Hello.

About me:

Patrick Lehner

Software Developer at XITASO

 @lehnerpat

 @xitasogmbh

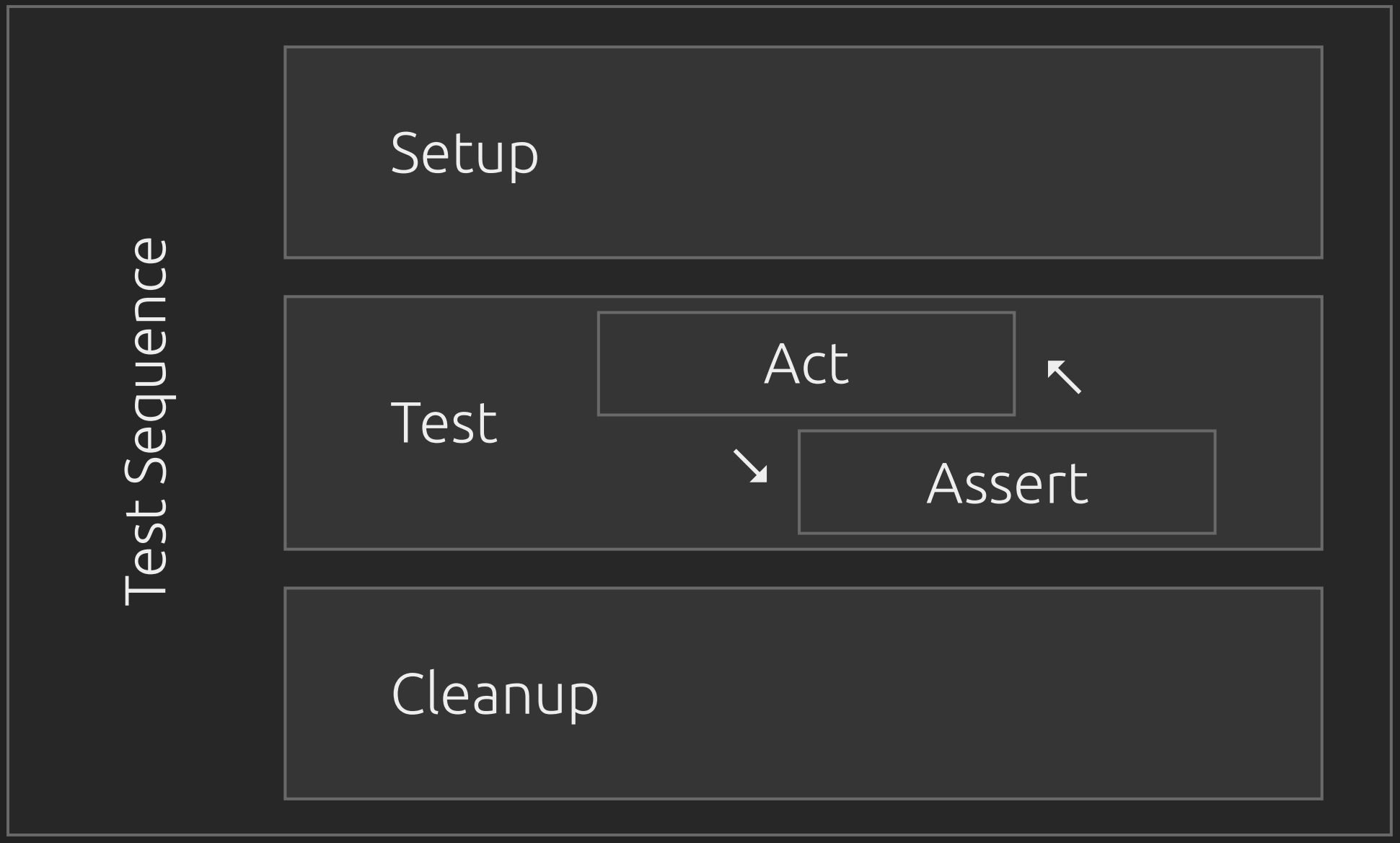
 blog.lehnerpat.com

 xitaso.com

In this talk, I present **a pattern**,
and **not a framework / library.**

Async Test Data Builders

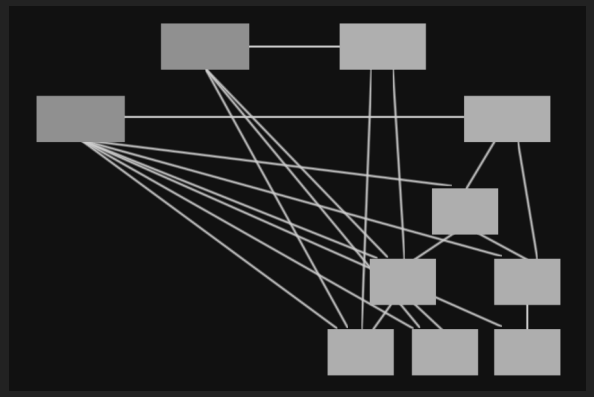
Motivation

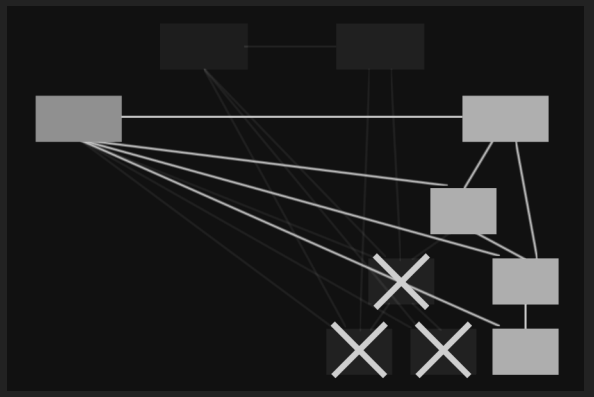

Test Sequence
Setup
Test
Act
Assert
Cleanup

```
const invoice = new Invoice(
    new Recipient("Sherlock Holmes",
        new Address("221b Baker Street",
                    "London",
                    new PostCode("NW1", "3RX"))),
    [
        new InvoiceLine("Deerstalker Hat",
            new PoundsShillingsPence(0, 3, 10)),
        new InvoiceLine("Tweed Cape",
            new PoundsShillingsPence(0, 4, 12)),
    ]);
```

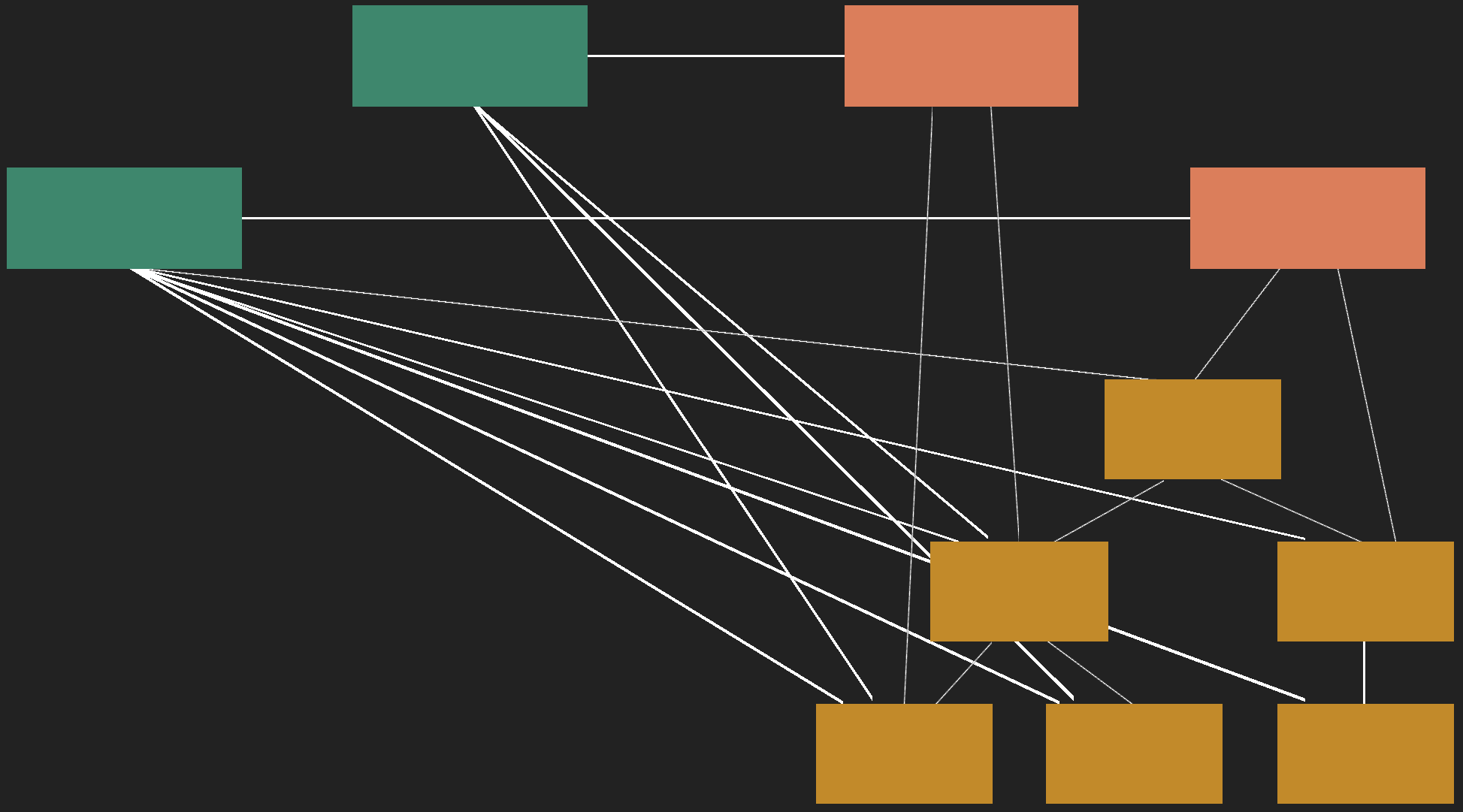
```
const invoice = new InvoiceBuilder()
    .withRecipient(new RecipientBuilder()
        .withPostCode("NW1", "3RX")
        .build())
    .withLines(InvoiceLineBuilder.buildRandomLines(2))
    .build();
```

Common Problems

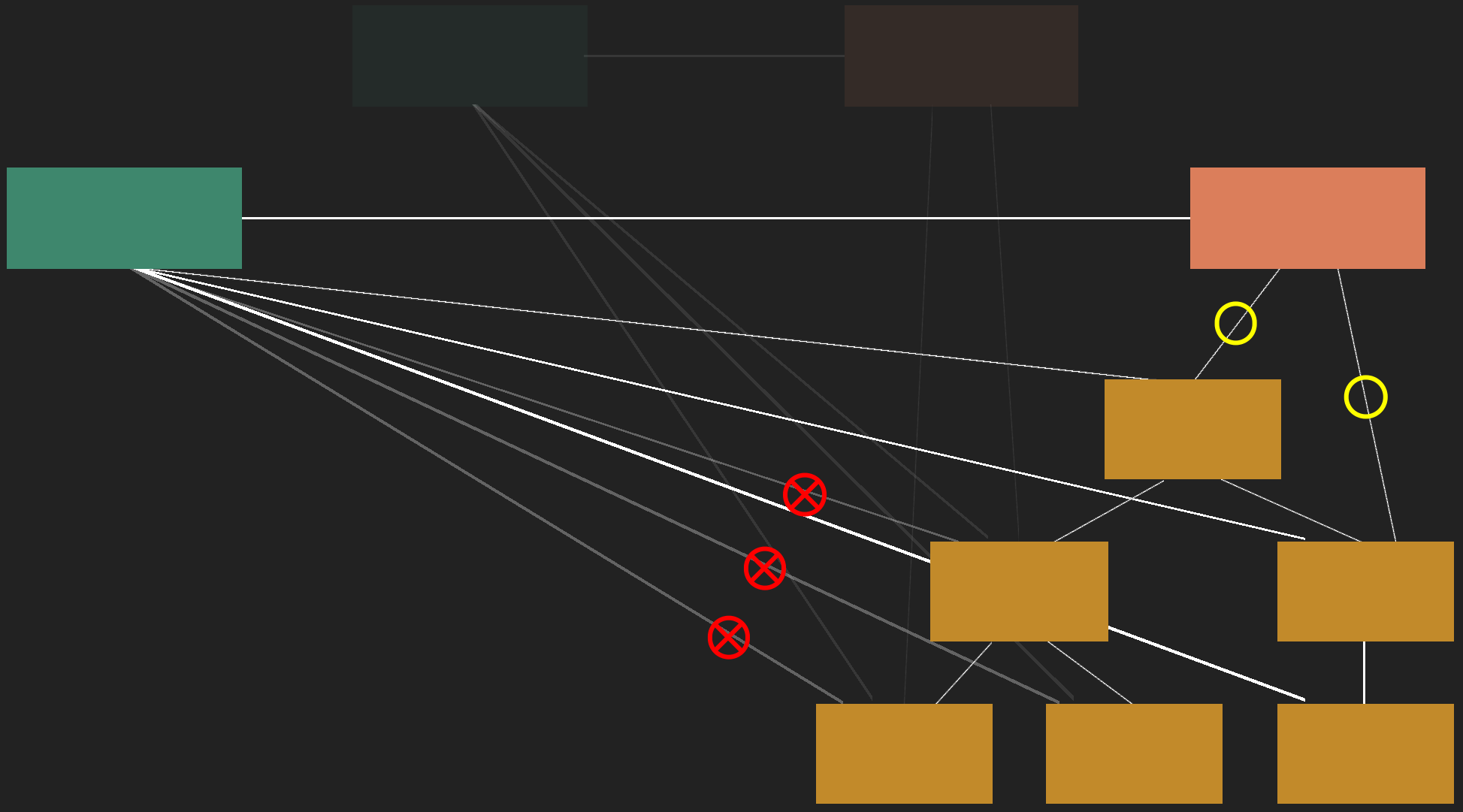




Tight Coupling



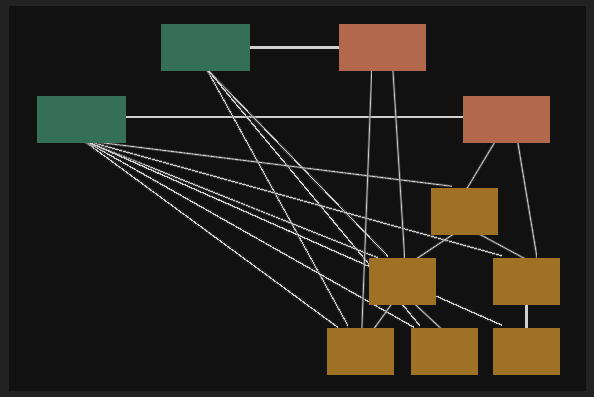
Invalid Models

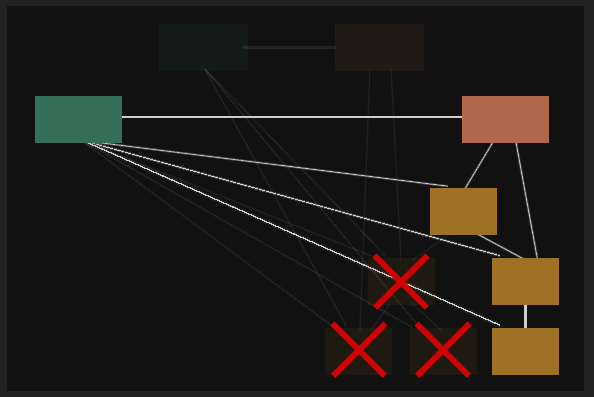


Test Setup Boilerplate

```
test("Can retrieve ticket", async () => {
    // setup data
    const response1 = await Axios.post(
        `${BaseUrl}/users`, {
            username: "saradoe98",
            createdAt: new Date()
        }
    );
    const userId = response1.data.id;

    const ticketTitle = "Do the thing";
    const response2 = await Axios.post(
        `${BaseUrl}/tickets`, {
            title: ticketTitle,
            creatorId: userId,
            createdAt: new Date()
        });
    const ticketId = response2.data.id;

    // act & assert
    const t = await new TicketService()
        .get(ticketId);
    expect(t.id).toBe(ticketId);
    expect(t.title).toBe(ticketTitle);
});
```

```
test("Can delete ticket", async () => {
    // setup data
    const response1 = await Axios.post(
        `${BaseUrl}/users`, {
            username: "saradoe98",
            createdAt: Date.now()
        }
    );
    const userId = response1.data.id;

    const ticketTitle = "Do the thing";
    const response2 = await Axios.post(
        `${BaseUrl}/tickets`, {
            title: ticketTitle,
            creatorId: userId,
            createdAt: Date.now()
        });
    const ticketId = response2.data.id;

    // act & assert
    await new TicketService()
        .delete(ticketId);
    await expect(Axios.get(BaseUrl +
            `/tickets/${ticketId}`))
        .rejects.toEqual({
            response: { stats: 404 }
        })
});
```

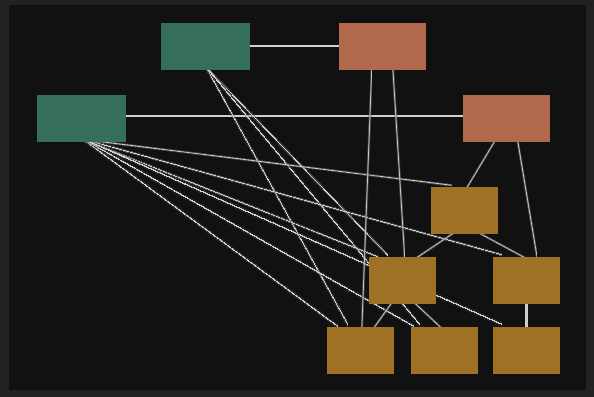


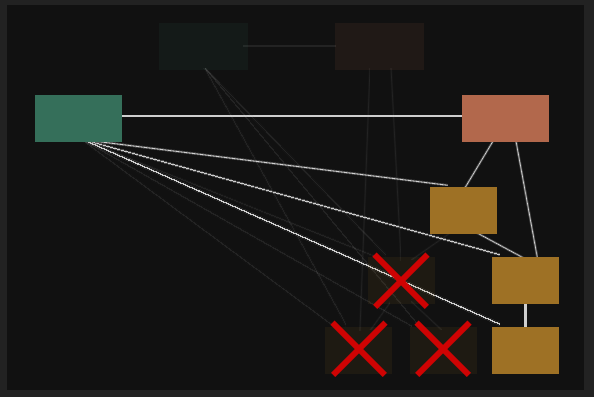


What are Test Data Builders?

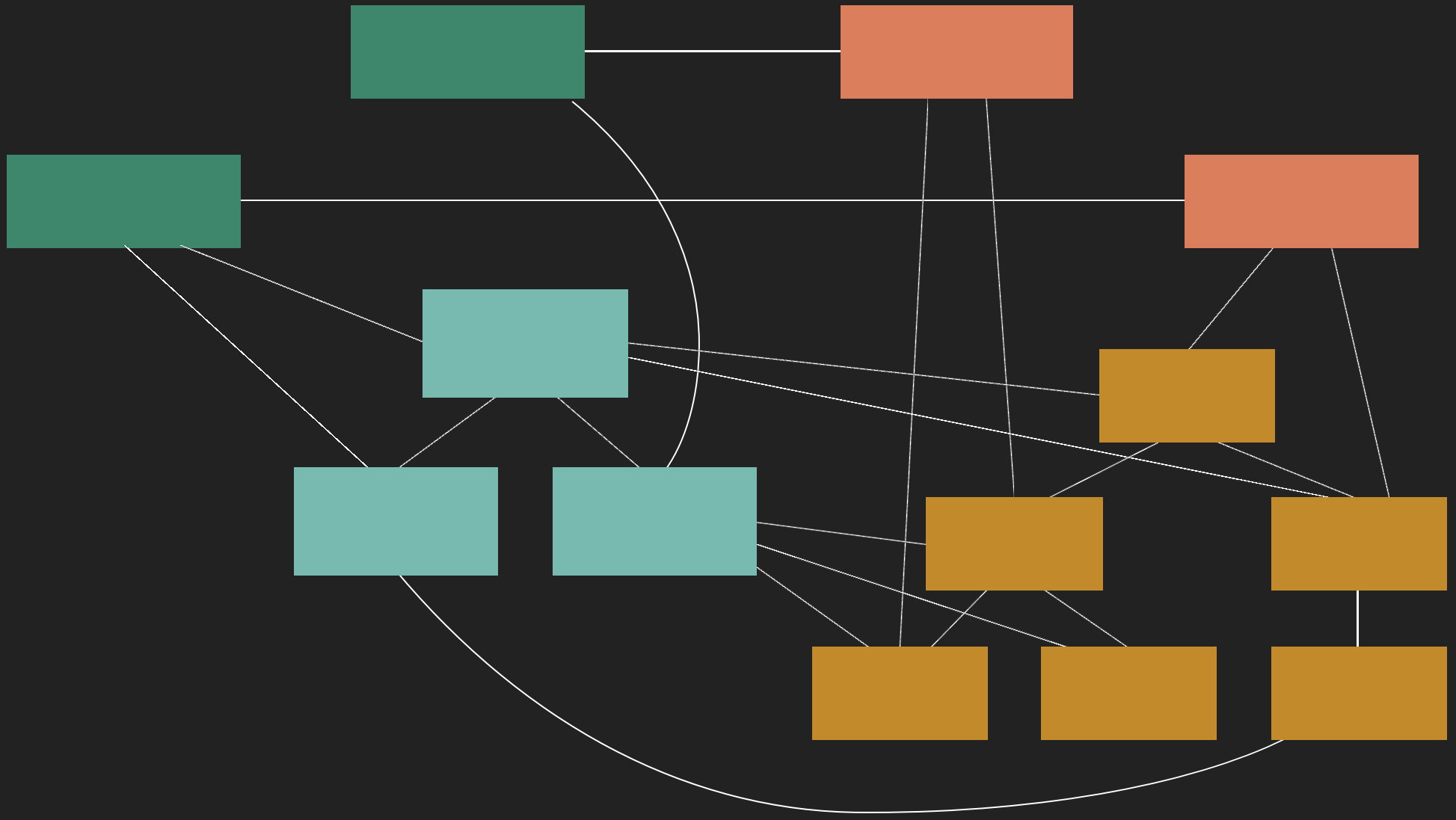
- tailored to fit your needs
- build (valid!) data model instances
- encode data model constraints
- manage persistence & communication
- manage cleanup where needed

How Does That Solve My Problems?

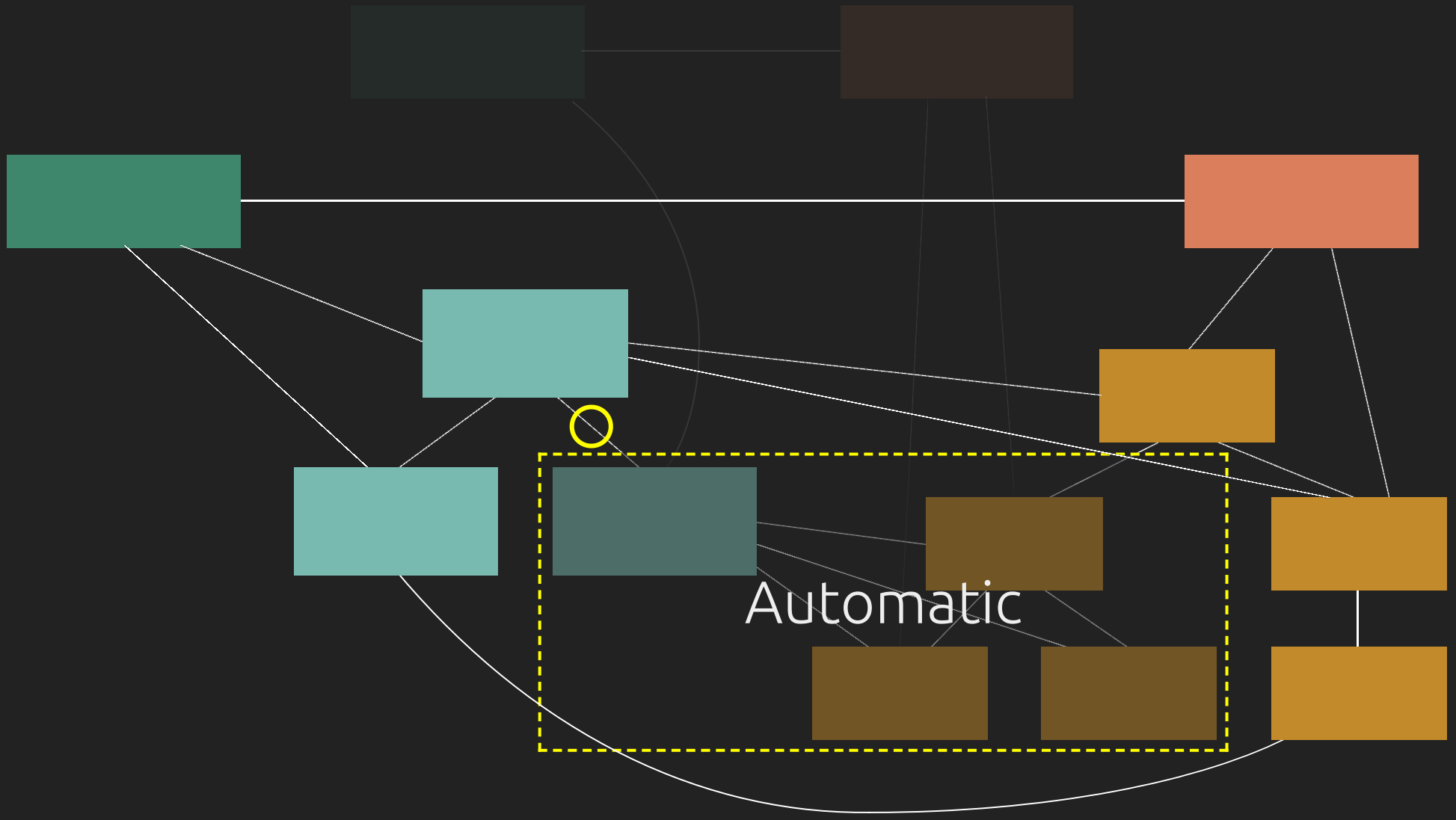




Tight Coupling → Indirection



Invalid Models → Valid Models

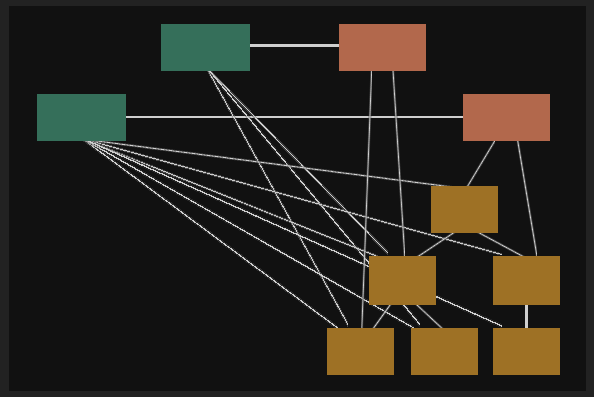
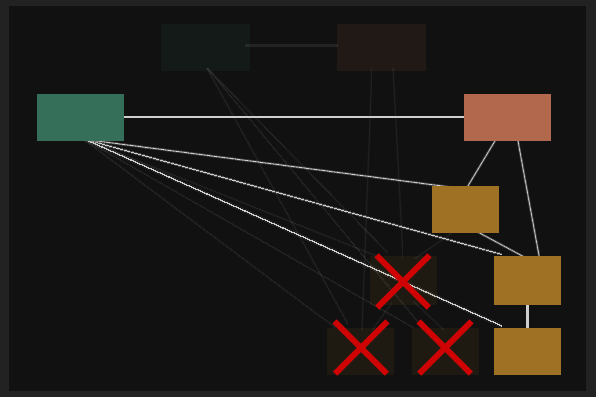


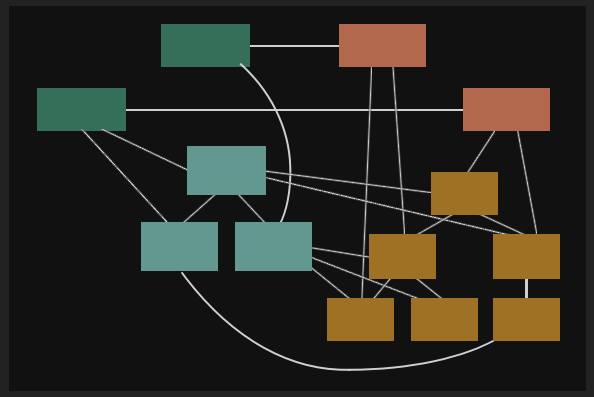
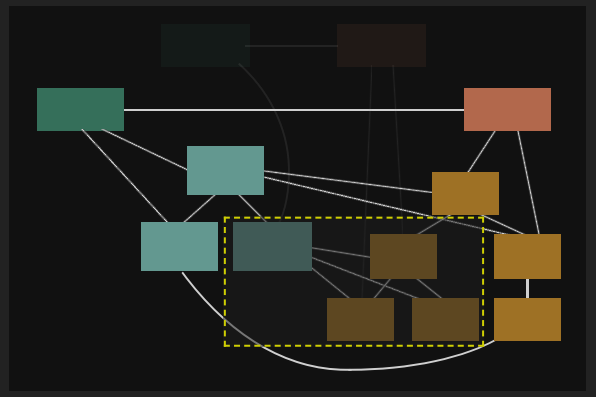
Test Setup Boilerplate → Greatly Reduced

```
test("Can retrieve ticket", async () => {
    // setup data
    const user = await new UserBuilder()
        .withDefaults()
        .resolve();

    const ticketTitle = "Do the thing";
    const tkt = await new TicketBuilder()
        .withTitle(ticketTitle)
        .createdBy(user)
        .resolve();

    // act & assert
    const t = await new TicketService()
        .get(tkt.id);
    expect(t.id).toBe(tkt.id);
    expect(t.title).toBe(tkt.title);
});
```

```
test("Can delete ticket", async () => {
    // setup data

    // or even shorter:
    const ticketTitle = "Do the thing";
    const tkt = await new TicketBuilder()
        .withTitle(ticketTitle)
        // creator filled by default
        .resolve();

    // act & assert
    await new TicketService()
        .delete(tkt.id);
    await expect(Axios.get(
            `${<base>}/tickets/${tkt.id}`))
        .rejects.toEqual({
            response: { stats: 404 }
        })
});
```

Recap


About me:

Patrick Lehner

Software Developer at XITASO

 @lehnerpat

 @xitasogmbh

 blog.lehnerpat.com

 xitaso.com

And now it's time

for your **questions**.

Where can I learn more?

- Test Data Builders: an alternative to the Object Mother pattern *by Nat Pryce* (→ on the Internet Archive)
- Generalised Test Data Builder *by Mark Seemann* (→ on the Internet Archive)
- Test Data Builders and Object Mother: another look *by Rafał Borowiec* (→ on the Internet Archive)
- kentan, an Angular library to "arrange reusable test data easily", and the accompanying blog post on kentan by *Gregor Woiwode*
- Example code from previous iterations of this talk:
  - Java (sync API)
  - TypeScript (async API)
- Search for Test Data Builders on Google

See you 👋