Creating middleware

Creating Middleware

Custom middleware is easy to create since it's just a function. Here's an example of creating a middleware and saving it as a variable which is used later to apply the middleware to the Express app:

```
const middleware = (request, response, next) => {
  // Modify request or response
  // Execute the callback when done
  next()
}
app.use(middleware)
```

Developers can also create middleware right in the `app.use()` call using an anonymous function definition:

```
app.use((request, response, next) => {
  // Modify request or response
  // Execute the callback when done
  next()
})
```

The first approach is better for modularizing because it's easier to move a named function or a function saved into a variable than an anonymous function.

Passing References

`request` is **always** the same object in the lifecycle of a single client request to the Express server. This allows us to implement a very useful pattern in which developers pass data from one middleware to another and to a request handler.

For example, developers can connect to a database in one middleware and make it available in all the subsequent middleware functions and request handlers (routes).

```
app.use(function (request, response, next) {
  DatabaseClient.connect(URI, (err, db) => {
    // error handling
    request.database = db
    next()
  })
})
```

In this middleware, `database` is available in the request object and we can run queries such as finding an application by the app ID:

```
app.use((req, res, next) => {
  req.database.collection('apps').findOne({appId: req.query.appId},
    // error handling
    req.app = app
    next()
  })
})
```

This makes moving routes and middleware to other files (modularization) straightforward, i.e., keeping code neat and organized.