Modular Express Application using router factories

To create a modular Express web application, you can use router factories. This allows you to build reusable and customizable route handlers.

Simple Example

Module:

A router factory that responds with a customizable greeting.

```
// greet.js
const express = require('express');

module.exports = function(options = {}) { //
Router factory
  const router = express.Router();

  // Respond with the greeting passed in
options
```

```
    router.get('/greet', (req, res, next) => {
      res.end(options.greeting);
    });

    return router;
  };
```

Application:

The main application using the greet.js module, passing the greeting "Hello world".

```
// app.js
const express = require('express');
const greetMiddleware =
require('./greet.js');

express()
  .use('/api/v1/', greetMiddleware({
greeting: 'Hello world' }))
  .listen(8080);
```

Result: When you visit http://:8080/api/v1/greet, it responds with "Hello world".

More Complex Example: Using Services

In this example, we introduce a service that generates custom greetings. This demonstrates the flexibility of using middleware factories.

Module:

This version of `greet.js` accepts a service object and generates personalized greetings.

```
// greet.js
const express = require('express');

module.exports = function(options = {}) { //
Router factory
  const router = express.Router();

  const { service } = options;  // Get
service from options

  router.get('/greet', (req, res, next) => {
    // Generate greeting using the service

res.end(service.createGreeting(req.query.nam
e || 'Stranger'));
  });
```

```
    return router;
};
```

Application:

The application creates instances of a GreetingService with different greetings and passes them into the greetMiddleware.

```
// app.js
const express = require('express');
const greetMiddleware =
require('./greet.js');

class GreetingService {
  constructor(greeting = 'Hello') {
    this.greeting = greeting;
  }

  createGreeting(name) {
    return `${this.greeting}, ${name}!`;
  }
}

express()
  .use('/api/v1/service1', greetMiddleware({
    service: new GreetingService('Hello'),
  }))
  .use('/api/v1/service2', greetMiddleware({
    service: new GreetingService('Hi'),
  }))
  .listen(8080);
```

Result:

Accessing `http://<hostname>:8080/api/v1/service1/greet?name=World` will respond with "Hello, World!"
Accessing `http://<hostname>:8080/api/v1/service2/greet?name=World` will respond with "Hi, World!"

Benefits:

The application is split into reusable modules (e.g., greet.js), making the code easier to maintain and scale.

Different greetings can be passed as options, allowing the service to be easily customized without modifying core logic.

You can reuse the same middleware across multiple routes with different configurations.

This structure helps keep the code clean, organized, and flexible for various use cases.