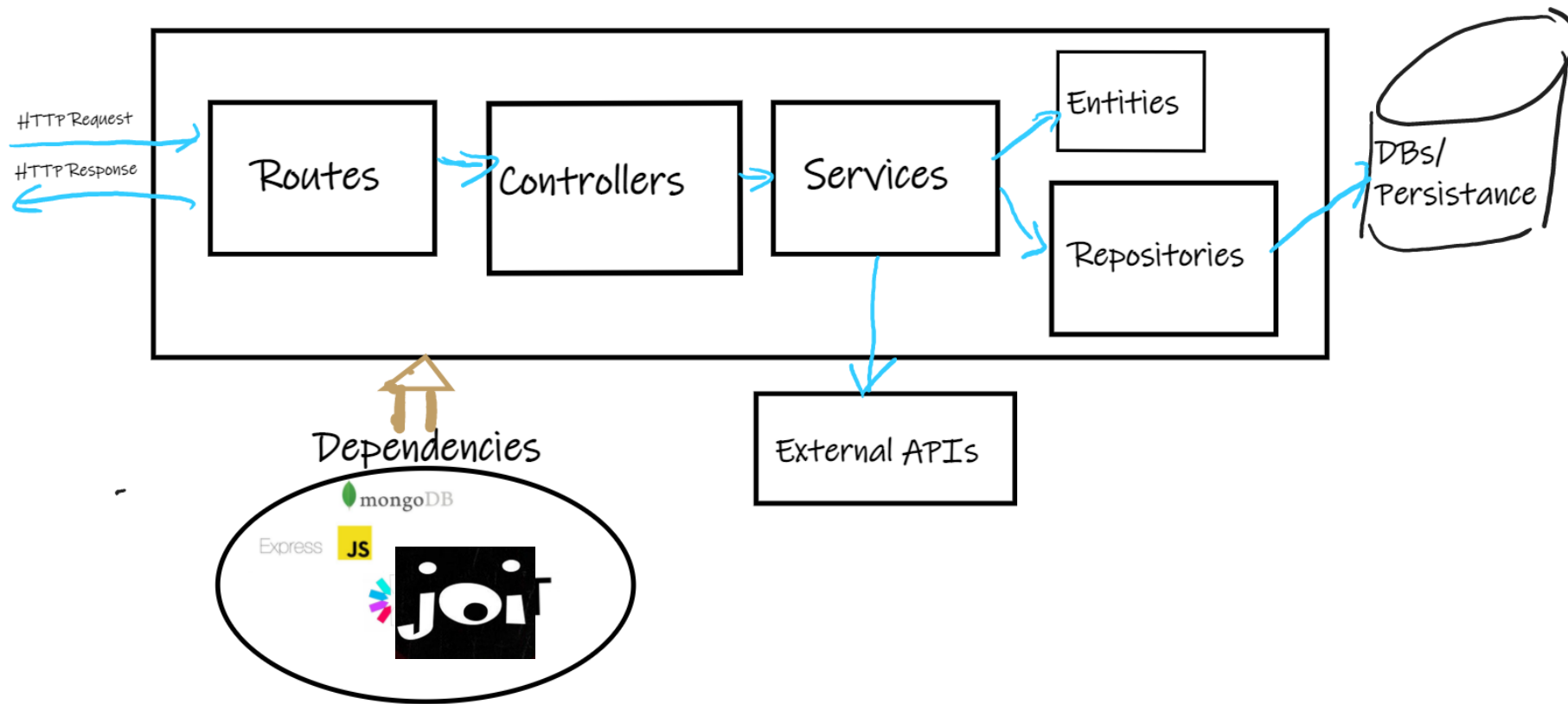


# The Joi of Validation

And some other stuff

# But first: Recap on Clean Architecture

Architecture we're following in the lab...




> controllers


> entities


> repositories

> routes

> services

 .babelrc

 .env

 .gitignore

 index.js

 package-lock.json

## Demo

---

Extend last weeks example to use in-memory DB

- Add Contacts Entity
- Add Contacts Repository
- Implement as In-Memory DB
- Add Contacts Controller
- Add Contacts Service

See Example Archive after lecture....

# Validation with Joi

# Context

- When creating a Contact you don't want this...

```
{  
  "userName": "a",  
  "name": "1234",  
  "password": "password",  
  "type": "A Friend",  
  "dob": "12 Jul",  
  "phone": "don't have one",  
  "email": "jonndjg"  
}
```

## Bad Data:

- Name should be alphanumeric at least
- Password is weak
- Type should be FRIEND, FAMILY, or OTHER
- DOB has no year
- Phone is not a phone number
- Email is not formatted properly


Prevent bad data making its way into your app: Use Data Validation

# Asside: Object Relational Mapping

- Mongoose, Sequelize, Knex provide for specification of validation constraints:
  - Handled in the application before persisting to DB
  - Good for enforcing Data layer rules (primary keys, no duplicate values)
  - Tightly coupled to the ORM implementation (what if I want to change DB?)
- In an API, data arriving via HTTP to endpoints
  - Good idea Validate at request level, using validation middleware/controller
  - In Express we can use middleware on the route.

# Joi

- Use this module to validate data at the request level.

joi 

17.6.0 • Public • Published 2 months ago

[Readme](#) [Explore](#) [5 Dependencies](#) [8,610 Dependents](#) [210 Versions](#)

## joi

The most powerful schema description language and data validator for JavaScript.

### Installation

```
npm install joi
```

Visit the [joi.dev](#) Developer Portal for tutorials, documentation, and support

### Useful resources

- [Documentation and API](#)
- [Versions status](#)

### Install

```
> npm i joi
```

### Repository


[github.com/sideway/joi](#)

### Homepage

[github.com/sideway/joi#readme](#)

### Weekly Downloads

5,985,882



# Using Joi: Define Schema

```
import Joi from 'joi';

const contactsSchema = Joi.object({
  email: Joi.string().email().lowercase().required(),
  password: Joi.string().min(4).required(),
  userName: Joi.string().min(1).required(),
  name: Joi.string(),
  dob: Joi.string(),
  type: Joi.string(),
  phone: Joi.string()
});

export default contactsSchema
```

email is a String, formatted as email, lowercase, and must be present(required)

Password is a String, length of 4 or longer, and must be present

Username is a String



# Using Joi: Include in Dependencies

- Include Schema as dependencies passed to Route

```
import ContactsRepositoryInMemory from './src/contacts/repositories/in-memory/ContactsRepository';  
import contactsSchema from './src/contacts/validators';  
  
const app = express();  
  
const dependencies = {  
  contactsRepository : new ContactsRepositoryInMemory(),  
  contactsValidator: contactsSchema  
};  
  
//Route Middleware  
app.use('/api/contacts', createContactsRouter(dependencies))
```


In keeping with clean architecture approach, Validator schema introduced using dependency injection.

# Validation Schema, Router

Validation Controller introduced onto route  
Request Response cycle stops if Validation fails .

```
export default (dependencies) => {  
  
  const { contactsValidator } = dependencies;  
  
  const validateContact = async (request, response, next) => {  
    // Input  
    try {  
      const validated = await contactsValidator.validateAsync(request.body)  
      request.body = validated  
      next()  
    } catch (err) {  
      next(new Error( err.message))  
    }  
  }  
  
  return {  
    validateContact  
  };  
}
```

```
import express from 'express';  
import ContactsController from '../controllers';  
import ContactsValidator from '../controllers/validation';  
  
const createRouter = (dependencies) => {  
  
  const router = express.Router()  
  const contactsController = ContactsController(dependencies);  
  const contactsValidator = ContactsValidator(dependencies);  
  
  router.post('/', contactsValidator.validateContact, contactsController.createContact);  
}
```



# Joi Validation: Dates

- Do 'npm install @joi/date' to install date format extension

```
import Joi from 'joi';
import JoiDate from '@joi/date';

const NewJoi = Joi.extend(JoiDate);

const contactsSchema = Joi.object({
  dob: NewJoi.date().format("DD/MM/YYYY"),
```

Adds Date extension to Joi

Checks date conforms to DD/MM/YYYY  
format

# Joi Validation: Enumerations

- Specify list of valid values:

```
// validators/registerValidators.js
import Joi from 'joi';
import JoiDate from '@joi/date';

const NewJoi = Joi.extend(JoiDate);

const contactsSchema = Joi.object({
  type: Joi.string().valid('FRIEND', 'FAMILY', 'OTHER').uppercase(),
  // ... other fields ...
});
```

Type must be in list of values

# Joi Validation: Regular Expressions

- Use Regular expressions to validate password/phone properties

```
const contactsSchema = Joi.object({  
  password: Joi.string().min(7).required().regex(/^(?=.*[A-Za-z])(?=.*\d)(?=.*[@$!%*#?&])[A-Za-z\d@$!%*#?&]{7,}$/),  
  phone: Joi.string().regex(/^s*\+?\s*([0-9][\s-]*){9,}$/),  
});
```

Phone number has length >9 and only contains numbers, spaces and '-'. Can begin with '+'

Ensures Password has length >7 and had letter, number and special char