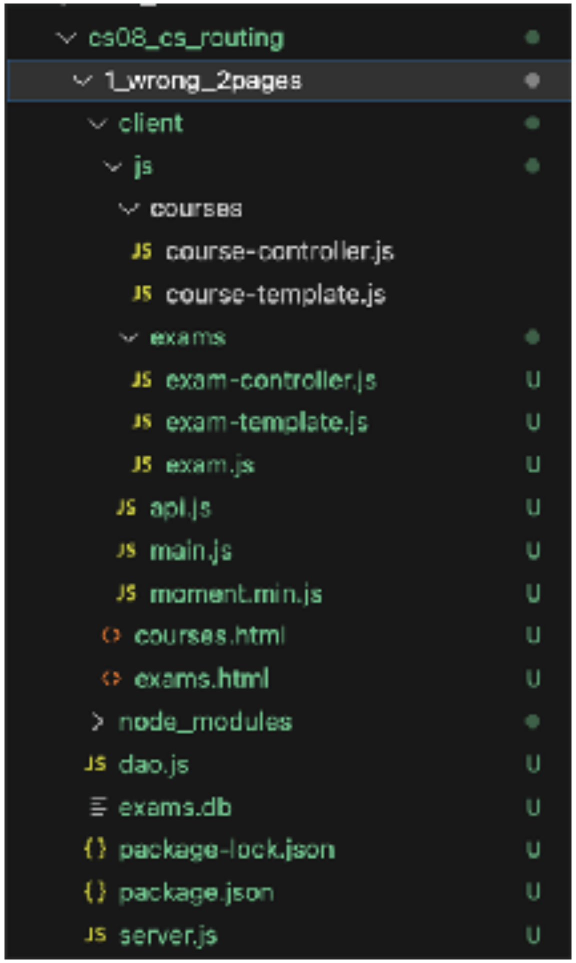
Single Page Application's approach

- Pros
  - No reloads are necessary
  - The same content is not passed multiple time, *if* the app heavily relies on JavaScript
  - Integrates in browser navigation
- Cons
  - Error-prone history mangling, *if* done manually
  - Needs to distinguish client's routes from server's one, *if* the web app is provided by the same REST server

Client-side Navigation

- Modify the location of the app (the URL)
- Interact with the HTML5 History API to enable "back", "forward", … buttons and features
- Determine which elements to render at a given location
- In principle, whenever the user clicks on a new URL
  - we prevent the browser from fetching the next page
  - we instruct the JS app to switch in&out elements

(Wrong) Approaches *Case 1*



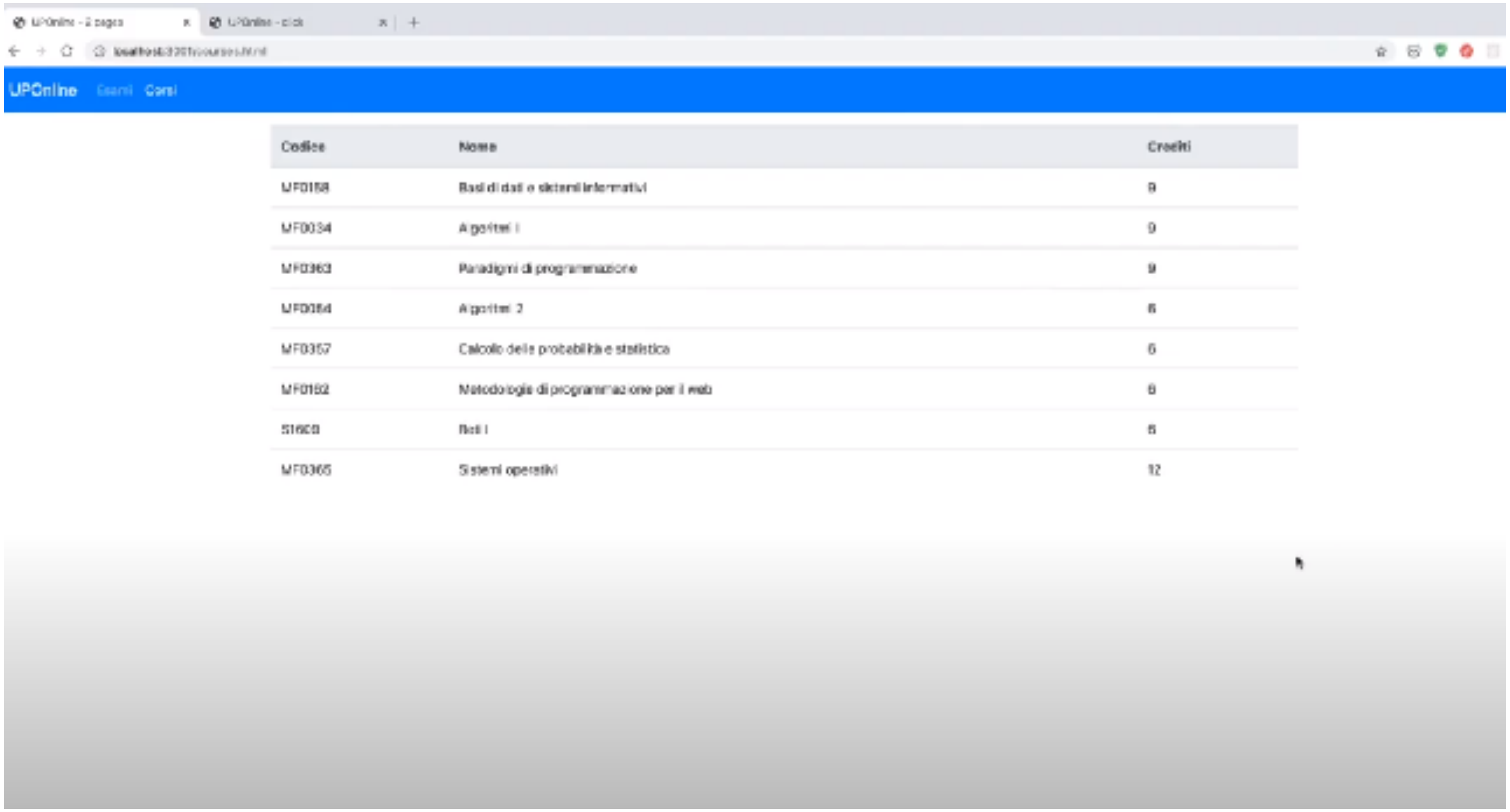
```
cs08_cs_routing
  1_wrong_2pages
    client
      js
        courses
          course-controller.js
          course-template.js
        exams
          exam-controller.js     U
          exam-template.js       U
          exam.js                U
        api.js                   U
        main.js                  U
        moment.min.js            U
      courses.html               U
      exams.html                 U
    node_modules
    dao.js                       U
    exams.db                     U
    package-lock.json            U
    package.json                 U
    server.js                    U
```

- For a web app that **heavily** relies on fetch + REST:
  - trying to adopt the server-side navigation

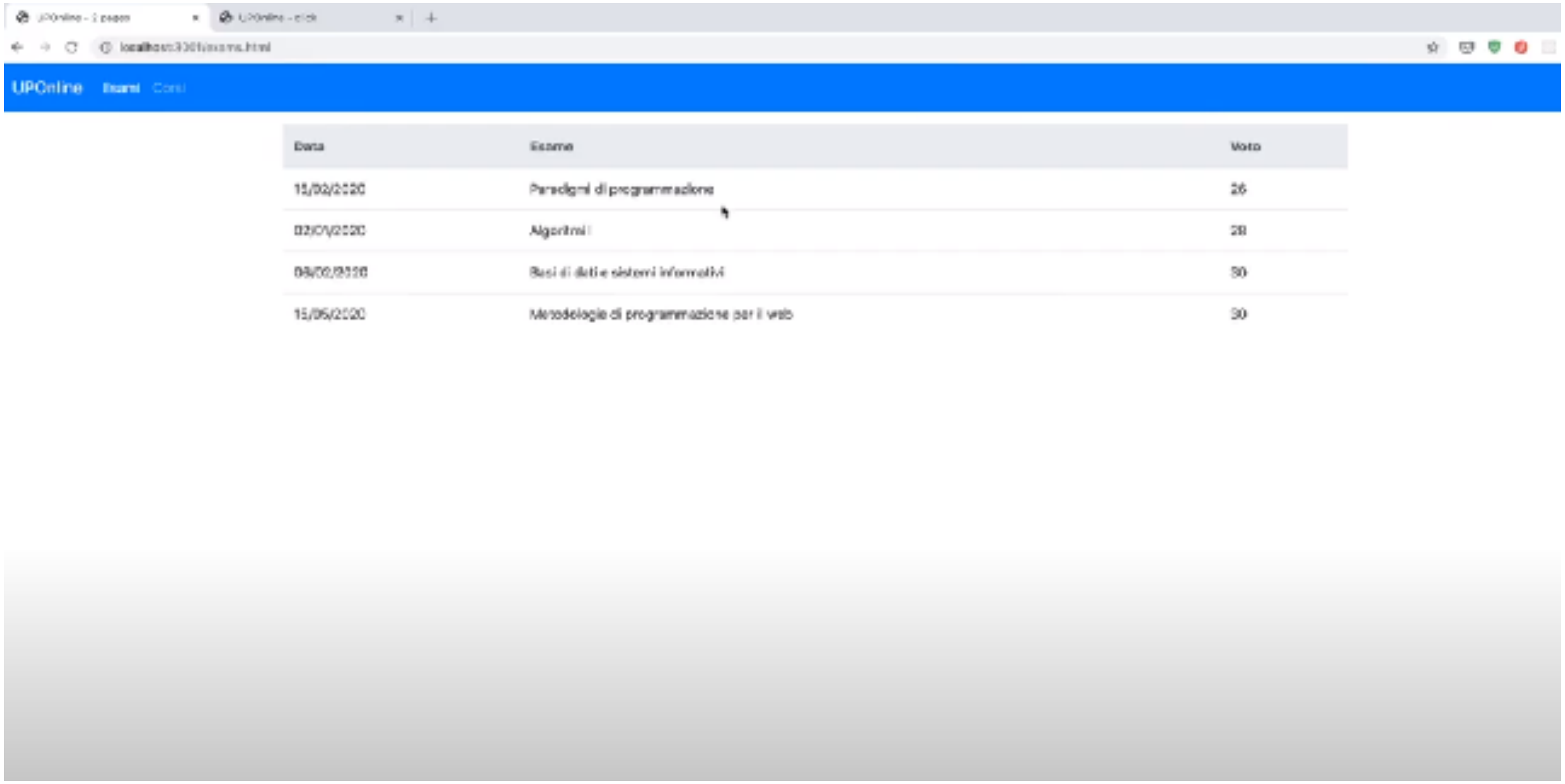
<- `courses.html` and `exams.html` are 99.99% identical

<- `course-controller.js` and `exam-controller.js` share a lot of code

<- `main.js` needs to check the URL to know which of the two classes should instantiate



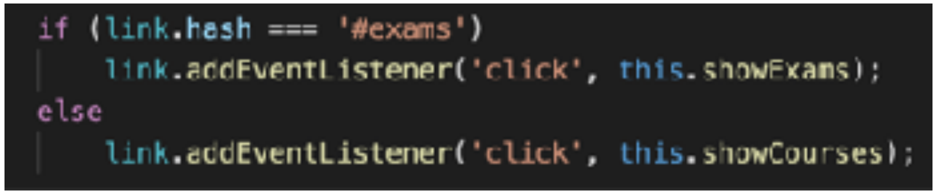
Codice	Nome	Crediti
MF0158	Basi di dati e sistemi informativi	9
MF0034	Algoritmi 1	9
MF0363	Paradigmi di programmazione	9
MF0364	Algoritmi 2	6
MF0357	Calcolo delle probabilità e statistica	6
MF0152	Metodologia di programmazione per il web	6
S1600	Reti 1	6
MF0365	Sistemi operativi	12



Data	Esame	Voto
15/02/2020	Paradigmi di programmazione	26
02/01/2020	Algoritmi I	28
08/02/2020	Basi di dati e sistemi informativi	30
15/06/2020	Metodologie di programmazione per il web	30

(Wrong) Approaches Case 2

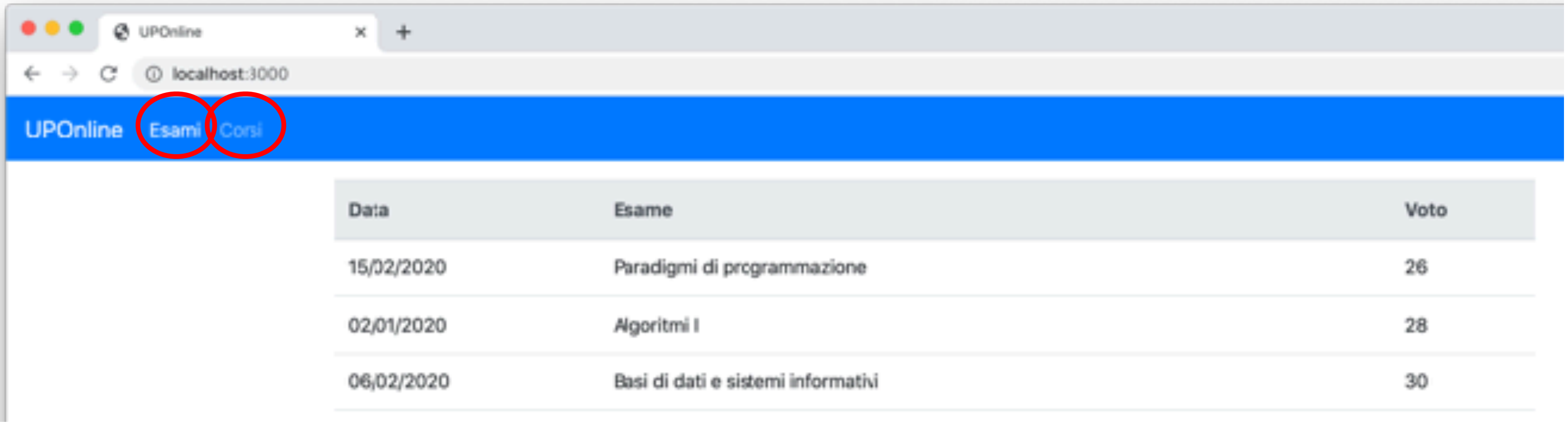
- For a web app that heavily relies on fetch + REST:
  - using click event listener to change "page"



```
if (link.hash === '#exams')
    link.addEventListener('click', this.showExams);
else
    link.addEventListener('click', this.showCourses);
```

<- it needs to have an `eventListener` explicitly defined for every link

<- it *breaks* browser navigation!!!



Data	Esame	Voto
15/02/2020	Paradigmi di programmazione	26
02/01/2020	Algoritmi I	28
06/02/2020	Basi di dati e sistemi informativi	30

The Page.js Router

Handling multiple pages in our web app

Routers

- The problems associated with multi-page navigation and URL management are usually handled by ***router*** libraries
- Fully-fledged SPA frameworks typically includes a router
  - or support some dedicated routers
- *Vanilla JS* applications (like ours) can choose among a few options, if they need a router (*most popular*)
  - ***Page.js***
  - navigo

Page.js

- A quite popular (and **updated**!) client-side router
- Documentation and examples
  - https://visionmedia.github.io/page.js/
- Integrate with the browser's native navigation features
- Use callbacks to show pages according to a specified route
- Easy to integrate and understand
  - it is Express-inspired!
- Available as ES6 module
- Tiny (~1200 byte)

Installation

- With a **global script tag**, in the HTML page

```
<script src="https://unpkg.com/page/page.js"></
script>
```

- With **modules**, in JavaScript (*preferred*)

```
import page from "//unpkg.com/page/page.mjs";
```

Usage: Route Definitions

- `page()` is the main function
- For defining routes, it accepts two arguments: a ***relative path*** and one or more ***callbacks***
  - callback(s) will be called, in order, when the URL matches the indicated path
- `page.base([path])` sets a base path for all the routes
- Example:

```
page('/', index);
page('/users/:id', () => {...});
page('/courses', prepare, load);
page('*', notFound);
page(); // register page's binding (e.g., click events)
```

Usage: Explicitly Changing Pages

- `page()` is still used
  - `page(path)` navigates to the given path
  - `page(fromPath, toPath)` redirects from one path to the other
- Examples:

```
page.base('/upo');
page('/exams');
page('/exams', '/courses');
```

Usage: Separating Concerns

- Page.js uses the same conventions that Express does, so things like ":id" and "*" work as you might expect
- Examples:

```
page('/users/:id', load, show);
page('/users/:id/edit', load, edit);
```

=

```
page('/users/*', load);
page('/users/:id', show);
page('/users/:id/edit', edit);
```

Advanced Usage: Parameters and Contexts

- Much like Express has request and response objects, Page.js has a `Context` object
- Example: `page('/user/:id', load, show)`
- Using this example, we can assign arbitrary properties to `ctx` to maintain ***state*** between callbacks
- To build `load()` that will load the user for subsequent routes you will need to access the ":id" passed

```
function load(ctx, next){
  const id = ctx.params.id;
  fetch('api/users/' + id).then(response => {
    response.json().then(user => {
      ctx.user = user;
      next();
    });  });
}
```

```
function show(ctx){
  return ctx.user.username;
}
```

- The `show()` might look something like this: