

Não use middlewares

```
app.post("/posts", authMiddleware, (req, res) => {  
  })
```



veja
alternativa

in-line middlewares



```
const [notAuth, authData] = requireAuth(req, res)
if (notAuth) return res.status(notAuth.status).json(notAuth.json)

const { userId } = authData
```

- ✓ **composable**
- ✓ **flexível**
- ✓ **type-safety**
- ✓ **controle do código, o que você vê,
você recebe**

**Mas quais os
pontos negativos
de usar
middlewares?**

**vamos por
partes**





```
const tweetsRouter = express.Router()
tweetsRouter.use(ensureAuthentication)
tweetsRouter.get("/", GetTweetsController.handle)
tweetsRouter.post(
  "/",
  ensureAuthentication,
  CreateTweetController.handle
)
tweetsRouter.put(
  "/:tweetId",
  ensureAuthentication,
  injectTweet,
  UpdateTweetController.handle
)
tweetsRouter.patch(
  "/:tweetId/visibility",
  ensureAuthentication,
  injectTweet,
  ensurePermissions("toggle-tweet-visibility"),
  ToggleTweetVisibilityController.handle
)
app.use("/tweets", tweetsRouter)
```

**foge do fluxo de
input e output**

Você vai escrever uma nova feature...

```
tweetsRouter.post("/new", (req, res) => {  
  // sua implementação  
})
```

mas **req** e **res** já vem adulterado
de outros middlewares



pode ter sido só **1**
middleware, ou **500**

**precisa sobescrever a
tipagem global para
satisfazer middlewares**

```
declare global {  
  export namespace Express {  
    interface Request {  
      userId: string  
    }  
  }  
}
```



```
tweetsRouter.post("/new", (req, res) => {  
  req.userId  
})
```


inline middlewares respeitam o fluxo de input e output, e com **typesafety**

```
app.get("/me", async function (req, res) {  
  const [notAuth, authData] = requireAuth(req, res)  
  if (notAuth) return res.status(notAuth.status).json(notAuth.json)
```


```
  const { userId } = authData
```

401



```
record<string, any> = record<string, any>>.json  
  message: "Você não está autenticado.";  
} | {  
  message: "Formato de token inválido.";  
  errorMessage: string;  
}
```

Componha seu endpoint!



o request que você trabalha, é o que o client enviou, agora você tem controle do código

```
app.post("/tweets/new", async function (req, res) {  
  const [notAuth, authData] = requireAuth(req, res)  
  if (notAuth) return res.status(notAuth.status).json(notAuth.json)  
  
  const [badInput, input] = parseInput(req.body, newPostSchema)  
  if(badInput) return res.status(badInput.status).json(badInput.json)  
  
  const { country, timestamp } = getUserLocation(req)  
  const { userId } = authData  
  const { text, media } = input
```


**Você consegue voltar em 2030 e
ainda saber 100% do que seu
endpoint responde**

```
const badInput: {  
  status: 400;  
  json: {  
    validations: {  
      message: string; ic  
      paths: string[]; re  
    }[]; status/  
    code: "BAD_INPUT";  
  };  
  (badInput) return res.status(bac
```

mas isso não adiciona mais verbosidade?

```
app.put("/random-endpoint/:subject", async function (req, res) {  
  const [failDepth, depth] = getDepth(req, res)  
  if (failDepth) return res.status(failDepth.status).json(failDepth.json)  
  
  const [failApple, apple] = getApple(req, res)  
  if (failApple) return res.status(failApple.status).json(failApple.json)  
  
  const [failBravo, bravo] = getBravo(req, res)  
  if (failBravo) return res.status(failBravo.status).json(failBravo.json)  
  
  const [failDelta, delta] = getDelta(req, res)  
  if (failDelta) return res.status(failDelta.status).json(failDelta.json)
```

A complexidade de software sempre vai existir

você que escolhe aonde quer deixá-la

❌ você pode ter ela distribuída pela codebase

✅ ou ter ela sob seu controle, dentro da sua visão

eu me alinho mais com a segunda proposta