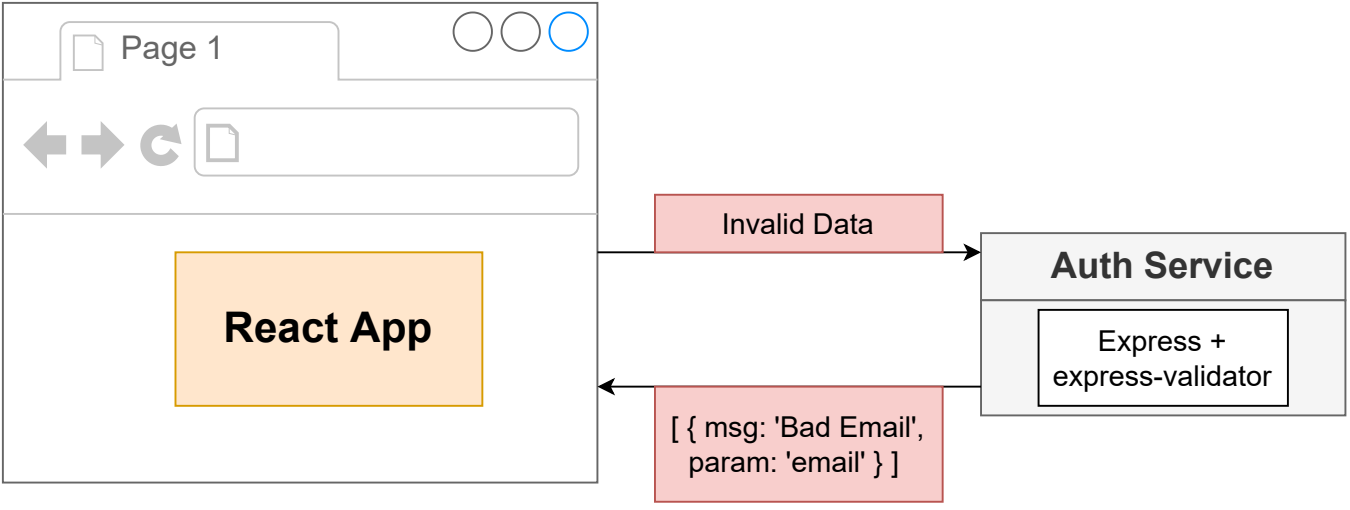
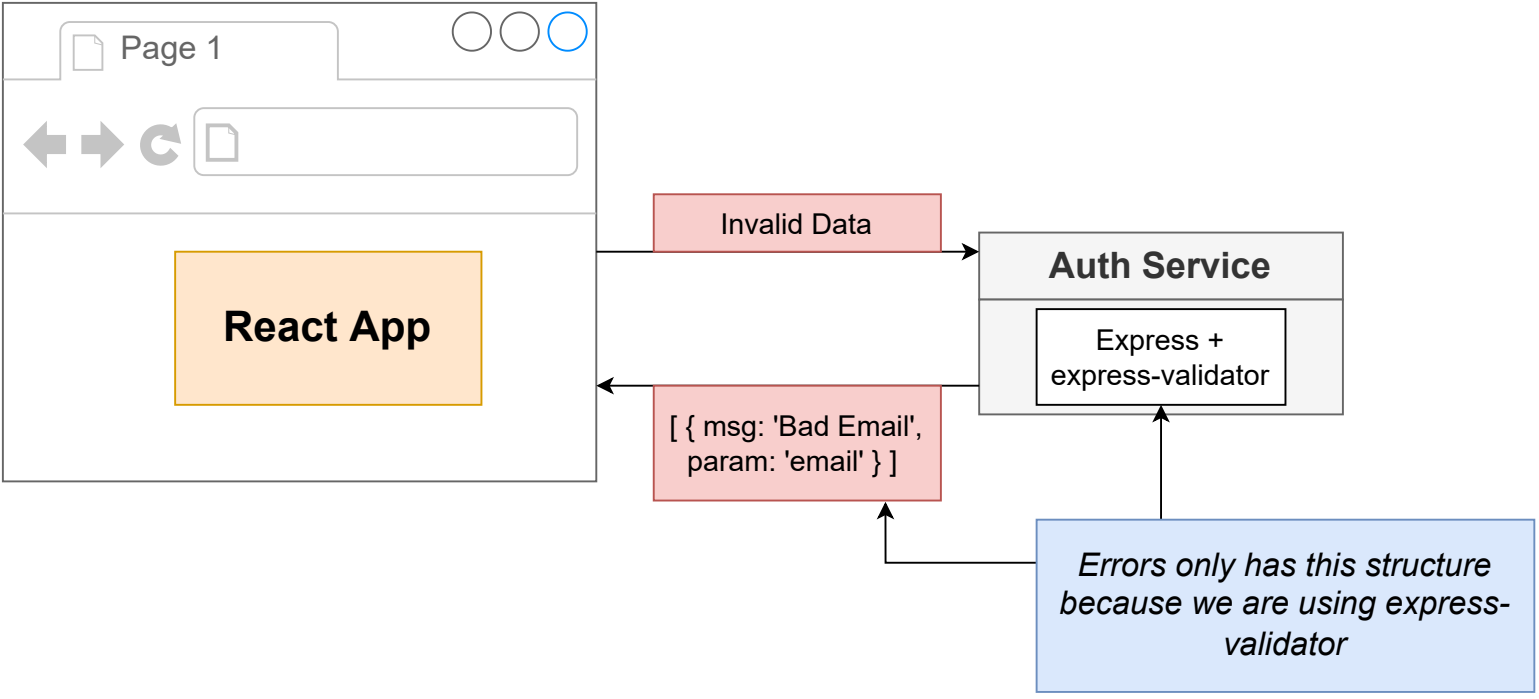
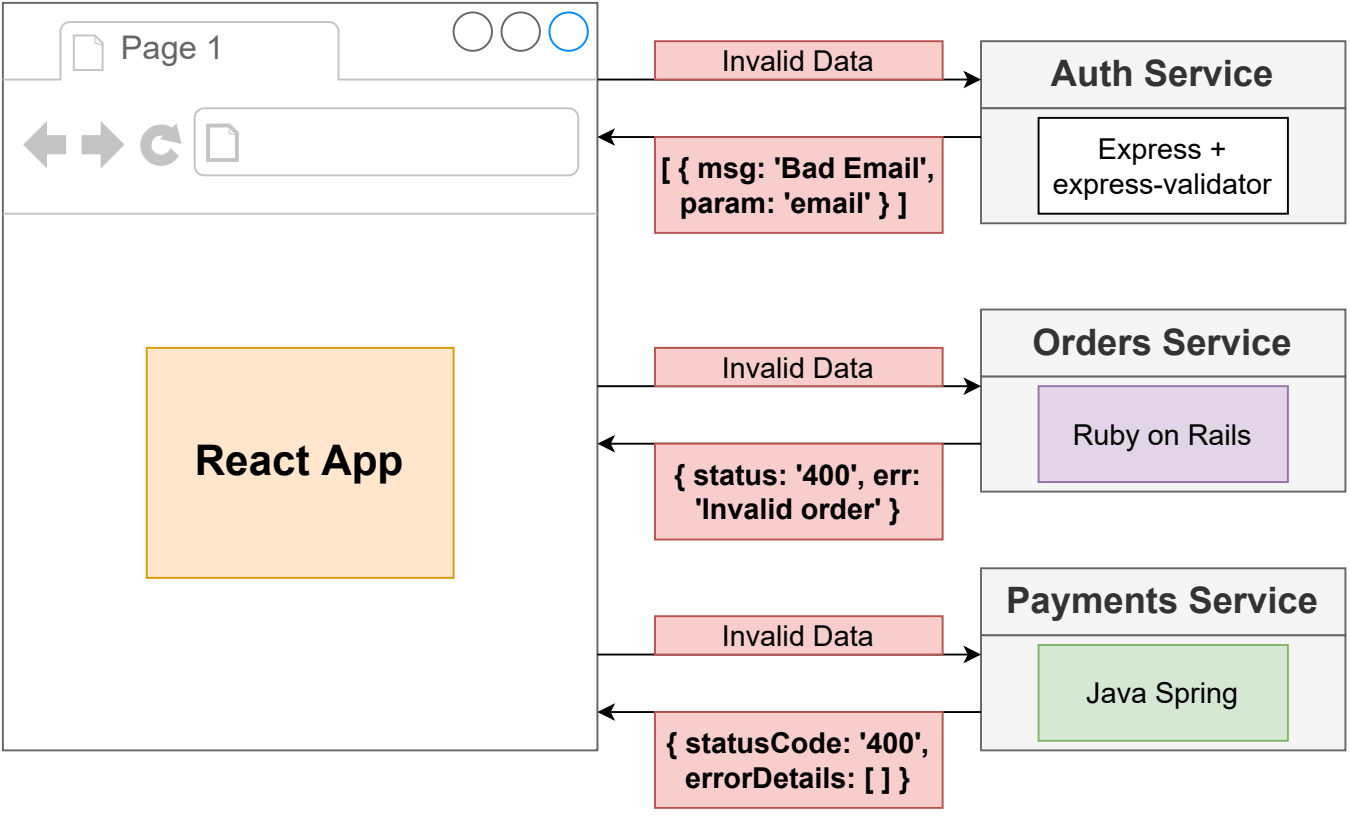
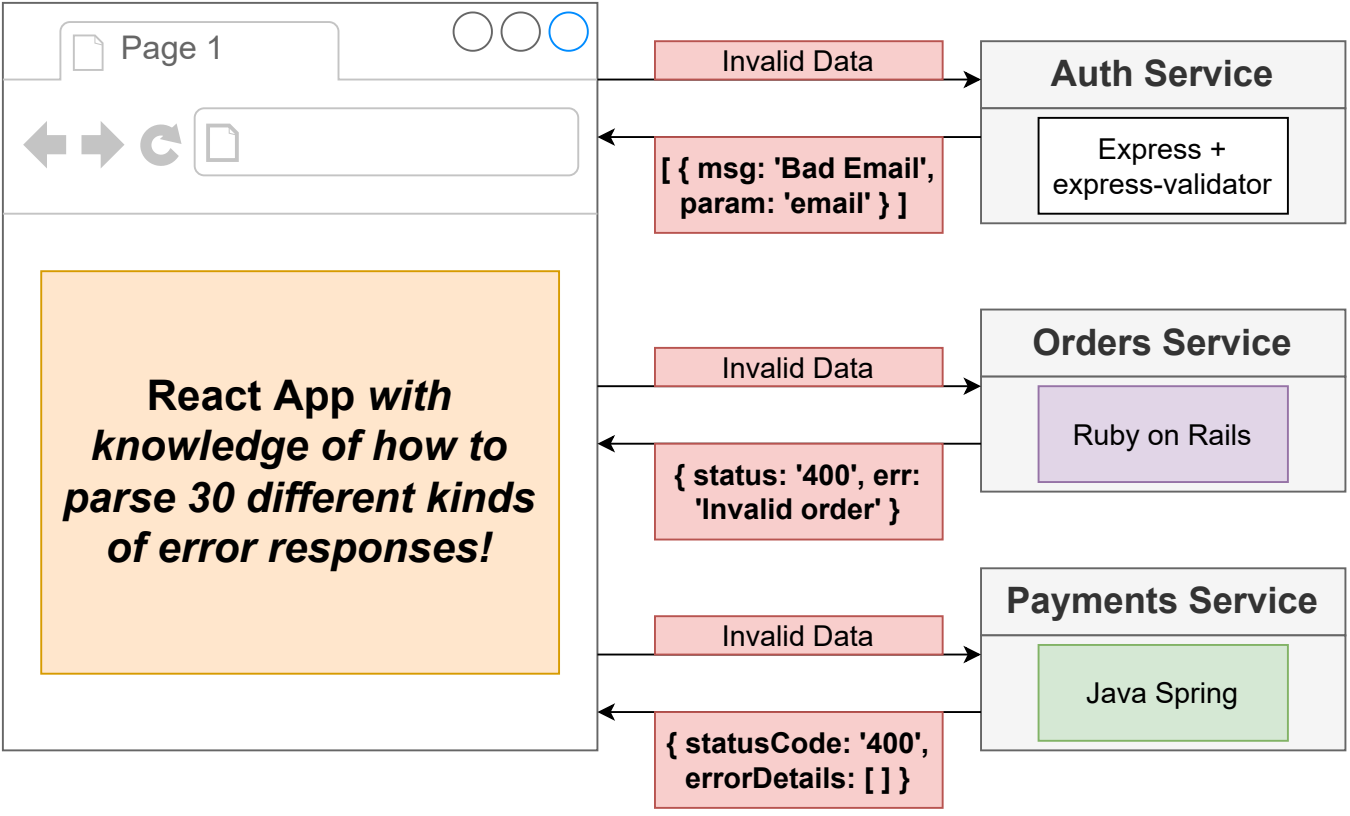
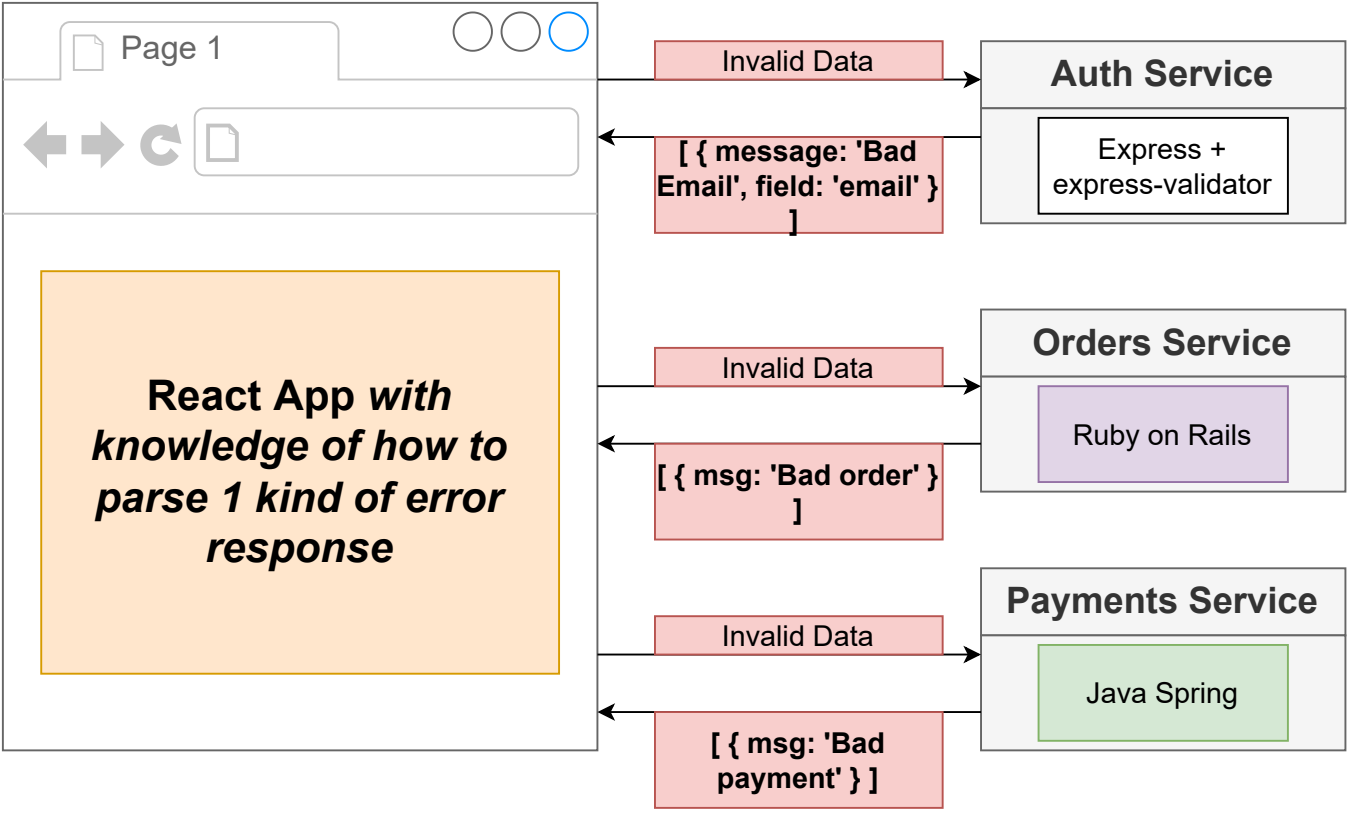
auth			
Route	Method	Body	Purpose
/api/users/signup	POST	{ email: string, password: string }	Sign up for an account
/api/users/signin	POST	{ email: string, password: string }	Sign in to an existing account
/api/users/signout	POST	{}	Sign out
/api/users/currentuser	GET	-	Return info about the user

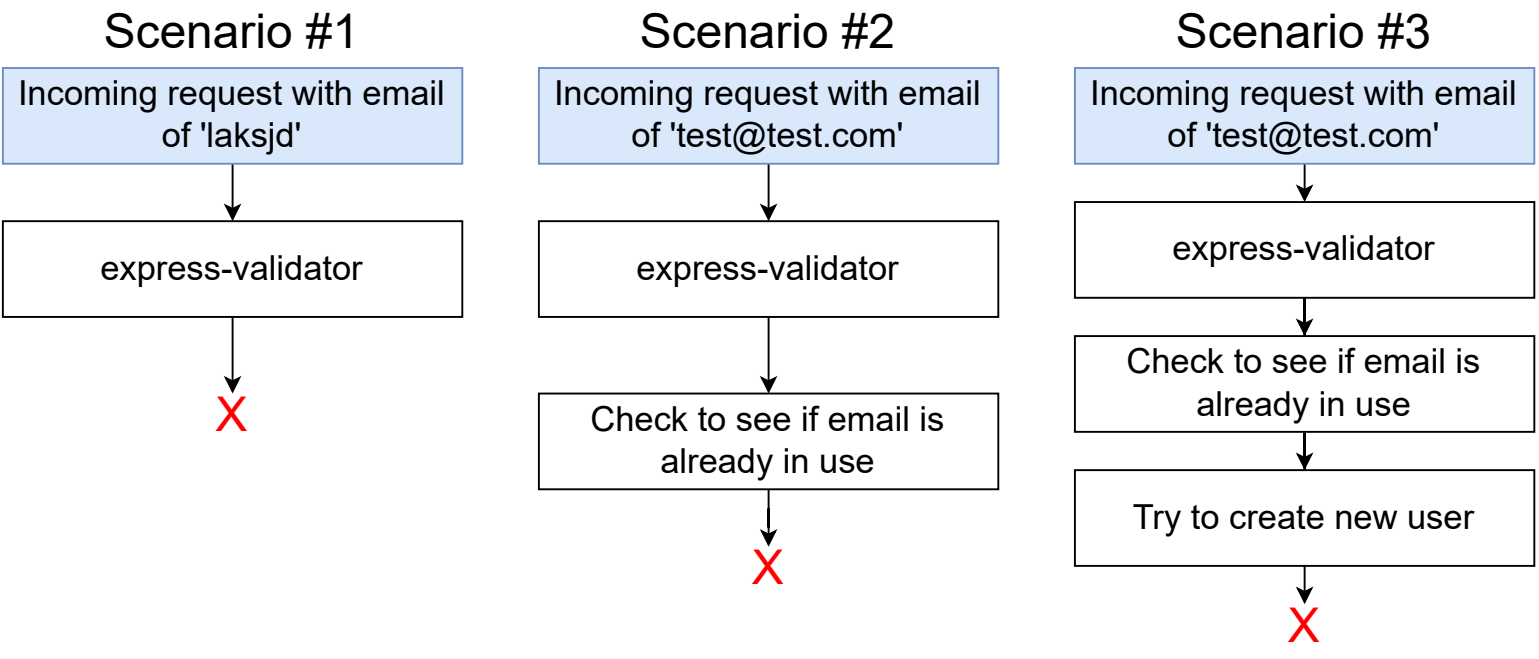

Page 1
React App
Invalid Data
Auth Service
Express +
express-validator
[{ msg: 'Bad Email',
param: 'email' }]

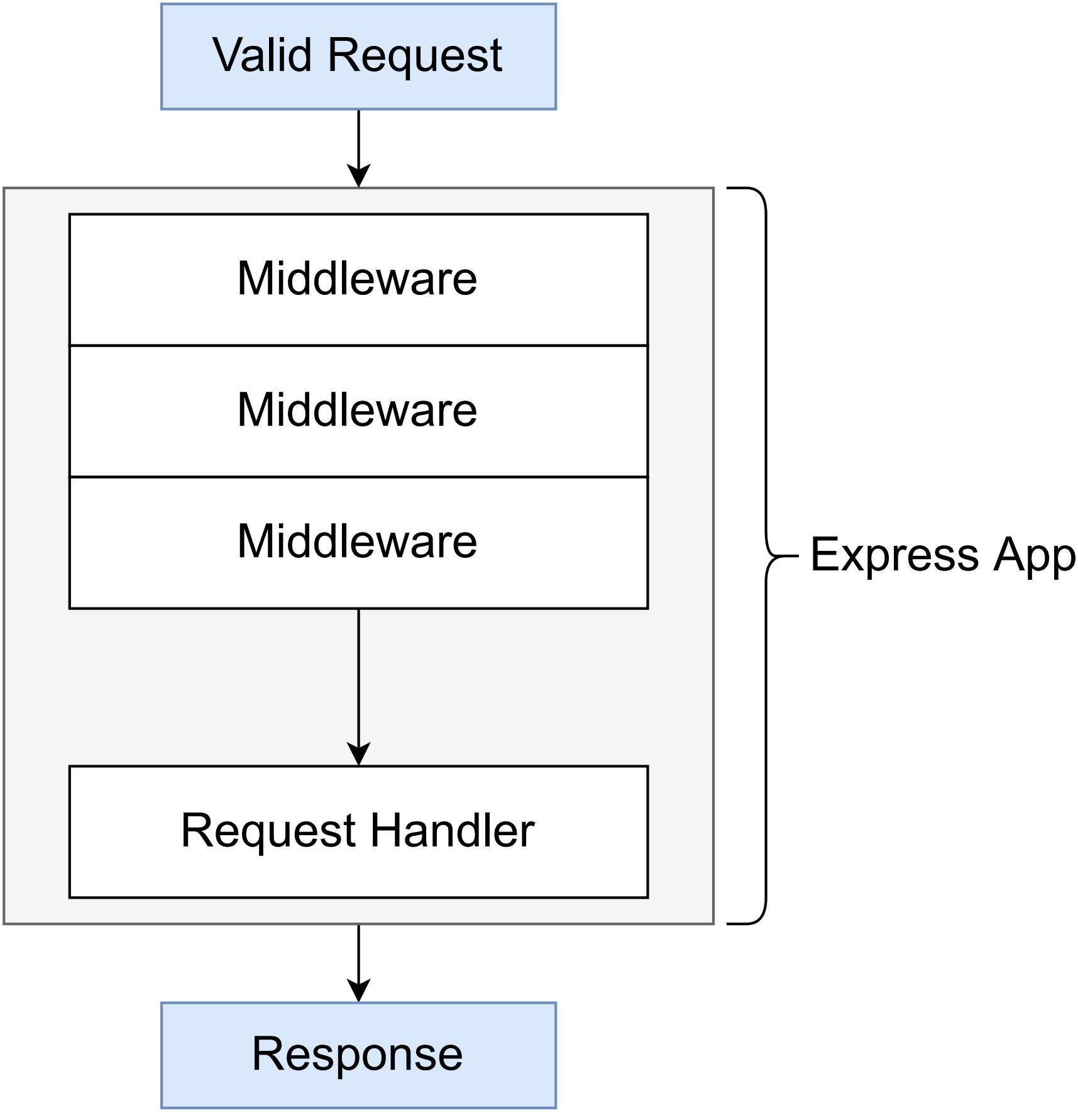

Page 1
React App
Invalid Data
Auth Service
Express +
express-validator
[{ msg: 'Bad Email',
param: 'email' }]
Errors only has this structure because we are using express-validator

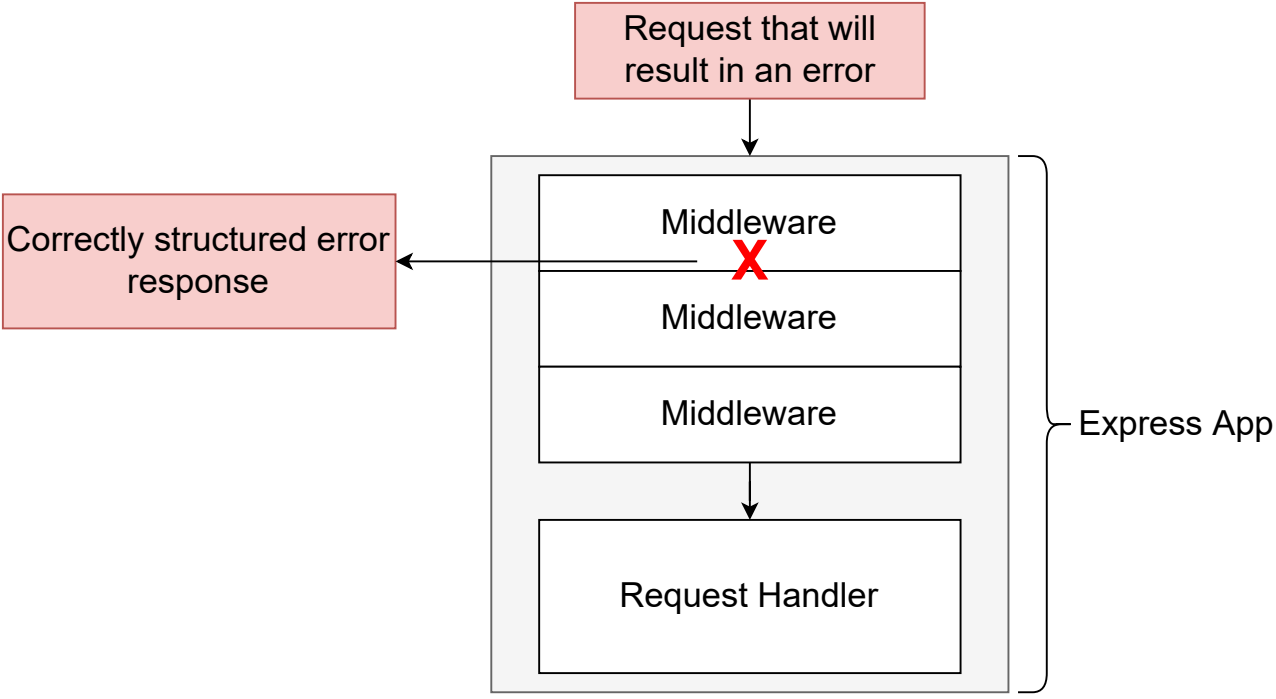

Page 1
React App
Invalid Data
[{ msg: 'Bad Email', param: 'email' }]
Auth Service
Express + express-validator
Invalid Data
{ status: '400', err: 'Invalid order' }
Orders Service
Ruby on Rails
Invalid Data
{ statusCode: '400', errorDetails: [ ] }
Payments Service
Java Spring


Page 1
React App *with knowledge of how to parse 30 different kinds of error responses!*
Invalid Data
[{ msg: 'Bad Email', param: 'email' }]
Auth Service
Express + express-validator
Invalid Data
{ status: '400', err: 'Invalid order' }
Orders Service
Ruby on Rails
Invalid Data
{ statusCode: '400', errorDetails: [ ] }
Payments Service
Java Spring


Page 1
React App *with knowledge of how to parse 1 kind of error response*
Invalid Data
[{ message: 'Bad Email', field: 'email' }]
Auth Service
Express + express-validator
Invalid Data
[{ msg: 'Bad order' }]
Orders Service
Ruby on Rails
Invalid Data
[{ msg: 'Bad payment' }]
Payments Service
Java Spring


Scenario #1
Incoming request with email of 'laksjd'
express-validator
X
Scenario #2
Incoming request with email of 'test@test.com'
express-validator
Check to see if email is already in use
X
Scenario #3
Incoming request with email of 'test@test.com'
express-validator
Check to see if email is already in use
Try to create new user
X


Valid Request
Middleware
Middleware
Middleware
Request Handler
Express App
Response


Request that will result in an error
Middleware
Middleware
Middleware
Request Handler
Express App
Correctly structured error response

Difficulty in Error Handling

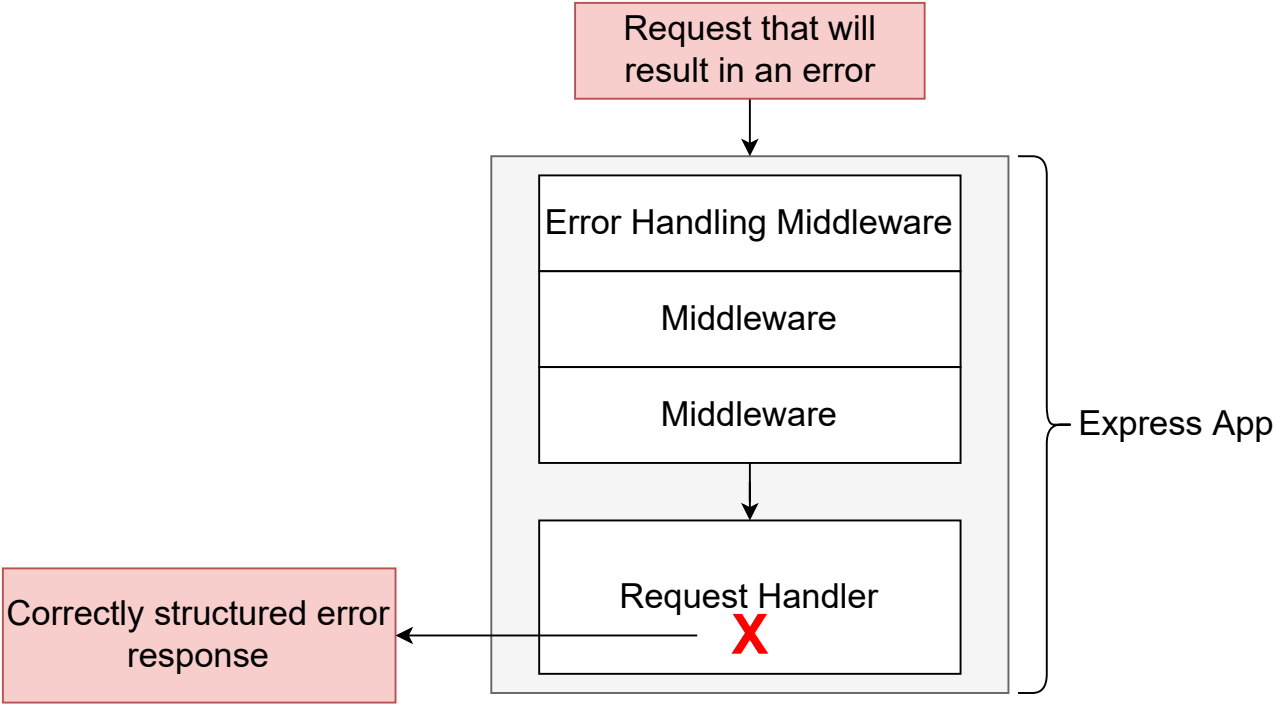
1	We must have a consistently structured response from *all* servers, no matter what went wrong
2	A billion things can go wrong, not just validation of inputs to a request handler. Each of these need to be handled consistently

Difficulty in Error Handling

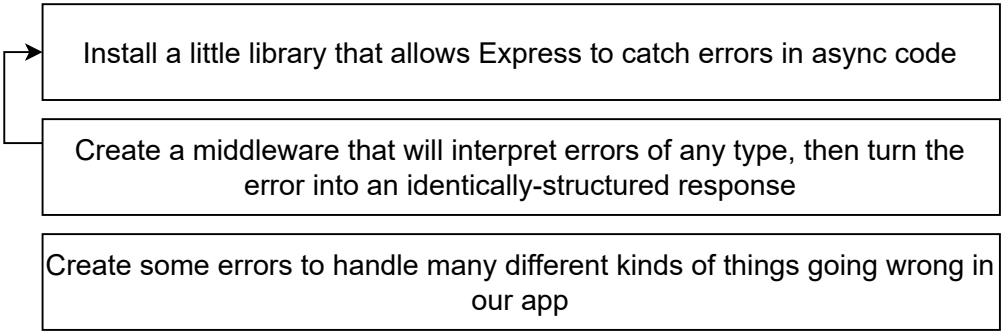
1	We must have a consistently structured response from *all* servers, no matter what went wrong	→ Write an error handling middleware to process errors, give them a consistent structure, and send back to the browser
2	A billion things can go wrong, not just validation of inputs to a request handler. Each of these need to be handled consistently	→ Make sure we capture all possible errors using Express's error handling mechanism (call the 'next' function!)

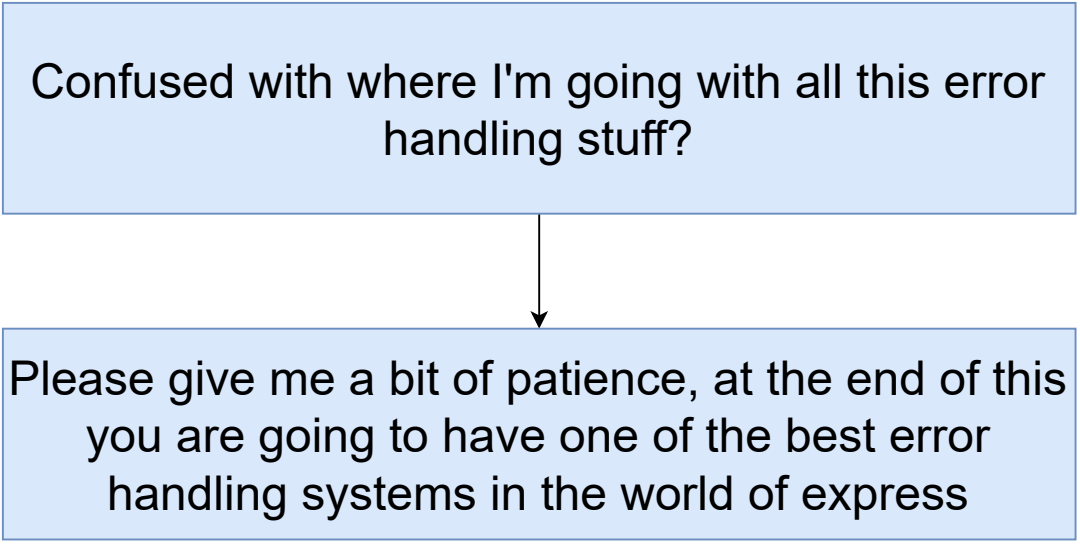
We need to have an easy-to-use system for handling *any kind of error* and sending an identically-structured response

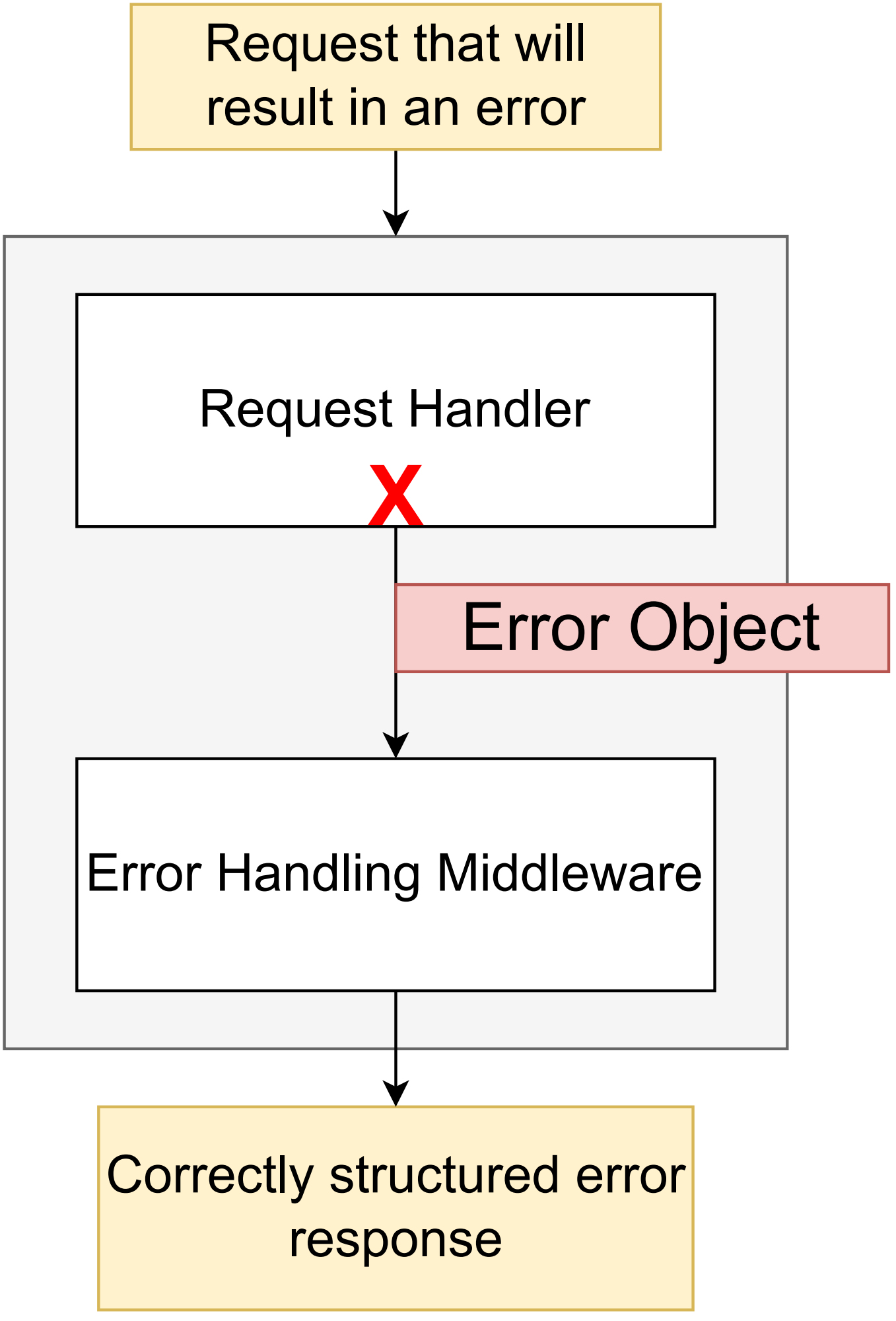
Unfortunately, Express's default error handling system makes this kind of hard, mostly because of async-await

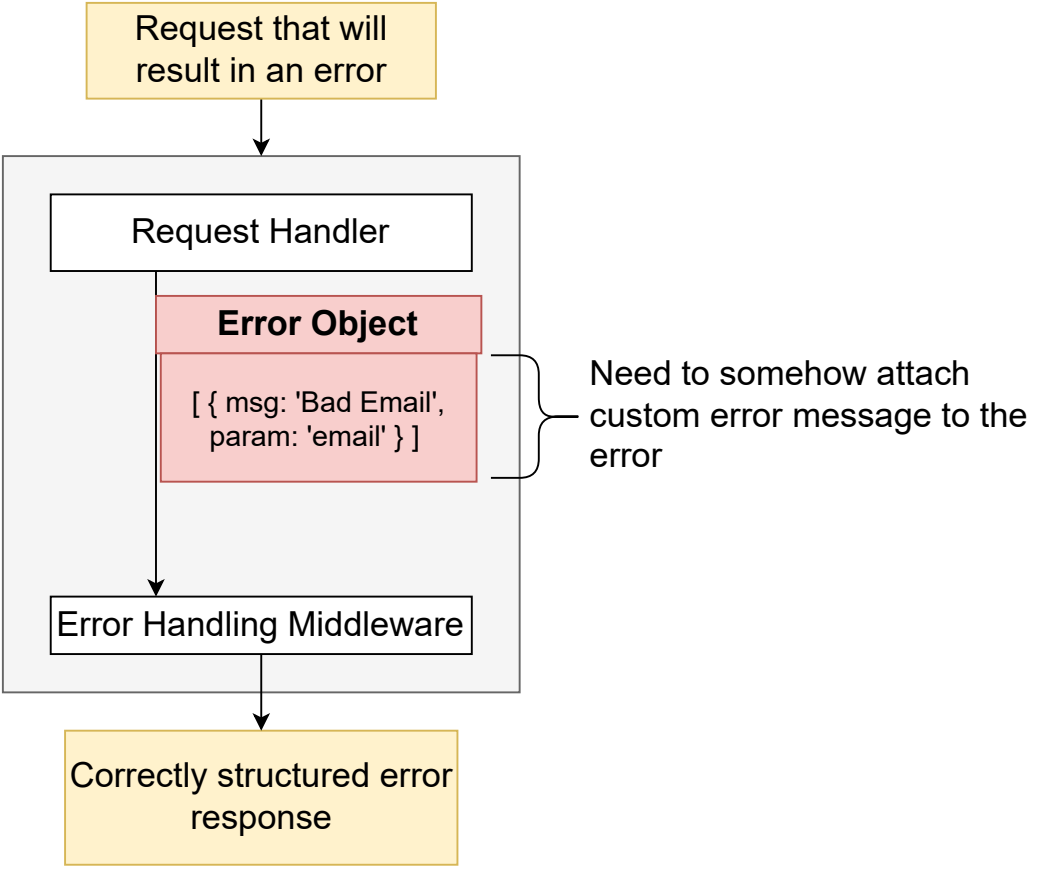

Request that will result in an error
Error Handling Middleware
Middleware
Middleware
Request Handler
X
Express App
Correctly structured error response

Error Handling Strategy



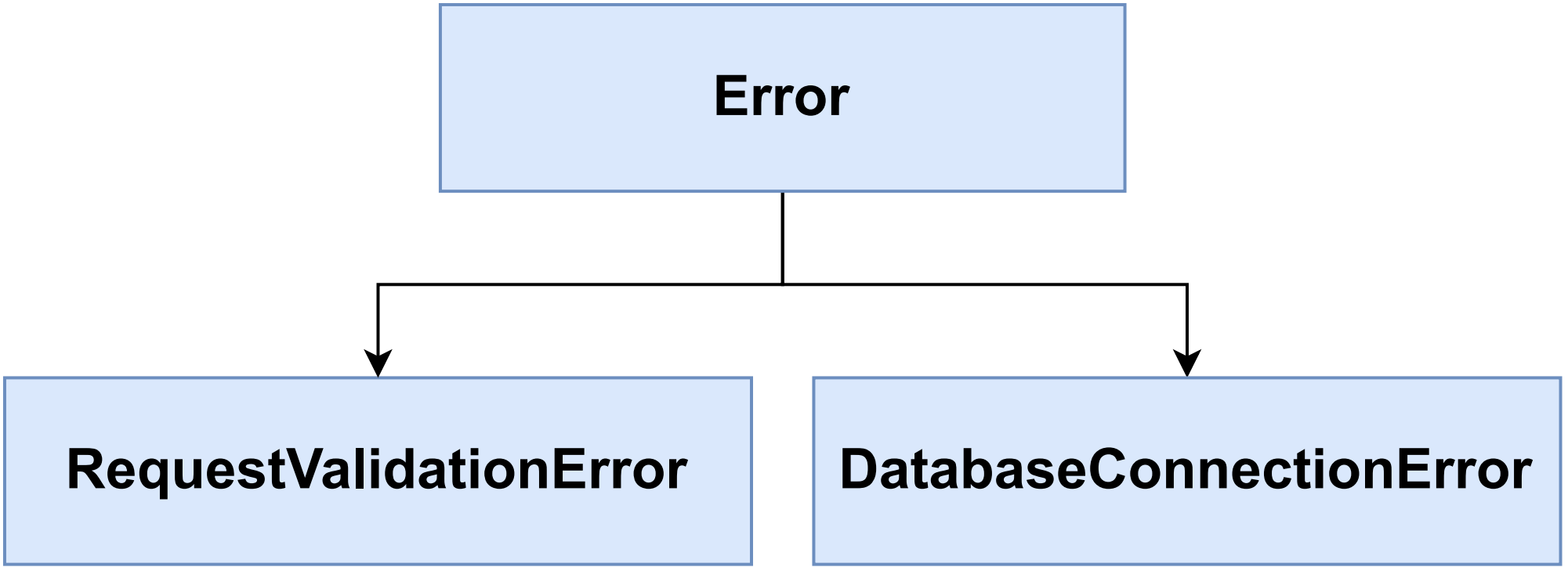

Confused with where I'm going with all this error handling stuff?
Please give me a bit of patience, at the end of this you are going to have one of the best error handling systems in the world of express

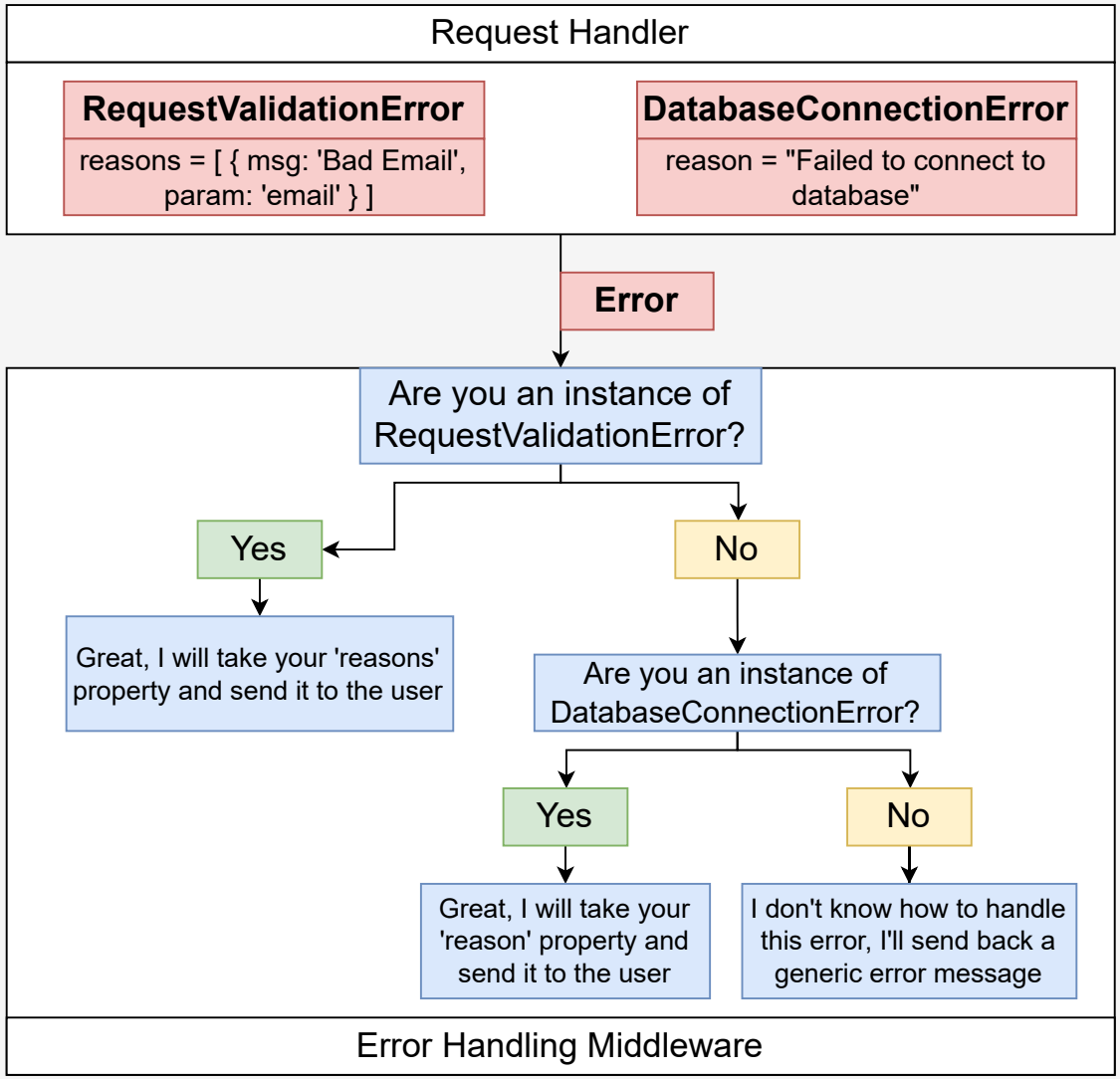

Request that will result in an error
Request Handler
X
Error Object
Error Handling Middleware
Correctly structured error response

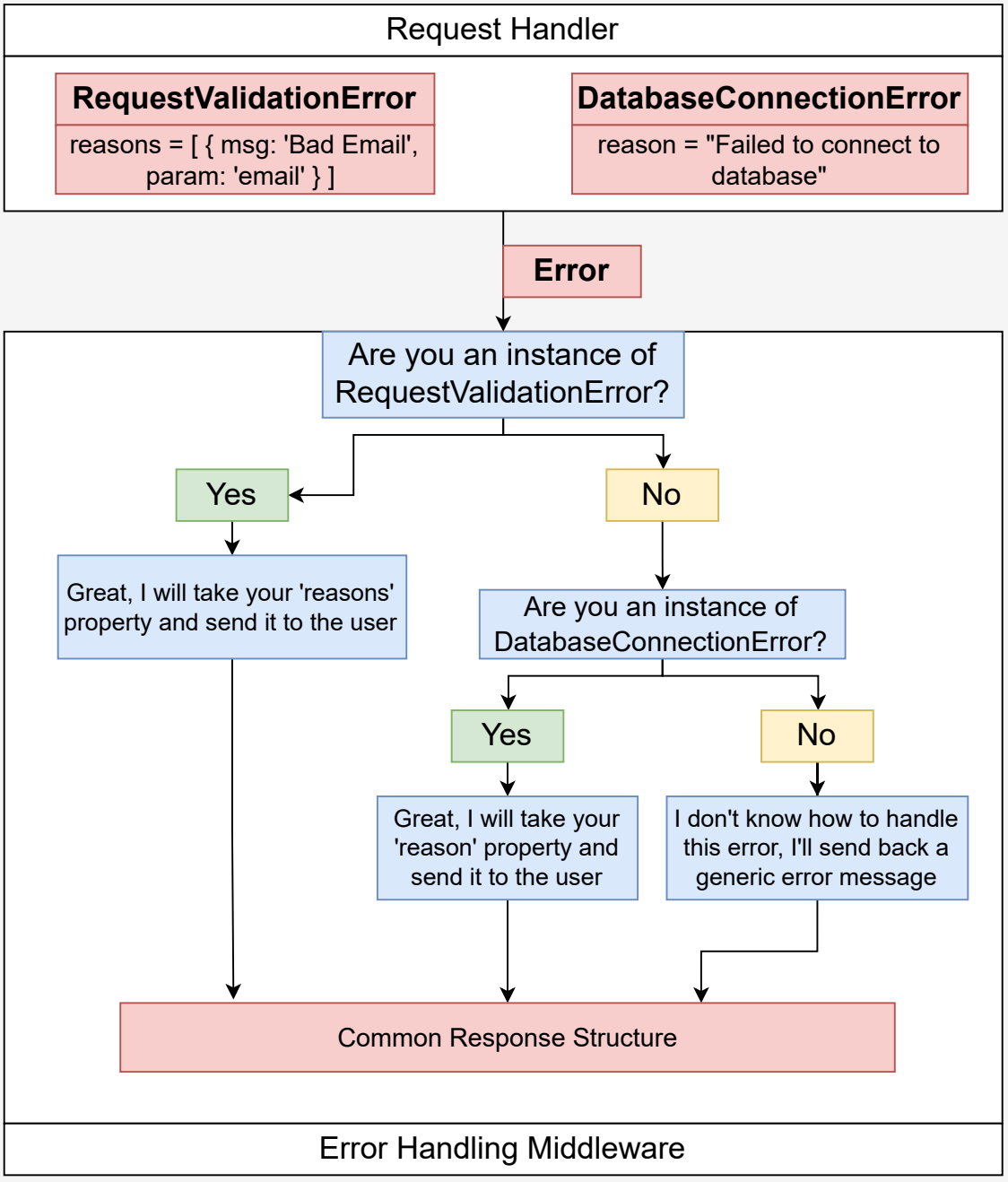

Request that will result in an error
Request Handler
Error Object
[{ msg: 'Bad Email', param: 'email' }]
Need to somehow attach custom error message to the error
Error Handling Middleware
Correctly structured error response

We want an object like an 'Error', but we want to add in some more custom properties to it

Usually a sign you want to subclass something!


Error
RequestValidationError
DatabaseConnectionError

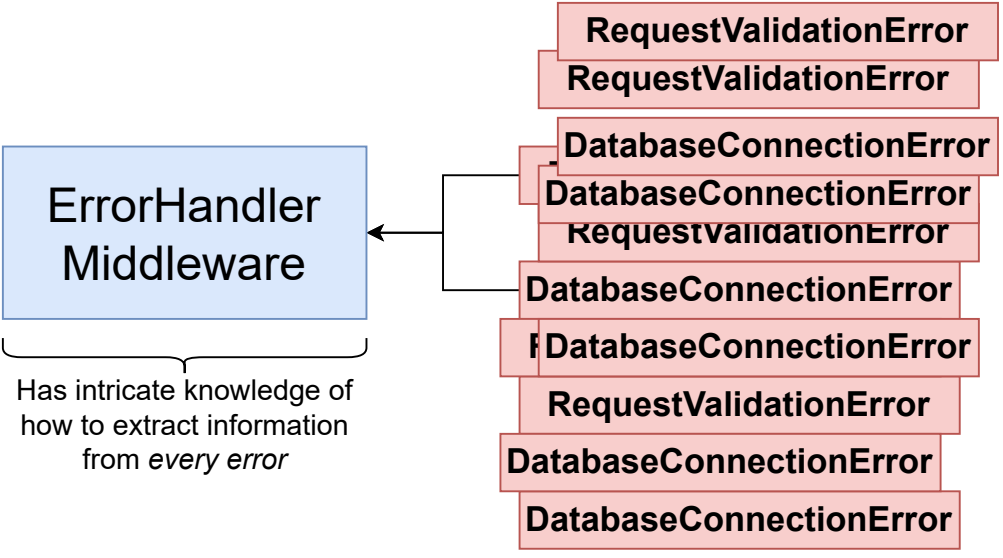

Request Handler
RequestValidationError
reasons = [{ msg: 'Bad Email', param: 'email' }]
DatabaseConnectionError
reason = "Failed to connect to database"
Error
Are you an instance of RequestValidationError?
Yes
No
Great, I will take your 'reasons' property and send it to the user
Are you an instance of DatabaseConnectionError?
Yes
No
Great, I will take your 'reason' property and send it to the user
I don't know how to handle this error, I'll send back a generic error message
Error Handling Middleware

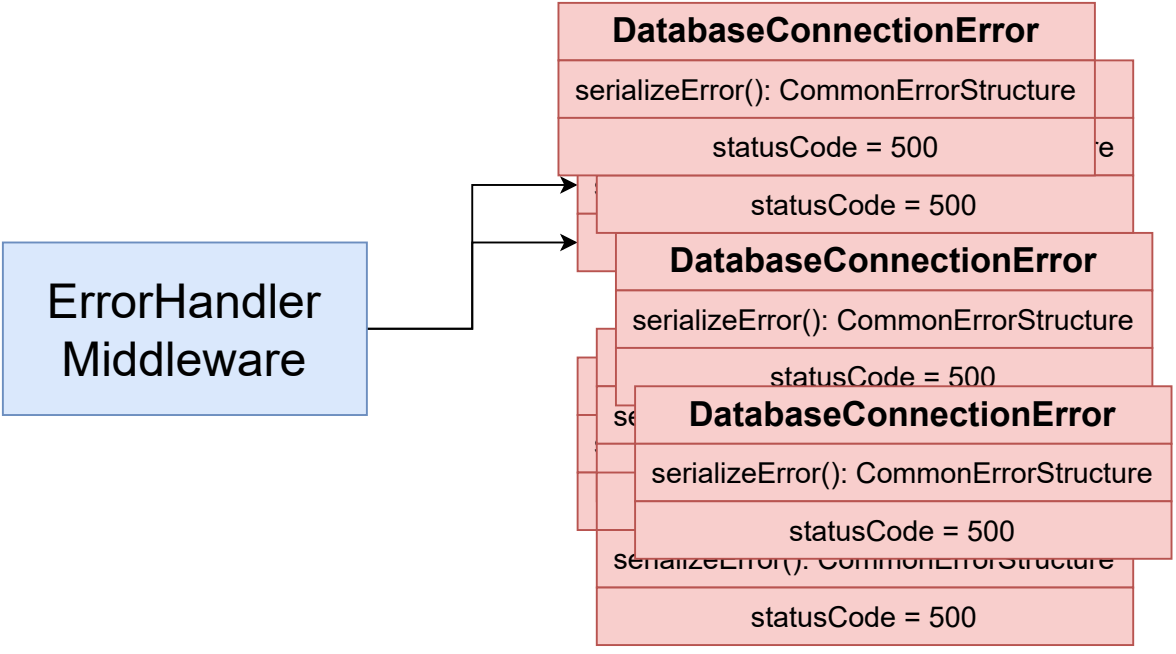

Request Handler
RequestValidationError
reasons = [{ msg: 'Bad Email', param: 'email' }]
DatabaseConnectionError
reason = "Failed to connect to database"
Error
Are you an instance of RequestValidationError?
Yes
No
Great, I will take your 'reasons' property and send it to the user
Are you an instance of DatabaseConnectionError?
Yes
No
Great, I will take your 'reason' property and send it to the user
I don't know how to handle this error, I'll send back a generic error message
Common Response Structure
Error Handling Middleware

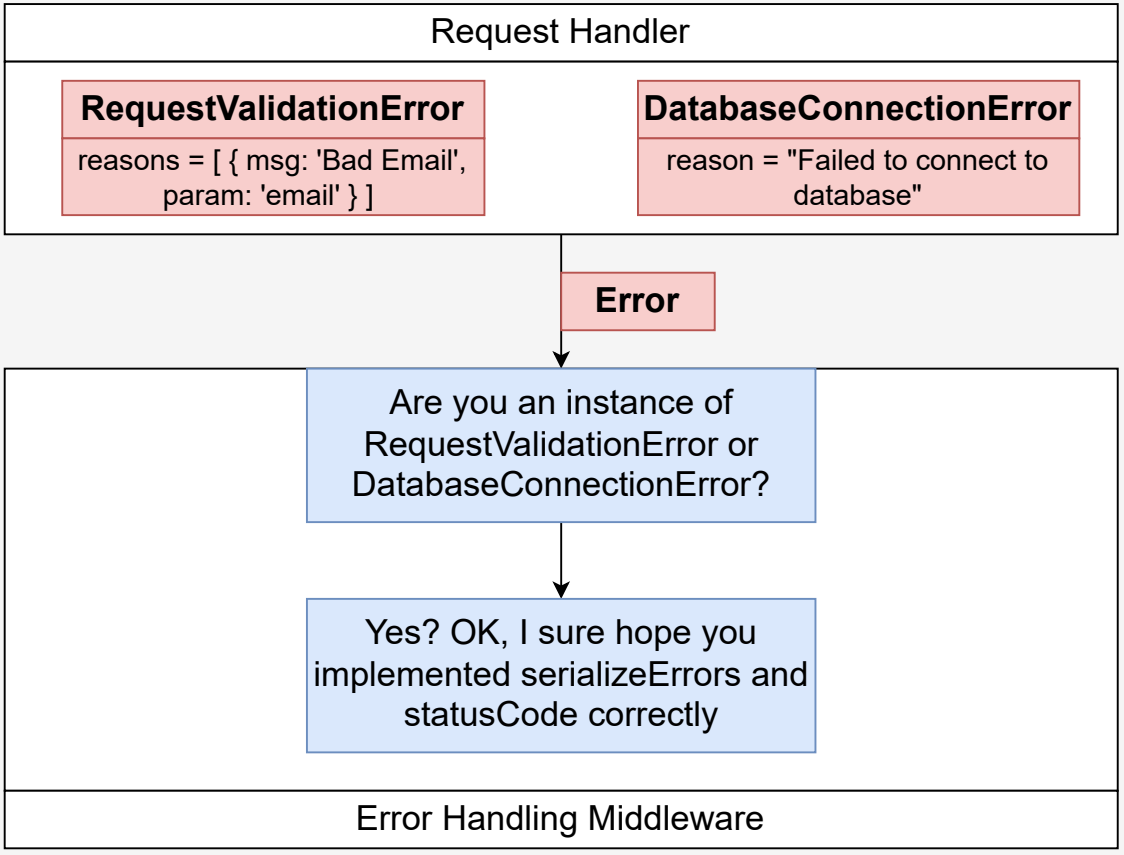
Common Response Structure

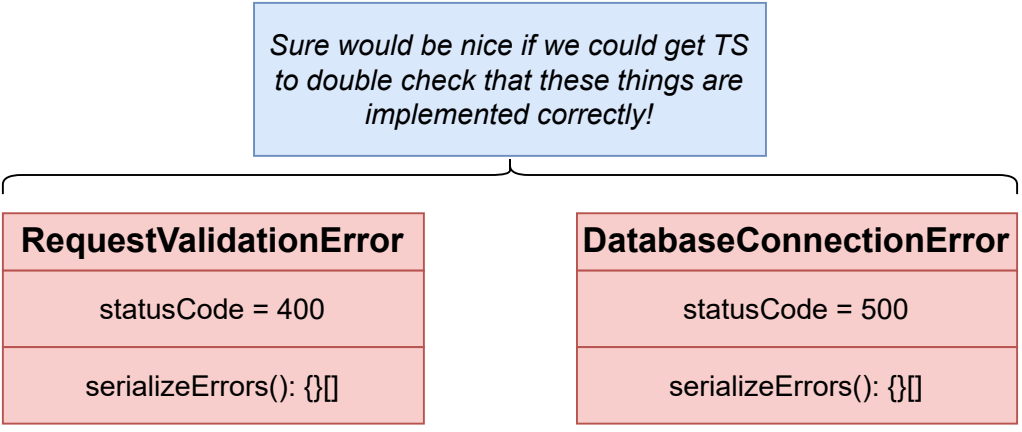
```
{
  errors: {
     message: string, field?: string
  }[ ]
}
```

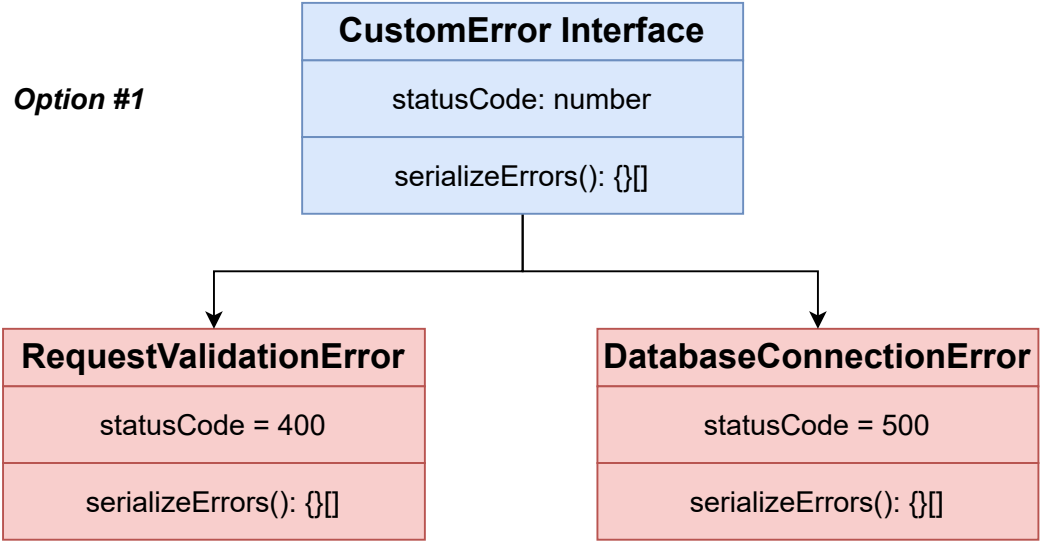
All error responses that we send out from any server should have this structure

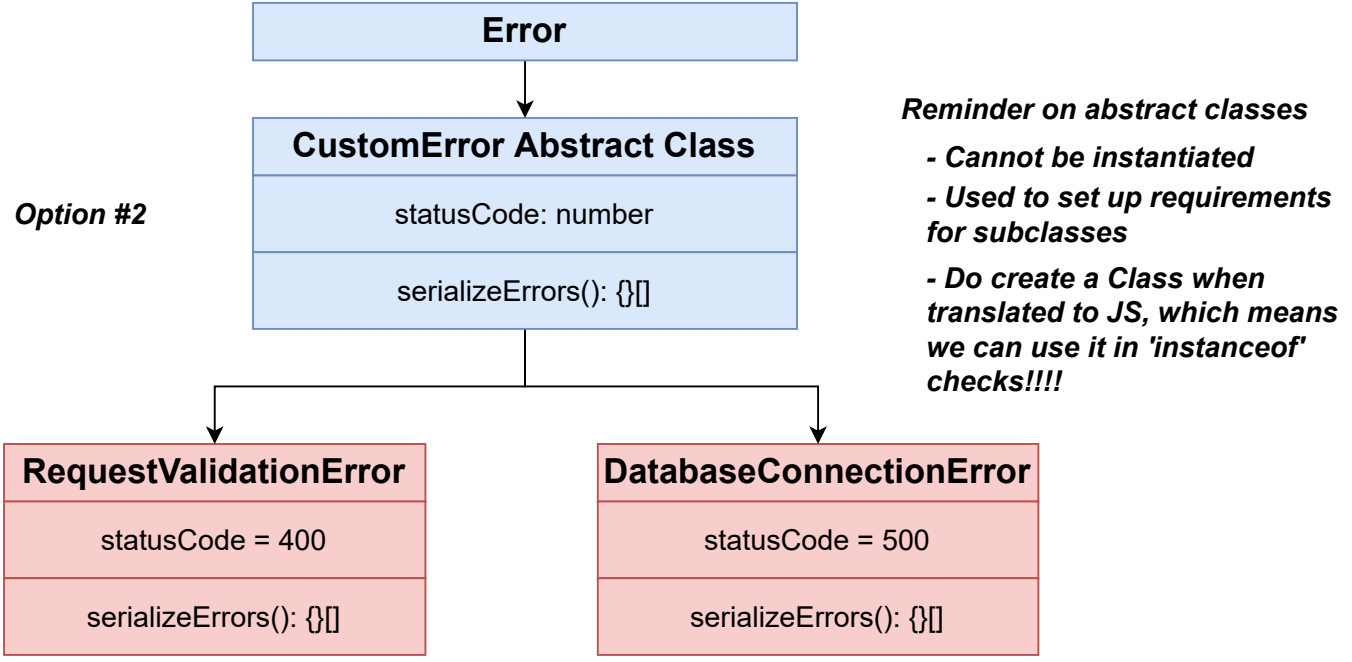

RequestValidationError
RequestValidationError
DatabaseConnectionError
DatabaseConnectionError
RequestValidationError
DatabaseConnectionError
DatabaseConnectionError
RequestValidationError
DatabaseConnectionError
DatabaseConnectionError
ErrorHandler Middleware
Has intricate knowledge of how to extract information from *every error*


ErrorHandler Middleware
DatabaseConnectionError
serializeError(): CommonErrorStructure
statusCode = 500
statusCode = 500
DatabaseConnectionError
serializeError(): CommonErrorStructure
statusCode = 500
DatabaseConnectionError
serializeError(): CommonErrorStructure
statusCode = 500
statusCode = 500


Request Handler
RequestValidationError
reasons = [{ msg: 'Bad Email', param: 'email' }]
DatabaseConnectionError
reason = "Failed to connect to database"
Error
Are you an instance of RequestValidationError or DatabaseConnectionError?
Yes? OK, I sure hope you implemented serializeErrors and statusCode correctly
Error Handling Middleware


Sure would be nice if we could get TS to double check that these things are implemented correctly!
RequestValidationError
statusCode = 400
serializeErrors(): {}[]
DatabaseConnectionError
statusCode = 500
serializeErrors(): {}[]


Option #1
CustomError Interface
statusCode: number
serializeErrors(): {}[]
RequestValidationError
statusCode = 400
serializeErrors(): {}[]
DatabaseConnectionError
statusCode = 500
serializeErrors(): {}[]


Error
CustomError Abstract Class
statusCode: number
serializeErrors(): {}[]
Option #2
Reminder on abstract classes
- Cannot be instantiated
- Used to set up requirements for subclasses
- Do create a Class when translated to JS, which means we can use it in 'instanceof' checks!!!!
RequestValidationError
statusCode = 400
serializeErrors(): {}[]
DatabaseConnectionError
statusCode = 500
serializeErrors(): {}[]