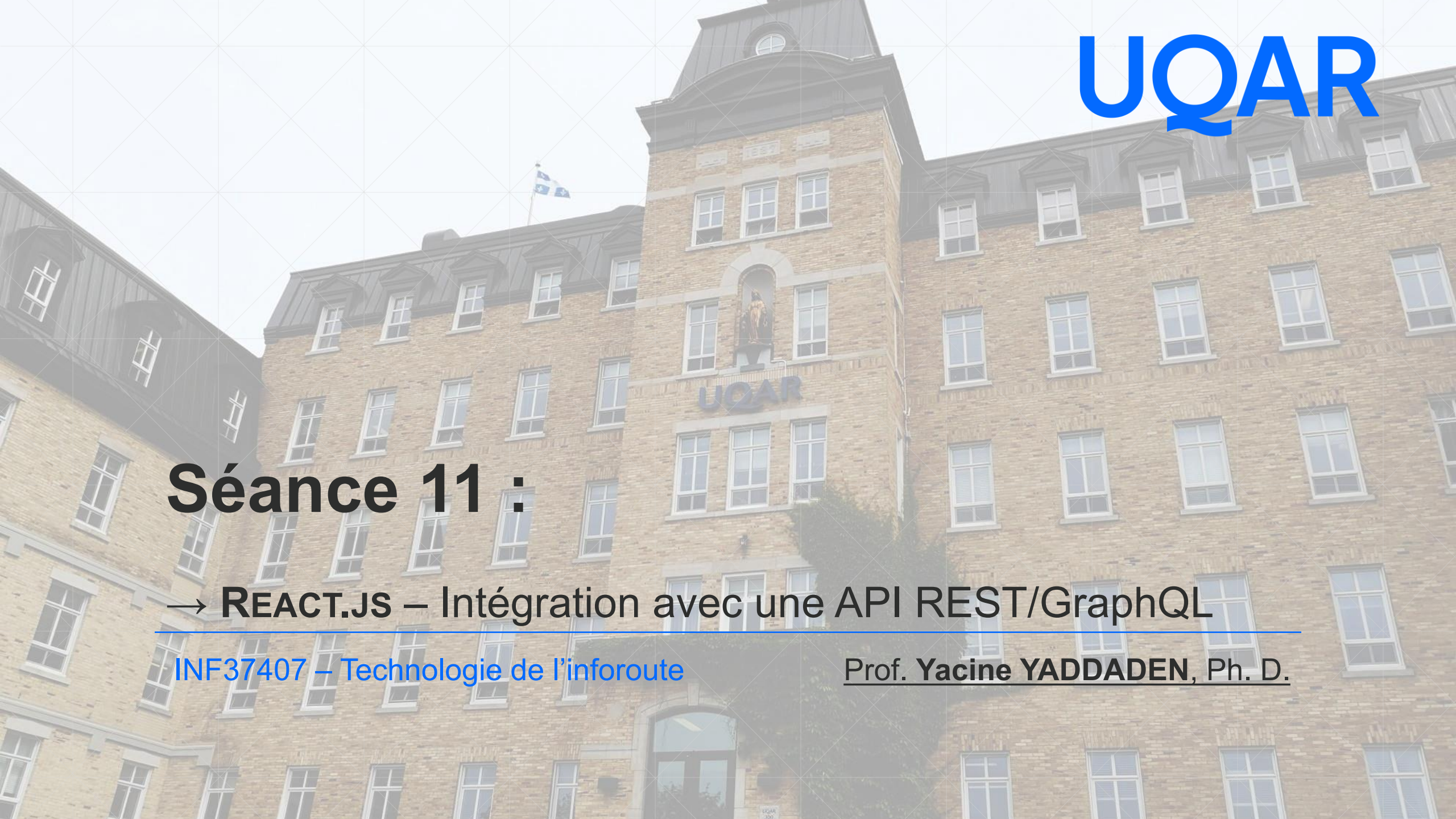

Séance 11 :

→ REACT.JS – Intégration avec une API REST/GraphQL

INF37407 – Technologie de l'inforoute

Prof. **Yacine YADDADEN**, Ph. D.

UQAR

Plan

1. Introduction & motivation
2. API REST — Théorie
3. API GraphQL — Théorie
4. Comparaison REST vs GraphQL
5. Outils utilisés : Redux Toolkit + RTK Query
6. Présentation Rick & Morty API
7. Exemple de code REST & GraphQL avec Redux Toolkit
8. Conclusion
9. Questions & Discussion

UQAR

Introduction & motivation

- **Pourquoi consommer des API dans React** ?
  - Aujourd'hui, **presque toutes les applications web** sont connectées à des données en ligne.
  - Les API permettent :
    - ☑ d'afficher des listes (produits, utilisateurs, films…)
    - ☑ de récupérer des informations dynamiques
    - ☑ d'interagir avec une base de données
    - ☑ de créer des interfaces vivantes et mises à jour en temps réel
- Pourquoi **REST** & **GraphQL** ?
  - **REST** : le standard historique, utilisé par 90% des API web.
  - **GraphQL** : nouvelle approche plus flexible et optimisée, utilisée par GitHub, Shopify, Meta, …
  - Les deux approches coexistent, et un développeur moderne doit savoir les intégrer.

Introduction & motivation

- **Pourquoi Redux Toolkit & RTK Query ?**
  - ☑ Simplifie énormément la gestion des requêtes
  - ☑ Cache intelligent → moins d'appels réseau
  - ☑ Hooks automatiques → code plus propre
  - ☑ Approche unifiée pour **REST et GraphQL**

UQAR

API REST — Théorie

→ C’est quoi une API REST ?

- REST = Representational State Transfer
- Architecture basée sur **des ressources** représentées sous forme d’URL
- Format principal : **JSON**
- Standard de facto pour toutes les applications web
- CRUD = **GET** → lire, **POST** → créer, **PUT/PATCH** → modifier, **DELETE** → supprimer
- **Bibliothèque utilisée** :
  - *fetch()* intégré
  - (Optionnel) **Axios** (*pnpm install axios*)
  - Pour Redux Toolkit → **RTK Query** (inclus dans RTK)

UQAR

API REST — Théorie

→ Comment consommer REST ?

- Envoyer une requête HTTP vers un endpoint
- Exemple : */api/character?page=1*
- Récupérer la réponse JSON
- Gestion :
  - Status HTTP, Erreurs, Loading, Pagination et Cache
- **Dans RTK Query :**

```
1  baseQuery: fetchBaseQuery({ baseUrl: '...' })
```

UQAR

API REST — Théorie

→ Exemple rapide REST

- Utiliser le *endpoint* → **GET** *https://rickandmortyapi.com/api/character?page=1*
- **Le résultat :**


```
{
  "info": {
    "count": 826,
    "pages": 42,
    "next": "https://rickandmortyapi.com/api/character?page=2",
    "prev": null
  },
  "results": [
    {
      "id": 1,
      "name": "Rick Sanchez",
      "status": "Alive",
      "species": "Human",
      "type": "",
      "gender": "Male",
      "origin": {
        "name": "Earth (C-137)",
        "url": "https://rickandmortyapi.com/api/location/1"
      },
      "location": {
```

UQAR

API GraphQL — Théorie

→ C’est quoi GraphQL ?

- Langage de requêtes pour API
- Un **seul *endpoint*** → */graphql*
- Le client décide quelles données obtenir
- Trois opérations :
  - **Query** : lire, **Mutation** : écrire et **Subscription** : temps réel.
- **Bibliothèque utilisée** :
  - *fetch()* intégré
  - Pour Redux Toolkit → **RTK Query** (inclus dans RTK)
  - (Optionnel) **Apollo** si besoin plus avancé (*pnpm install @apollo/client*)

UQAR

API GraphQL — Théorie

→ Comment consommer GraphQL ?

- Toujours envoyer une requête **POST**
- Body = un **JSON** contenant une chaîne de caractères **GraphQL**
- Exemple :

```
{
  "query": "{ characters { results { id name } } }"
}
```

UQAR

API GraphQL — Théorie

→ Exemple de requête GraphQL

```
query {
  characters(page: 1) {
    results {
      id
      name
      species
    }
  }
}
```

UQAR

Comparaison REST vs GraphQL

REST

- ☑ Simple
- ☑ Standard
- ☑ Intégré partout
- ☒ Récupère souvent trop de données

GraphQL

- ☑ Une seule requête = données optimisées
- ☑ Idéal pour interfaces complexes
- ☒ Overhead côté serveur
- ☒ Exige de structurer un schéma GraphQL

UQAR

Outils utilisés : Redux Toolkit + RTK Query

→ Pourquoi Redux Toolkit ?

- Facilite la consommation d'API REST & GraphQL
- RTK Query gère automatiquement :
  - **Caching → Mise en cache**
  - **Invalidation → Invalidation du cache**
  - **Re-fetch → Rafraîchissement automatique** ou **Requête automatique**
  - **Loading / error → États de chargement / erreur**
  - **…**
- Aucune librairie externe nécessaire.

UQAR

Outils utilisés : Redux Toolkit + RTK Query

→ RTK Query pour REST

- *baseQuery* = *fetchBaseQuery*
- *endpoints* = *builder.query* & *builder.mutation*
- *hooks* auto-générés
- parfait pour les APIs simples

UQAR

Outils utilisés : Redux Toolkit + RTK Query

→ RTK Query pour GraphQL

- Même logique
- *baseQuery* identique
- Autorise **POST** avec body GraphQL
- Pas besoin de **Apollo**

UQAR

Présentation Rick & Morty API

→ Rick & Morty REST API

- **URL :** *https://rickandmortyapi.com/api*
- Pas d'authentification
- *Endpoints* : */character*, */location*, */episode*
- Paramètres : *page*, *name*, *status*, *species*
- Idéal pour étudiants
- Très rapide

UQAR

Présentation Rick & Morty API

→ Rick & Morty REST API

- **URL :** *https://rickandmortyapi.com/graphql*
- Même contenu que REST
- *Endpoints* : */character*, */location*, */episode*
- Mais via Query & Mutation
- Très utile pour montrer la différence entre les deux styles

UQAR

Exemple de code avec Redux Toolkit
→ API **REST**

UQAR

Exemple de code avec Redux Toolkit

→ API **REST**

```
export const rickApi = createApi({
  reducerPath: "rickApi",
  baseQuery: fetchBaseQuery({
    baseUrl: "https://rickandmortyapi.com/api/",
  }),
  endpoints: (builder) => ({
    getCharacters: builder.query({
      query: (page = 1) => `character?page=${page}`,
    }),
  }),
});

export const { useGetCharactersQuery } = rickApi;
```

UQAR

Exemple de code avec Redux Toolkit

→ API **REST**

```jsx
export default function CharactersRest() {
  const { data, isLoading } = useGetCharactersQuery(1);

  if (isLoading) return <p>Loading...</p>;

  return (
    <div>
      <h2>REST Characters</h2>
      {data.results.map((c) => (
        <div key={c.id}>
          {c.name} – {c.status}
        </div>
      ))}
    </div>
  );
}
```

UQAR

Exemple de code avec Redux Toolkit

→ API **GraphQL**

UQAR

Exemple de code avec Redux Toolkit

→ API GraphQL

```
export const rickGraphqlApi = createApi({
  reducerPath: "rickGraphqlApi",
  baseQuery: fetchBaseQuery({
    baseUrl: "https://rickandmortyapi.com/graphql",
  }),
  endpoints: (builder) => ({
    getCharacters: builder.query({
      query: () => ({
        method: "POST",
        body: {
          query: `
            query {
              characters(page: 1) {
                results { id name species }
              }
            }
          `,
        },
      }),
    }),
  }),
});

export const { useGetCharactersQuery } = rickGraphqlApi;
```

UQAR

Exemple de code avec Redux Toolkit
→ API GraphQL

```jsx
export default function CharactersGraphql() {
  const { data, isLoading } = useGetCharactersQuery();

  if (isLoading) return <p>Loading ...</p>;

  const characters = data?.data?.characters?.results || [];

  return (
    <div>
      <h2>GraphQL Characters</h2>
      {characters.map((c) => (
        <div key={c.id}>
          {c.name} – {c.species}
        </div>
      ))}
    </div>
  );
}
```

UQAR

Conclusion

- **REST** : simple, plusieurs *endpoints*
- **GraphQL** : un *endpoint*, requêtes puissantes
- Redux Toolkit + RTK Query facilite l’intégration des deux

UQAR

Questions & Discussion

UQAR

Bibliographie

1. Templier, Thierry & Gougeon, Arnaud (2007). JavaScript pour le Web 2.0 Programmation objet, DOM, Ajax, Prototype, Dojo, Script.aculo.us, Rialto. Éditions Eyrolles.
2. Porteneuve, Christophe (2008). Bien développer pour le Web 2.0 : Bonnes pratiques Ajax. Éditions Eyrolles.
3. Engels, Jean (2012). HTML5 et CSS3 : Cours et exercices corrigés. Éditions Eyrolles.
4. Martin, Michel (2014). HTML5, CSS3 & jQuery : Créez votre premier site web. Éditions Pearson.