

Transformando
código spaghetti
em código...

Lasagna



Tallysson

@tallyssonoc

tallyssonoc.github.io

Web dev / Codeminer42



O que é código spaghetti?





Código spaghetti



Código spaghetti dividido em “services”

Como identificar?

- Difícil de encontrar coisas
- Difícil de testar
- Nomes de classes que indicam “fazedores” (-er, -or)
- Acoplamento entre as unidades
- Adicionar features implica em mexer em vários lugares
- Difícil de modificar



“

...if you're afraid to change
something it is clearly poorly
designed.

- Martin Fowler

Código spaghetti não tem estrutura fácil de mudar

Código spaghetti não segue uma boa arquitetura

O que é arquitetura?

O que não é arquitetura?

- Não é conjunto de tecnologias
- Não é conjunto de bibliotecas e framework usado
- Não é onde sua aplicação roda (web, desktop, CLI)

O que é arquitetura?

- Separar responsabilidades
- Separar conceitos
- Deixar explícito o que sua aplicação realmente faz
- Priorizar a parte mais importante da sua aplicação

Responsabilidades

```
const { User, Address } = require('../models');  
const UserWelcomeMailer = require('../mailers');
```

```
app.post('/users', async (req, res) => {
```

```
  const userData = req.body.user;
```

```
  if(!userData.name) {
```

```
    return res.status(422).json({ error: 'Name is required' });
```

```
  }
```

```
  const { address } = userData;
```

```
  if(!address || !address.street || !address.number || !address.zipCode) {
```

```
    return res.send(422).json({ error: 'Address is required' });
```

```
  }
```

```
  try {
```

```
    const user = await User.create(userData);
```

```
    user.address = await Address.create(address);
```

```
    res.status(201).json({ user });
```

```
    UserWelcomeMailer.send({ user });
```

```
  } catch(error) {
```

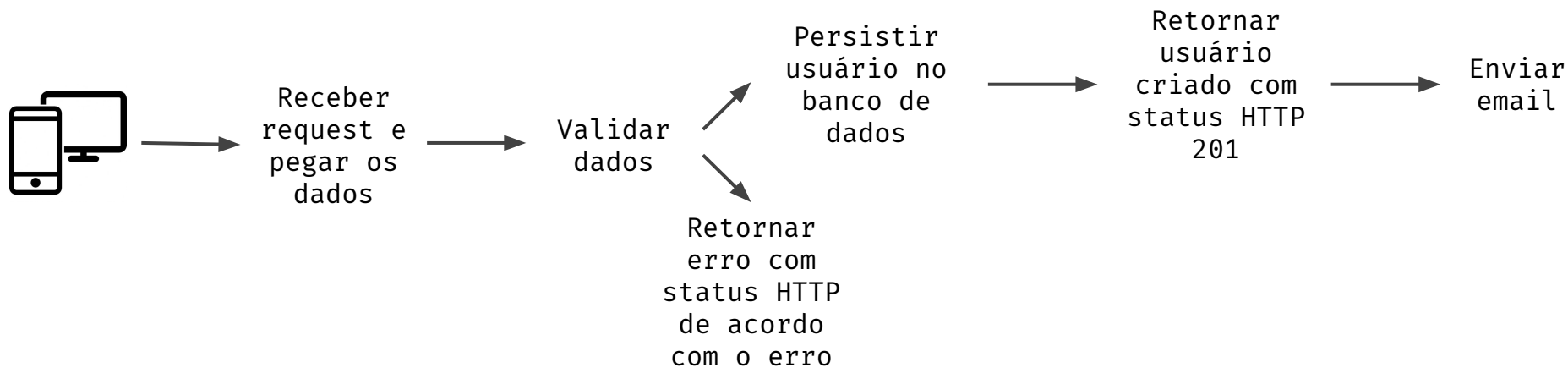
```
    res.send(400).json({ error });
```

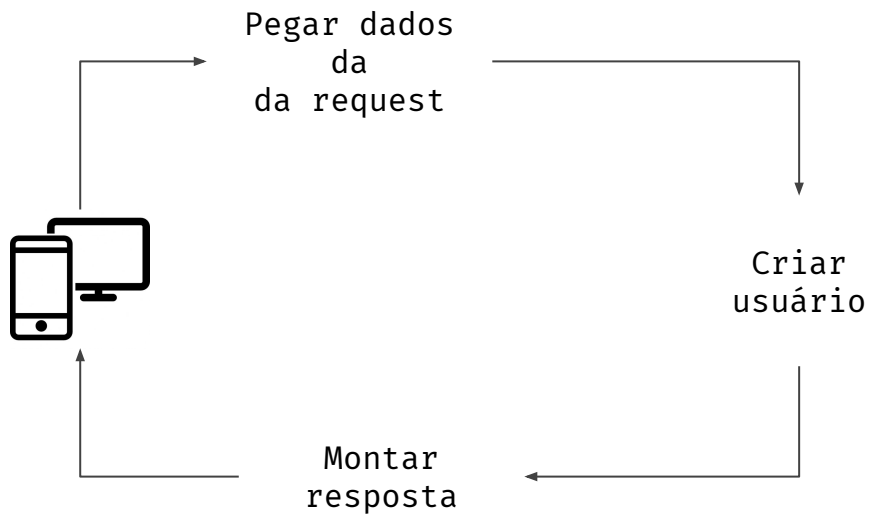
```
  }
```

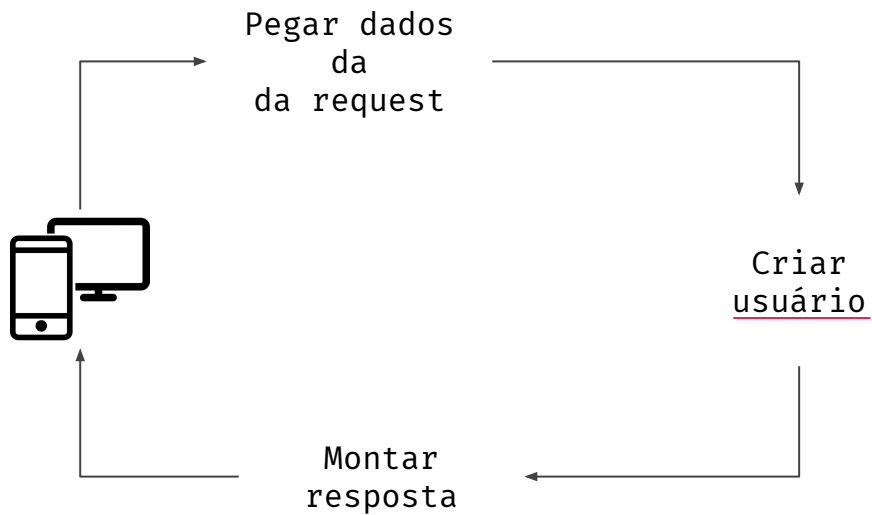
```
});
```



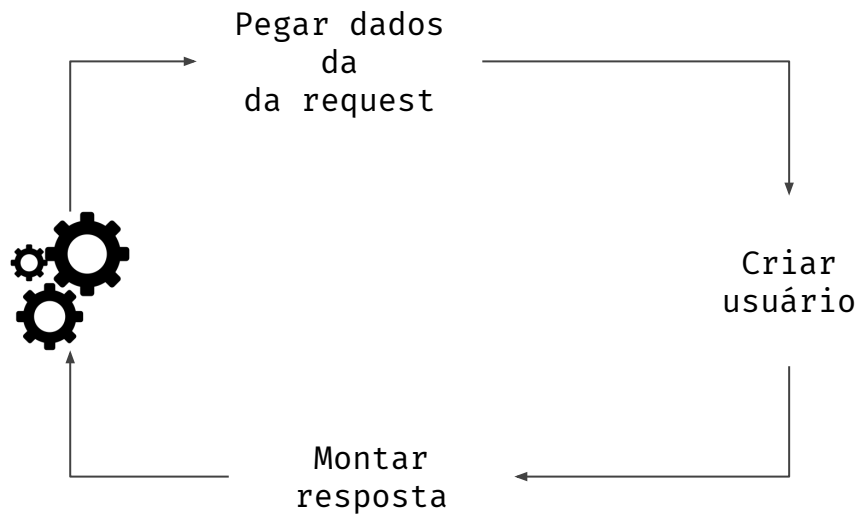
Qual a parte mais importante deste fluxo?

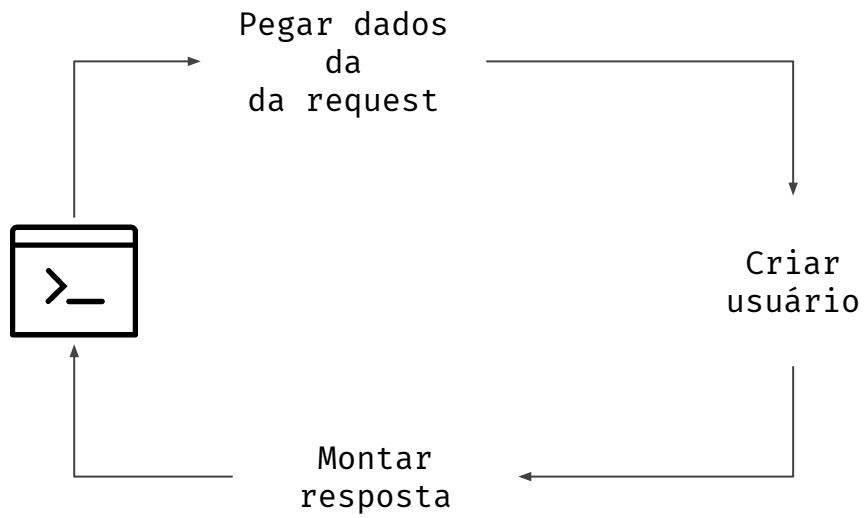


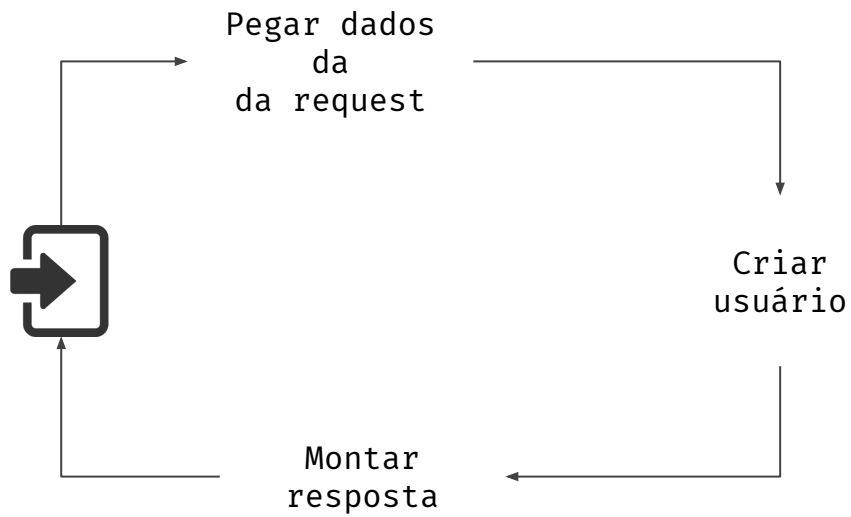




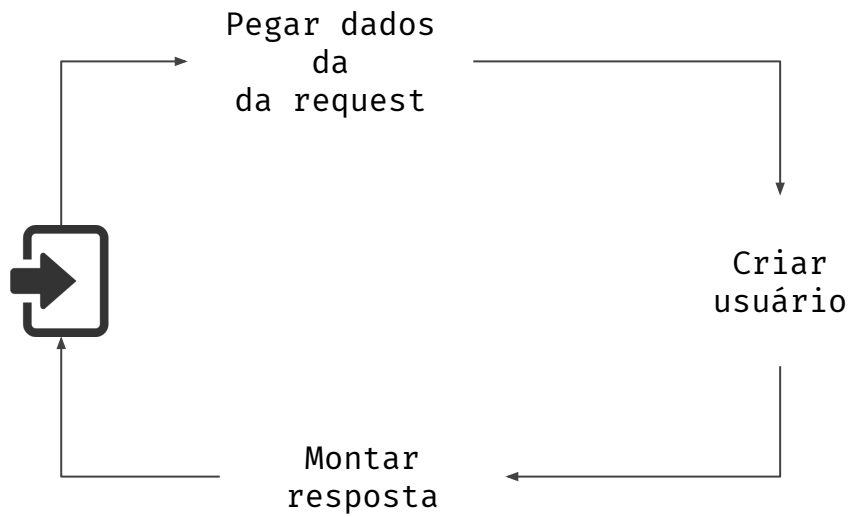
Caso de uso da aplicação





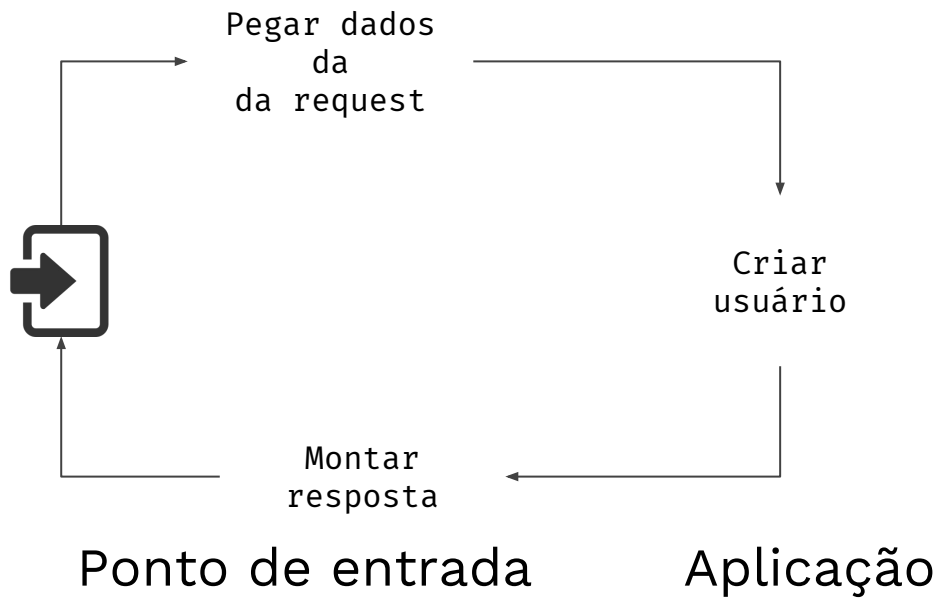


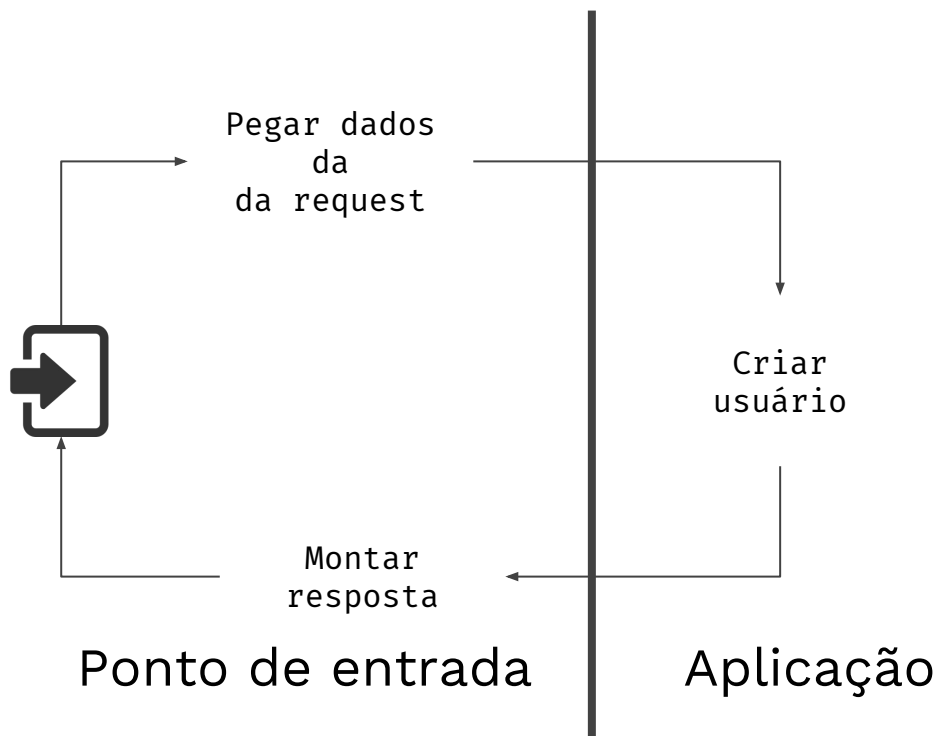
Ponto de entrada

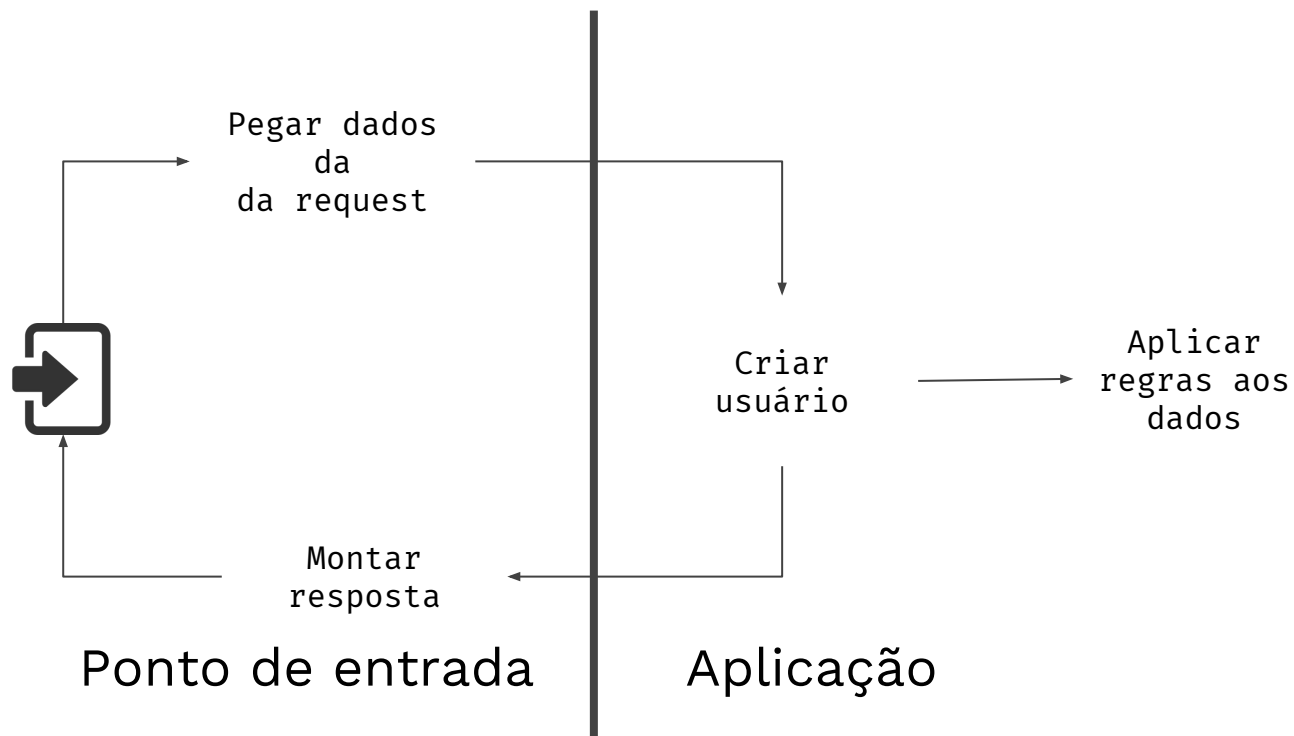


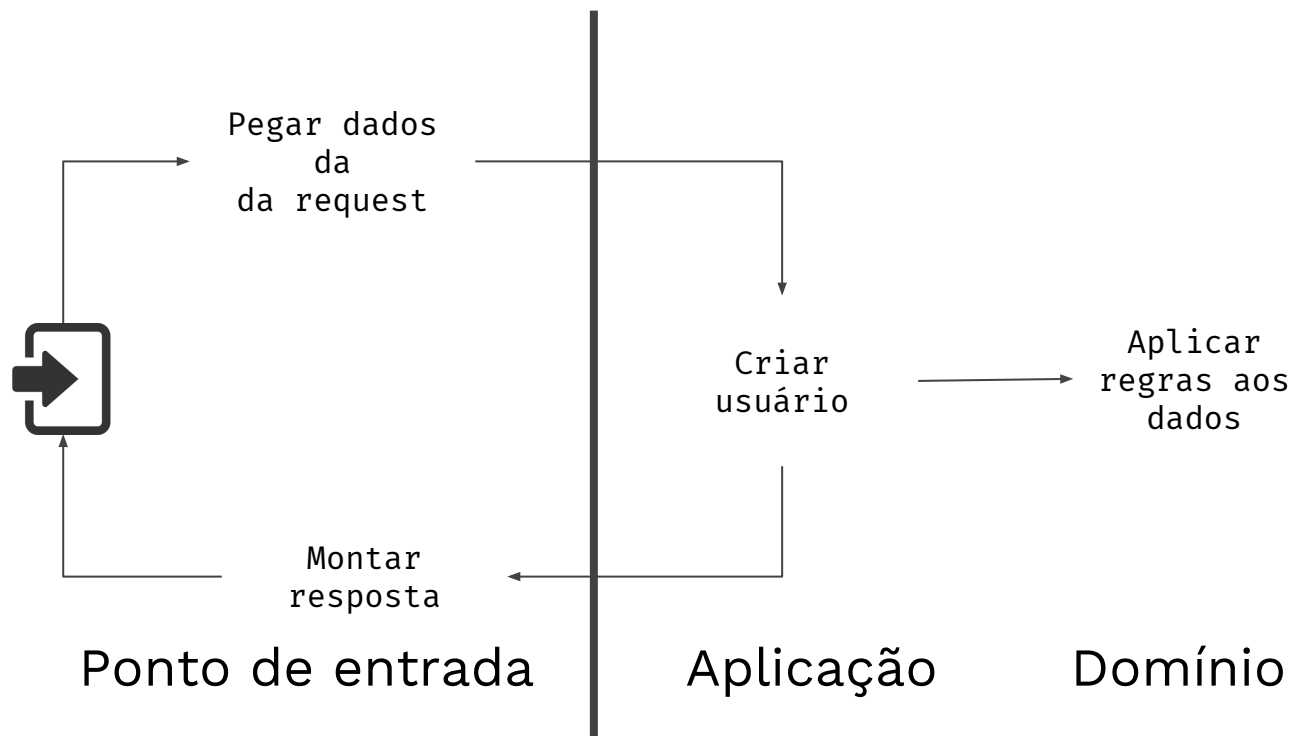
Ponto de entrada

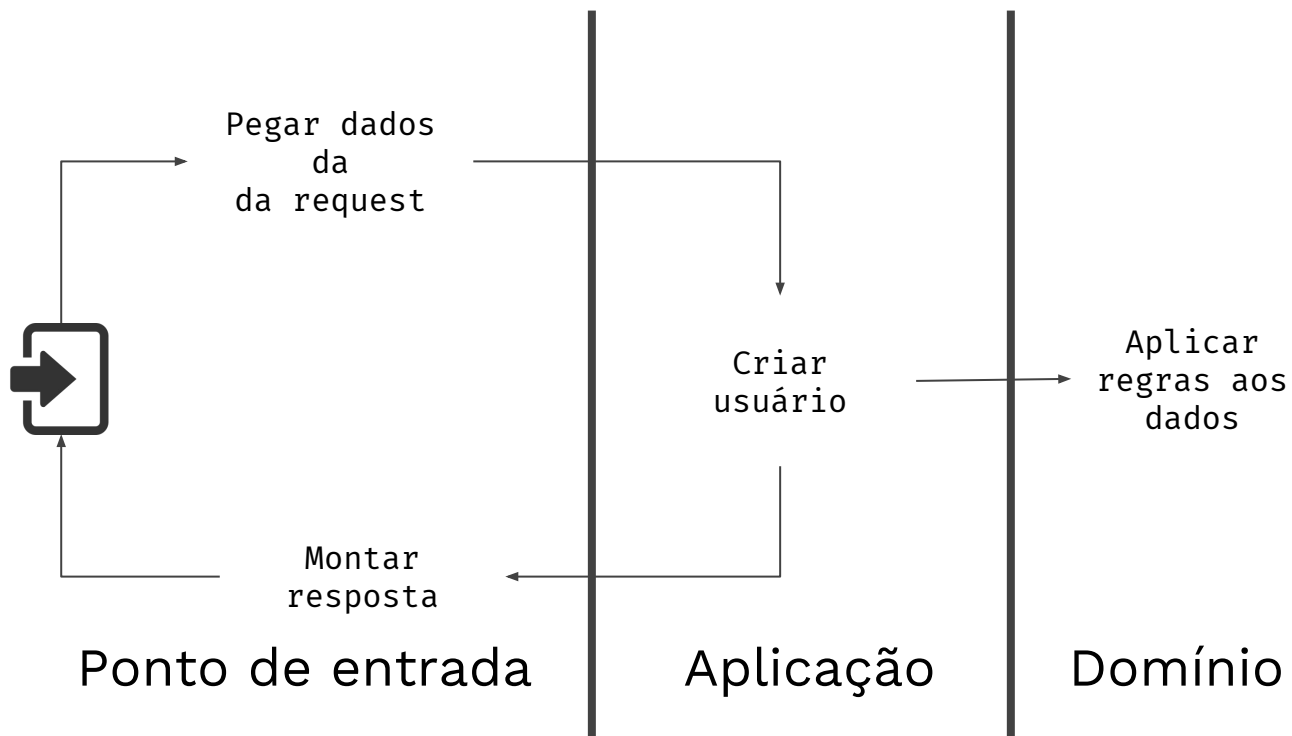
Caso de uso

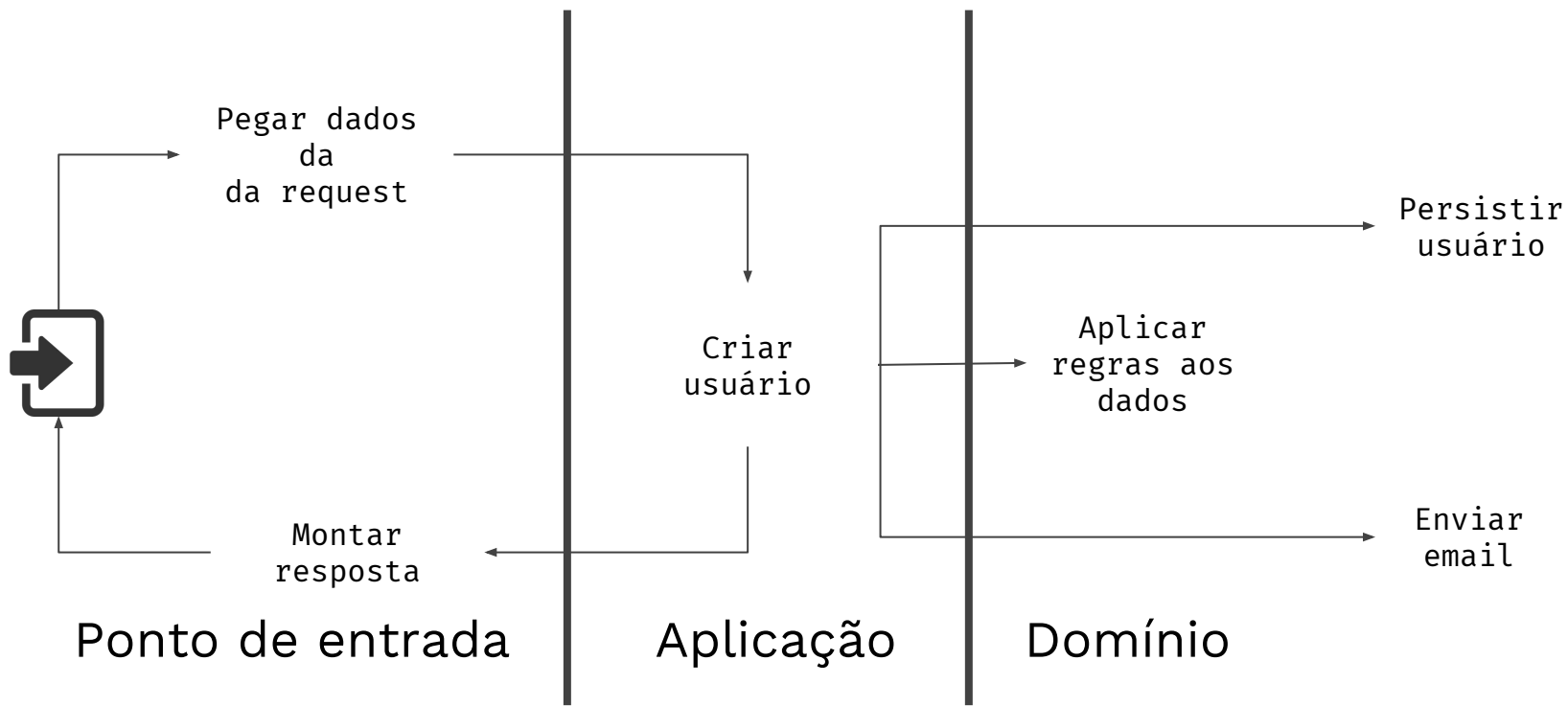


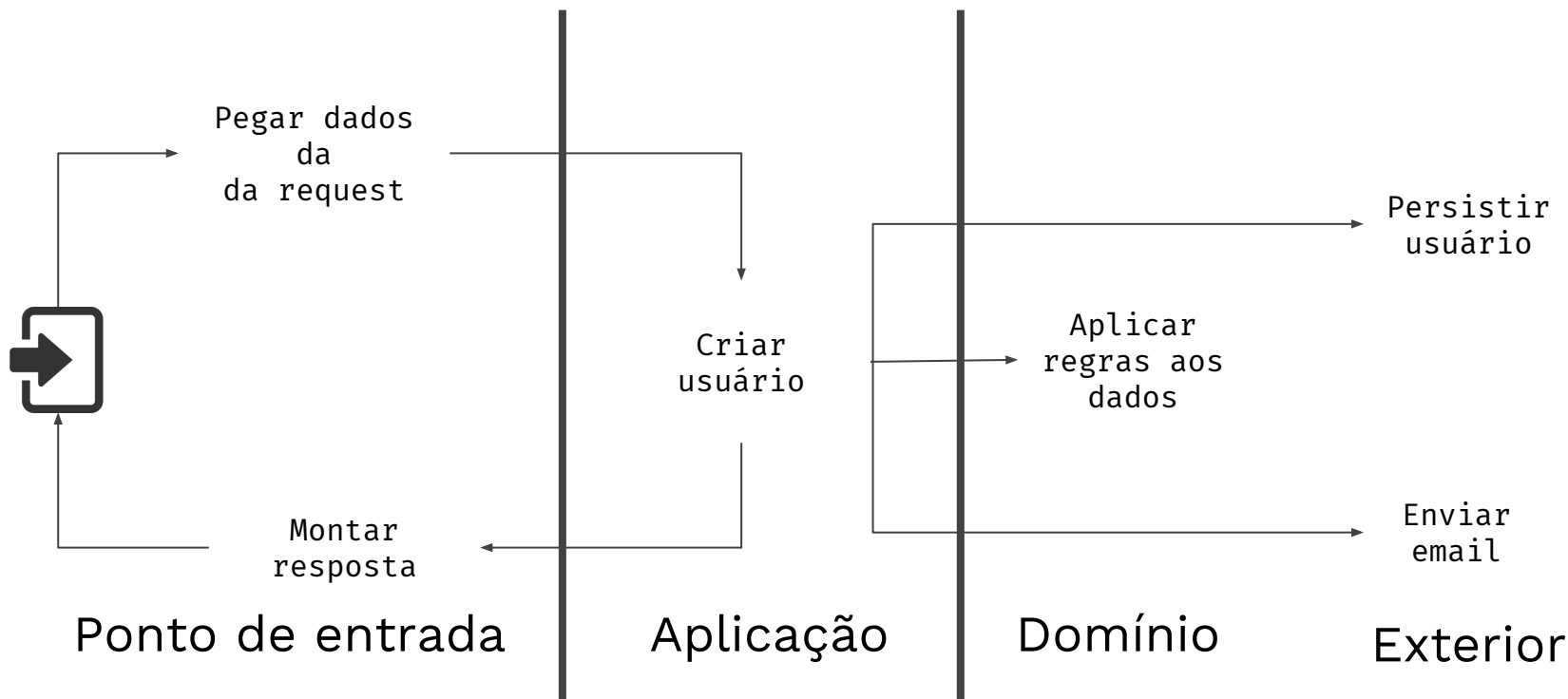


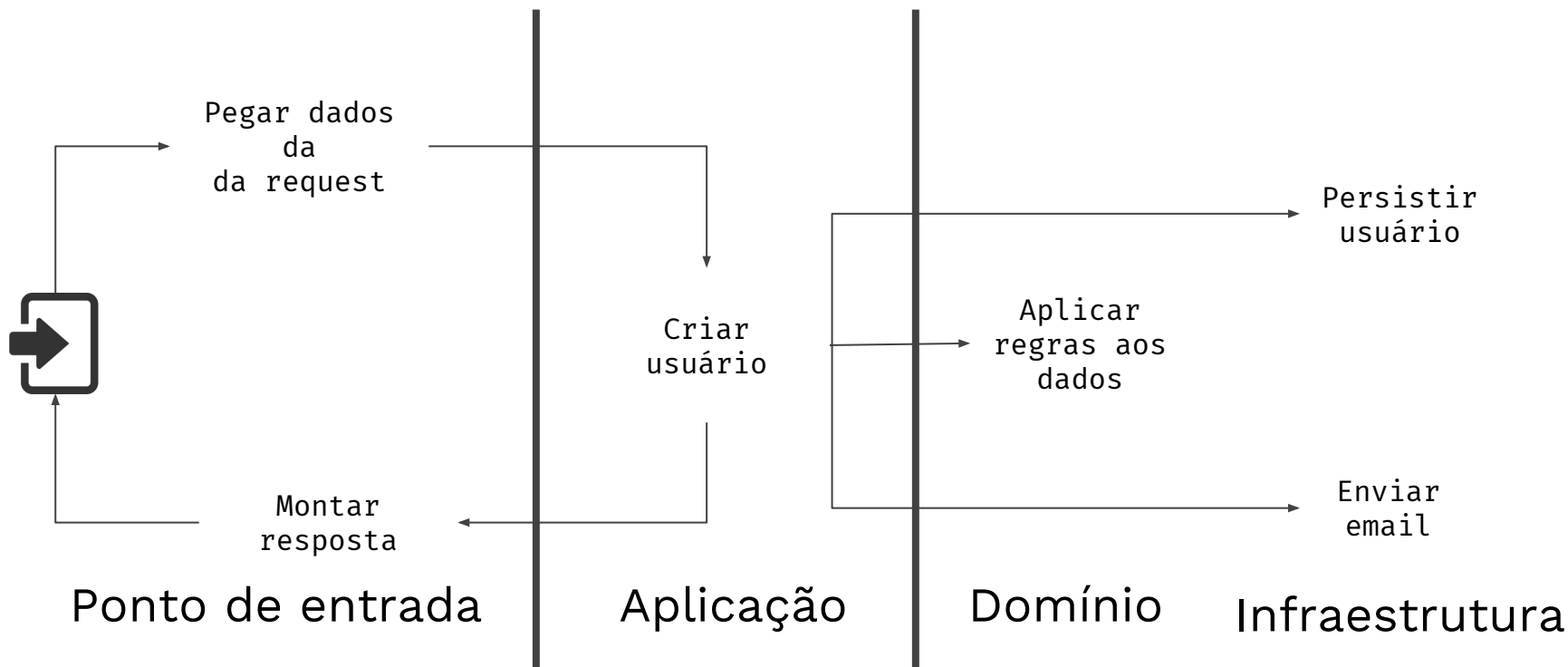


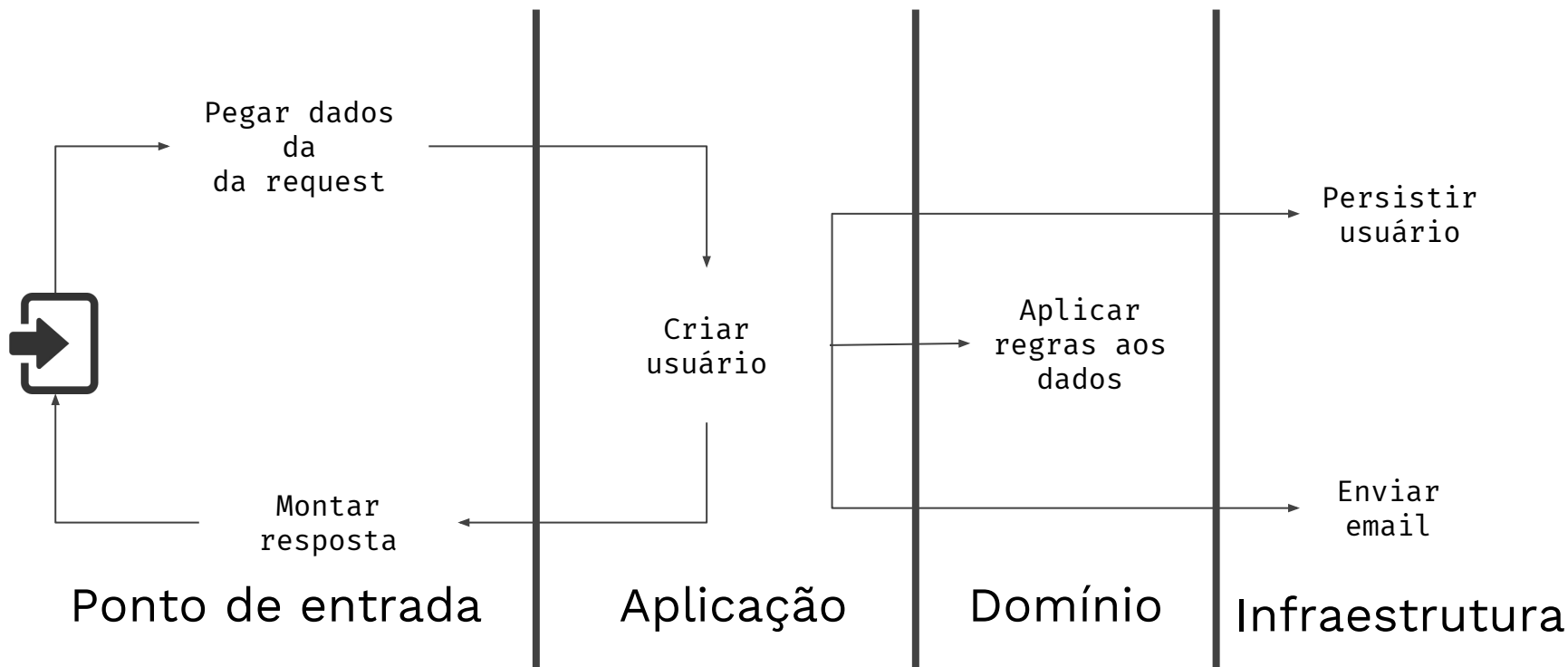


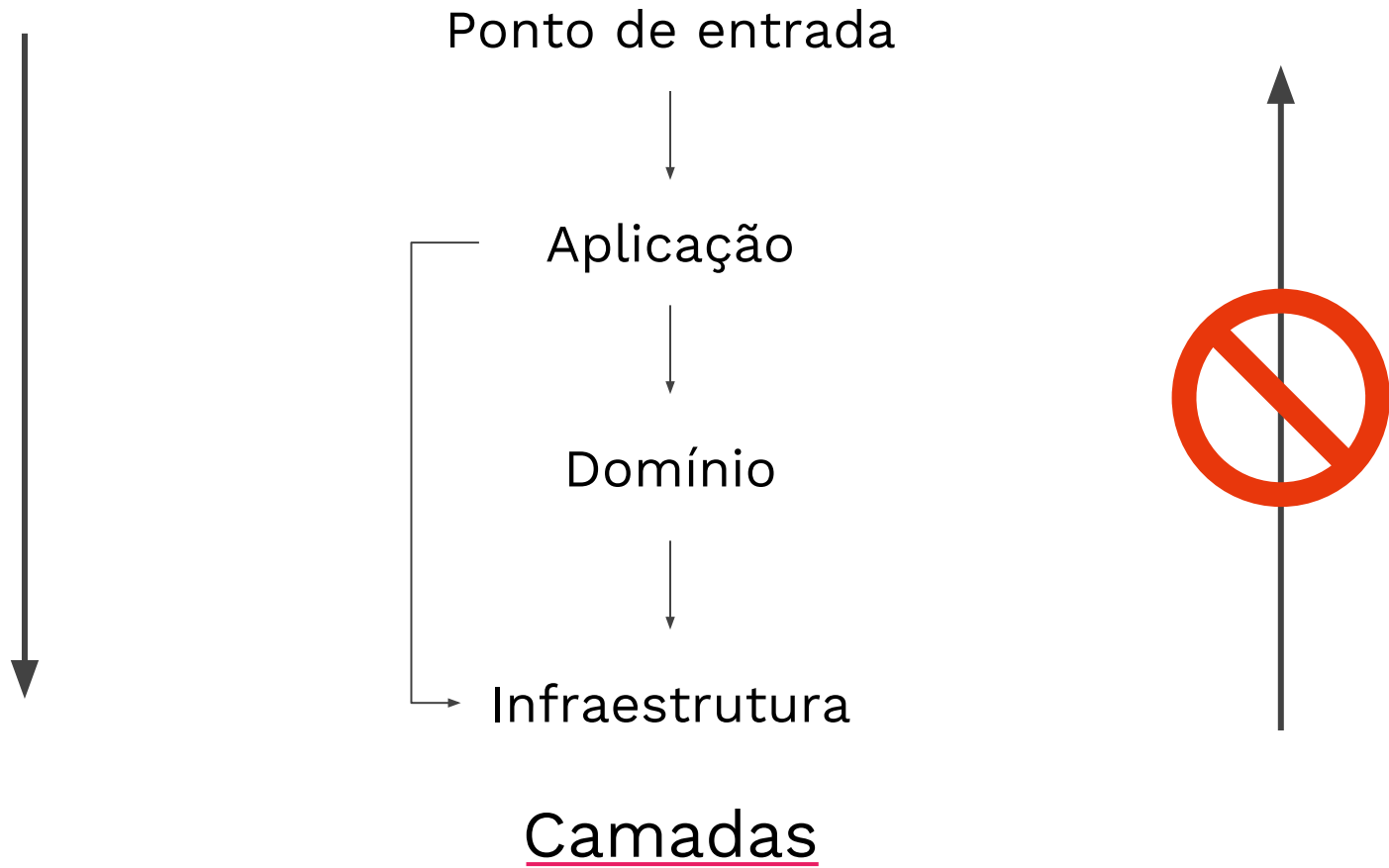














Antes



Depois

≈ Camadas ≈

Por ordem de importância

Domínio

- Entidades e regras de negócio
- Camada mais isolada e *importante*
- Usada pela camada de aplicação para definir casos de uso
- Ex.: User, Group, JoinGroupRule

```
class User {  
    constructor({ name, age }) {  
        this.name = name;  
        this.age = age;  
    }  
  
    isValid() {  
        return Boolean(this.name) && this.isOfAge();  
    }  
  
    isOfAge() {  
        return this.age >= 18;  
    }  
}
```

- Casos de uso e funcionalidades
- Realiza a interação entre *unidades* de domínio
- Casos de uso tem mais de dois possíveis resultados
- Também é um *adapter* para a camada de infraestrutura
- Ex.: JoinGroup, CreateUser, EmailService

Aplicação

```
class CreateUser extends EventEmitter {  
  constructor({ userRepository }) {  
    this.userRepository = userRepository;  
  }  
  
  async execute(userData) {  
    const user = new User(userData);  
  
    if(!user.isValid()) { return this.emit('invalidUser'); }  
  
    try {  
      const newUser = await this.userRepository.add(user);  
      this.emit('success', newUser);  
    } catch(error) {  
      this.emit('error', error);  
    }  
  }  
}
```

```
const createUser = async (userData, { onSuccess, onError }) => {  
  const user = new User(userData);  
  
  if(!user.isValid()) {  
    return onError(new Error('Invalid user'));  
  }  
  
  try {  
    const newUser = await UserRepository.add(user);  
  
    onSuccess(newUser);  
  } catch(error) {  
    onError(error);  
  }  
};
```

Infraestrutura

- Comunicação direta com o exterior do software
- A mais baixa das camadas
- Tratada como *detalhe de implementação*
- Ex.:
UserRepository/UserStorage,
MailchimpService,
PayPalService


```
class UserRepository {  
  async add(user) {  
    const userDbData = UserMapper.toDatabase(user);  
  
    const newUser = await UsersTable.insert(userDbData);  
  
    return UserMapper.toEntity(newUser);  
  }  
}
```

```
const UserRepository = {
  async add(userData) {
    const response = await fetch('http://api.example.com/users', {
      method: 'POST',
      body: JSON.stringify(userData)
    });

    const newUserData = await response.json();

    const newUser = new User(newUserData);

    return newUser;
  }
};
```

- Menos importante das camadas
- Sem *nenhum* tipo de regra de negócio
- Pega dados da entrada e passa para a camada de aplicação
- Ex.: controllers (HTTP), workers (filas), CLI, actions (Redux)

Pontos de Entrada

```
const UsersController = {
  create(req, res, next) {
    const createUser = new CreateUser();
    createUser
      .on('success', (user) => {
        res.status(201).json(userSerializer.serialize(user));
      })
      .on('invalidUser', (error) => {
        res.status(400).json({
          type: 'ValidationError',
          details: error.details
        });
      })
      .on('error', next);

    createUser.execute(req.body);
  }
};
```

```
const createUserAction = (userData) => (dispatch) => {  
  dispatch(startCreateUser());  
  
  createUser(userData, {  
    onSuccess: (newUser) => dispatch(successCreateUser(newUser)),  
    onError: (error) => dispatch(failCreateUser(error));  
  });  
};
```

Conectando camadas


- Comunicação direta causa acoplamento
- Injeção de dependência
- Inversão de controle (IoC)
- Não deve fazer com que se crie *mais* acoplamento
- Soluções

```
const UsersController = {
  create(req, res, next) {
    const createUser = new CreateUser();
    createUser
      .on('success', (user) => {
        res.status(201).json(userSerializer.serialize(user));
      })
      .on('invalidUser', (error) => {
        res.status(400).json({
          type: 'ValidationError',
          details: error.details
        });
      })
      .on('error', next);

    createUser.execute(req.body);
  }
};
```



Acoplamento

```
class CreateUser extends EventEmitter {  
  constructor({ userRepository }) {  Injeção de dependência  
    this.userRepository = userRepository;  
  }  
}
```

```
async execute(userData) {  
  const user = new User(userData);  
  
  if(!user.isValid()) { return this.emit('invalidUser'); }  
  
  try {  
    const newUser = await this.userRepository.add(user);  
    this.emit('success', newUser);  
  } catch(error) {  
    this.emit('error', error);  
  }  
}
```


Awilix

NodeJS

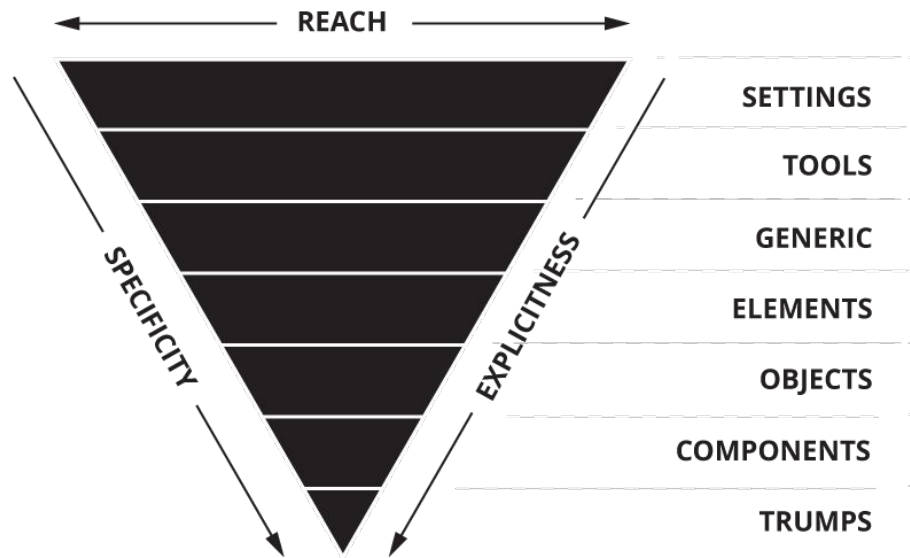
- Injeção de dependência de maneira isolada
- Não causa mais acoplamento
- Não atrapalha testar
- Fácil de usar
- Possui adapters para Express, Koa e Hapi
- npmjs.com/package/awilix

- Injeção de dependência diretamente nas actions
- Não modifica o fluxo do Redux
- Não adiciona complexidade
- <https://github.com/reduxjs/redux-thunk#injecting-a-custom-argument>

withExtraArgument

Redux + Redux-Thunk

Também vale
para CSS!



ITCSS

- Separação demais no código causa complexidade
- A aplicação não pode ficar difícil de entender
- Encontre o equilíbrio
- Não precisa aplicar tudo



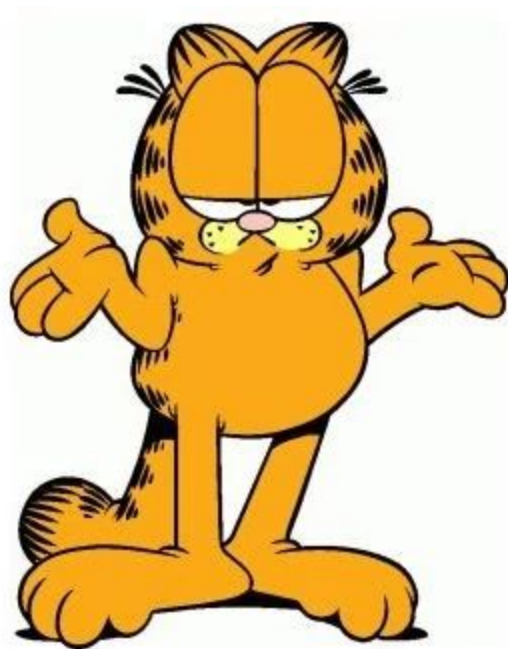
Sem exageros!

Show me the code!

Node API Boilerplate

<https://goo.gl/tgQH8v>





- Convenção sobre configuração
- Ferramentas já estabelecidas da comunidade
- Baixa curva de aprendizagem
- Arquitetura escalável
- Segue ideias do 12 Factor App
- Pronto para sair produzindo
- @talyssonoc / #euquerolasagna
- talyssonoc@gmail.com

Lasagna framework

Links

- Bob Martin - Architecture the lost years: <https://youtu.be/WpkDN78P884>
- Rebecca Wirfs-Brock - Why We Need Architects (and Architecture) on Agile Projects: <https://youtu.be/Oyt4Ru7Xzq0>
- Mark Seemann - Functional architecture - The pits of success: <https://youtu.be/US8QG9l1XW0>
- Bob Martin - The Clean Architecture: <https://goo.gl/2N92AV>
- Domain-Driven Design Books - <https://goo.gl/27bjVK>
- Eu mesmo - NodeJS and Good Practices: <https://goo.gl/YPNpoh>
- Eu mesmo 2 - NodeJS e boas práticas: <https://goo.gl/HNn9Xm>
- Jeff Hansen - DI in NodeJS: <https://goo.gl/jasFHm>
- Iago Dahlem - How to Organize your Styles with ITCSS: <https://goo.gl/YopDzz>

Obrigado!

@tallyssonoc
tallyssonoc.github.io