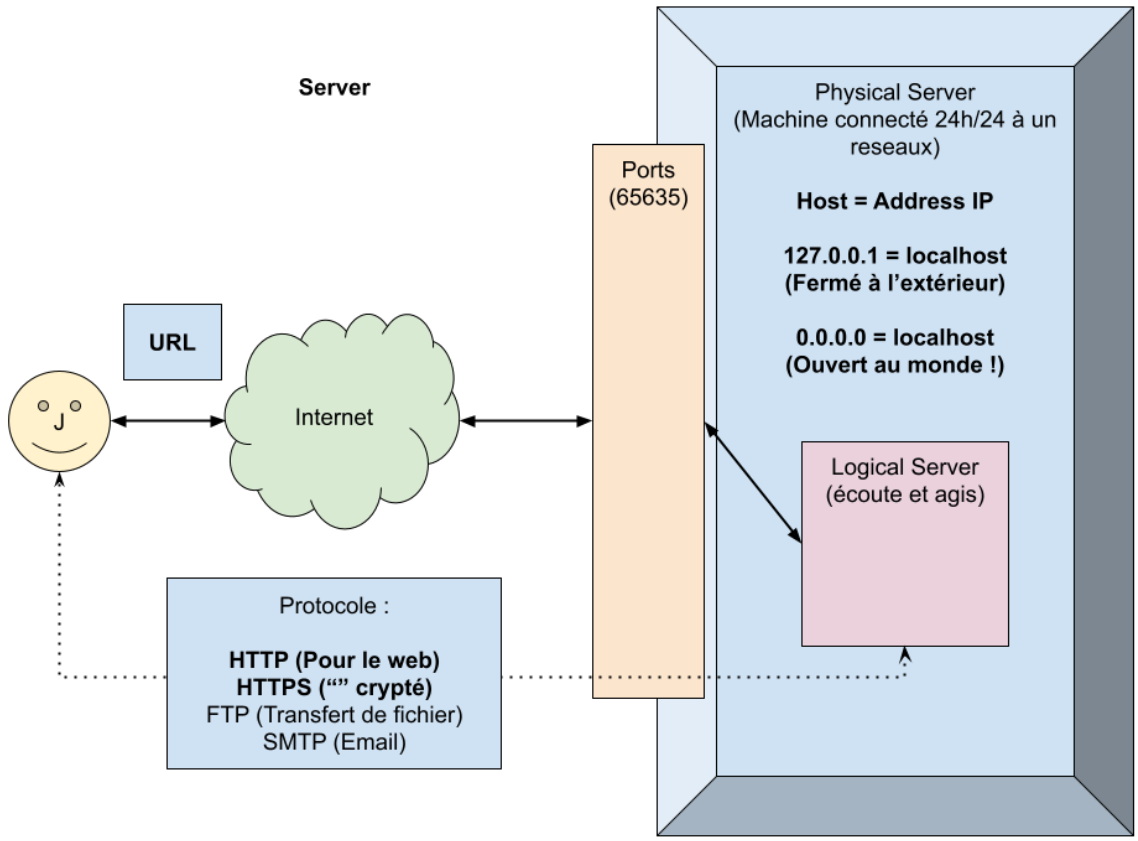
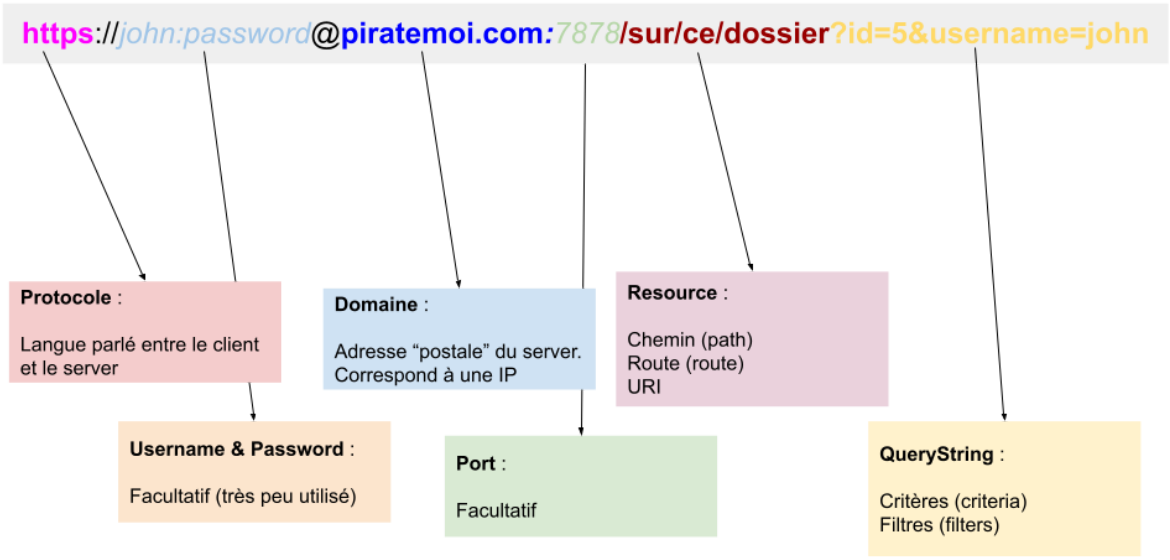
Découvrir fastify

Fastify est une librairie nodejs permettant de créer des server logique HTTP.

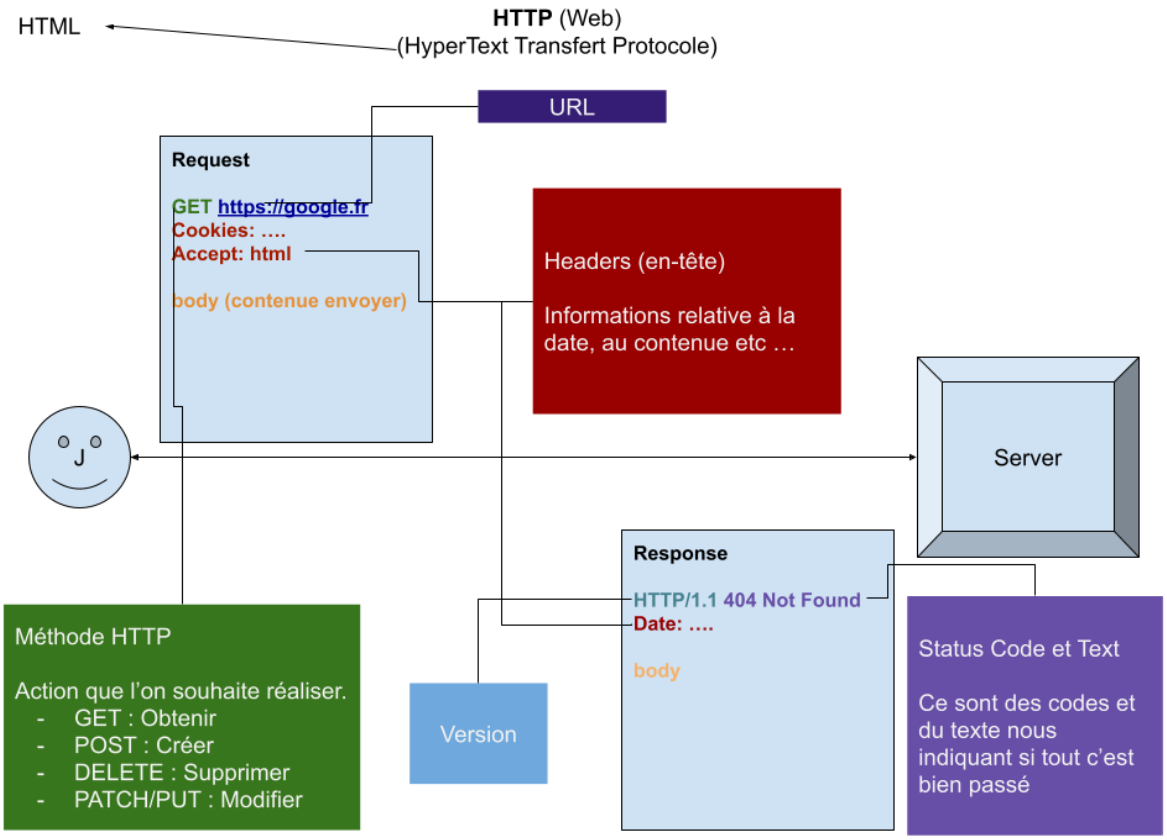
Les serveurs



Les urls



Le protocol HTTP



Fastify

Nodejs à été conçu dans l'optique de créer des serveurs logique HTTP. Pour cela il existe plusieurs librairie :

- **Express** : La toute première librairie créé pour faires de serveurs HTTP en nodejs
- **Fastify** : Plus récente et plus rapide, elle s'inspire d'express et supporte typescript, c'est celle que nous allons utilisé dans ce cours.
- **NestJS** : Framework complet basé sur le paradigm MVC, plus difficile d'accès mais bien plus complet !

Installer fastify

Pour installer fastify :

```
npm i fastify
```

Créer une application fastify

Afin de pouvoir créer notre premier server, nous devons créer une application fastify :

```
import fastify from 'fastify'

// Création d'une application fastify
const app = fastify()
```

Lancer le serveur HTTP (listen)

Afin de démarrer notre serveur nous devons demander à notre application fastify d'écouter un port et un host :

```
// Écoute un port et un host
app.listen({ port: 5353, host: '127.0.0.1' }, () => {
  // Affiche un message dans la console nous indiquant que le serveur est démarré
  console.log("Le serveur http est prêt sur l'address : http://127.0.0.1:5353")
})
```

> ATTENTION : Il est fortement conseillé de mettre le port et le host dans des variales d'environements.

Ajouter des routes

De base un serveur http ne fais rien du tout, pour ajouter des « actions » à notre serveur, nous allons devoir lui ajouter des route. Par éxemple je souhaiterais que mon serveur me dise bonjour :

```
// On utilise l'application fastify pour ajouter des routes. Chaque route possède une méthode HTTP et
// un chemin (Resource, Path) :
app.get('/', () => {
  return 'Bonjour les amis'
})
app.post('/', () => { ... })
app.delete('/', () => { ... })
```

Les paramètres de routes

Il est possible en fastify de créer des routes dites "dynamique". Ce sont des routes qui accépte un ou plusieurs paramètre dans leur URI :

```
// Ici nous déclarons une route « dynamique ». Et nous devons spécifier via un
générique
// le type des nos params :
app.get<{ Params: { name: string } }>('/bonjour/:name', request => {
  // Nous utilisons la request pour récupérer le contenu du paramètre
  // « name » :
  const name = request.params.name

  return `Bonjour ${name}`
})
```

Pour plus de lisibilité il est tout à fait possible de créer notre propre type :

```
/**
 * On déclare le type de nos params
 */
type SalutParams = {
  name: string
}

/**
 * On déclare la route :
 */
app.get<{ Params: SalutParams }>('/bonjour/:name', request => {
  // Nous utilisons la request pour récupérer le contenu du paramètre
  // « name » :
  const name = request.params.name

  return `Bonjour ${name}`
})
```

On peut aussi créer un type pour l'intégralité du générique :

```
/**
 * On déclare le type de nos params
 */
type SalutParams = {
  name: string
}

/**
 * On déclare un type pour notre route
 */
type SalutRoute = {
  Params: SalutParams
}

/**
 * On déclare la route :
```

```
 */
app.get<SalutRoute>('/bonjour/:name', request => {
  // Nous utilisons la request pour récupérer le contenu du paramètre
  // « name » :
  const name = request.params.name

  return `Bonjour ${name}`
})
```

Généralement la solution la plus répandu est un mélange des 2 solutions plus haut :

```
/**
 * Création d'un type pour notre route :
 */
type SalutRoute = {
  Params: {
    name: string
  }
}

/**
 * On déclare la route :
 */
app.get<SalutRoute>('/bonjour/:name', request => {
  // Nous utilisons la request pour récupérer le contenu du paramètre
  // « name » :
  const name = request.params.name

  return `Bonjour ${name}`
})
```

Request & Reply

En fastify il éxiste 2 paramètre envoyé à la fonction de traitement d'une route :

- La request
- La Reply

Ces deux paramètres permettent de récupérer les informations relative à la Request envoyé par le client et personnaliser la réponse du serveur (reply).

Il est possible de récupérer tout un tas d'informations ainsi que de personnaliser notre réponse en utilisant ces deux objets correctement

> Nous avons dèja vu plus haut comment utiliser la request ! Nous avons aussi utiliser un générique afin de typer les données de la request. Dans ce généric nous pouvons aussi personnaliser d'autre données comme les Querystring, Body, Headers, Params mais aussi la Reply : https://www.fastify.io/docs/latest/Reference/TypeScript/#getting-started

Les query string

Il est aussi possible de « typer » et manipuler des query string (filtres) :

```
/**
 * Création d'un type pour ma route salutation
```

```
 */
type SalutationRoute = {
  Params: {
    name: string
  }
  Querystring: {
    upcase?: boolean
  }
}

/**
 * On déclare la route :
 */
app.get<SalutationRoute>('/salutation/:name', request => {
  // Nous utilisons la request pour récupérer le contenu du paramètre
  // « name » :
  const name = request.params.name

  // On récupére le filtre "upcase"
  const upcase = request.query.upcase

  return upcase ? `Bonjour ${name}`.toUpperCase() : `Bonjour ${name}`
})
```

Personaliser le status de réponse

Il est parfois essentiel que notre serveur retourne le bon status. Par éxemple le status `200 Ok` est utilisé lorsque tout ce passe bien, cependant si une erreur survient notre serveur doit répondre le bon status :

- `400` : Une erreur c'est produit côté client
- `500` : Une erreur c'est produit sur le serveur
- `404` : La page n'éxiste pas

Vous retrouverez la liste des stautus http ici :

Liste des status HTTP

Pour personnaliser le status :

```
app.get('/test', (request, response) => {
  // Pour personnaliser le status, nous utilisons la réponse :
  response.code(400)

  return 'Une erreur est survenue'
})
```

Le format JSON

Lorsque l'on souhaite échanger des données entre un client et un serveur, le format de prédilection est JSON. Simple, lisible, léger c'est le plus répandu au monde (anciennement c'était le `xml`).

Voici un exemple de fichier json :

Les string

```
"Coucou les amis"
```

> Très similaire à javascript sauf attention, uniquement les guillemets double sont valide !

Les numbers

```
12
12.5
-56
```

Les boolean

```
true
false
```

Les array

```
["coucou", "les", "amis", 12, false]
```

Les objets

```
{
  "nom": "Doe",
  "prenom": "John",
  "age": 32
}
```

> En json les guillemets sont obligatoire pour les clefs de vos objets !

Exemple de json une fiche de présence :

```
{
  "AMIN Ali": {
    "email": "....",
    "presence": {
      "lundi": {
        "matin": "P",
        "après midi": "P"
      },
      "mardi": {
        "matin": "P",
        "après midi": "P"
      },
      "mercredi": {
        "matin": "P",
        "après midi": "P"
      }
    }
```

```
  }
}
```

Transmettre des données à notre serveur

Lorsque l'on fais des requêtes HTTP à un serveur nous dévons spécifier une méthode HTTP :

- GET : Obtenir
- POST : Créer
- DELETE : Effacer
- PATCH : Modifier une partie
- PUT : Modifier l'intégralité

Certaines de ses actions pour s'éxécuter doivent envoyer de la données à notre serveur ! C'est le cas des actions `POST`, `PATCH` et `PUT`.

Pour envoyer des données en utilisant le format JSON il faut, dans notre requête HTTP spécifier un en-tête `Content-Type`. Cet en-tête http accépte un `MIME Type` qui est `application/json`

```
POST http://monserver.com/articles
Content-Type: application/json

{
  "title": "Mon voyage en espagne",
  "description": "Super voyage ...",
  "content": "lorem ipsum dolor sit amet ..."
}
```

Récupérer ses données dans notre route fastify

Pour récupérer les données json envoyé par un client il faut utiliser le `request.body` (par éxemple, je souhaite récupérer le title envoyé en json : `request.body.title`).

Attention ! En typescript ce `request.body` doit être typé !

```
import fastify from 'fastify'

const app = fastify()

// Type contenant la définition des Params, Querystring mais aussi le body
type CreateArticleRoute = {
  Body: {
    title: string
    description: string
    content: string
  }
}

// Création d'une route post pour ajouter un nouvelle article
app.post<CreateArticleRoute>('/articles', request => {
  // Récupérer le titre de mon article
  const title = request.body.title

  // Enregistrer l'article dans une base de données (par éxemple MongoDB) ...
```

```
})

app.listen(....)
```

Comprendre le générique envoyé à la route

Un route à besoin d'un type générique afin de définir ce que contient la request et plus spécifiquement les éléments suivant :

```
// Type permettant de dire à typescript ce que contient une request
type MaRoute = {
  // Définie ce ques contient les Params de la route
  Params: {
    id: string
  }
  // Définie ce que contient les Querystring de la route
  Querystring: {
    orderBy: string
  }
  // Définie ce que contient le body de la route
  Body: {
    title: string
  }
  // Définie les en-tête http que doit contenir notre request
  Headers: {
    'Content-Type': string
  }
}

app.get<MaRoute>('/test', request => {
  // Ici request.params doit contenir MaRoute['Params']
  console.log(request.params) // { id: '...' }
  // request.query doit contenir MaRoute['Querystring']
  console.log(request.query) // { orderBy: '...' }
  console.log(request.body) // { title: '...' }
  console.log(request.headers) // { "Content-Type": '.....' }
})
```

Les plugins

Dans une véritable API Web, il est possible d'avoir un très grand nombres routes (parfois même des centaines). On ne vas pas mettre toutes les routes dans le même fichier. Pour séparer nos routes en plusieurs fichier, fastify à mis en place un système de « plugin » (Des petites extensions).

Pour pouvoir utiliser les plugins nous avons besoin d'installer un package :

```
npm i fastify-plugin
```

Généralement, les différentes routes de notre applications sont rangé dans un dossier : `src/routes`

Dans ce dossier nous allons pouvoir créer nos premiers plugins :

```
// src/routes/users.ts

/**
 * Un plugin est une fonction asynchrone recevant l'application fastify :
 */
export default async function userRoute(app: FastifyInstance) {
  /**
   * Grâce à l'application fastify, nous pouvons facilement
   * déclarer des routes
   */
  app.get('/users', async () => {
    return { ... }
  })
}
```

Maintenant que nous avons notre premier plugin, nous pouvons l'assembler (ou le connécter) dans notre fichier principal :

```
// src/index.ts
import fastify from 'fastify'
import fp from 'fastify-plugin'
import userRoute from './routes/users'

/**
 * Création d'un app fastify
 */
const app = fastify()

/**
 * Maintenant nous pouvons connécter notre plugin :
 */
app.register(fp(userRoute))
```

La décoration

Fastify offre la possibilité d'enregistrer dans l'application des données et des fonctions. Cela vous nous permettre de transmettres des informations entre les différents plugins. Pour cela nous pouvons décorer l'application :

```
// src/routes/users.ts

/**
 * Création du plugin
 */
export default async function userRoute(app: FastifyInstance) {
  /**
   * Nous pouvons décorer l'application :
   */
  app.decorate('collection', 'user_collection')
}
```

Maintenant grace à la décoration nous pouvons récupérer la variable `collection` n'importe ou !

```ts
// src/routes/pizzas.ts

/**
 * Création du plugin
 */
export default async function pizzaRoute(app: FastifyInstance) {
  /**
   * Route récupérer les pizzas
   */
  app.get('/pizzas', async () => {
    // Je peux récupére la collection :
    app.collection // 'user_collection'
  })
}
```