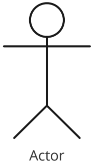
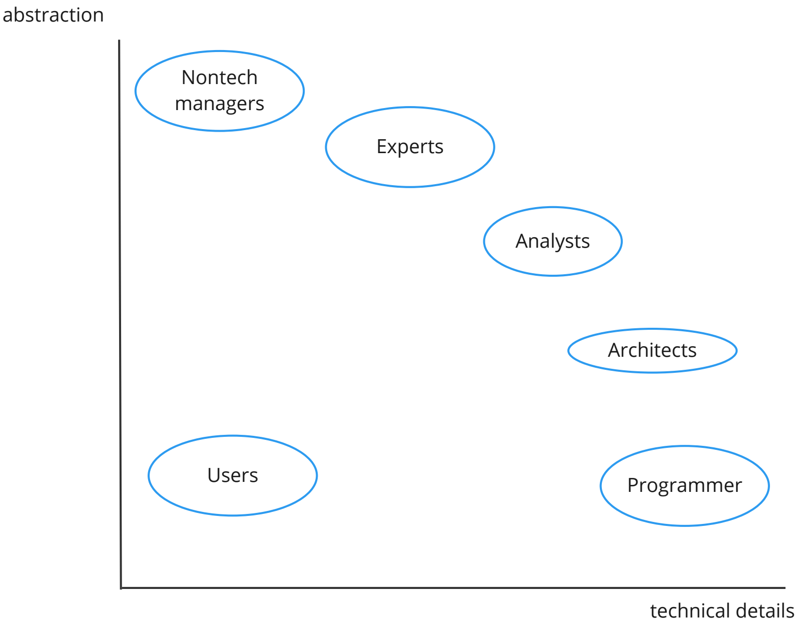
Who is the audience?



- Nontechnical managers
- Domain experts
- Users
- Analyst
- Designers
- Software architects
- Programmers

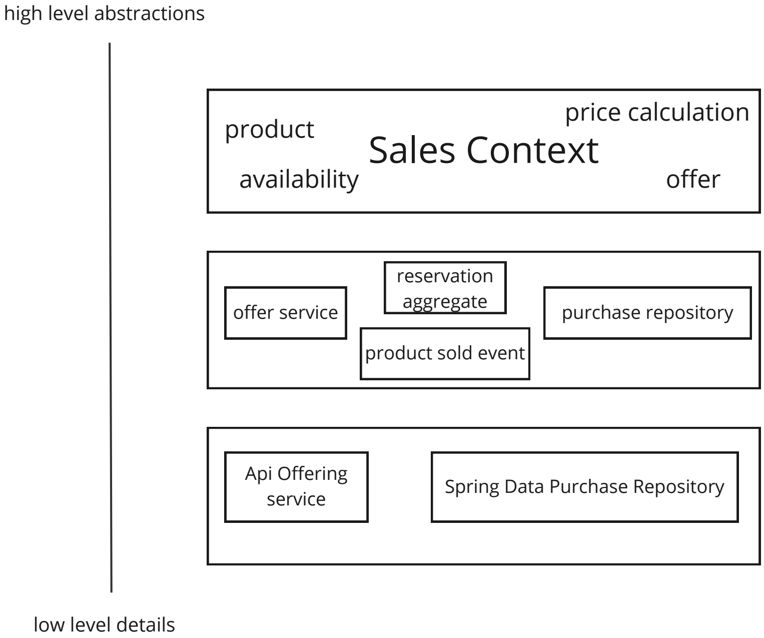


What is the required level of details?

- Strategic level design
  - Ubiquitous Language
  - Bounded Context

- Tactical level design
  - Responsibilities
  - Software building blocks

- Operational level design
  - Low level code
  - Implementation



high level abstractions

product

Sales Context

price calculation

availability

offer

offer service

reservation aggregate

product sold event

purchase repository

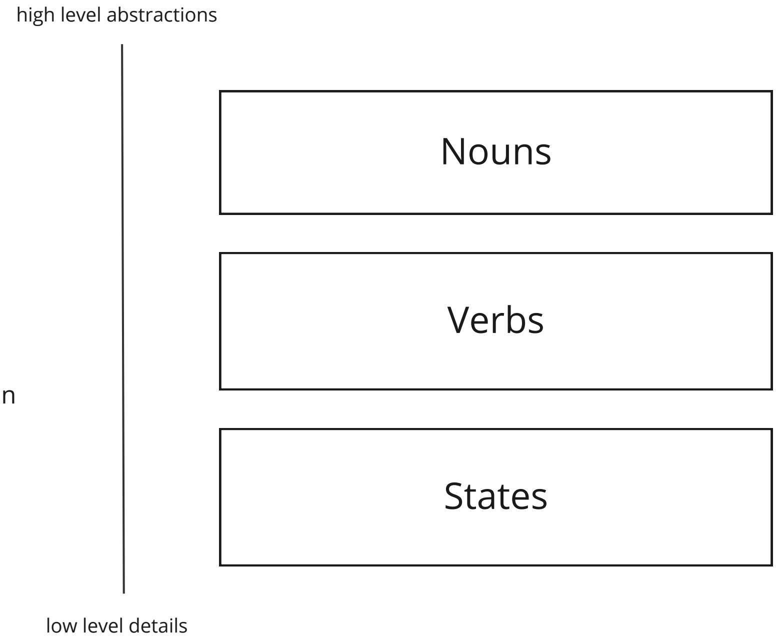
Api Offering service

Spring Data Purchase Repository

low level details

What needs to be expressed?

- Entities
    - What is it? (String / Int)
    - What is relation (1 - n)
- Behaviours
    - what are responsibilities?
    - what is it do?
    - how it affects environment?
    - who / what is source of action?
    - who / what required / involved in order to complete?
    - what are the rules of action?
- Transformation
    - How behaviours are changing under certain circumstances



high level abstractions

Nouns

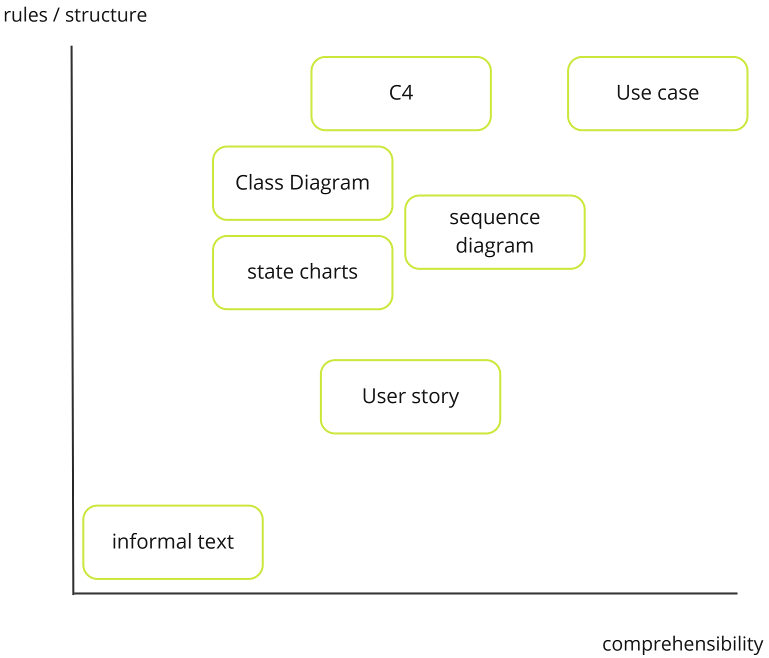
Verbs

States

low level details

Available tools

- Informal text
- User story
- Use case specification
- Use case diagram
- Sequence diagram
- Class diagram
- State charts
- C4 https://c4model.com/
- Java Code (is it readable?)
- BPMN
- others



User story

As a customer, I want to be able to browse a gallery of photos and select the ones that I like, so that I can create a personalised collection of images that could be bought in the future

Given:

- There is a gallery of 100 photos

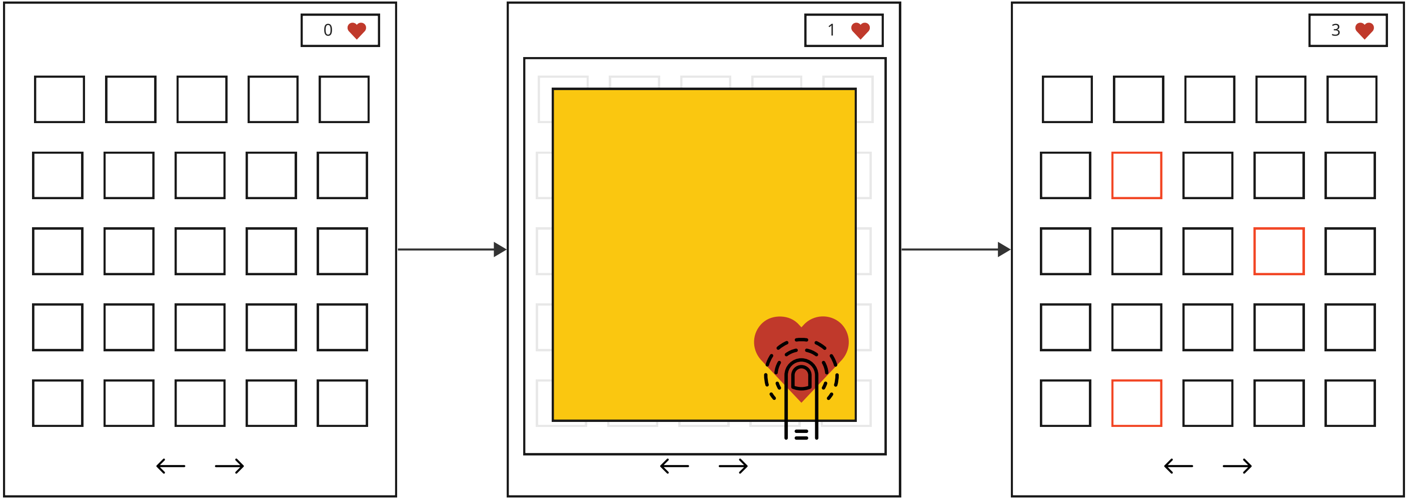
When:

- I select 3 images

Then:

- 3 images are available in my personal collection
- There is possibility to buy them

Screen flow



Use case scenario

Title: Handle photo purchase
Scope: Stock Platform
Primary Actor: Customer
Secondary Actors: Payment gateway
Stakeholder:

- Customer: would like to purchase stock photo
- Customer: would like to select and finally buy photos without paying attention that someone else would buy it first
- Company: would like to restrict licence for phot just to single customer
- Company: would like to have history of sold photos
- Payment gateway: would like to have payment registered

Precondition:

- Customer has an account on the stock platform and is logged in.
- There are photos available

Trigger: Customer selects photos to purchase on the stock platform.
Main Flow:

1. Customer selects one or more photos on the stock platform.
2. The system puts the selected photos on hold for 30 minutes, making them unavailable for purchase by other customers.
3. Customer proceeds to checkout and enters their payment information.
4. The system register payment request at the Payment Gateway
5. The system redirects the customer to an external payment gateway for payment processing.
6. The external payment gateway processes the payment and sends a response to the system.
7. If the payment is successful, the system sends an email to the customer with the purchased photos attached.

Alternate Flow:

- If the payment is unsuccessful, the system displays an error message and the customer is given the option to try again or cancel the purchase.

Postcondition:

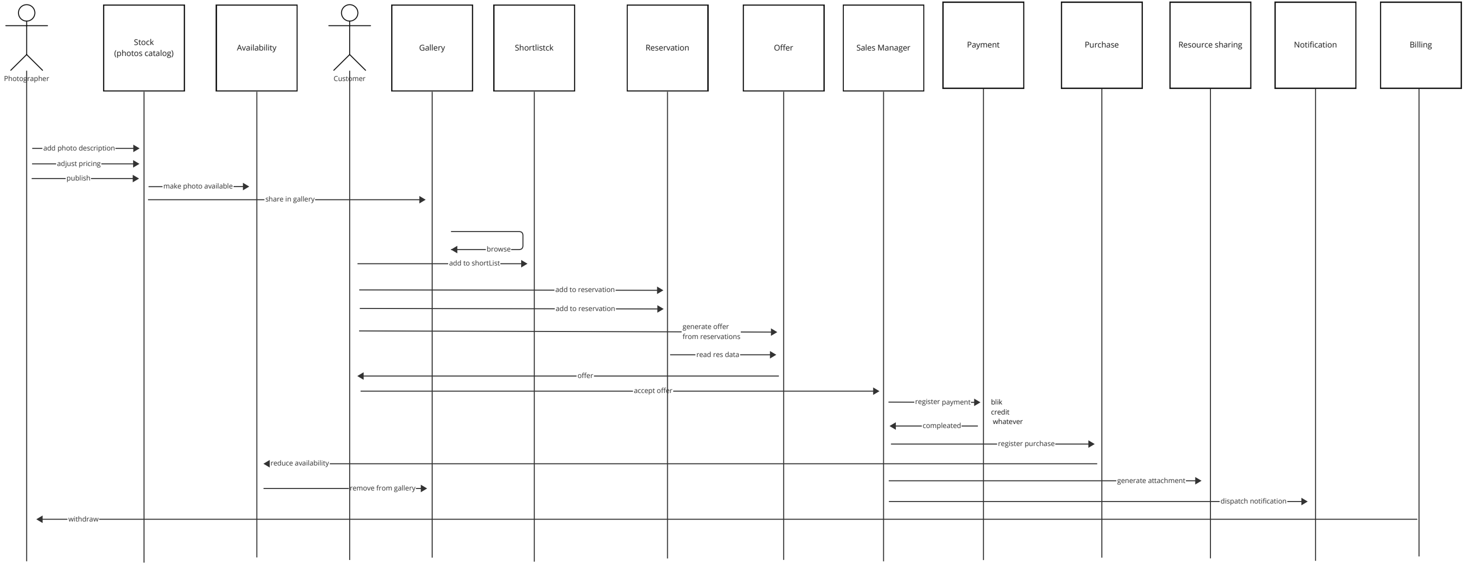
- The customer has successfully purchased the selected photos and received them via email.
- The photos are no longer available for purchase on the stock platform.

Additional requirements:

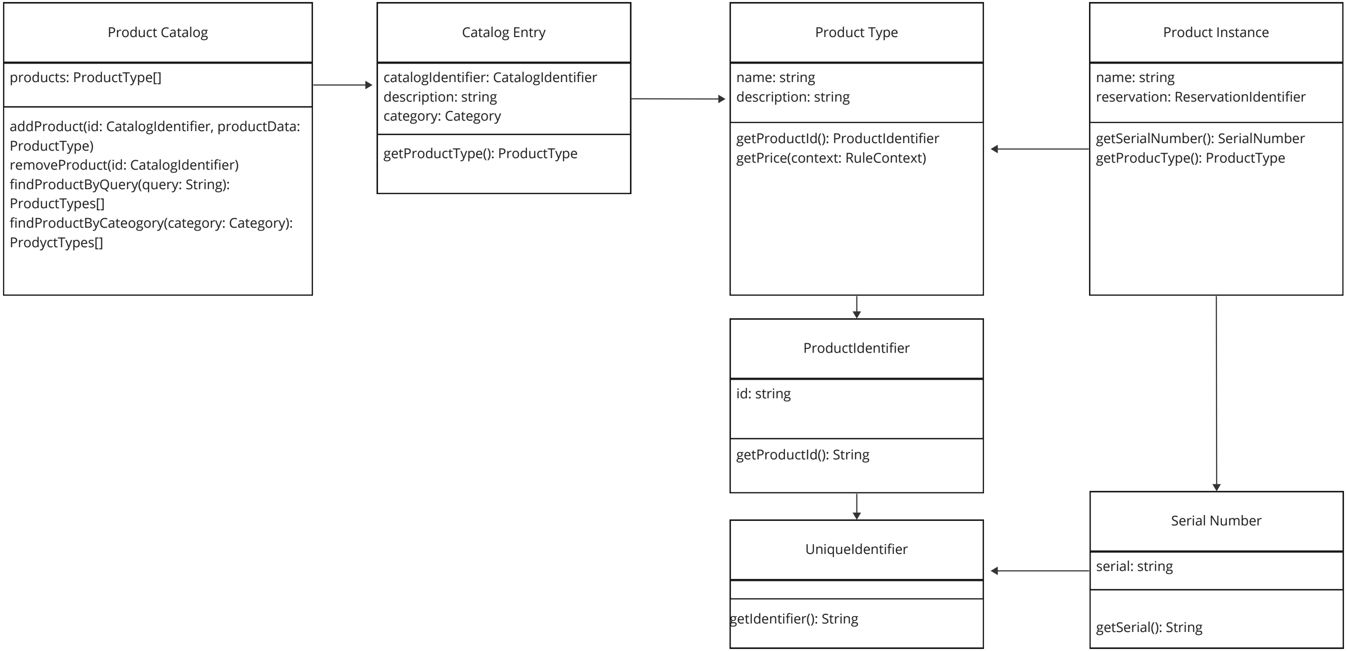
- Credit cards authorisation must be accomplished below 30 s in 90% of cases

Frequency of actions: continuous

Sequence diagram - happy path



Class diagram UML



Product Catalog

products: ProductType[]

addProduct(id: CatalogIdentifier, productData: ProductType)
removeProduct(id: CatalogIdentifier)
findProductByQuery(query: String): ProductTypes[]
findProductByCateogory(category: Category): ProdyctTypes[]

Catalog Entry

catalogIdentifier: CatalogIdentifier
description: string
category: Category

getProductType(): ProductType

Product Type

name: string
description: string

getProductId(): ProductIdentifier
getPrice(context: RuleContext)

Product Instance

name: string
reservation: ReservationIdentifier

getSerialNumber(): SerialNumber
getProducType(): ProductType

ProductIdentifier

id: string

getProductId(): String

UniqueIdentifier

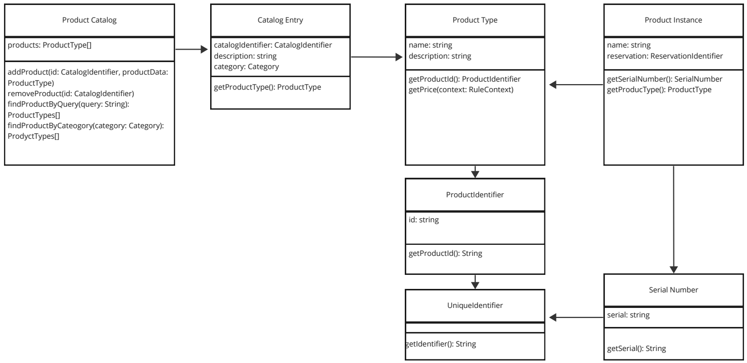
getIdentifier(): String

Serial Number

serial: string

getSerial(): String

Class diagram UML + Business context documents



Product Catalog

products: ProductType[]

addProduct(id: CatalogIdentifier, productData: ProductType)
removeProduct(id: CatalogIdentifier)
findProductByQuery(query: String): ProductTypes[]
findProductByCateogory(category: Category): ProdyctTypes[]

Catalog Entry

catalogIdentifier: CatalogIdentifier
description: string
category: Category

getProductType(): ProductType

Product Type

name: string
description: string

getProductId(): ProductIdentifier
getPrice(context: RuleContext)

Product Instance

name: string
reservation: ReservationIdentifier

getSerialNumber(): SerialNumber
getProducType(): ProductType

ProductIdentifier

id: string

getProductId(): String

UniqueIdentifier

getIdentifier(): String

Serial Number

serial: string

getSerial(): String

+

Narrative
+
embedded model
fragments

ProductCatalog manager operations

Operation:

addProductType (productType : ProductType) *Adds a ProductType to the ProductCatalog by creating a unique CatalogEntry for it*

addProductType (productType : ProductType, catalogIdentifier : String) *Adds a ProductType to the ProductCatalog by assigning it to an Existing catalog entry*

This operation is required only where a business system must allow for one-to-many mappings of CatalogEntries to ProductTypes