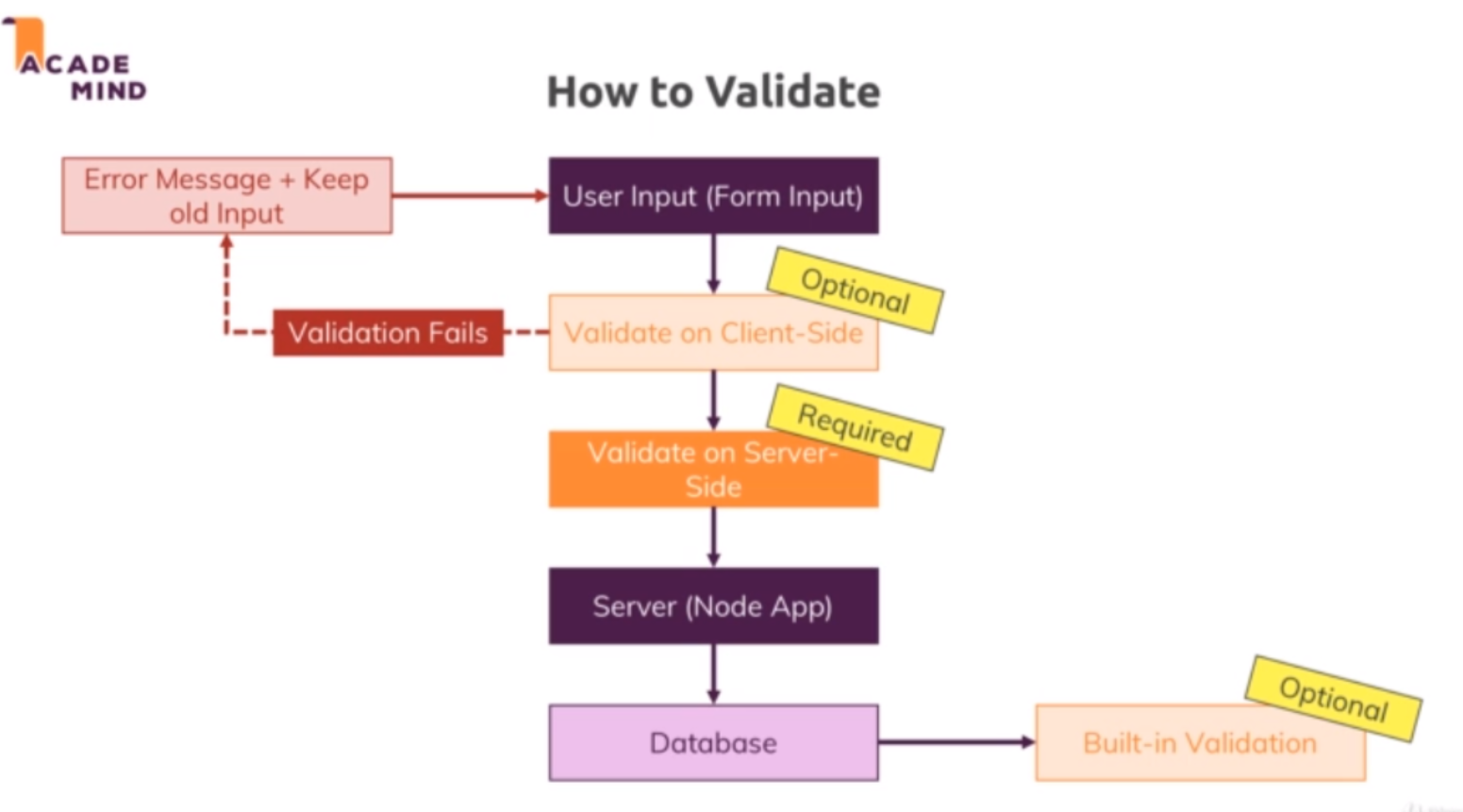
**Lecture 289**

**Why should we use validation?**

* Refer notes

**Lecture 290**

**How to validate input?**

* Client side validation can definitely improve the user experience and therefore you might want to consider using it but it is optional because since we use client side javascript, so javascript code that runs in the browser, the user can see that code, the user could change that code and the user can of course disable javascript. So this is not a protection that secures you against incorrect data being sent to your server, this is not a secure solution, it's only there to improve the user experience because of course there you can validate and you can show a nice error message and keep the old input so that the user doesn't have to re-enter that email address where the @ sign was forgotten but it's only for a user experience thing, it's not really secure.
* 
* So your next option is to validate on the server side. Now this is what we'll focus on in this module and in this course because this is of course what we do with nodejs. We have to do that because this code can't be seen or changed by the user, the user can't disable us using that code because it happens on the server, not in the browser and this is the crucial part where we have to add validation, where we really have to filter out invalid values, so this is a must have, it's absolutely required and it is what we'll focus on. And this then ensures that we only work with valid data in our node app and ultimately if we do plan on storing it, that we do store correct data.
* Now also important, for some database engines and for most database engines actually, like for example mongodb, there is also a built in validation which you can turn on, I do cover that for the example in my mongodb course if you want to learn more about that. It's also optional because this can be a last resort but if you have good server side validation in place as you should have, then this might not be required because there is not really a scenario where invalid data could reach your database because you filter it out in that server side validation already ready. No matter which approach you have if you validate on the server side and or in the database, though or is not really an option you should validate on the server side at all means but no matter what you choose, in the end this can of course fail and then you should always return an error message, a helpful error message if possible and never reload the page but always keep the data the user already inserted because that of course is a horrible user experience which we all know that you enter something incorrect and you get back hey this password is not known or this e-mail address is unknown and you have to enter it all again.

**Lecture 291-292**

**Setup and Basic Validation**

* Refer code 01-setup-and-basic-validation
* Npm install –save express-validator 🡪 3rd party package for validation
* Routes/auth.js 🡪 adding the validation middleware for /signup route.
* Controllers/auth.js 🡪 postSignup method 🡪 showing validation methods
* Signup.ejs 🡪 adding *novalidate* attribute to form to disable the default browser validation of fields based on its type
* <https://express-validator.github.io/docs/>

**Lecture 293**

**Builtin and Custom validators**

* Refer code 03-built-in-and-custom-validators
* Routes/auth.js 🡪 adding more validation for email in /signup route

**Lecture 294**

**More Validators**

* Refer code 04-more-validators
* Routes/auth.js 🡪 adding validation for password /signup route

**Lecture 295**

**Checking for field equality**

* Refer code 04-checking-for-field-equality
* Routes/auth.js 🡪 adding validation for password and confirm password equality

**Lecture 296**

**Adding Async Validation**

* Refer code 05-added-async-validation
* Routes/auth.js 🡪 async validation for email existence
* Controllers/auth.js -🡪 removed validation for email existence
* Currently the validation of whether the email already exists or not is done in auth controller. We have to do that also in the auth.js routes
* The express validator package will check for a custom validator to return true or false, to return a thrown error or to return a promise. If it's a promise as it is the case with this because here we ultimately return a promise because every then block implicitly returns a new promise, so if we return a promise then express validator will wait for this promise to be fulfilled and if it fulfills with in our case nothing, so basically no error, then it treats this validation as successful. If it resolves with some rejection in the end though which will happen if we make it into this if block, then express validator will detect this rejection and will store this as an error, this message will then be stored as an error message. And this is how we can add our own asynchronous validation, asynchronous because we have to reach out to the database which of course is not an instant task but express validator will kind of wait for us here.

**Assignment Solution**

* Refer code assignment-solution.
* Controllers/auth.js 🡪 postLogin method
* Routes/auth.js 🡪 adding validation to /login route

**Lecture 297**

**Keeping user input**

* Refer code 06-keeping-user-input
* Controllers/auth.js 🡪 postSignup, postLogin method – passing entered values as oldInput

getSignup, getLogin method – passing empty values in oldInput attribute

**Lecture 298**

**Adding Conditional CSS Classes**

* Refer code 06-adding-conditional-css-classes
* Controllers/auth.js 🡪 passing validationErrors array to view in postSignup and getSignup
* Signup.ejs 🡪 adding conditional css classes
* Forms.css 🡪 adding invalid css class

**Lecture 299**

**Adding Validation to Login**

* Refer code 07-adding-validation-to-login
* Controllers/auth.js 🡪 getLogin , postLogin methods
* Login.ejs 🡪 adding conditional css classes

**Lecture 300**

**Sanitizing Data**

* Refer code 08-sanitizing-data
* Sanitizing – trimming white spaces, converting to lower case etc to store the data in a uniform way

**Lecture 301,302**

**Validating Product Addition and Editing**

* Refer code 09-validating-product-editing
* Auth.js routes 🡪 adding validation for /add-product and /edit-product
* Auth.js controller 🡪 getAddProduct , postAddProduct , getEditProduct , postEditProduct
* Edit-product.js 🡪 adding newly passed variable *hasError*, and *validationErrors*