Oliver Zenger 13.05.2024

M295 Backend Applikation Realisieren

learnyounode

Das **fs-Modul** in Node.js ermöglicht die Interaktion mit dem Dateisystem. Es bietet Funktionen zum Lesen, Schreiben, Erstellen und Löschen von Dateien.

- fs.readFile(): Lesen einer Datei asynchron.
- fs.readFileSync(): Lesen einer Datei synchron.
- fs.writeFile(): Schreiben in eine Datei asynchron.
- fs.writeFileSync(): Schreiben in eine Datei synchron.
- fs.existsSync(): Überprüfen, ob eine Datei oder ein Verzeichnis existiert.
- fs.mkdir(): Erstellen eines Verzeichnisses.
- fs.unlink(): Löschen einer Datei.
- fs.readdir(): Lesen eines Verzeichnisses.

Das **path-Modul** in Node.js ermöglicht die Arbeit mit Datei- und Verzeichnispfaden, wie das Normalisieren und Verbinden von Pfaden.

1. path.join(): Verbindet Pfadsegmente zu einem vollständigen Pfad. Dies ist hilfreich, um plattformunabhängige Pfade zu erstellen, da es automatisch den richtigen Trennungszeichen für das Betriebssystem hinzufügt.
2. path.resolve(): Löst einen absoluten Pfad aus einem oder mehreren Pfadsegmenten auf. Es wird verwendet, um den vollständigen Pfad einer Datei oder eines Verzeichnisses zu erhalten.
3. path.basename(): Gibt den letzten Teil eines Pfads zurück, normalerweise den Dateinamen.
4. path.dirname(): Gibt den Verzeichnisnamen eines Pfads zurück.
5. path.extname(): Gibt die Dateierweiterung eines Pfads zurück.
6. path.normalize(): Normalisiert einen Pfad, indem Doppelte Schrägstriche und Punkte entfernt werden, um einen konsistenten Pfad zu erzeugen.

In Node.js kannst du einen **HTTP-Client** erstellen, um Anfragen an Server zu senden und Antworten zu empfangen. Du verwendest das integrierte http-Modul, um dies zu tun.

Callbacks:

Ein Callback in JavaScript ist eine Funktion, die einer anderen Funktion als Argument übergeben wird und später aufgerufen wird, wenn eine bestimmte Aktion abgeschlossen ist.

```
function duplicate(number, callback) {
  callback(number * 2);
}

duplicate(5, function(result) {
  console.log('Das Ergebnis ist:', result);
});
```

```
function selection(number, positive, negative) {
  if (number >= 0) {
    positive(number);
  } else {
    negative(number);
  }
}

selection(-1, function(number) {
  console.log("positive number", number);
}, function(number) {
  console.log("negative number", number);
})

selection(1, function(number) {
  console.log("positive number", number);
}, function(number) {
  console.log("negative number", number);
})
```

Promises:

Promises sind ein Muster in JavaScript, das es ermöglicht, asynchrone Operationen einfacher und lesbarer zu handhaben. Sie repräsentieren den Erfolg oder das Scheitern einer asynchronen Operation und ermöglichen es, sequenziellen Code für asynchrone Abläufe zu schreiben.

```
function upperString(str) {
  return new Promise((resolve, reject) => {
    if(str == '') {
      reject('Empty string')
    } else {
      let upperStr = str.toUpperCase();
      resolve(upperStr);
    }
  })
}

upperString('')
.then(result => console.log(result))
.catch(error => console.log(error));
```

Async/Await in JavaScript

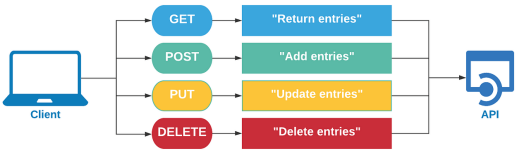
Async/Await: Async/Await baut auf Promises auf und bietet eine syntaxfreundlichere Möglichkeit, asynchrone Operationen zu behandeln. Durch die Verwendung des async-Schlüsselworts wird eine Funktion als asynchron markiert, und innerhalb dieser Funktion kann das await-Schlüsselwort verwendet werden, um auf den Abschluss einer Promise zu warten.

```
// Definition einer Funktion, die eine Verzögerung simuliert
async function simuliereVerzoegerung(ms) {
  // Verzögerung um die angegebene Zeit in Millisekunden
  await new Promise(resolve => setTimeout(resolve, ms));
}

// Definition einer Funktion, die zwei Zahlen addiert, nach einer
Verzögerung
async function addiereNachVerzoegerung(a, b, ms) {
  // Warten auf die Beendigung der Verzögerung
  await simuliereVerzoegerung(ms);
  // Durchführung der Addition nach der Verzögerung
  const sum = a + b;
  // Ausgabe der Summe in der Konsole
  console.log('Das Ergebniss ist ', sum);
}

// Aufruf der Funktion addiereNachVerzoegerung mit den Werten 2 und 3,
nach einer Verzögerung von 1000 Millisekunden
addiereNachVerzoegerung(3, 7, 2000);
```

HTTP(Hypertext Transfer Protocol) Grundlagen

HTTP ist das Protokoll, das den Datenaustausch zwischen Webbrowsern und Webservern im Internet ermöglicht.

Endpoints sind spezifische URLs, die auf bestimmte Funktionen oder Ressourcen auf einem Webserver verweisen.



Post:

```
app.post('/names', (request, response) => {
  console.log(request.query.name)
  let newName = request.query.name;
  list.push(newName);
  response.send('Name added');
});

app.get('/name', (request, response) => {
  response.send(list);
});
```

Delete:

```
app.delete('/names', (request, response) => {
  const nameOfDelete = request.query.name;
  const NameIndex =
list.indexOf(nameOfDelete);
  if (NameIndex > -1) {
    list.splice(NameIndex, 1);
    response.send('Name deleted');
  } else {
    response.send('Name not found');
  }
  respond.send(list);
});
```

Patch:

```
app.patch('/me', (request, response) => {
  const updatedData = request.body;
  Object.assign(me, updatedData);
  response.send('Data updated');
});
```

File hello-world.js:

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

const app = express();
const port = 3000;

app.get('/', (request, response) => {
  response.send('Hallo Welt!');
});

app.listen(port, () => {
  console.log(`Beispiel-App hört auf Port ${port}`);
});
```

Zeit:

```
app.get('/now', (request, response) => {
  let timezone = request.query.tz || 'UTC';
  let currentTime = new Date().toLocaleString("de-
CH", { timeZone: timezone });
  response.send(currentTime + ' ' + timezone);
});
```

Webseite abrufen:

```
app.get('/zli', (request, response) => {
  response.redirect('https://www.zli.ch');
});
```

Read files and send Data:

```
app.get('/html', (request, response) => {
  fs.readFile('/Users/oliver/Development/UEK295/
Block3/3-3/index.html', 'utf8', (error, data) => {
    if (error) {
      response.status(500).send('Fehler beim Lesen
der Datei');
    } else {
      response.send(data);
    }
  } );
});
```

Status 418 zurück gibt.

```
app.get('/teapot', (request, response) => {
  response.status(418).send('Status 418');
});
```

JSON Objekt zurück gibt

```
app.get('/me', (request, response) => {
  const userInfo = {
      "Vorname": "Oliver",
      "Nachname": "Mustermann",
      "Alter": 17,
      "Wohnort": "Schweiz",
      "Augenfarbe": "Blau"
  };
  response.send(userInfo);
});
```