Routage

Le routage fait référence à la manière dont les points de terminaison (URI) d'une application répondent aux demandes des clients.

Pour définir le routage à l'aide des méthodes de l'objets Express app qui correspondent aux méthodes HTTP ;

app.get() pour gérer les requêtes GET

app.post pour gérer les requêtes POST.

app.all() pour gérer toutes les méthodes HTTP

app.use() pour spécifier le middleware comme fonction de rappel

L'application "écoute" les requêtes qui correspondent à la ou aux routes et méthodes spécifiées, et lorsqu'elle détecte une correspondance, elle appelle la fonction de rappel spécifiée.

En fait, avec plusieurs fonctions de rappel, il est important de fournir ***next*** un argument à la fonction de rappel, puis d'appeler next() dans le corps de la fonction pour transférer le contrôle au rappel suivant.

```
const express = require('express')
const app = express()

app.get('/', (req, res) => {
  res.send('Bonjour')
})
```

Les méthodes de routage

Une méthode de route est dérivée de l'une des méthodes HTTP et est attachée à une instance de la classe express.

Exemple de routes définies pour les méthodes GET et POST vers la racine de l'application.

```
// GET route
app.get('/', (req, res) => {
  res.send('GET vers homepage')
})

// POST route
app.post('/', (req, res) => {
  res.send('POST vers homepage')
})
```

Express prend en charge les méthodes qui correspondent à toutes les méthodes de requête HTTP : get, post, …

Une méthode de routage spéciale, app.all(), est utilisée pour charger les fonctions middleware sur un chemin pour toutes les méthodes de requête HTTP.

Exemple, les requêtes vers la route "/secret", GET, POST, PUT, DELETE ou toute autre méthode de requête HTTP prise en charge.

```
app.all('/secret', (req, res, next) => {
  console.log('Accès a la section secret ...')
  next()
})
```

Chemins de routage

Les chemins de routage, associés à une méthode de demande, définissent les points de terminaison auxquels les demandes peuvent être effectuées. Les chemins de routage peuvent être des chaînes, des modèles de chaîne ou des expressions régulières.
Les caractères ?, +, * et () sont des sous-ensembles de leurs homologues d'expression régulière. Le trait d'union (-) et le point (.) sont interprétés littéralement par des chemins basés sur des chaînes.
Le caractère dollar ($) échappe dans une chaîne de chemin, placez-le entre ([et]).
Exemple : la chaîne de chemin d'accès pour les requêtes à " /data/$book" serait " /data/([\$])book".

- **Exemples de chemins de routage basés sur des chaînes.**

Chemin d'accès correspondra aux requêtes vers /about.

```
app.get('/about', (req, res) => {
  res.send('about')
})
```

Chemin d'accès correspondra aux requêtes vers /random.text.

```
app.get('/random.text', (req, res) => {
  res.send('random.text')
})
```

- **Exemples de chemins de routage basés sur des modèles de chaîne.**

Chemin d'accès correspondra à acd et abcd.

```
app.get('/ab?cd', (req, res) => {
  res.send('ab?cd')
})
```

Chemin d'accès correspondra à abcd, abbcd, abbbcd, etc.

```
app.get('/ab+cd', (req, res) => {
  res.send('ab+cd')
})
```

Chemin d'accès correspondra à abcd, abxcd, abRANDOMcd, ab123cd, etc.

```
app.get('/ab*cd', (req, res) => {
  res.send('ab*cd')
})
```

Chemin d'accès correspondra à /abe et /abcde.

```
app.get('/ab(cd)?e', (req, res) => {
  res.send('ab(cd)?e')
})
```

- **Exemples de chemins de routage basés sur des expressions régulières :**

Chemin d'accès correspondra à tout ce qui contient un "a".

```
app.get(/a/, (req, res) => {
  res.send('/a/')
})
```

Chemin d'accès correspondra à butterfly et dragonfly, mais pas butterflymanà , dragonflyman, etc.

```
app.get(/.*fly$/, (req, res) => {
  res.send('/.*fly$/')
})
```

Paramètres d'itinéraire

Les paramètres de route sont des segments d'URL nommés qui sont utilisés pour capturer les valeurs spécifiées à leur position dans l'URL.

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

Pour définir des itinéraires avec des paramètres d'itinéraire, spécifiez simplement les paramètres d'itinéraire dans le chemin de l'itinéraire comme indiqué ci-dessous.

```
app.get('/users/:userId/books/:bookId', (req, res) => {
  res.send(req.params)
})
```

Le nom des paramètres de route doit être composé de « caractères de mot » ([A-Za-z0-9_]).

Puisque le trait d'union (-) et le point (.) sont interprétés littéralement, ils peuvent être utilisés avec les paramètres d'itinéraire à des fins utiles.

```
Route path: /flights/:from-:to
Request URL: http://localhost:3000/flights/LAX-SFO
req.params: { "from": "LAX", "to": "SFO" }
Route path: /plantae/:genus.:species
Request URL: http://localhost:3000/plantae/Prunus.persica
req.params: { "genus": "Prunus", "species": "persica" }
```

Pour avoir plus de contrôle sur la chaîne exacte qui peut être mise en correspondance par un paramètre de route, on pourra ajouter une expression régulière entre parenthèses (()) :

```
Route path: /user/:userId(\d+)
Request URL: http://localhost:3000/user/42
req.params: {"userId": "42"}
```

Gestionnaires d'itinéraire

On peut fournir plusieurs fonctions de rappel qui se comportent comme un middleware pour gérer une requête. La seule exception est que ces rappels peuvent invoquer next('route') pour contourner les rappels de route restants. Utilisons ce mécanisme pour imposer des conditions préalables à un itinéraire, puis passer le contrôle aux itinéraires suivants s'il n'y a aucune raison de continuer avec l'itinéraire actuel.

Les gestionnaires de route peuvent se présenter sous la forme d'une fonction, d'un tableau de fonctions ou d'une combinaison des deux. Une seule fonction de rappel peut gérer une route.

Exemple :

```
app.get('/exemple/a', (req, res) => {
  res.send('Bonjour A')
})
```

Plusieurs fonctions de rappel peuvent gérer une route (spécifier l'objet next).

Exemple :

```
app.get('/exemple/b', (req, res, next) => {
  console.log('La réponse sera envoyé à la fonction suivante ...')
  next()
}, (req, res) => {
  res.send('Bonjour B')
})
```

Un tableau de fonctions de rappel peut gérer une route.

```
const cb0 = function (req, res, next) {
  console.log('CB0')
  next()
}

const cb1 = function (req, res, next) {
  console.log('CB1')
  next()
}

const cb2 = function (req, res) {
  res.send('Bonjour C')
}

app.get('/example/c', [cb0, cb1, cb2])
```

Une combinaison de fonctions indépendantes et de tableaux de fonctions peut gérer une route.

```
const cb0 = function (req, res, next) {
  console.log('CB0')
  next()
}

const cb1 = function (req, res, next) {
  console.log('CB1')
  next()
}

app.get('/example/d', [cb0, cb1], (req, res, next) => {
  console.log('La réponse sera envoyé à la fonction suivante ...')
  next()
}, (req, res) => {
  res.send('Bonjour D')
})
```

Méthodes de réponse

Les méthodes sur l'objet de réponse (res) dans le tableau suivant peuvent envoyer une réponse au client et terminer le cycle demande-réponse. Si aucune de ces méthodes n'est appelée à partir d'un gestionnaire de route, la demande du client sera laissée en suspens.

Méthode	Description
res.download()	Invite un fichier à télécharger.
res.end()	Terminez le processus de réponse.
res.json()	Envoyez une réponse JSON.
res.jsonp()	Envoyez une réponse JSON avec prise en charge de JSONP.
res.redirect()	Rediriger une demande.
res.render()	Effectuez le rendu d'un modèle de vue.
res.send()	Envoyez une réponse de différents types.
res.sendFile()	Envoyer un fichier sous forme de flux d'octets.
res.sendStatus()	Définissez le code d'état de la réponse et envoyez sa représentation sous forme de chaîne en tant que corps de la réponse.

app.route()

Il permet de créer des gestionnaires d'itinéraire chaînés pour un chemin d'itinéraire. Étant donné que le chemin est spécifié à un emplacement unique, la création d'itinéraires modulaires est utile, tout comme la réduction de la redondance et des fautes de frappe.

Exemple de gestionnaires d'itinéraires chaînés définis à l'aide de app.route().

```
app.route('/book')
  .get((req, res) => {
    res.send('Get')
  })
  .post((req, res) => {
    res.send('Add')
  })
  .put((req, res) => {
    res.send('Update')
  })
```

express.Router

express.Router permet de créer des gestionnaires de routage modulaires et montables. Une instance est un middleware complet et un système de routage ; On l'appelle souvent une « mini-application ».

L'exemple suivant crée un routeur en tant que module, y charge une fonction middleware, définit certaines routes et monte le module de routeur sur un chemin dans l'application principale.

Créez un fichier de routeur nommé birds.js dans le répertoire de l'application :

```
const express = require('express')
const router = express.Router()

// middleware
router.use((req, res, next) => {
  console.log('Time: ', Date.now())
  next()
})
// home page route
router.get('/', (req, res) => {
  res.send('Birds')
})
// about route
router.get('/about', (req, res) => {
  res.send('About')
})

module.exports = router
```

Ensuite, chargez le module routeur dans l'application :

```
const birds = require('./birds')

// ...

app.use('/birds', birds)
```

L'application sera désormais en mesure de gérer les requêtes vers /birds et /birds/about, ainsi que d'appeler la fonction middleware timeLog spécifique à la route.

Exercice

QCM interactif :

Cree un QCM avec 5 questions, qui auront chacune 3 réponses possibles. Y ajouter un bouton « Corriger » qui envoie le formulaire au serveur.

Faire en sorte que initialement seule la première question soit proposée. Les bonnes réponses sont cachées sur le serveur, et la question suivante apparait lorsque la bonne réponse a été donnée. Tant que la réponse n'est pas la bonne, l'utilisateur peu ressayer.

Associer une route à chaque question, pensez à utiliser des routes dynamiques.