ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

React JS
React Router

Δαμιανός Μαρινάτος

SPA - Single Page Application

- Σε αντίθεση με τις παραδοσιακές web εφαρμογές, **οι εφαρμογές μίας σελίδας (SPA**) χρειάζεται να ενημερώνουν μόνο το τμήμα της σελίδας που προβάλλεται τη δεδομένη στιγμή, προκειμένου να ανταποκριθούν στις ενέργειες του χρήστη· έτσι αποφεύγεται η διαρκής φόρτωση νέων σελίδων από τον διακομιστή. Αυτή η προσέγγιση δεν διακόπτει την εμπειρία του χρήστη μεταξύ διαδοχικών σελίδων, κάνοντας την εφαρμογή να συμπεριφέρεται περισσότερο σαν desktop application.
- Όλος ο απαραίτητος κώδικας, HTML, JavaScript και CSS, **φορτώνεται με μία μόνο αρχική κλήση**. Στη συνέχεια, οι επιμέρους πόροι φορτώνονται δυναμικά και ενσωματώνονται στη σελίδα όποτε χρειάζεται, επιτρέποντας την πλοήγηση μέσω συνδέσμων ή άλλων διαδραστικών στοιχείων χωρίς πλήρη ανανέωση του περιεχομένου.

SPA vs MultiPage Applications

Πλεονεκτήματα	Μειονεκτήματα
Ταχύτητα & άμεση απόκριση κατά την πλοήγηση· αλλάζει μόνο το δυναμικό τμήμα της σελίδας.	Αργότερο αρχικό φόρτωμα· όλο το bundle κατεβαίνει μία φορά στην αρχή.
Μικρότερος φόρτος server / bandwidth μετά το πρώτο load· ακολουθούν μόνο API κλήσεις.	Δεν βολεύει εφαρμογές πολύ βαριές σε περιεχόμενο (π.χ. τεράστιες landing pages με SEO focus).
Offline λειτουργία & caching μέσω Service Workers.	SEO: δυσκολίες αν δεν γίνει SSR ή prerender.
Γρηγορότερη ανάπτυξη UI (component reuse, hot reload).	Πιο σύνθετο client-side scripting & state-management.
Πλούσια διαδραστικότητα – μοιάζει με desktop app.	Θέματα ασφάλειας αν δεν προστατευτούν σωστά τα public APIs.
Καλύτερη κλιμάκωση για μεγάλα apps (code-splitting, lazy routes).	Πιθανή ασυμβατότητα με παλαιούς browsers / συσκευές.
Ευκολότερη συντήρηση – κώδικας χωρισμένος σε modules/components.	Δυσκολότερο debugging (hydration, χρονομετρητές, race conditions).
Cross-platform συμβατότητα με σύγχρονα web standards (PWA).	Θέματα προσβασιμότητας αν δεν προσεχθεί aria / focus management.
Μεγάλη ευελιξία UI/UX (animations, transitions).	Δεν ταιριάζει σε κάθε project.

History API

- Το **History API** παρέχει πρόσβαση στο ιστορικό περιήγησης της τρέχουσας συνεδρίας(session) του browser μέσω του global object **history**.
- Προσφέρει χρήσιμες μεθόδους και ιδιότητες που μας επιτρέπουν να:
  - αλλάζουμε το url χωρίς να γίνει νέα αίτηση στον server.
  - πλοηγούμαστε μπρος και πίσω στο ιστορικό του χρήστη, και
  - τροποποιούμε το περιεχόμενο της στοίβας του ιστορικού (history stack).
- Κύριες μέθοδοι:
  - **history.pushState()**: Προσθέτει ένα νέο state στο ιστορικό και αλλάζει το URL.
  - **history.replaceState()**: Αντικαθιστά το τρέχον state χωρίς να προσθέσει νέο.
  - **window.onpopstate**: Εκτελείται όταν ο χρήστης πάει πίσω/μπροστά με τα βελάκια.

React Router

- Είναι βιβλιοθήκη της React για **client-side routing** που συνδέει *URL paths* με React Components.
- Χρησιμοποιεί το **History API** (pushState, popstate)· αλλάζει τη γραμμή διεύθυνσης **χωρίς** να ζητά νέο HTML.
- Κάθε “route” φορτώνει/απο-φορτώνει components, ενημερώνει τίτλο σελίδας, στέλνει analytics κ.λπ.
- Υποστηρίζει **δυναμικά & nested paths**(/users/:id), ακριβώς όπως σε παραδοσιακό backend routing.
- **Διατηρεί το global state** ζωντανό μεταξύ σελίδων.

React Router - Core Blocks

- **<BrowserRouter>:** Περικλείει όλη την εφαρμογή και συνδέει το React Router με το History API του browser. Παρακολουθεί τις αλλαγές στο URL χωρίς full-page reload.
- **<Routes> / <Route>:** Δηλώνουν ποιο component θα εμφανιστεί σε ποιο path**.**
- **<Link>:** χρησιμοποιείται για πλοήγηση χωρίς ανανέωση σελίδας. Καλεί το pushState και ο Router κάνει render το νέο component. Το <a> προκαλεί πλήρη ανανέωση της σελίδας.
- **<NavLink>:** κάνει κλασικό client-side navigation και ελέγχει αν το path είναι ενεργό. Όταν ταιριάζει το URL, προσθέτει αυτόματα class="active" (και aria-current="page") για εύκολο styling.
- **useNavigate / <Navigate>:** Χρησιμοποιούνται για ανακατευθύνσεις.
- **useParams:** Διαβάζει παραμέτρους από το URL (/users/:id).

Εγκατάσταση React Router

- Ανοίγουμε το terminal και γράφουμε την παραπάνω εντολή στο root directory του project μας.

```
npm i react-router
```

- Μόλις εγκατασταθεί θα δούμε ότι η βιβλιοθήκη έχει προστεθεί στο αρχείο package.json

```
"dependencies": {
  "@tailwindcss/vite": "^4.1.7",
  "lucide-react": "^0.511.0",
  "react": "^19.1.0",
  "react-dom": "^19.1.0",
  "react-router": "^7.6.1",
  "tailwindcss": "^4.1.7"
},
```

Simple Routes


```
<Routes>
  <Route index element={<HomePage />} />
  <Route path="name-changer" element={<NameChangerPage />} />
</Routes>
```

- Συνδέουμε τις σελίδες **HomePage** και **NameChangerPage** με τα routes.
- Χρησιμοποιούμε το **Layout Component** για να δημιουργήσουμε μια κοινή δομή για όλες τις σελίδες (π.χ. header, footer).
- Η **index** διαδρομή αντιστοιχεί στην αρχική σελίδα (/).

Nested Routes


```
<Routes>
  <Route index element={<HomePage />} />
  <Route path="examples">
    <Route path="name-changer" element={<NameChangerPage />} />
    <Route path="online-status" element={<OnlineStatusPage />} />
  </Route>
</Routes>
```

- Η σελίδα **/examples** λειτουργεί ως γονικό route.
- Οι σελίδες **name-changer** και **online-status** είναι child routes του /examples.
- Το τελικό URL για κάθε child είναι /examples/name-changer και /examples/online-status.

Dynamic Segments


```
    <Routes>
      <Route path="users/:userId" element={<UserPage />} />
    </Routes>
  </Layout>
```

- Το :userId είναι δυναμικό segment — ταιριάζει οποιαδήποτε τιμή στη θέση του (π.χ. /users/5).
- Μέσα στο UserPage, χρησιμοποιούμε useParams() για να διαβάσουμε την τιμή του userId.
- Το component μπορεί να φορτώσει δεδομένα χρήστη από API ή να εμφανίσει πληροφορίες ανάλογα με το userId

Optional Segments


```
<Routes>
  <Route path="users?/:userId" element={<UserPage />} />
  <Route path="examples?/name-changer" element={<NameChangerPage />} />
</Routes>
```

- Μπορούμε να δηλώσουμε τμήματα του path ως **προαιρετικά**, χρησιμοποιώντας το **?** στο τέλος του segment.
- Το ? στο path δηλώνει ότι **αυτό το κομμάτι μπορεί να υπάρχει ή να λείπει**.
- Στο 1ο Route, μπορούμε να καλέσουμε το UserPage Component καλώντας με το **/users/userId** ή με το **/userId** (πχ. /users/10 ή /10).
- Στο 2ο Route, μπορούμε να καλέσουμε το NameChangerPage Component καλώντας με το **/examples/name-changer** ή με το **/name-changer**.

Catchall Segments


```
<Routes>
  <Route path="files/*" element={<File />} />
</Routes>
```

```
let params : Readonly<Params<string>> = useParams();
let filePath : string | undefined = params["*"];
```

- Όταν ορίζουμε μια διαδρομή με * σημαίνει ότι οποιαδήποτε διαδρομή ξεκινάει με /files/ θα ταιριάζει, ανεξάρτητα από το τι ακολουθεί μετά.
- Αυτή η τεχνική λέγεται catch-all routing και μας επιτρέπει να "πιάσουμε" ό,τι απομένει στο URL.
- Μέσα στο component, μπορούμε να χρησιμοποιήσουμε το useParams() για να πάρουμε το το υπόλοιπο path.
- Χρησιμεύει για φόρτωση αρχείων ή περιεχομένου από δυναμική διαδρομή, για Nested routing και για Fallback routes (π.χ. για custom 404 σε συγκεκριμένο section)

Layout Routes (1)


```jsx
<Routes>
  <Route element={<RouterLayout />}>
    <Route index element={<HomePage />} />
  </Route>
  <Route path="examples" element={<RouterExamplesLayout />}>
    <Route index element={<Examples />} />
    <Route path="name-changer" element={<NameChangerPage />} />
    <Route path="online-status" element={<OnlineStatusPage />} />
  </Route>
</Routes>
```

- Επιτρέπουν να ορίσουμε **κοινή διάταξη (layout)** για ένα σύνολο child routes, όπως header, footer ή sidebar.
- Το layout περιέχει ένα <Outlet />, όπου “γεμίζει” δυναμικά το περιεχόμενο του αντίστοιχου child route.
- Βοηθάει στη **δομική οργάνωση** και **επαναχρησιμοποίηση UI** (π.χ. /examples/* έχει το δικό του layout ξεχωριστό από την αρχική σελίδα).
- Στο παράδειγμα το <RouterLayout /> τυλίγει μόνο την αρχική σελίδα (/). Και το <RouterExamplesLayout /> τυλίγει όλες τις διαδρομές που ξεκινούν από /examples/* σε συγκεκριμένο section)

Layout Routes (2)


```
import { Outlet } from "react-router"
import Header from "./Header";
import Footer from "./Footer";

const RouterLayout : () => Element  = () : Element  => {  Show usages   new *
  return (
    <>
      <Header />
      <div className="container mx-auto min-h-[95vh] pt-24">
        <Outlet />
      </div>
      <Footer />
    </>
  );
};

export default RouterLayout;  Show usages   new *
```

Linking


```
<Link to="/examples/name-changer">Name Changer</Link>

<NavLink
  to="/examples/online-status"
  className={({ isActive } : NavLinkRenderProps ) : "text-cf-dark-red underline" | "text-cf-g... =>
    isActive ? " text-cf-dark-red underline" : " text-cf-gray"
  }
>
  Online Status
</NavLink>
```

- Το **<Link>** χρησιμοποιείται για **πλοήγηση** χωρίς ανανέωση σελίδας (client-side navigation).
- Το **<NavLink>** κάνει κλασικό client-side navigation και ελέγχει αν το path είναι ενεργό. Όταν ταιριάζει το URL, προσθέτει αυτόματα το **class="active"** και το **aria-current="page"**. Είναι ιδανικό για μενού, tabs ή breadcrumbs όπου θέλουμε να ξεχωρίζει ποια σελίδα είναι επιλεγμένη.
- Το **isActive** είναι ένα property που παρέχεται από το React Router και δείχνει αν το path του NavLink ταιριάζει με την τρέχουσα τοποθεσία (URL).

Navigating


```
const navigate : NavigateFunction = useNavigate();
navigate( to: "/");
```

- Είναι ένα Hook της βιβλιοθήκης React Router που επιτρέπει πλοήγηση (navigation) χωρίς την αλληλεπίδραση του χρήστη.
- Το χρησιμοποιούμε μετά από υποβολή φόρμας, σε auto-logout λόγω αδράνειας, σε χρονομετρημένες διεπαφές (countdowns) κ.α.

Route Params


```
import { useParams } from "react-router";

const UserPage : () => Element = () : Element => {
  const { userId } = useParams();
```

- Τα **route parameters** είναι δυναμικά τμήματα ενός URL path που δηλώνονται με **":"**
- Παράδειγμα <Route path="/users/:userId" element={<UserPage />} />. Εδώ το :userId είναι placeholder και μπορεί να πάρει οποιαδήποτε τιμή από το URL, π.χ. /users/42.
- Η **useParams()** είναι ένα Hook που μας δίνει πρόσβαση στα dynamic route parameters.

Search Params


```
const [searchParams] = useSearchParams();
const query : string | null  = searchParams.get( name: "query");
const page : string | null  = searchParams.get( name: "page");
```

- Η **useSearchParams()** είναι ένα Hook που μας δίνει πρόσβαση στα query params.
- Με την **get("key")** μπορούμε να πάρουμε την κάθε παράμετρο.
- Χρησιμοποιείται για search, filtering, sorting κ.ά.