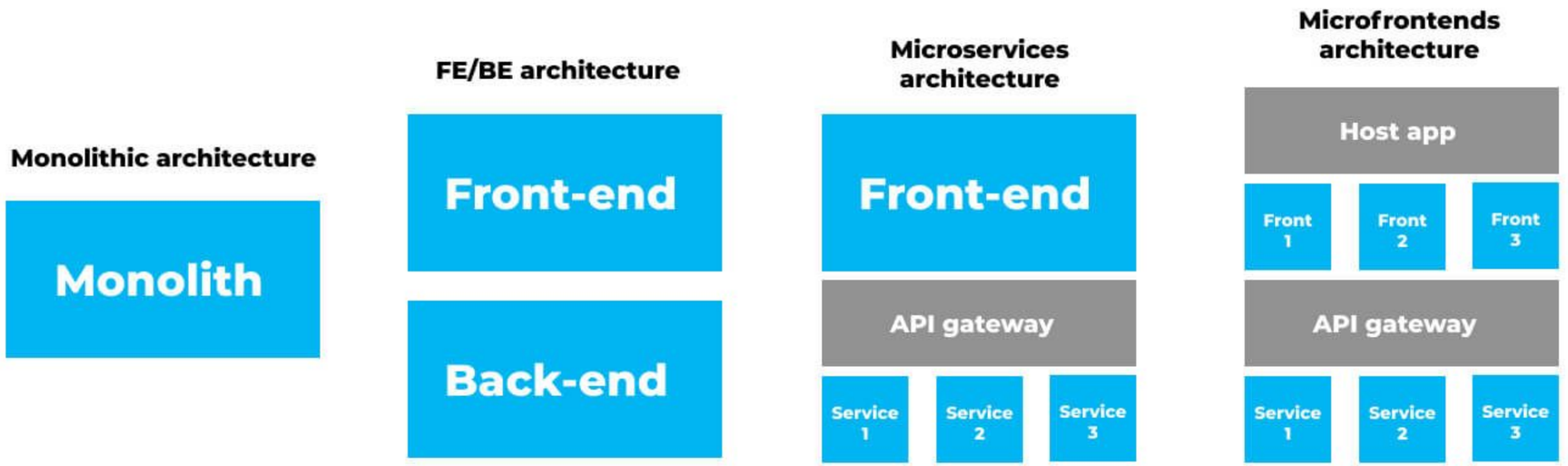
Arquitectura Front

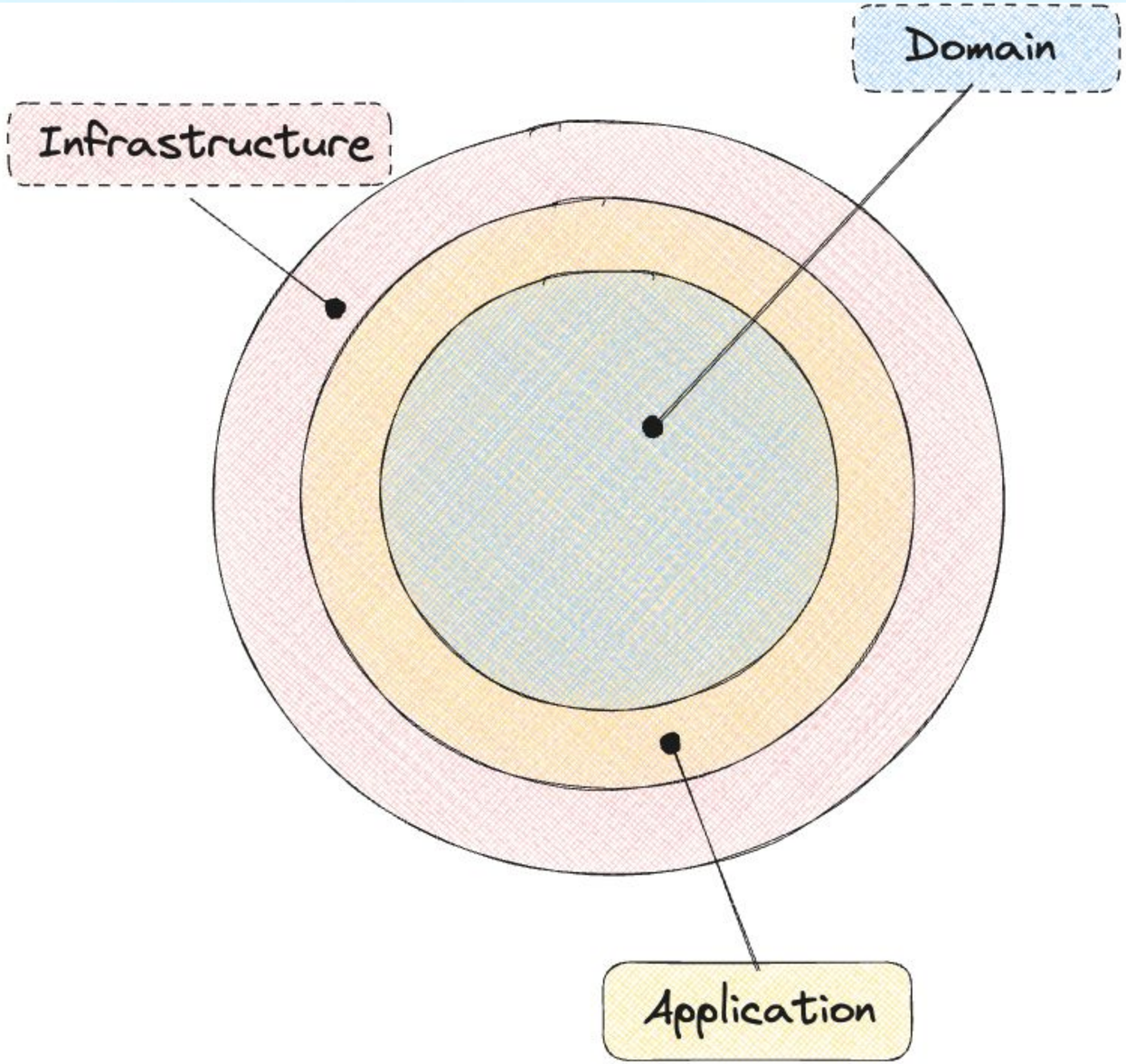
Un corte de sostenibilidad

Adrián Ferrera González, Thu 29 Feb 2024

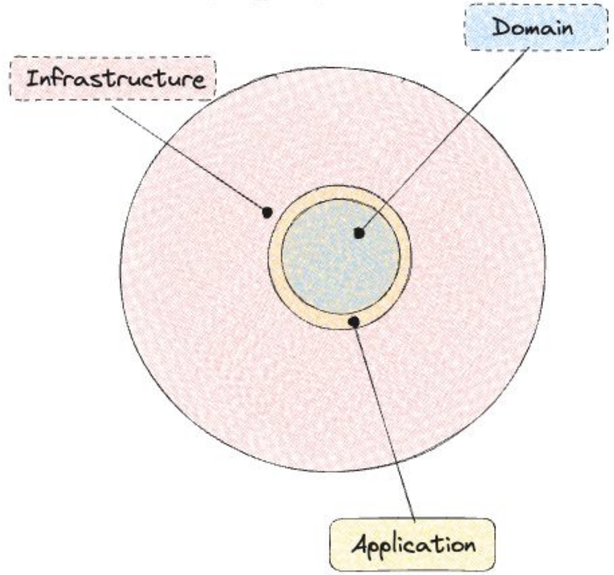
@AdrianFerrera91

Tienes LA solución

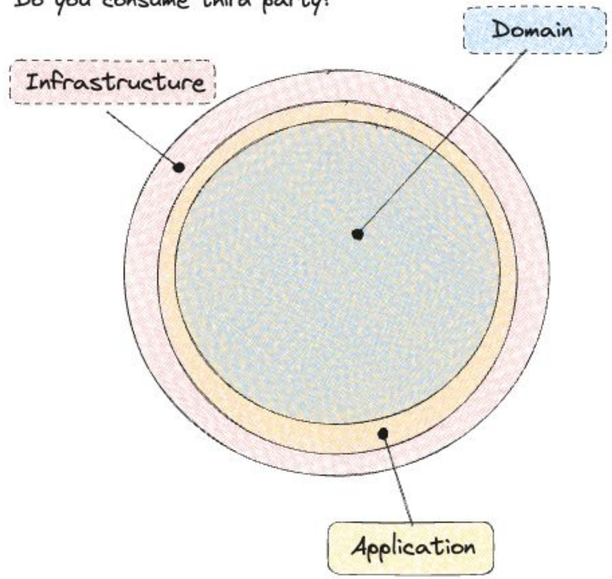




Are you the backends owner?
Your front looks like this:



Do you consume third party?



Responsabilidad,
Cohesión,
Acoplamiento
+ Testing
~~Sostenibilidad~~

¿Qué hacemos en Frontend?

- Pintar componentes
- Usamos librerías de terceros
- Usamos recursos del navegador
- Gestionar estado
- Ejecutamos nuestra lógica

Pintando componentes

```
export const MyPaymentContainer = ({
  purchase = purchaseUseCase()
}) => {
  const onPurchase = () => { {/* Do some stuff */} }
  const onCancel = () => { /* Do some stuff */ } }

  return (
    <MyPayment
      onPurchase={onPurchase}
      onCancel={onCancel}
    />
  )
}
```

```
export const MyButton = ({children, onClick}) => (
  <button onClick={onClick}>{children}</button>
)

export const MyConfirmationModal: React.FC<Props> = ({ children, onAccept,
onCancel, onClosed }) => {
  return (
    <div>
      <MyButton onClick={onClosed}>Close</MyButton>
      <div>{children}</div>
      <MyButton onClick={onAccept}>Accept</MyButton>
      <MyButton onClick={onCancel}>Open</MyButton>
    </div>
  )
}

export const MyPayment: React.FC<Props> = (
  { onPurchase, onCancel }
) => {
  let paymentModal = false;
  const onOpen = () => { paymentModal = true; }
  const onClose = () => { paymentModal = false; }
  const handleCancel = () => { paymentModal = false; onCancel(); }
  const handlePurchase = () => {
    onPurchase();
    onClose();
  }

  return (
    <div>
      {/* ... */}
      <MyButton onClick={onOpen}>Pay</MyButton>
      {paymentModal && <MyConfirmationModal
        onAccept={handlePurchase}
        onCancel={handleCancel}
        onClosed={onClose}>
          <p>Do you accept the payment?</p>
      </MyConfirmationModal>}
      {/* ... */}
    </div>
  )
}
```

¿Cómo probarlo?

```
// Unit Test
describe('MyButton Component', () => {
  it('triggers click event', () => {
    const givenText = 'Irrelevant test'
    const onClickMock = vi.fn()
    render(
      <MyButton onClick={onClickMock}>
        {givenText}
      </MyButton>
    )

    fireEvent.click(screen.getByText(givenText))

    expect(onClickMock).toHaveBeenCalled()
  })
})
```

```
// Social Unit Test
describe('MyPaymentContainer Component', () => {
  it('calls purchase use case', () => {
    const purchaseUseCaseMock = vi.fn()
    render(
      <MyPaymentContainer
        purchase={purchaseUseCaseMock} />
    )
    // ... simulate user operations
    fireEvent.click(screen.getByText('Accept'))

    expect(purchaseUseCaseMock).toHaveBeenCalled()
  });
});
```

Usamos librerías de terceros

```
import {MyFancyFormLibrary} from "MyFancyFormLibrary.ts";
import {config} from "./config.ts";

export const MyForm1 = () => {
  const handleChange = () => {
    MyFancyFormLibrary.validateForm({
      id: 'form1',
      exhaustive: true,
      rules: config
    })
  };
  return <form id="form1"><input onChange={handleChange}/></form>;
}

export const MyForm2 = () => {
  const handleSubmit = () => {
    MyFancyFormLibrary.validateForm({
      id: 'form2',
      exhaustive: true,
      rules: config
    })
  };
  return <form id="form2" onSubmit={handleSubmit}>{/**/}</form>;
}
```

```
import {MyFancyFormLibrary} from "MyFancyFormLibrary.ts";
import {config} from "./config.ts";

export const validateForm = (formId: string) => {
  MyFancyFormLibrary.validateForm({
    id: formId,
    exhaustive: true,
    rules: config
  })
}

export const MyForm1 = () => {
  const handleChange = () => {
    validateForm('form1')
  };
  return <form id="form1"><input onChange={handleChange}/></form>;
}

export const MyForm2 = () => {
  const handleSubmit = () => {
    validateForm('form2')
  };
  return <form id="form2" onSubmit={handleSubmit}>{/**/}</form>;
};
  return <form id="form2" onSubmit={handleSubmit}>{/**/}</form>;
}
```

¿Y los tests?

```
// Do we really need to test the library?
// Do we really need to test a wrapper?
describe('MyForm Component', () => {
  it('can be submitted when information is valid', () => {
    const mock = vi.fn().mockImplementation(validateForm)
    mock.mockImplementationOnce(() => true)
    render(<MyForm2 />)
    // ... simulate user operations
    const submitButton = screen.getByText('Submit')
    expect(submitButton).toBeEnabled()
  });
});
```

Usamos recursos del navegador

```
const get = async <T>(url: string, headers: any) => {
  const response = await fetch(url, {
    method: 'GET',
    headers: {...headers}
  })
  return await response.json() as T
}

export const httpClient = {get}
```

```
const get2 = async <T>(url: string, headers: any) =>
{
  const response = await axios.get(url, {
    method: 'GET',
    headers: {...headers}
  })
  return await response.data as T
}

export const httpClient = {get}
```

Gestionamos el estado

¿O es una caché?

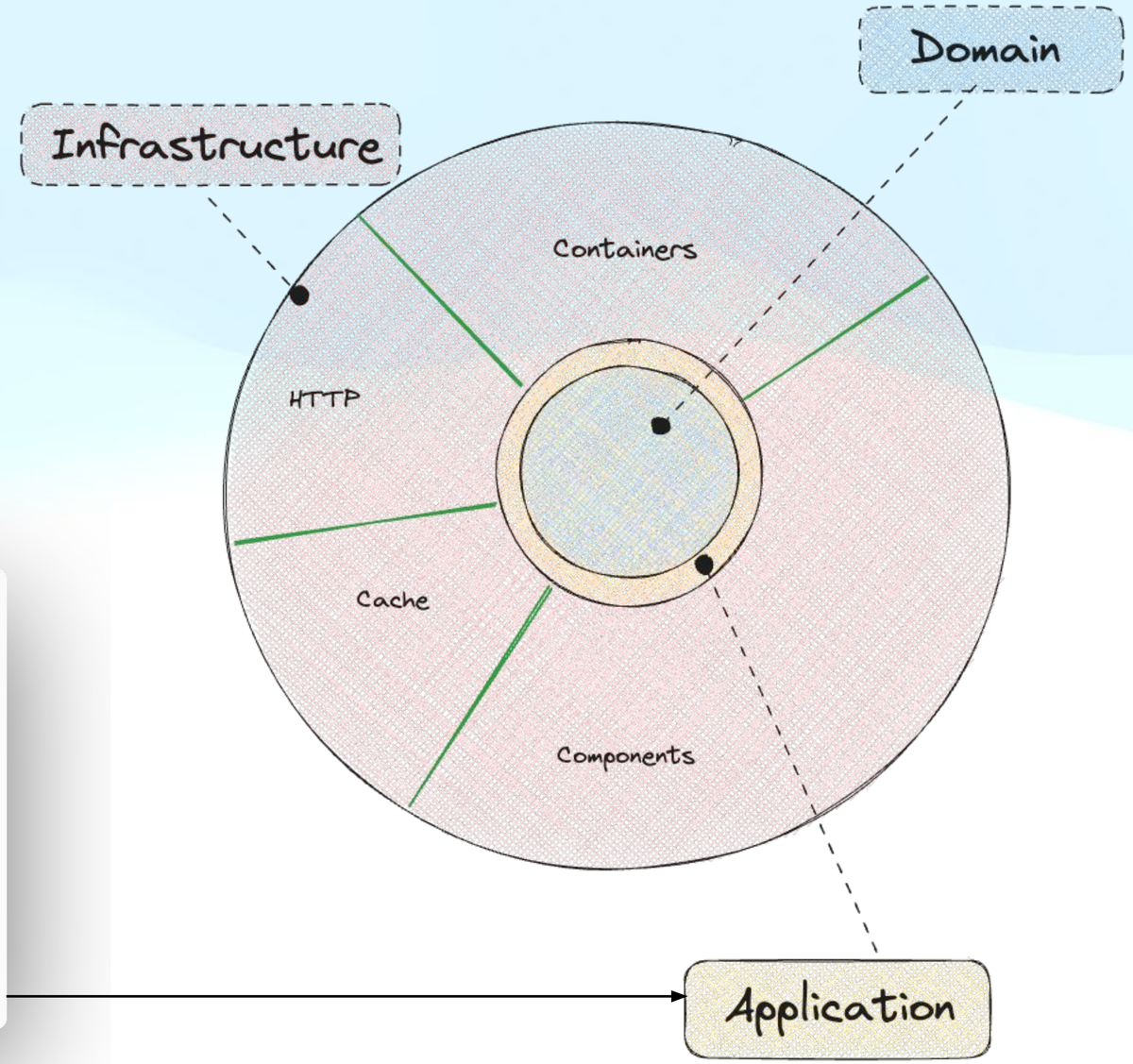
```
import useSWR, {useSWRConfig} from "swr";

export const useData = ({key, fetcher} :UseData<T>) => {
  const {data, error, isValidating} = useSWR(key, fetcher)
  return {data, error, loading: isValidating}
}

export const useDataMutation = ({key, mutation} ) => {
  const { mutate: swrMutate } = useSWRConfig()
  const mutate = async (data) => {
    return await swrMutate(key, mutation(data))
  }

  return {mutate}
}
```

```
export const MyPaymentContainer = ({
  purchase = purchaseUseCase()
}) => {
  const {mutate} = useDataMutation({
    key: 'api.purchase',
    mutation: purchase
  })
  const onPurchase = () => {
    mutate()
  }
  const onCancel = () => {
    // make some API call
  }
  return (
    <MyPayment
      onPurchase={onPurchase}
      onCancel={onCancel}
    />
  )
}
```

Ejecutamos nuestra lógica

```
const useCase = purchaseUseCase(purchaseClient, paymentClient)
export const MyPaymentContainer = ({
  purchase = useCase
}) => {
  const {mutate} = useDataMutation({
    key: 'api.purchase',
    mutation: purchase
  })
  const onPurchase = () => { mutate() }
  const onCancel = () => { }
  return (
    <MyPayment
      onPurchase={onPurchase}
      onCancel={onCancel}
    />
  )
}
```

```
export const purchaseUseCase = async ({
  purchaseClient,
  paymentClient
}: Dependencies) => async (items, billingInfo) => {
  const itemsMap = items.map(Items.from(items))
  const order = await paymentClient.createOrder({billingInfo,
items: itemsMap})
  await purchaseClient.notifyOrder({id: order.id})
}
```



Por supuesto…

```
describe('Purchase use case', () => {
  it('notifies an order when user order amount of items', async () => {
    const purchaseClient = { notifyOrder: vi.fn(async () => {}) }
    const paymentClient = { createOrder: vi.fn(async () => ({id: 'orderId'})) }

    const items = [{id: 'itemId', quantity: 1}];
    const billingInfo = {name: 'John Doe', creditCard: '1234 5678 9012 3456'};
    await purchaseUseCase({purchaseClient, paymentClient})(items, billingInfo)

    expect(purchaseClient.notifyOrder)
      .toHaveBeenCalledWith({id: 'orderId'})
  })
})
```

Test E2E

```
test('User make purchase at platform', async ({page}) => {
  const email = 'adrian.ferrera@leanmind.es'
  await signIn(page, email, 'irrelevant')
  await page.waitForNavigation({
    timeout: 20000
  })
  await addItemstoCart(page, items)
  await checkoutCart(page)
  await fillBillingInfo(page, billingInfo)
  await confirmPurchase(page)

  await page.waitForSelector(`text=${confirmation.title}`)
  await expect(page.getByText(confirmation.title)).toBeVisible()
})
```

Enlaces de interés

- https://softwarecrafters.io/
- https://leanmind.es/es/blog/
- https://danielirvine.com/
- https://adrianferrera.dev/