Swift côté serveur - démo

- Créer un nouvel projet

```
$ vapor new projetdemo
```

- Builder

```
$ cd projetdemo
$ vapor build
```

- Lancer

```
$ vapor run
```

- Elements importants
  - App.json
  - Package.swift
  - Config/servers.json
  - Sources*App*main.swift

- Main.swift
  - **Droplet** permet de gérer les requêtes http, configurer le serveur web…
  - **droplet.get** prend en param l'objet httprequest et doit retourner un objet qui se conforme au protocole RessourceRepresentable (qui peut se convertir en réponse http). On peut retourner directement un String car Vapor a rendu l'objet String conforme à ce protocole.
  - **droplet.run()** - lance le serveur web

```
vapor build && vapor run
```

  - Si on veut répondre un JSON on a qu'à créer un objet JSON et dans le param node passer un dictionnaire d'éléments à retourner.

```
return try JSON(node: ["message": "Hello from Vapor!"])
```

- **Routes**:
  - on met l'url dans le param du get `/hello`

```
drop.get("hello") { request in
      return try JSON(node: [
          "message": "Hello, again!"
      ])
}
```

  - pour imbriquer des chemins on passe des multiples params
    `/hello/there`

```
drop.get("hello", "there") { request in
      return try JSON(node: [
          "message": "Hello, again and again, again !"
      ])
}
```

  - pour passer un paramètre on met le type du param dans le path
    `/bieres/12`

```
drop.get("bieres", Int.self) { request, bieres in
      return try JSON(node: [
          "message": "Une bière pour toi il ne reste plus que \(beers - 1) ..."
      ])
  }
```

  - Création d'une route qui retourne une liste d'utilisateurs
    `/users`

```
drop.get("users") { req in
    return try JSON(node: [
                ["name": "Tim", "beer": 12],
                ["name": "Craig", "beer": 39],
                ["name": "Phil", "beer": 52]
        ])
```

```
}
```

- Création d'un model
  - Création d'un fichier User.swift dans **Sources***App***Models/**

```
touch Sources/App/Models/Users.swift
```

```
import Foundation
import Vapor

struct User: Model {
    var exists: Bool = false
    var id: Node?
    let name: String
    let beer: Int?

    init(name: String, beer: Int?) {
        self.name = name
        self.beer = beer
    }

    //Intializable from a Node
    init(node: Node, in context: Context) throws {
        id = try node.extract("id")
        name = try node.extract("name")
        beer = try node.extract("beer")
    }

    //Node represantable
    func makeNode(context: Context) throws -> Node {
        return try Node(node: ["id": id,
                               "name": name,
                               "beer": beer])
    }

    //Database preparation
```

```
    static func prepare(_ database: Database) throws {
        try database.create("users") {users in
            users.id()
            users.string("name")
            users.int("beer", optional: true)
        }
    }

    static func revert(_ database: Database) throws {
        try database.delete("users")
    }
}
```

Pour se conformer au protocole Model on doit ajouter deux propriétés:

- **id**: le type Node? qui est `nil` tant que le modèle n'est pas enregistré dans la base de donné (ou récupéré de la base).
- **exists**: qui informe si l'instance a été récupérée de la base de donnée ou pas.

et quelques fonctions:

- pour initialiser un objet à partir d'un Node (**init(node..**),
- pour représenter l'objet comme un Node (**makeNode**),
- pour préparer la base de donnée en créant la table dans laquelle l'objet sera enregistré (**prepare**),
- et pour supprimer la table de l'objet (créée par le prepare) (**revert**).

- on remplace dans la route pour utiliser les objets

```
drop.get("users") { req in
    return try JSON(node: [
                User(name: "Tim", beer: 12),
                User(name: "Craig", beer: 39)["name": "", "beer": 39],
                 User(name: "Phil", beer: 52)
        ])
}
```

- Ajout du provider Postgres

`https://github.com/vapor/postgresql-provider`

- Dans `Package.swift` ajout de a la ligne

```
        .Package(url: "https://github.com/vapor/postgresql-provider",
majorVersion: 1, minor: 1)
```

- Utiliser Postgres dans `main.swift`

```
import VaporPostgreSQL

let drop = Droplet(
    preparations: [User.self],
    providers: [VaporPostgreSQL.Provider.self]
)
```

- Pour builder le driver il faut installer Postgres, au moins lib-dev (sous linux/docker si l'installation est ailleurs)

```
apt-get install libpq-dev
```

- Configuration de la connexion a la base de données

  Créer le fichier `Config/secrets/postgresql.json`

```
{
    "host": "172.17.0.1",
    "user": "test",
    "password": "testpwd",
    "database": "testdb",
    "port": 5432
}
```

- Ajout d'une route qui crée quelques utilisateurs

```
drop.get("createusers") { req in
```

```
    var user =  User(name: "Tim", beer: 12)
    try user.save()
    user =  User(name: "Craig", beer: 39)
    try user.save()
    user =  User(name: "Phil",  beer: nil)
    try user.save()
    return try JSON(node: User.all().makeNode())
}
```

- Modifier la route `/users` pour récupérer tous les utilisateurs et `/user/<id>` pour récupérer utilisateur

```
drop.get("users") { req in
    return try JSON(node: User.all().makeNode())
}

drop.get("user", Int.self) { req, userID in
    guard let user = try User.find(userID) else {
        throw Abort.notFound
     }
    return try user.makeJSON()
}
```

- creation d'une route pour créer un user via une requête POST

```
drop.post("user") { req in
    var user = try User(node: req.json)
    try user.save()
    return try user.makeJSON()
}
```

- creation d'une router pour appeler une API de récupération de temp

```
drop.get("temperature") { req in
```

```
    return "Montpellier temperature is: <todo> °C"
}
```

- appeler une API

```
drop.get("temperature") { req in
    //service URL
    let URL = "http://api.openweathermap.org/data/2.5/weather"

    //the query data
    let QUERY =  "Montpellier,FR"
    //the APPID to use with openweathermap service
    let APIKEY = "6a7e58c89803de6f97e8aa4581b41d8e"

    //make the query with passing the query params
    let response = try drop.client.get(URL, query: ["q": QUERY, "APPID":APIKEY,
"units":"metric"])

    print(response)

    return "La temperature à Montpellier est : <todo> °C"
}
```

- regarder le JSON retourné

```
{
  "coord": {
    "lon": 3.88,
    "lat": 43.61
  },
  "weather": [
    {
      "id": 800,
      "main": "Clear",
      "description": "clear sky",
      "icon": "01d"
    }
```

```
  ],
  "base": "stations",
  "main": {
    "temp": 19.73,
    "pressure": 1013,
    "humidity": 64,
    "temp_min": 16,
    "temp_max": 21
  },
  "visibility": 10000,
  "wind": {
    "speed": 5.1,
    "deg": 180
  },
  "clouds": {
    "all": 0
  },
  "dt": 1495128600,
  "sys": {
    "type": 1,
    "id": 5588,
    "message": 0.0157,
    "country": "FR",
    "sunrise": 1495080944,
    "sunset": 1495134407
  },
  "id": 2992166,
  "name": "Montpellier",
  "cod": 200
}
```


- on veut juste garder `main/temp`

```
guard let temperature = response.data["main", "temp"]?.string else {
       return "Temperature inconnue pour Montpellier"
}

return "Montpellier temperature is: \(temperature) °C"
```