API & Domain Driven Design

PeiSong@Rest.Studio

About

10+ years server development, mainly Java/Spring

3+ years iOS development

1 year modern web development

Tech lead of a few small startups

Creator of Rest.Studio

Domain Driven Design

Maintain Model / Data Integrity

What is DDD (Domain-driven Design)

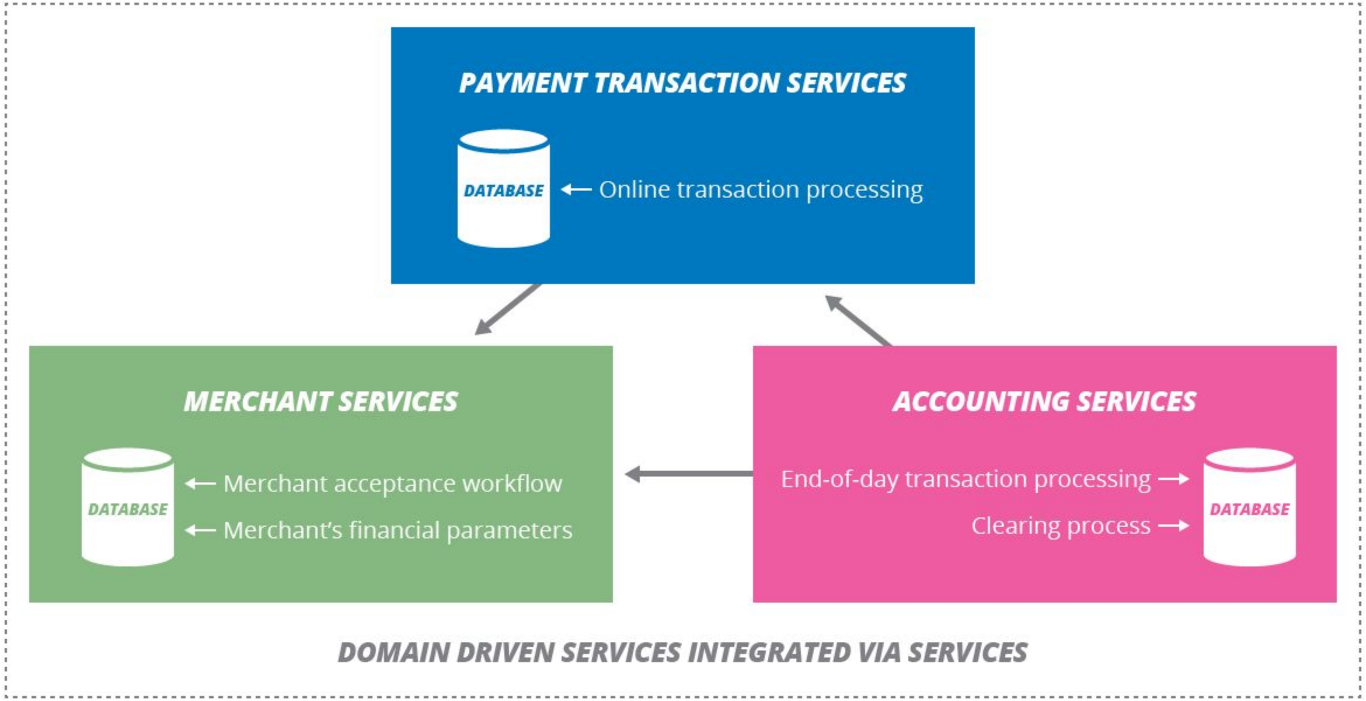
DDD is an approach to software development for **complex needs** by connecting the implementation to an **evolving model**.

- wikipedia

DDD Basic Concept

- **Context**
  The setting in which a word or statement appears that determines its meaning
- **Domain**
  A sphere of knowledge, influence, or activity
- **Model**
  A system of abstractions that describes selected aspects of a domain and can be used to solve problems related to that domain
- **Ubiquitous Language**
  A language structured around the domain model and used by all team members to connect all the activities of the team with the software

Example of Context



PAYMENT TRANSACTION SERVICES
DATABASE
Online transaction processing
MERCHANT SERVICES
DATABASE
Merchant acceptance workflow
Merchant's financial parameters
ACCOUNTING SERVICES
End-of-day transaction processing
Clearing process
DATABASE
DOMAIN DRIVEN SERVICES INTEGRATED VIA SERVICES

DDD Common Building Blocks

- Entity
- Value Object
- Repository
- Aggregate

- Database
- DAO / Repository
- Service
- Controller

Example of Building Blocks

```
OrderDelivery {
    user: User,
    address: Address,
    orderItems: OrderItem[]
}
```

OrderDelivery: Aggregate

OrderItem, Address: Value Object

User: Entity

Users/UserService: Repository

More About Model

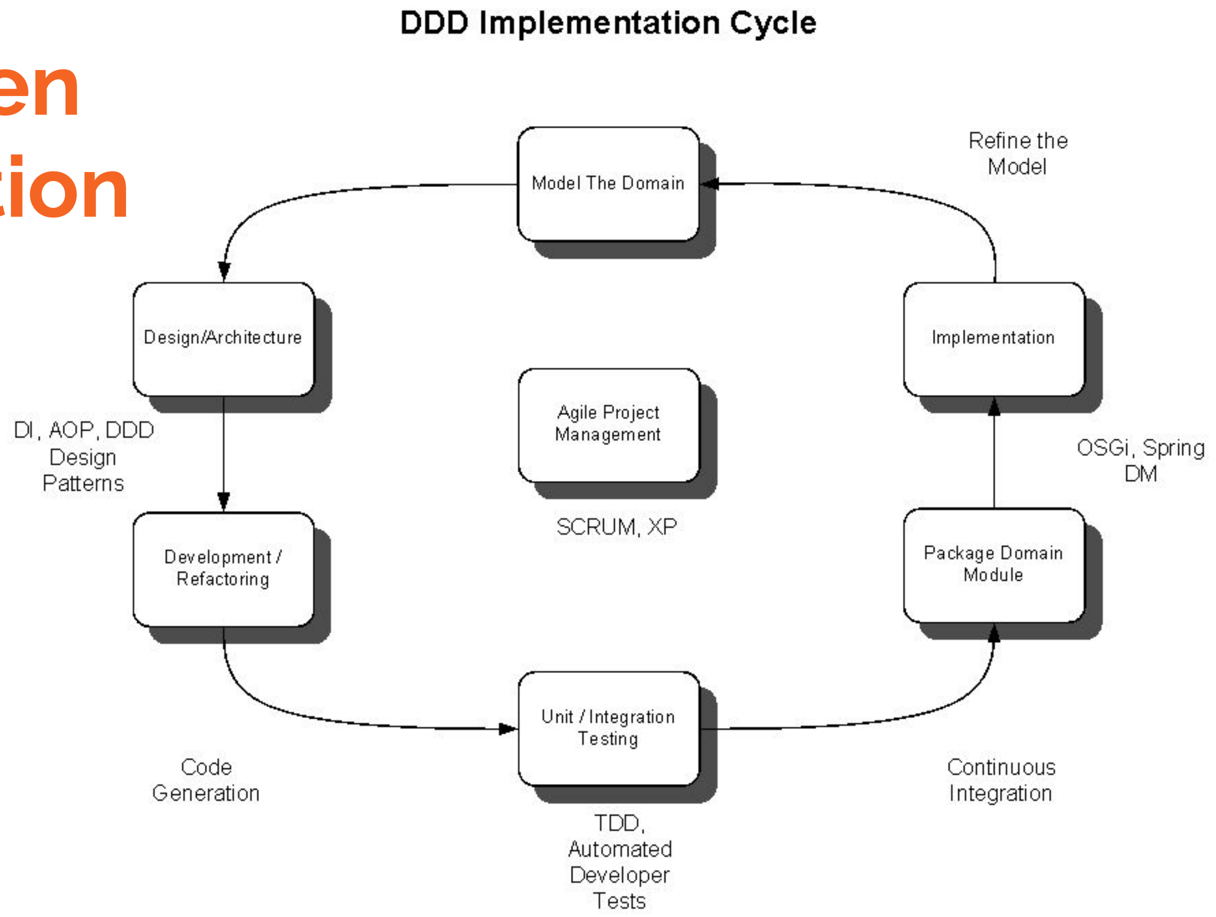
Between Domain Expert and System Designer

Mission: describe how model works in real world accurately

Among System Designer and Developers

Mission: how to implement a robust system that keeps data integrity

Domain-driven Implementation Cycle



However

DDD is Expensive and not for Everyone

Model with JSON Schema

The Ubiquitous Language for JSON REST API

Why JSON Schema

We need reliable JSON

We need universal description language for JSON validation, not language-dependent

JSON Schema is developed to validate JSON input

JSON Schema itself is an JSON object

JSON Schema Example

```
{
     title: ‘User’,
     type: ‘object’,
     required: [‘username’, ‘user_id’],
     properties: {
          username: {
               type: string,
               minLength: 6
          },
          user_id: {
               Type: integer
          },
          last_online: {
               type: integer,
               format: int64
          }
     }
}
```

JSON Schema

```
{
     title: ‘User’,
     type: ‘object’,
     required: [‘user_id’],
     properties: {
          username: {
               type: string,
               minLength: 6
          },
          user_id: {
               Type: integer
          },
          last_online: {
               type: integer,
               format: int64
          }
     }
}
```

Data To Validate

```
{
     username: ‘tom’
}

{
     user_id: 1234
     username: ‘wonderful’,
     last_online: ‘1 hour ago’
}

{
     user_id: 1234
     username: ‘wonderful’
}
```

No Schema/Model

```
GET /users
{
     success: true,
     users: [
          {
               username: string,
               user_id: string,
               last_online: long
          }
     ]
}

GET /users/{{user_id}}
{
     username: string,
     user_id: string,
     last_online: long
}
```

With Schema/Model

```
GET /users
{
     success: true,
     users: [User]
}

GET /users/{{user_id}}
User

Model:User
{
     username: string,
     user_id: string,
     last_online: long
}
```

Concept only, not a valid JSON schema

Validation

Client Side

Client side SHOULD always validate request and response.

However it is optional.

Server Side

Server side MUST validate request data. It is dangerous to assume the request data conforms to required schema.

Conditional Logic Validation

Schema validation does not solve conditional logic validation by default.

Conditional logic validation is still required even schema seems valid.

Consider

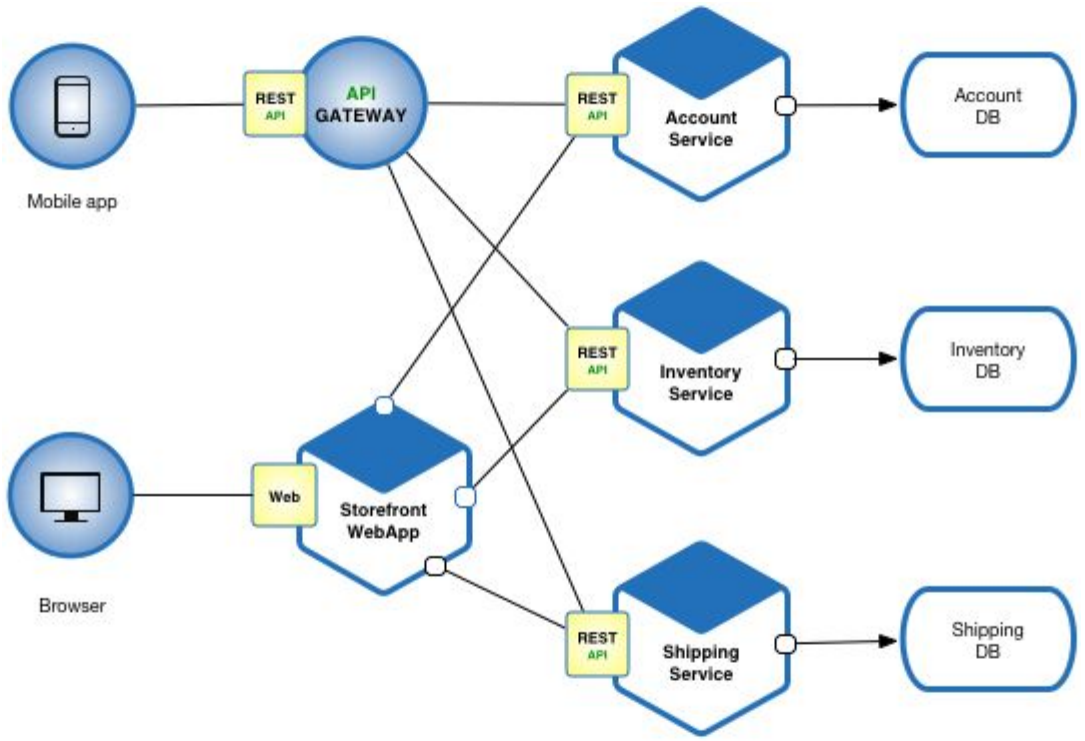
Design unconditional schema and split condition into multiple APIs.

Microservices with DDD

What is Microservices

A microservice is a piece of application functionality factored out into **its own code base/database**, speaking to other microservices over a **standard protocol**

Microservice Example

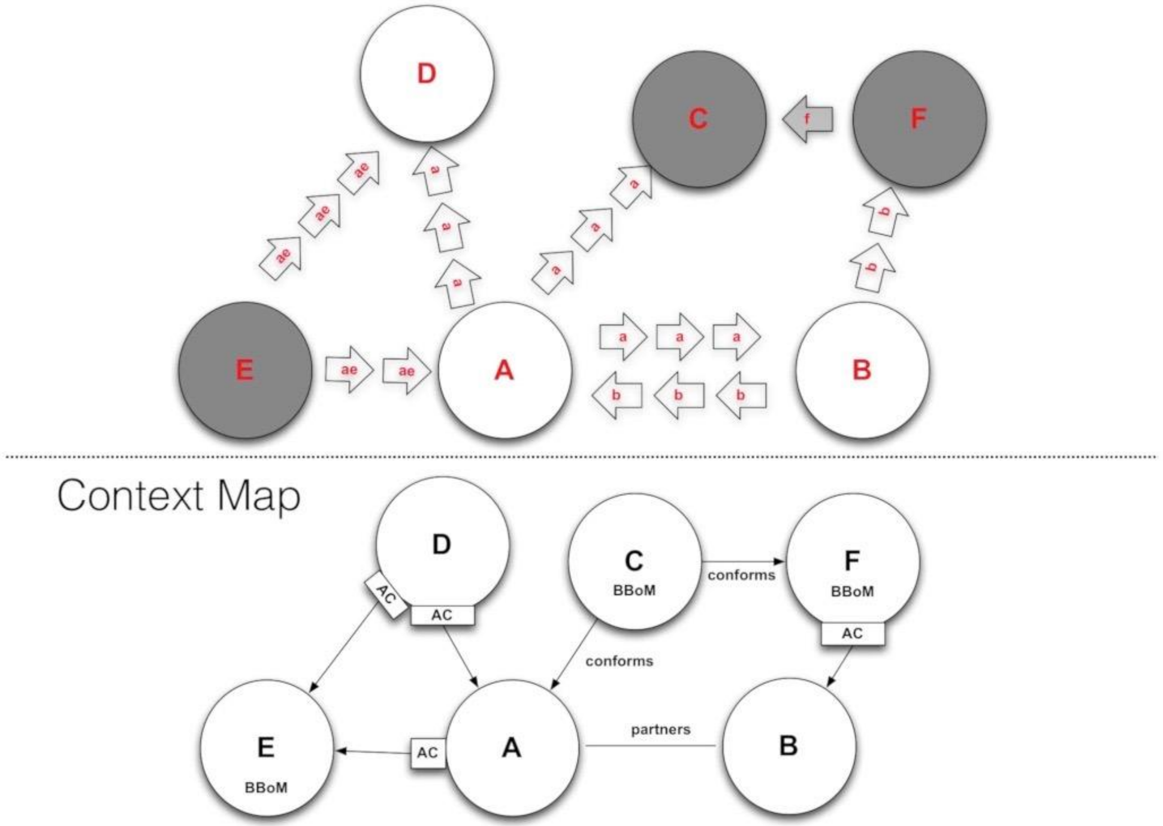


Microservices Problems

Model Visibility

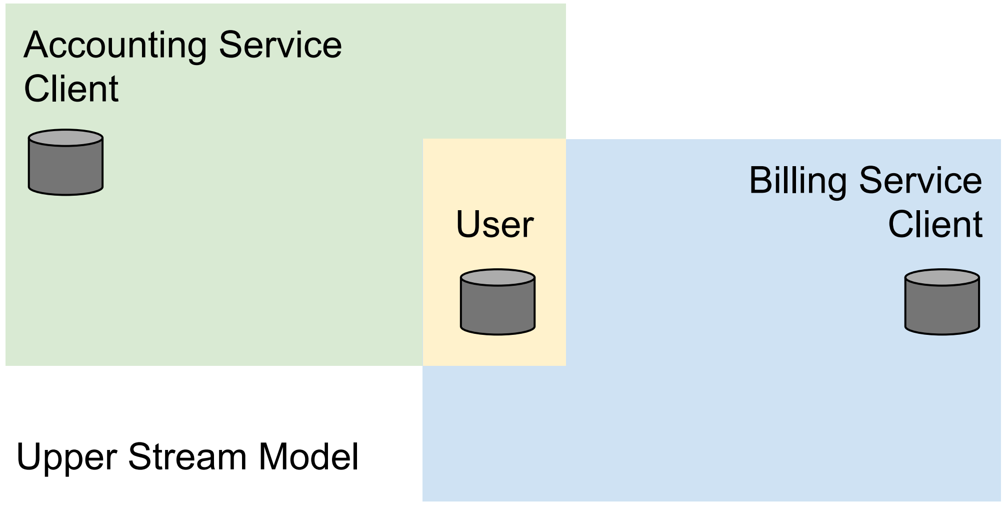
Model changes in on sub system are not visible to other systems.

The Problems



A Few Strategies

Shared Kernel



Anti-Corruption Layer

Add Adapter

Accept object that conforms to upstream schema but convert it into its own format

Test with Data Samples

Unit test and continuous integration is particularly important for adapter

Continuous Integration

Validate Model, Anti-Corruption Layer

When we think of data from REST API, we do not think about model

Test with Data Samples

- Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Spec First, Code Later

Fast Iteration

Develop API spec first. Gather feedback from API consumer as soon as possible.

Two Way Agreement

Only when both API producer and consumer agree with the spec, start to code.

Mock Data

Mock Request/Response

Mock request/response data to validate the data structure, avoid possible errors.

Create Schema and Document

Once producer/consumer are good with the mock data, create schema/document as spec and start to code following the spec.

Follow the Spec

Follow Spec

Spec/document becomes the standard to evaluate server/client developer’s responsibility since it is a two way agreement.

Open API: Beyond Model

JSON Schema For API

JSON Schema is used for REST API modelling.

JSON Schema can be used for API spec too.

Swagger Example

API Definition

```
"/pets":
  get:
    description: Returns all pets
    produces:
      - application/json
    responses:
      '200':
        description: A list of pets.
        schema:
          type: array
          items:
            "$ref": "#/definitions/Pet"
```

Model Definition

```
Pet:
  type: object
  required:
  - id
  - name
  properties:
    id:
      type: integer
      format: int64
    name:
      type: string
    tag:
      type: string
```

References

GOTO 2015 • DDD & Microservices: At Last, Some Boundaries! • Eric Evans: https://www.youtube.com/watch?v=yPvef9R3k-M&t=1564s

Domain Driven Design and Development In Practice
https://www.infoq.com/articles/ddd-in-practice

Wrap Up

1. Maintain data/model integrity
2. Validate input/output
3. Spec first
4. Mock first
5. Consider Open API Spec for REST API