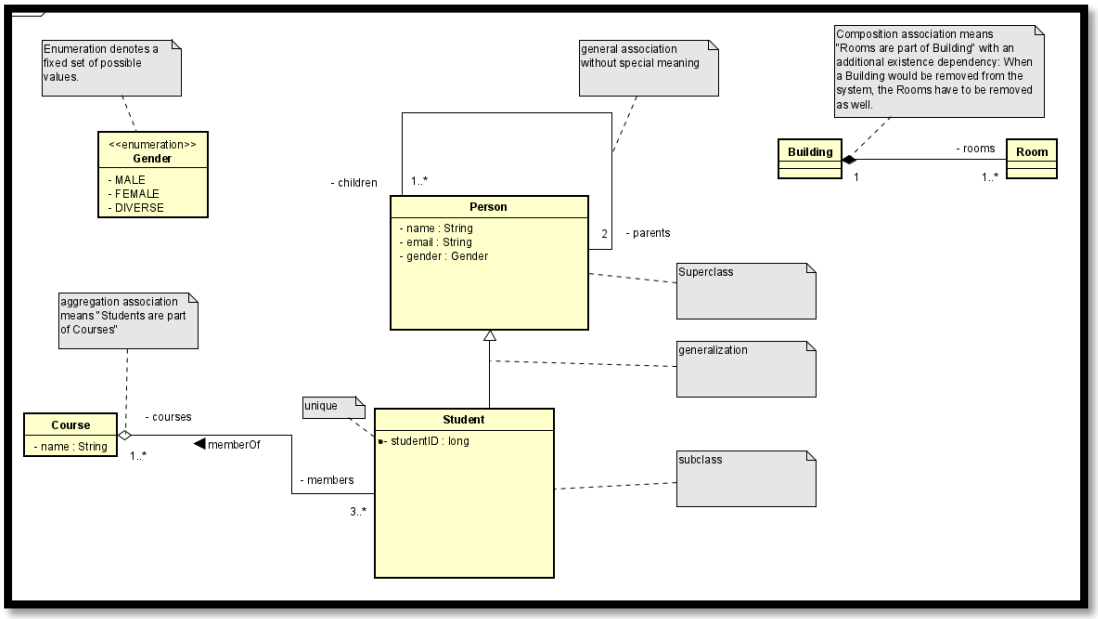
Basic Class Diagram

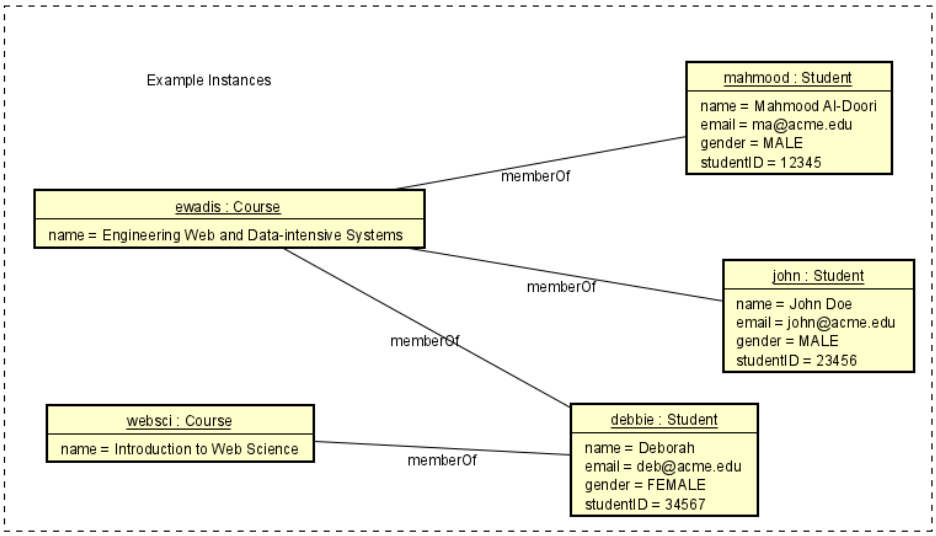
UML class diagrams consist of:

- classes with
  - name
  - attributes with a data type
  - operations

- associations between classes
  - role names
  - multiplicities
  - name
  - type of association (e.g. aggrgate, composite)

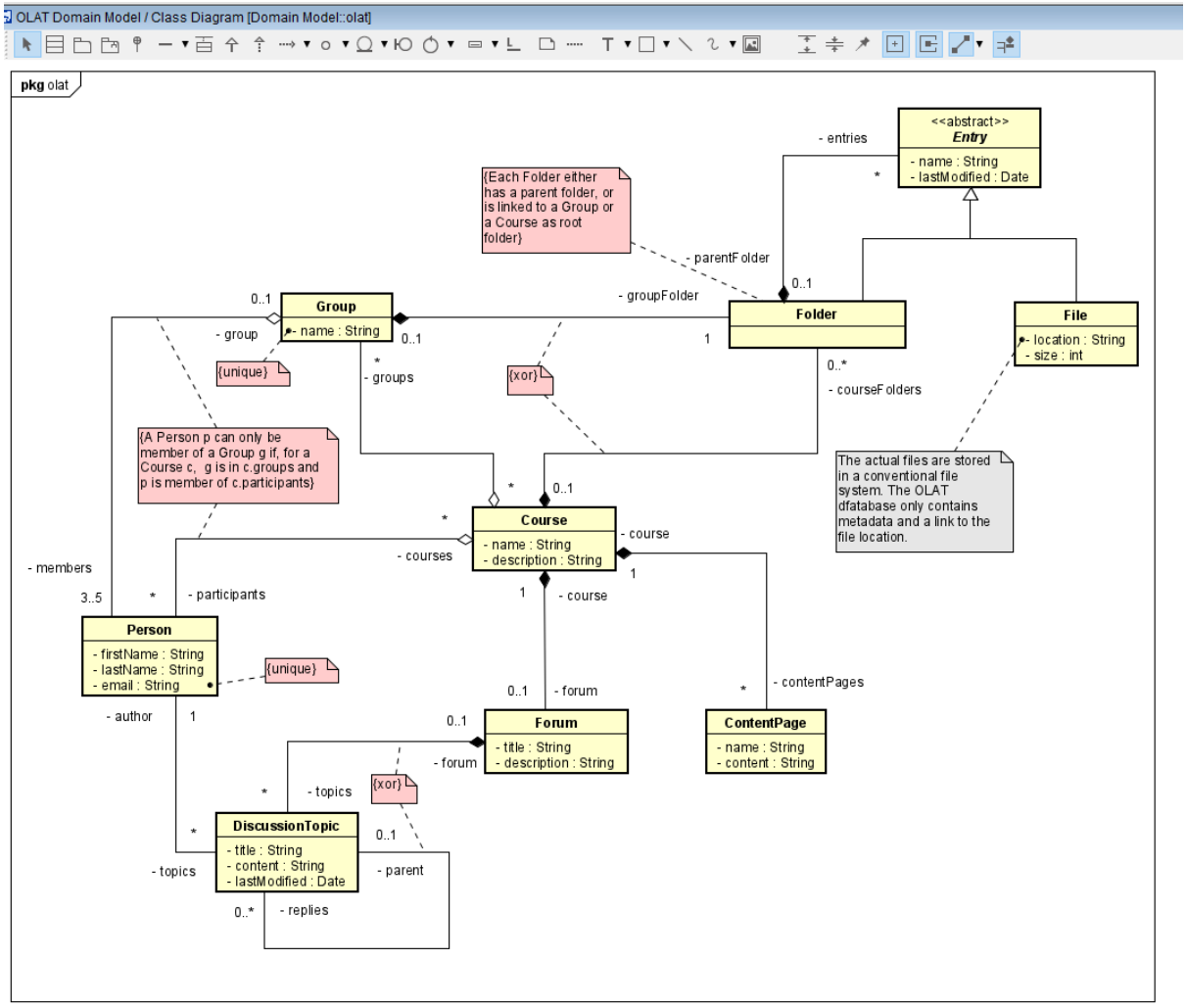


Instance of Class Diagram

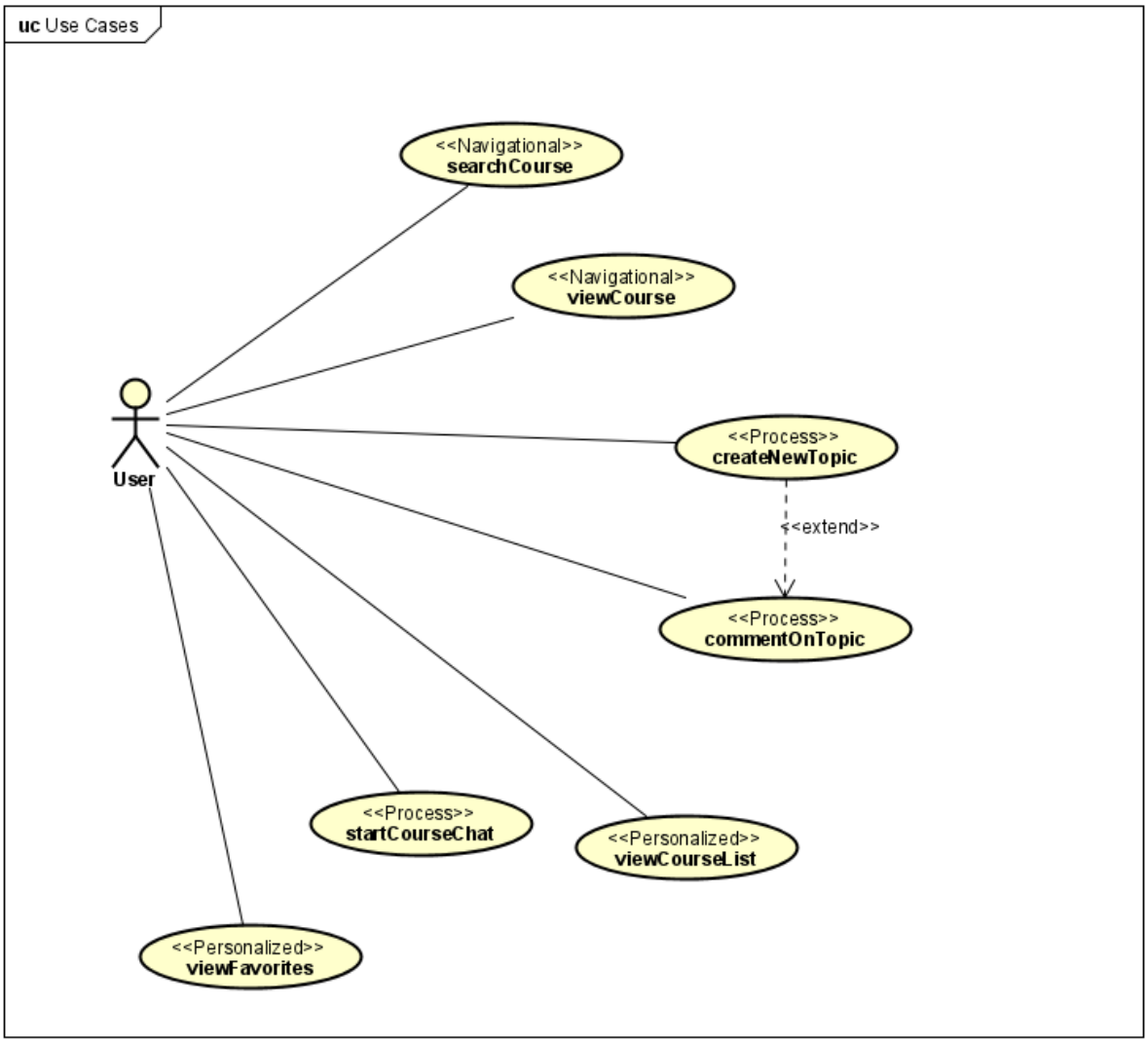


OLAT System

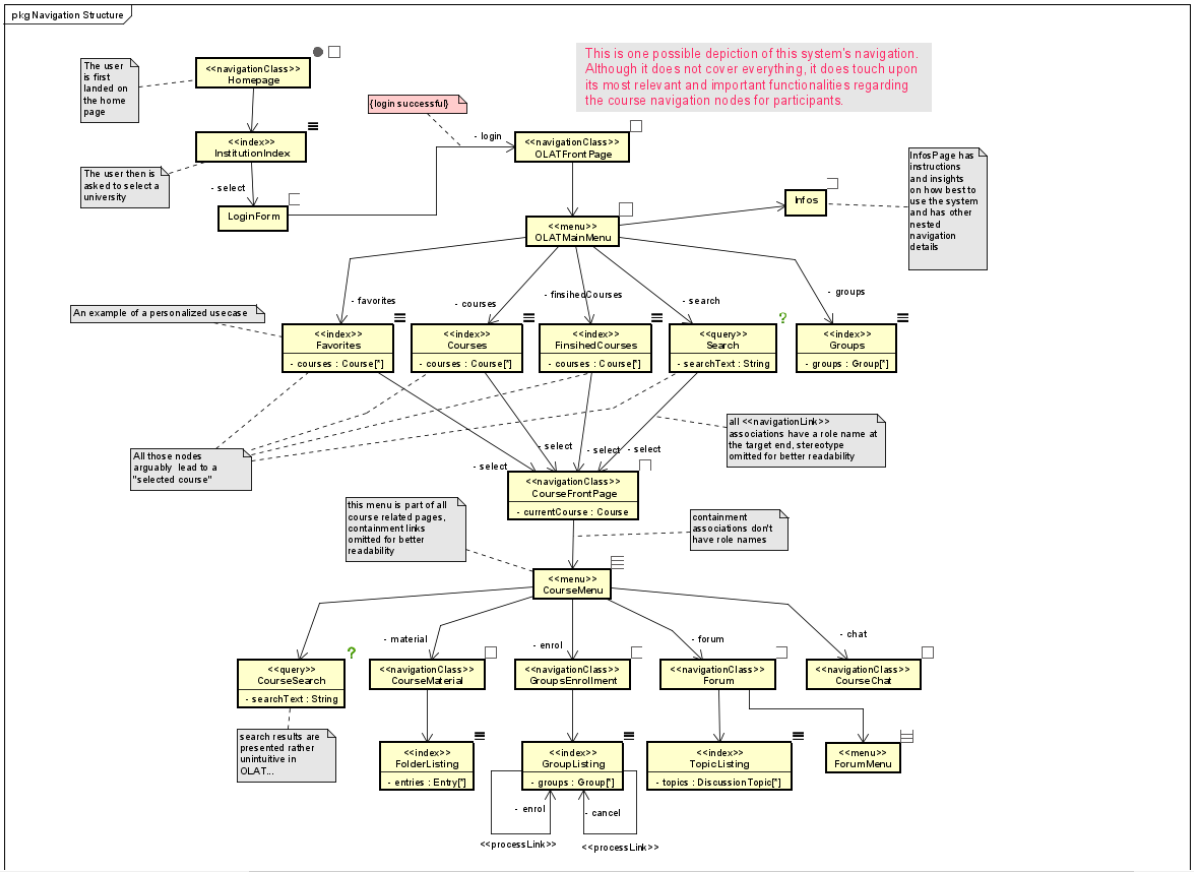
Class/Domain Diagram



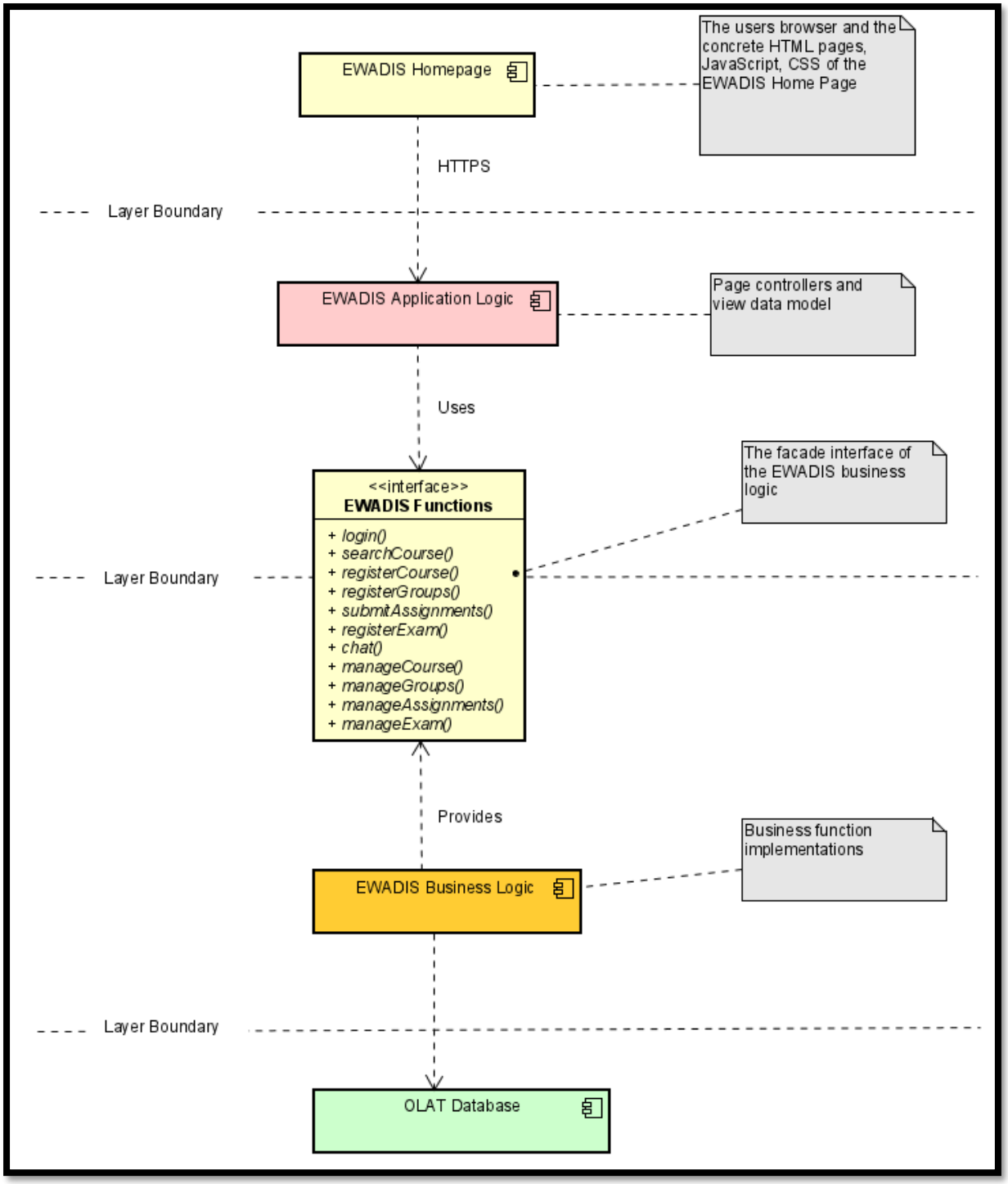
Use Case


uc Use Cases
<<Navigational>>
searchCourse
<<Navigational>>
viewCourse
User
<<Process>>
createNewTopic
<<extend>>
<<Process>>
commentOnTopic
<<Process>>
startCourseChat
<<Personalized>>
viewCourseList
<<Personalized>>
viewFavorites

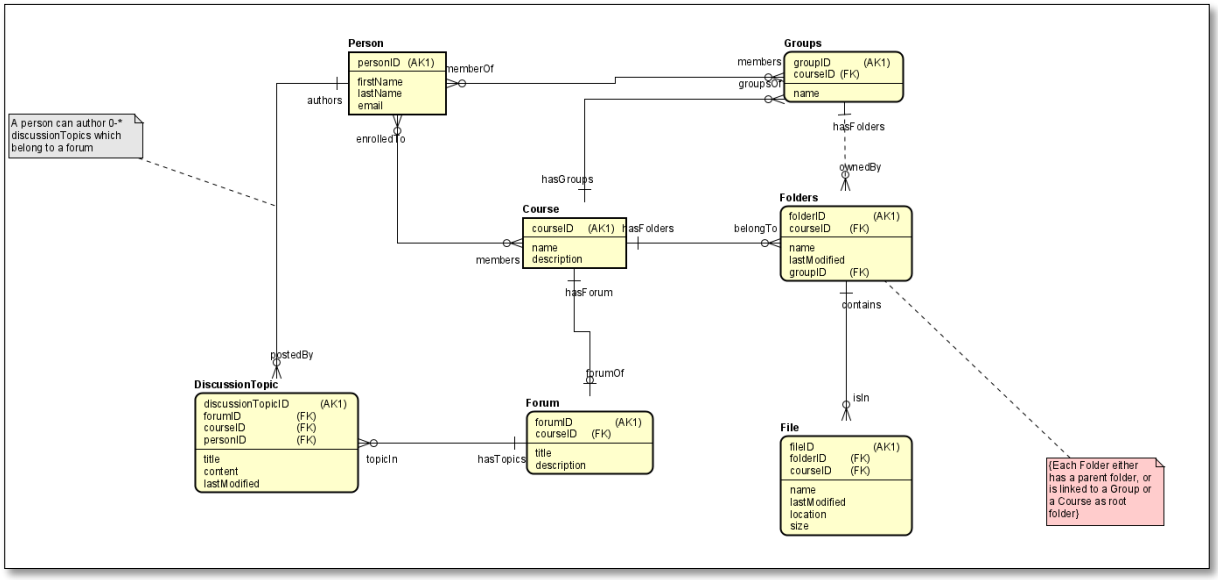
Navigation Model



Component Diagram



Relational DB Model – (E/R Diagram)



SQL Script with CREATE TABLE, ALTER TABLE, and CREATE INDEX commands to construct the relational schema.

CREATE TABLE Course (
courseId INT NOT NULL,
courseName CHAR(50),
description CHAR(100)
);

ALTER TABLE Course ADD CONSTRAINT PK_Course PRIMARY KEY (courseId);
ALTER TABLE Groups ADD CONSTRAINT FK_Groups_0 FOREIGN KEY (courseID) REFERENCES Course (courseID);
CREATE UNIQUE INDEX ID ON File (fileID);

SQL Script

Try to provide two SQL SELECT... queries to answer the following questions. If you think that there can be no solution (with a single query), state why!

• Given a Course (referenced by it's ID), please provide a sorted list of it's Groups with the number of members in each group.

SELECT g.groupID, COUNT(p.personID)
FROM Course as c, Groups as g, Person as p
WHERE courseID = 1223, c.courseID = g.courseID, g.groupID = p.groupID
GROUPBY g.groupID
ORDERBY g.groupID

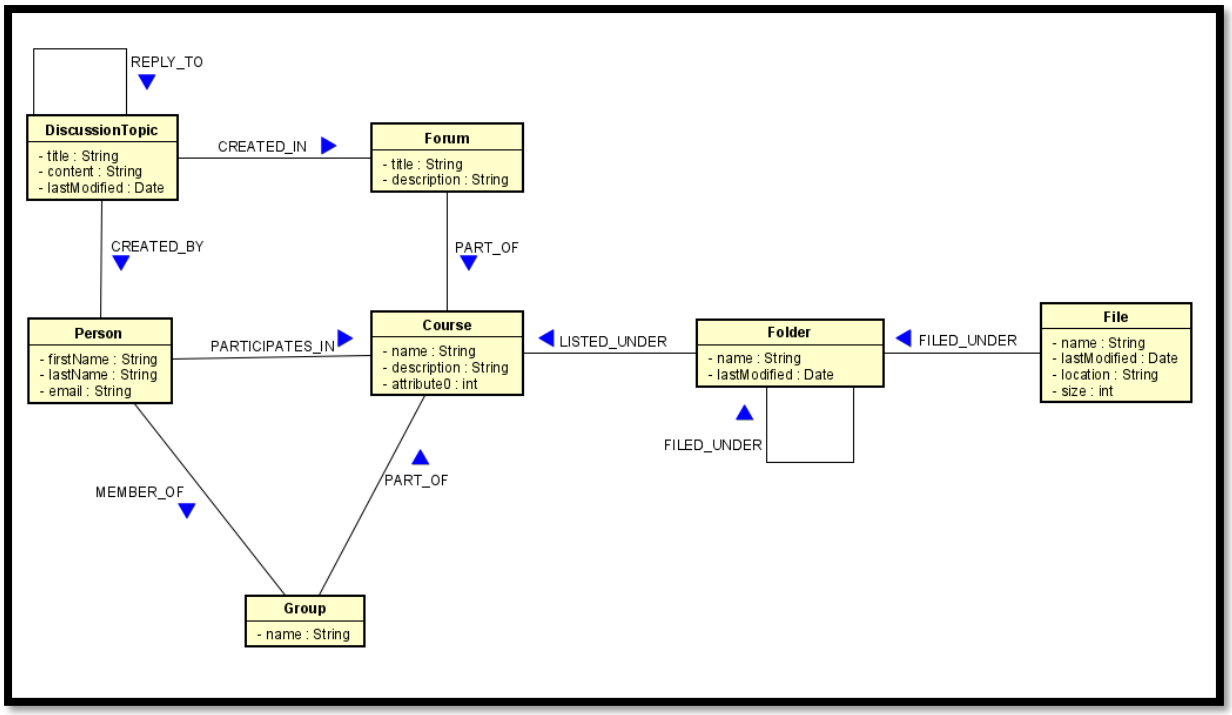
• For each top-level DiscussionTopic, compute the set of Persons who contributed to the topic and it's replies.

SELECT P.personID, p.firstname, COUNT(d.discussionTopicID)

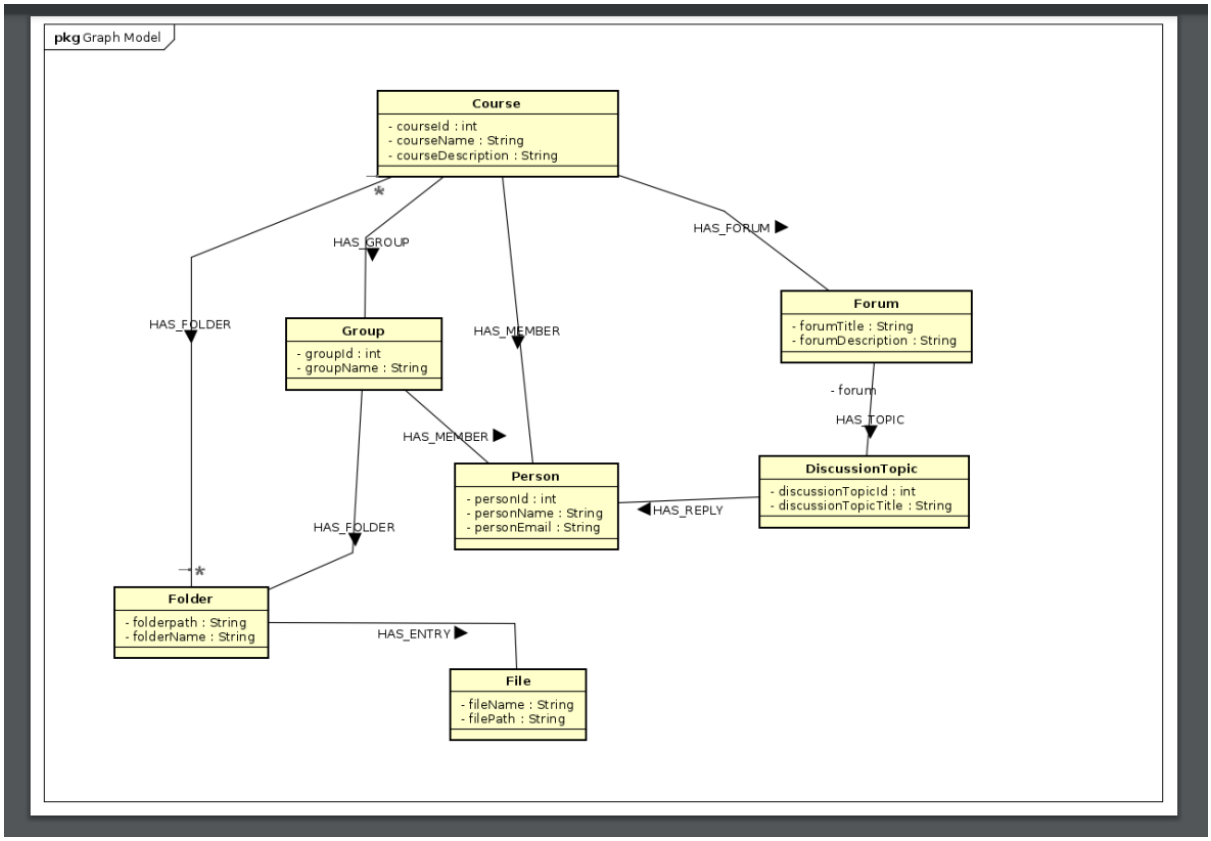
FROM Person as p, DiscussionTopic as d
WHERE P.personID = D.personID
GROUPBY P.personID

Graph Schema

Version 1



Version 2 (More correct)



Cypher Queries

Please populate your graph database with example data such that you provide one (1) Course. Add data such that you get around 5 instances of all associations and classes. The deliverable here is a Cypher file containing CREATE statements.

Example of few queries

CREATE (n:Course
{courseId:"1", courseName:"EWADIS", courseDescription:"Engineering Web and Data Intensive Systems"});

CREATE (n:Person
{personId : "1", personName: "Priya"})

MATCH (a:Course
{courseName:"EWADIS"}
),
(b:Group)
MERGE
(a)-[r:HAS_GROUP] ->(b);

Other examples of merge with conditions

MATCH (a:Course
{courseName:"EWADIS"}
),

```
        (b:Forum {forumName:"Forum_1"})
        MERGE
        (a)-[r:HAS_FORUM] ->(b);

MATCH (a:Group
        {groupName:"Group_1"}),
        (b:Person)
        where b.personName in ["Person_1", "Person_2", "Person_3"]
        MERGE
        (a)-[r:HAS_MEMBER] ->(b);
```

Write Cypher Queries:
Try to provide two Cypher MATCH... queries to answer the following questions. If you think that there can be no solution (with a single query), state why!
• Given a Course (referenced by its ID), please provide a sorted list of its Groups with the number of members in each group.

```
MATCH (course:Course {courseName:"EWADIS"})-[r1:HAS_GROUP]-> (group:Group)
MATCH (group)-[r2:HAS_MEMBER]-> (member:Person)
        return group.groupName, count(member)
        order by count(member) DESC
```

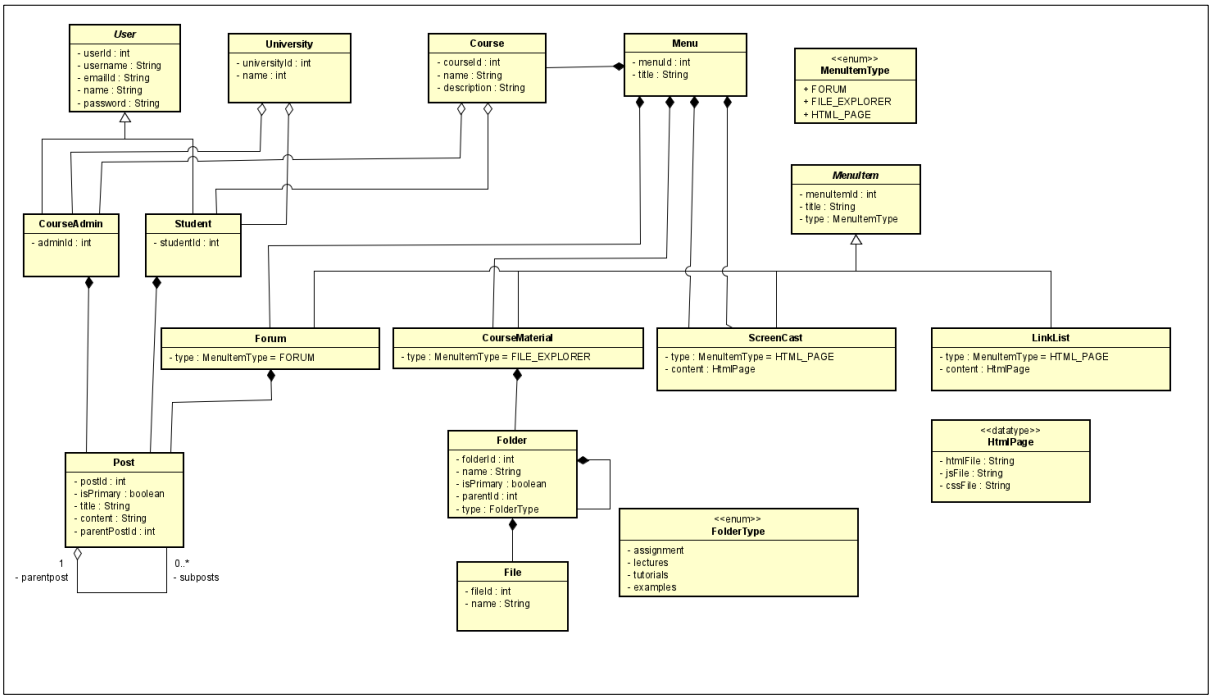
• For each top-level DiscussionTopic, compute the set of Persons who contributed to the topic and its replies.

```
MATCH (course:Course {courseName:"EWADIS"})-[r1:HAS_FORUM]-> (forum:Forum)
MATCH (forum)-[r2:HAS_TOPIC]-> (topic:DiscussionTopic)
MATCH (topic)-[r3:HAS_REPLY]-> (member:Person)
        return topic.discussionTopicName, member.personName , r3.reply
        order by topic.discussionTopicName
```

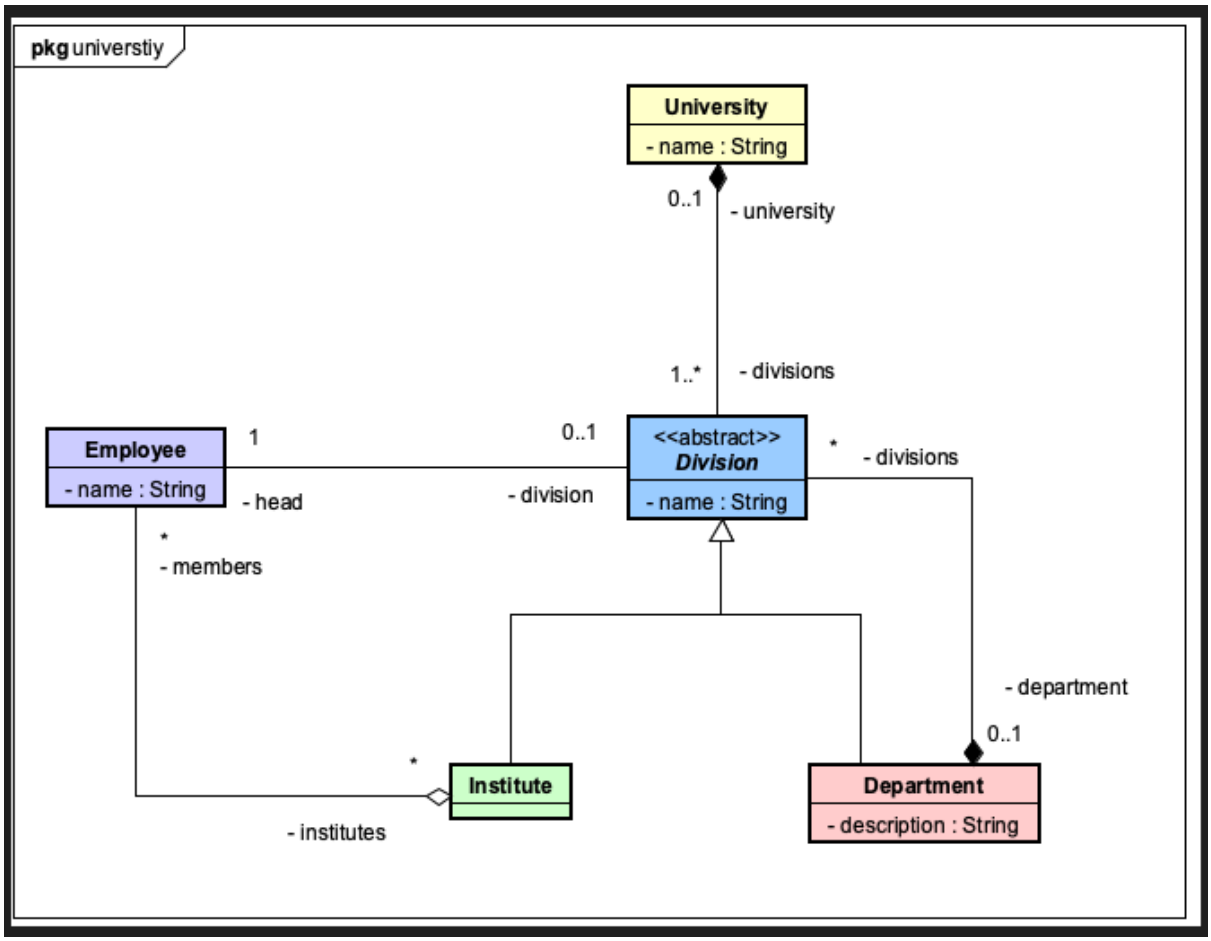
```
MATCH (n) DETACH DELETE n

MATCH (g:Group) -[:HAS_MEMBER]-> (p)
  RETURN g.name, count(p)
  ORDER BY g.name

MATCH (f:Forum) -[:HAS_TOPIC]-> (t) -[:HAS_REPLY*0..]-> () -[:HAS_AUTHOR]-> (p)
  RETURN t, collect(p)
```

University Example

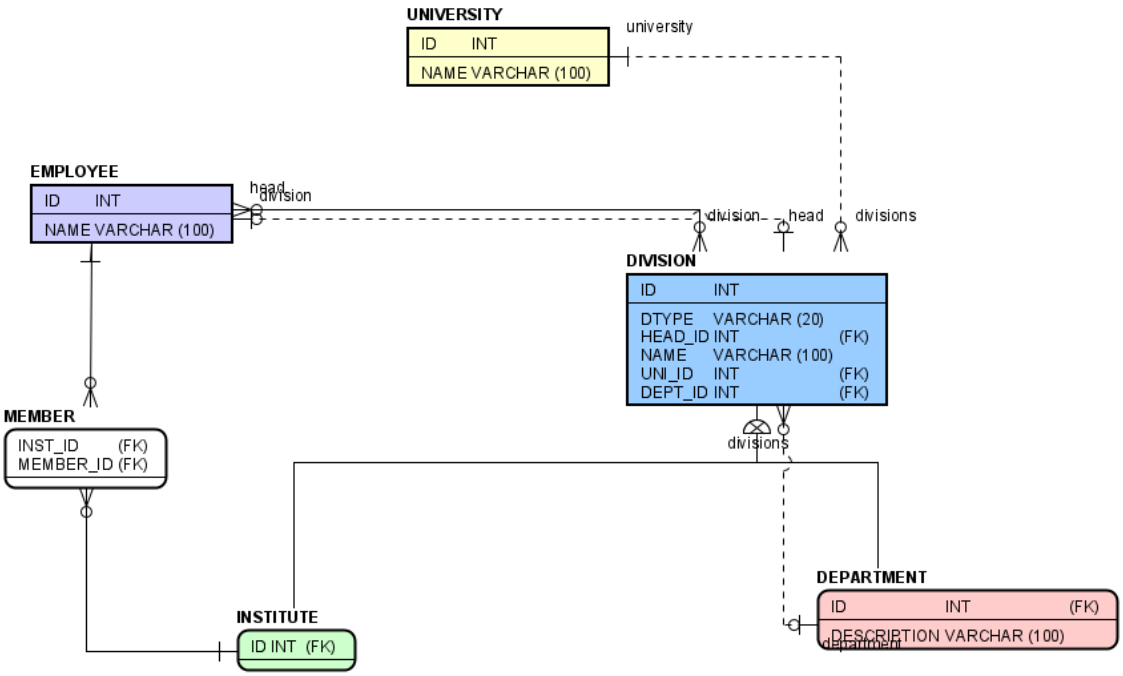
Class Diagram



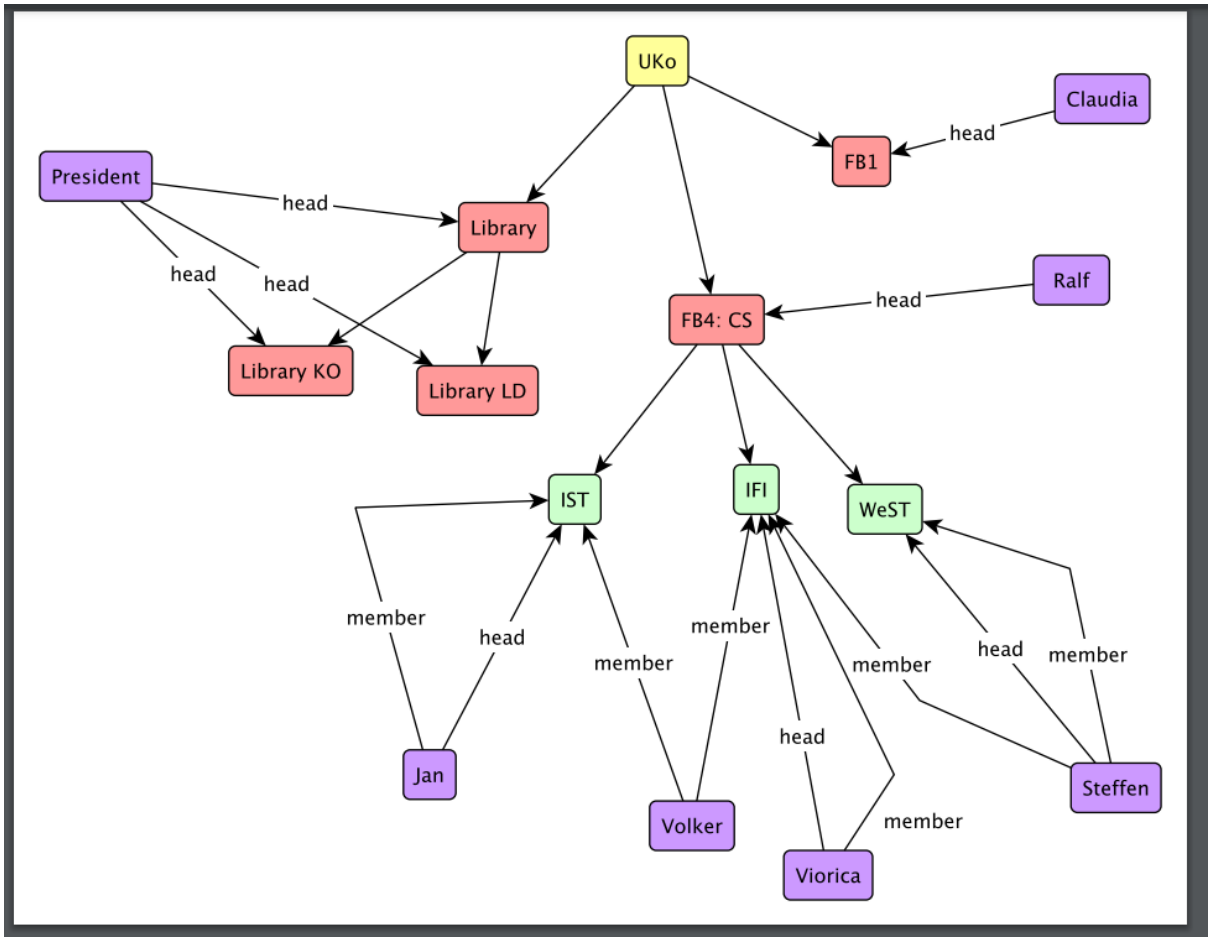
Another Class Diagram



Relational DB Model

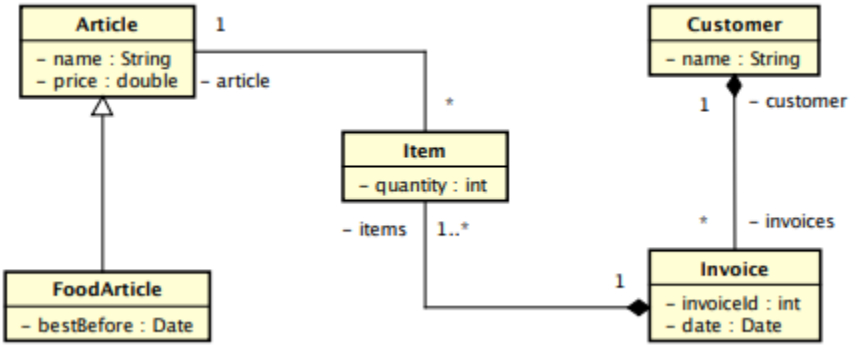


Graph Schema Instance

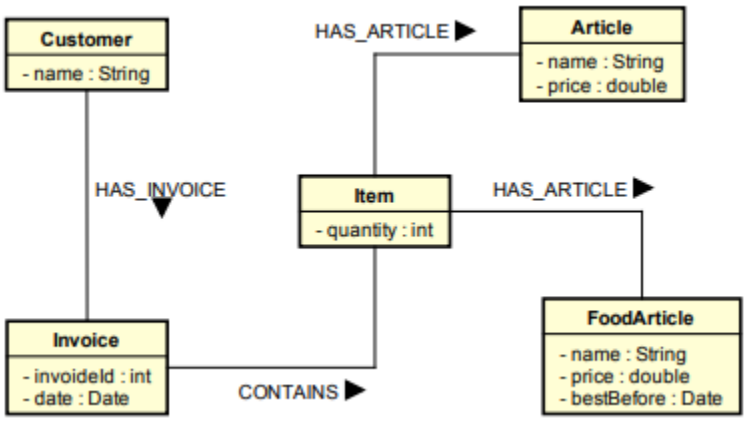

UKo
Claudia
head
FB1
President
head
Library
head
head
Ralf
head
FB4: CS
Library KO
Library LD
IST
IFI
WeST
member
head
member
member
member
head
member
head
Jan
Steffen
Volker
member
Viorica

Assignment 5 WebShop

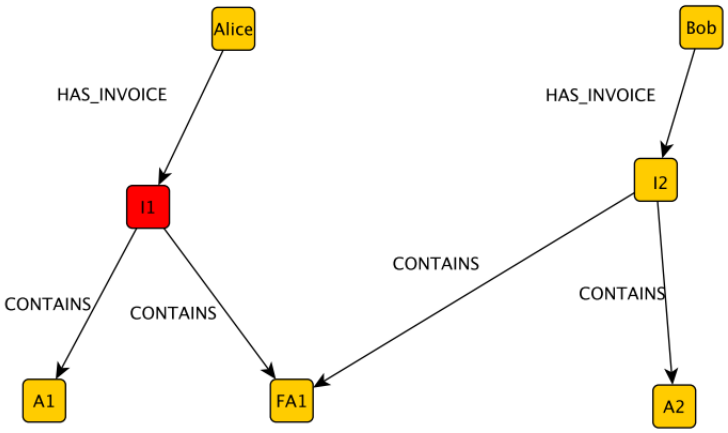
Domain Model



Graph Schema



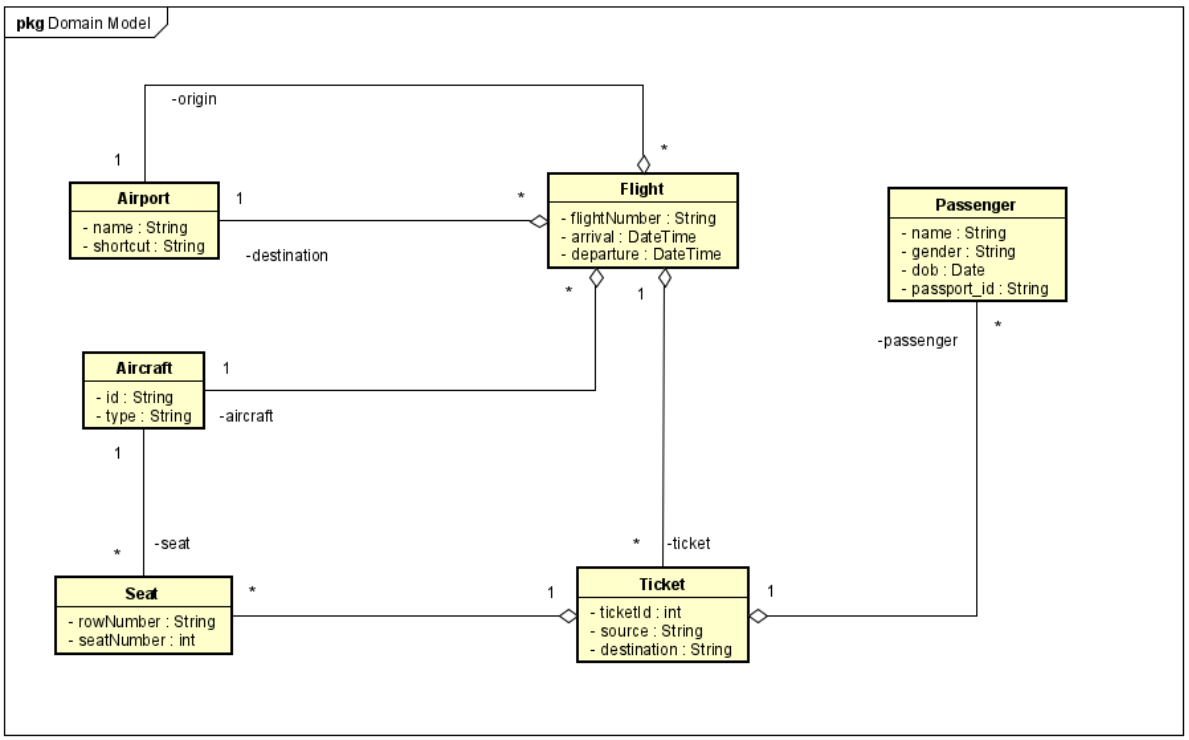
Graph Instance

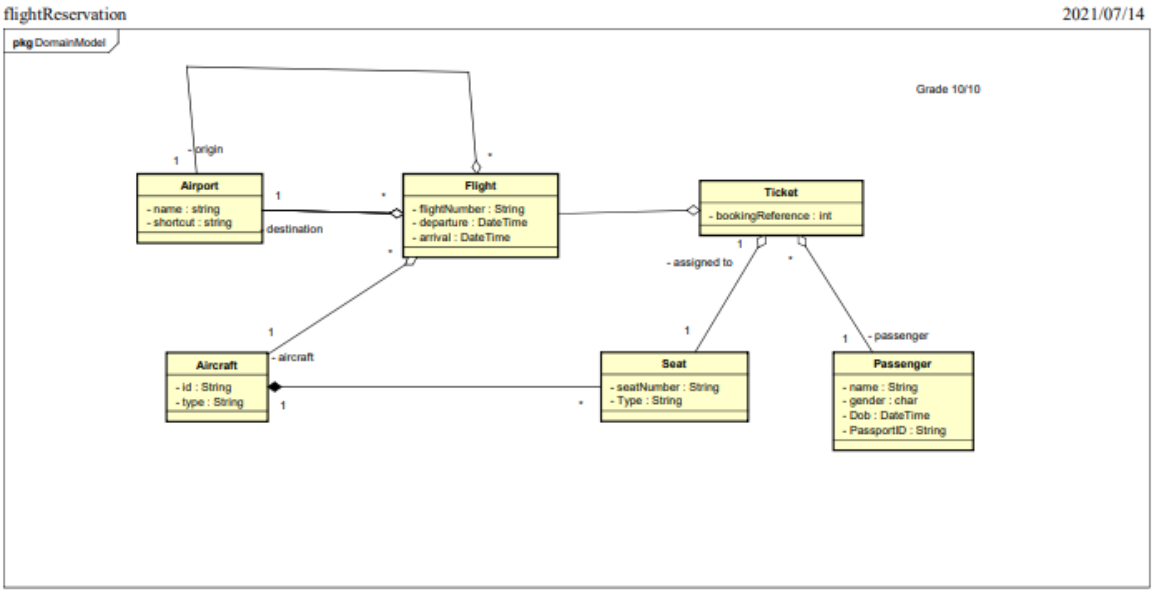


```
cc: Current Customer
ci: Current Invoice
oi: Other Invoice
ca: Current Article (part of ci)
ra: Recommended Article (part of oi)
oc: Other customer

MATCH
(cc :Customer) -[:HAS_INVOICE]->
(ci :Invoice {invoiceId: 1})
  -[:CONTAINS]-> (ca)
  <-[:CONTAINS]- (oi)
  -[:CONTAINS]-> (ra),

(oi) <-[:HAS_INVOICE]- (oc)

WHERE
  cc <> oc
  AND ra <> ca
  AND NOT (ra) <-[:CONTAINS]- (ci)
  AND oi.date + duration("P3M") >= date()

RETURN
  ra
```

Flight Reservation (Airport)

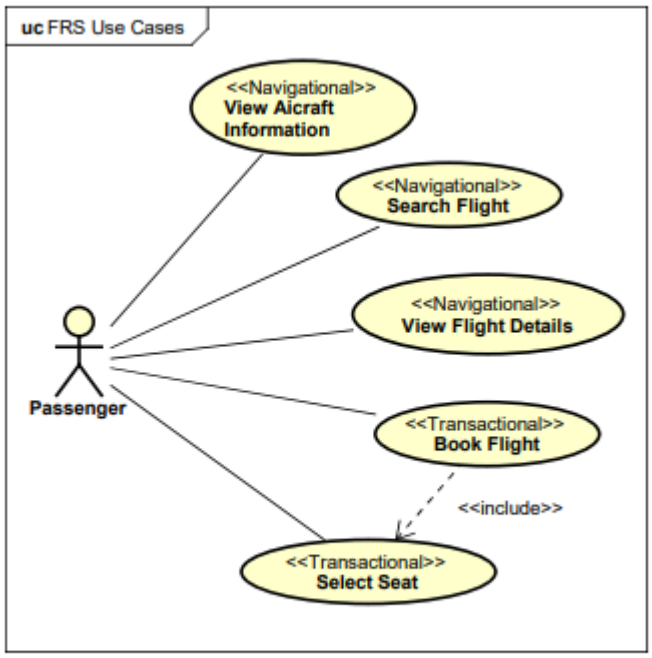
Domain Model



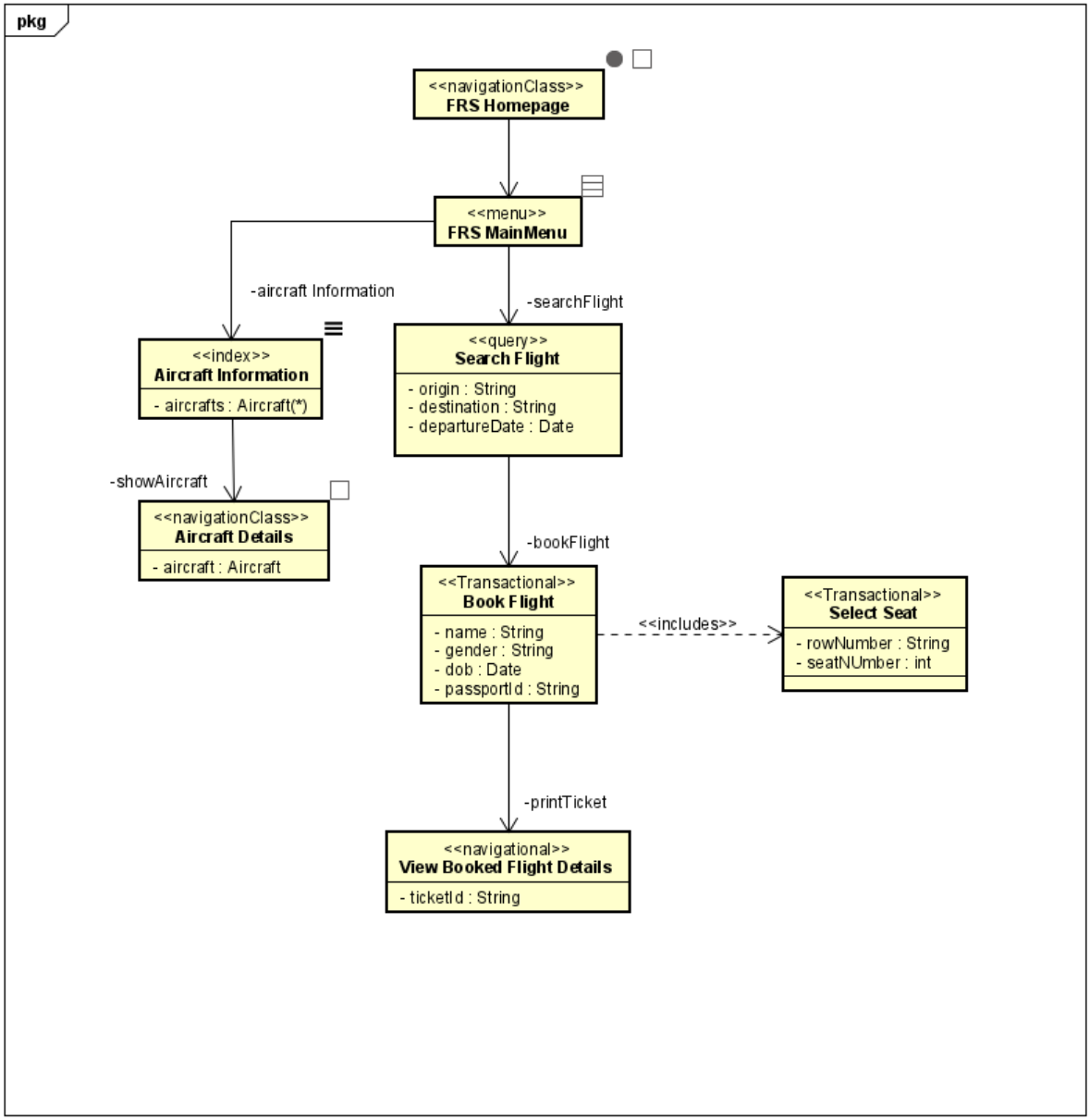


Use Case

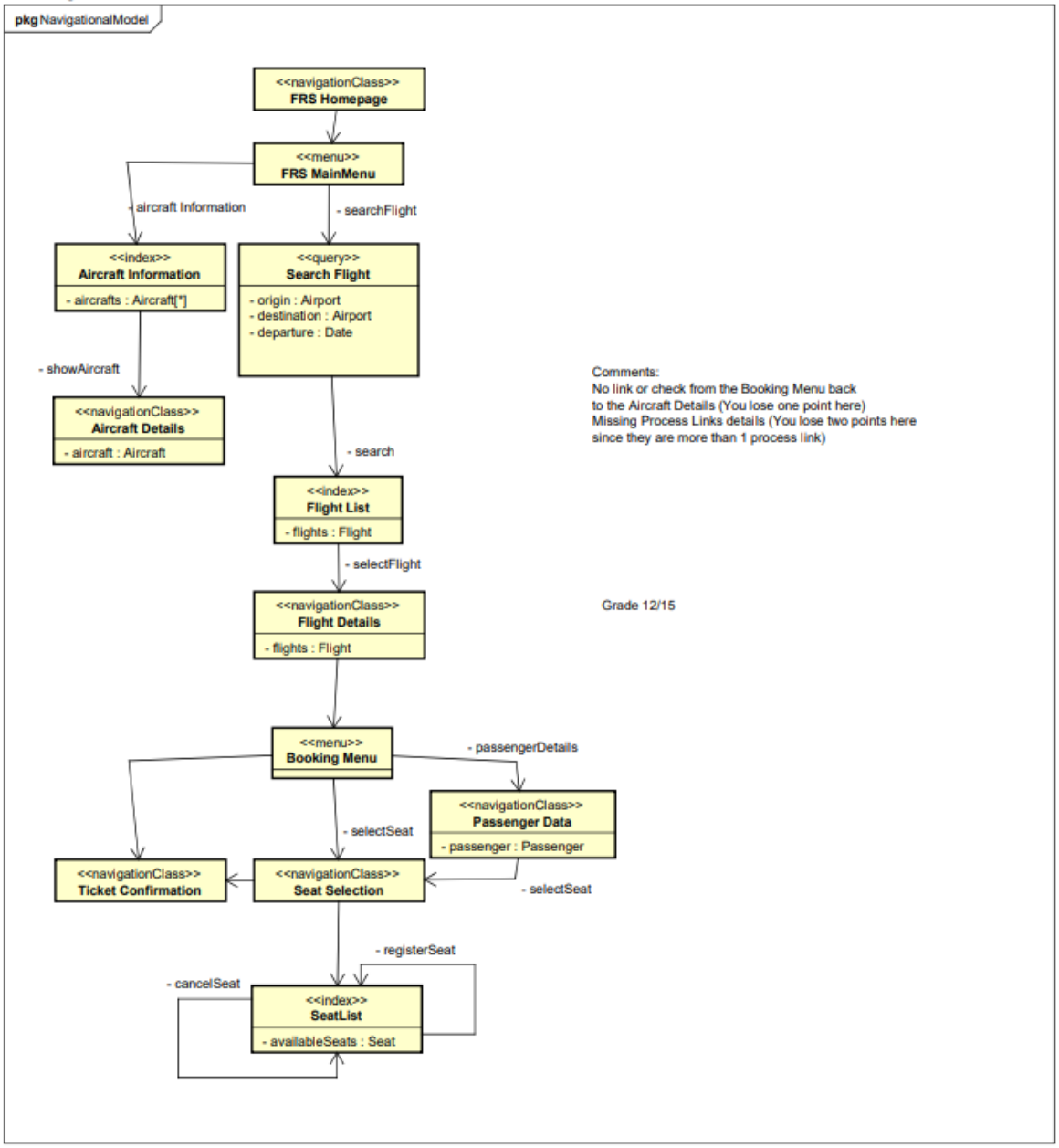
FRS Use Cases



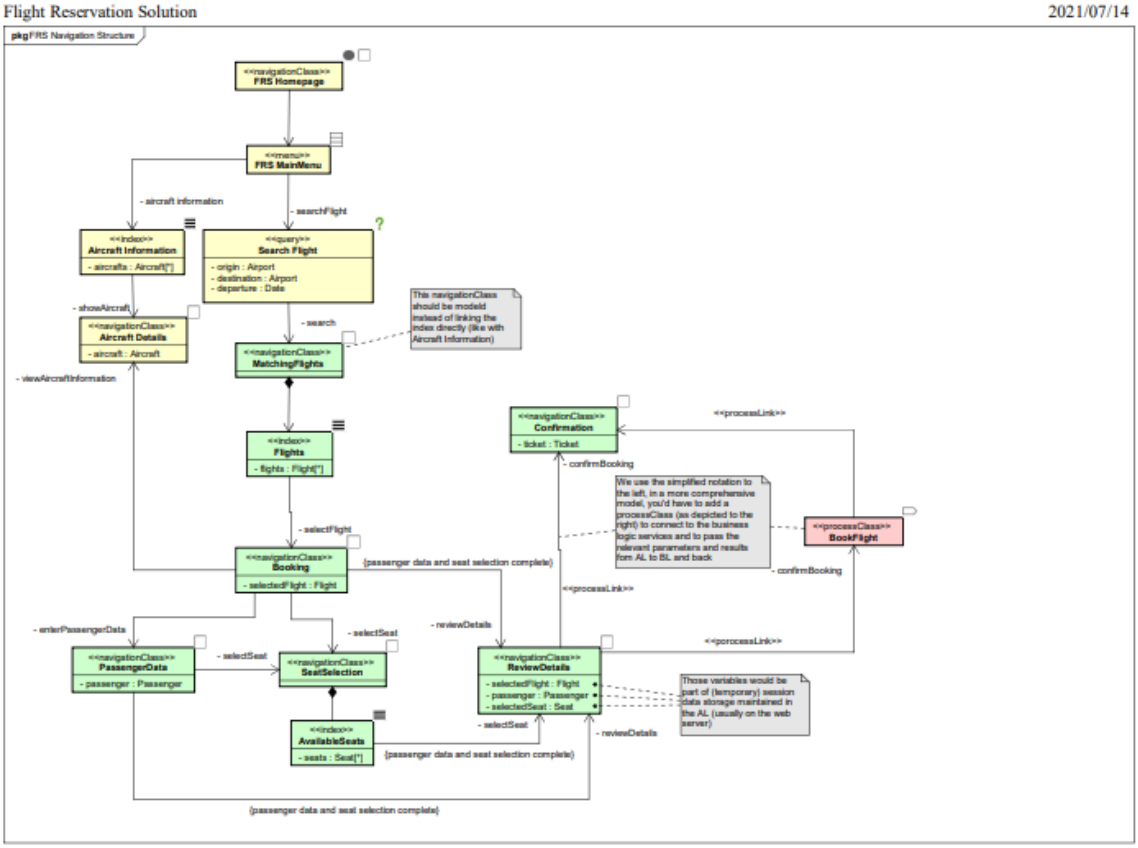
Navigation Diagram


pkg
<<navigationClass>>
FRS Homepage
<<menu>>
FRS MainMenu
-aircraft Information
-searchFlight
<<index>>
Aircraft Information
- aircrafts : Aircraft(*)
<<query>>
Search Flight
- origin : String
- destination : String
- departureDate : Date
-showAircraft
<<navigationClass>>
Aircraft Details
- aircraft : Aircraft
-bookFlight
<<Transactional>>
Book Flight
- name : String
- gender : String
- dob : Date
- passportId : String
<<includes>>
<<Transactional>>
Select Seat
- rowNumber : String
- seatNUmber : int
-printTicket
<<navigational>>
View Booked Flight Details
- ticketId : String

Class Diagram0 2021/07/14

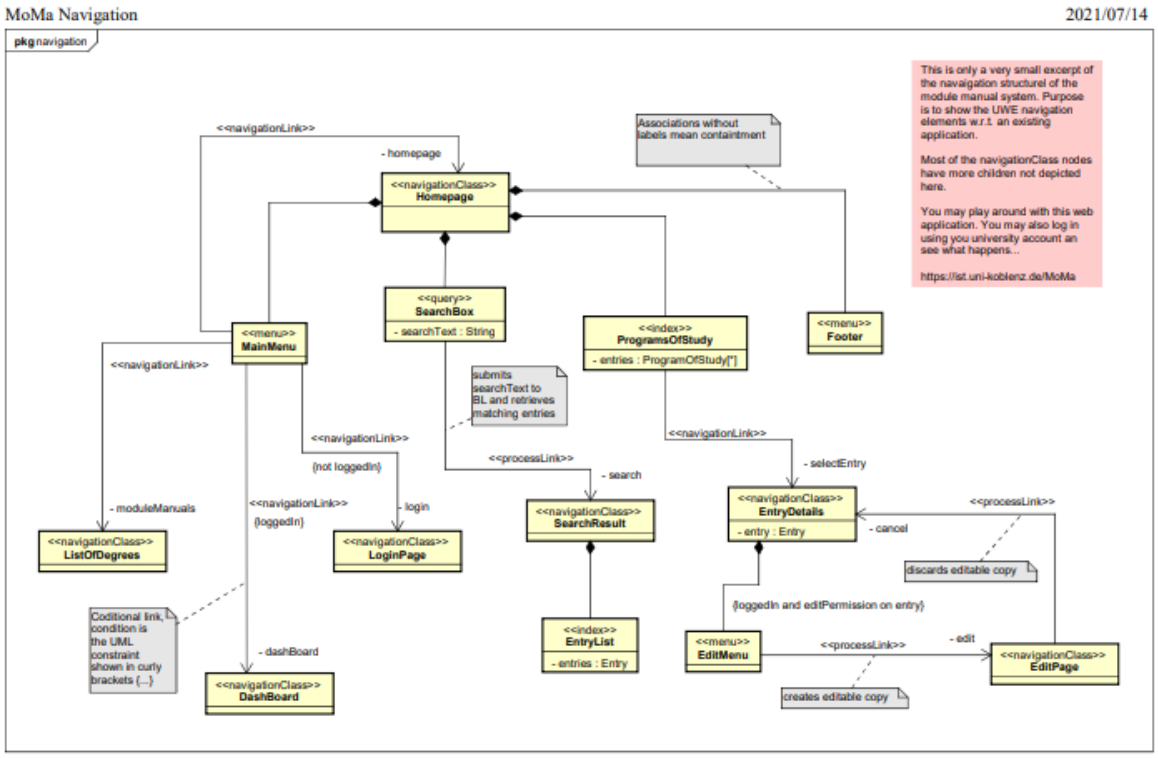


Solution


pkg FRS Navigation Structure
«navigationClass»
FRS Homepage
«menu»
FRS MainMenu
- aircraft information
- searchFlight
«index»
Aircraft Information
- aircrafts : Aircraft[*]
«query»
Search Flight
- origin : Airport
- destination : Airport
- departure : Date
- showAircraft
«navigationClass»
Aircraft Details
- aircraft : Aircraft
- viewAircraftInformation
- search
«navigationClass»
MatchingFlights
This navigationClass should be modeld instead of linking the index directly (like with Aircraft Information)
«index»
Flights
- flights : Flight[*]
- selectFlight
«navigationClass»
Booking
- selectedFlight : Flight
{passenger data and seat selection complete}
- enterPassengerData
- selectSeat
«navigationClass»
PassengerData
- passenger : Passenger
- selectSeat
«navigationClass»
SeatSelection
«index»
AvailableSeats
- seats : Seat[*]
{passenger data and seat selection complete}
- reviewDetails
«navigationClass»
ReviewDetails
- selectedFlight : Flight
- passenger : Passenger
- selectedSeat : Seat
- selectSeat
- reviewDetails
Those variables would be part of (temporary) session data storage maintained in the AL (usually on the web server)
{passenger data and seat selection complete}
«navigationClass»
Confirmation
- ticket : Ticket
- confirmBooking
«processLink»
«processLink»
We use the simplified notation to the left, in a more comprehensive model, you'd have to add a processClass (as depicted to the right) to connect to the business logic services and to pass the relevant parameters and results from AL to BL and back
«processClass»
BookFlight
- confirmBooking
«processLink»

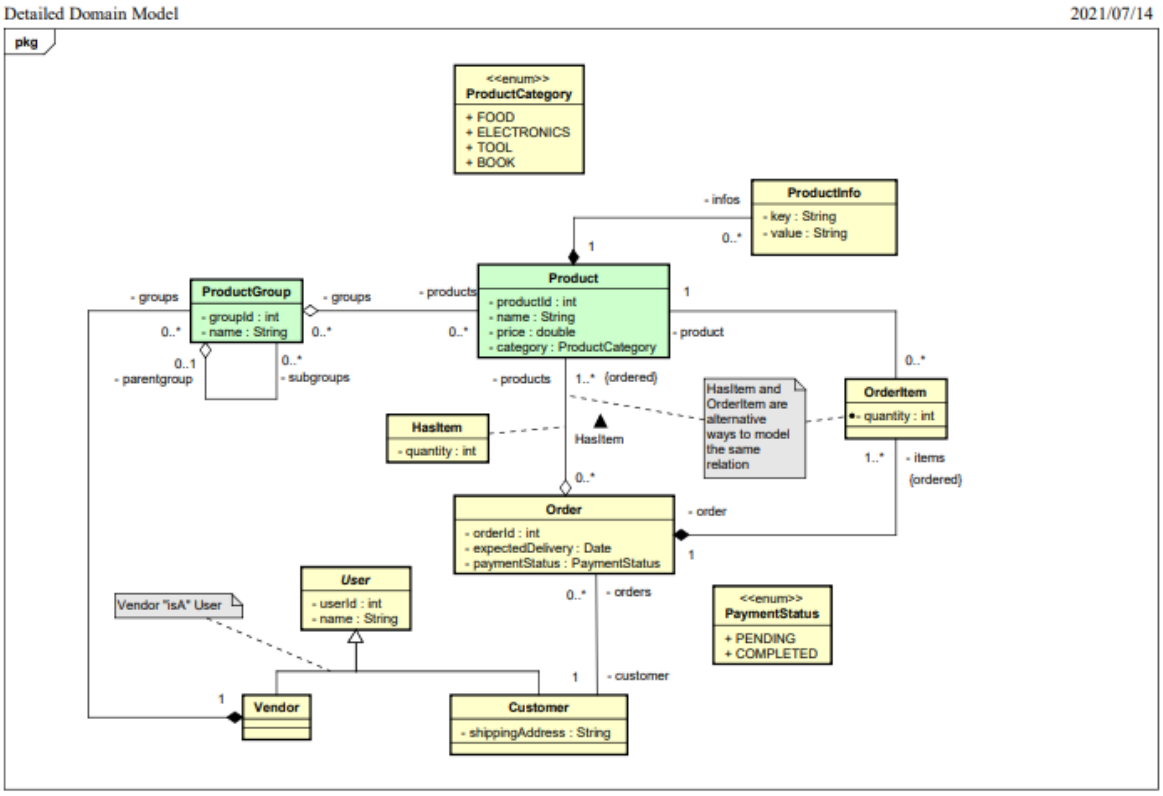
MOMA

Navigation Model

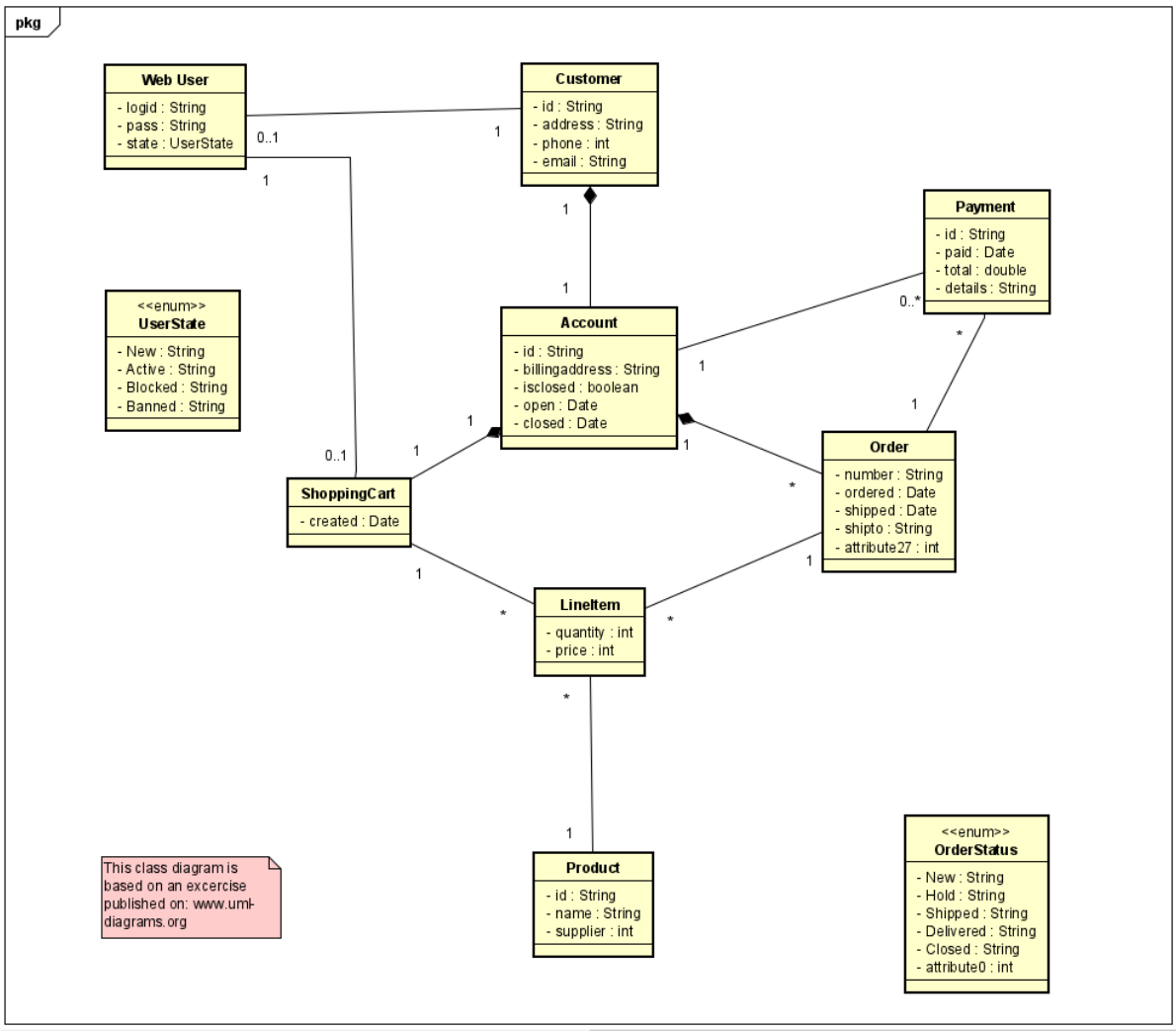


Web Shop

Domain Model

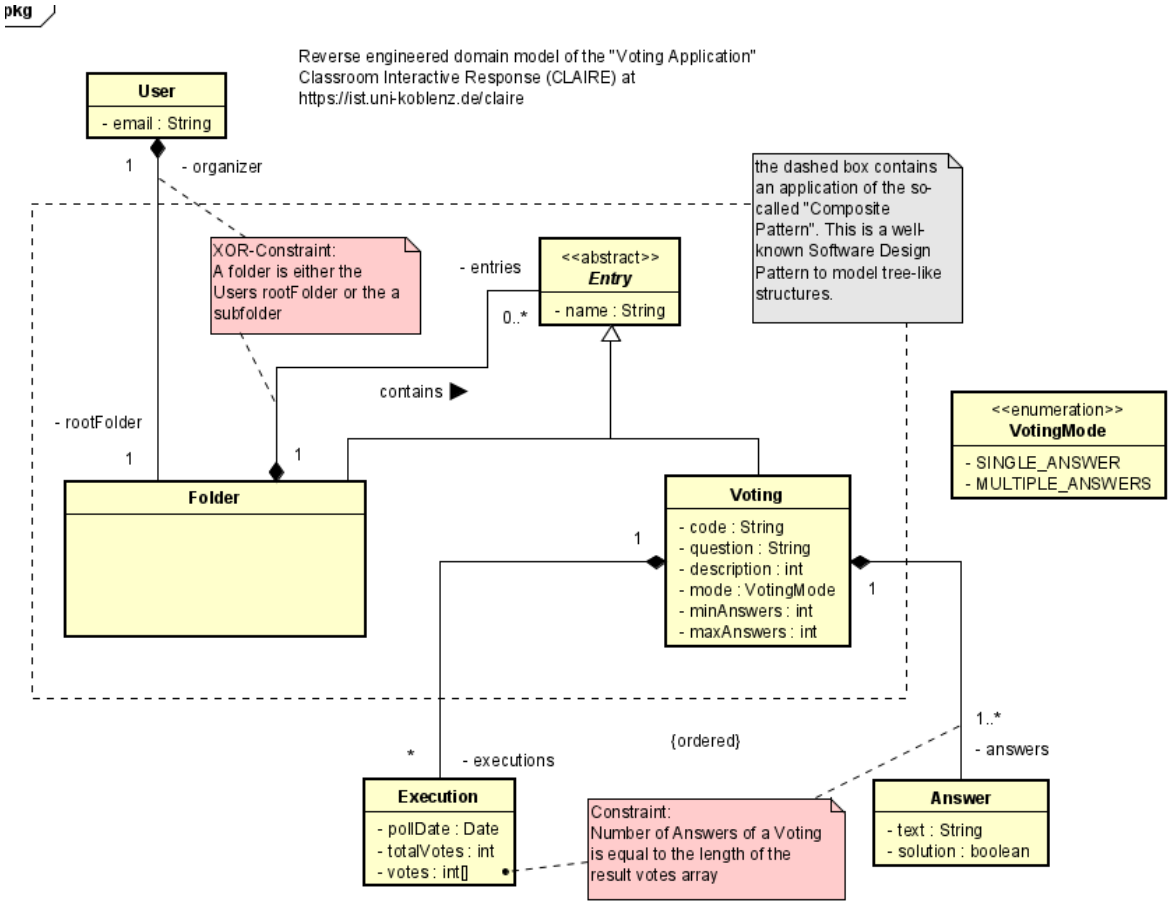


Online Shopping System

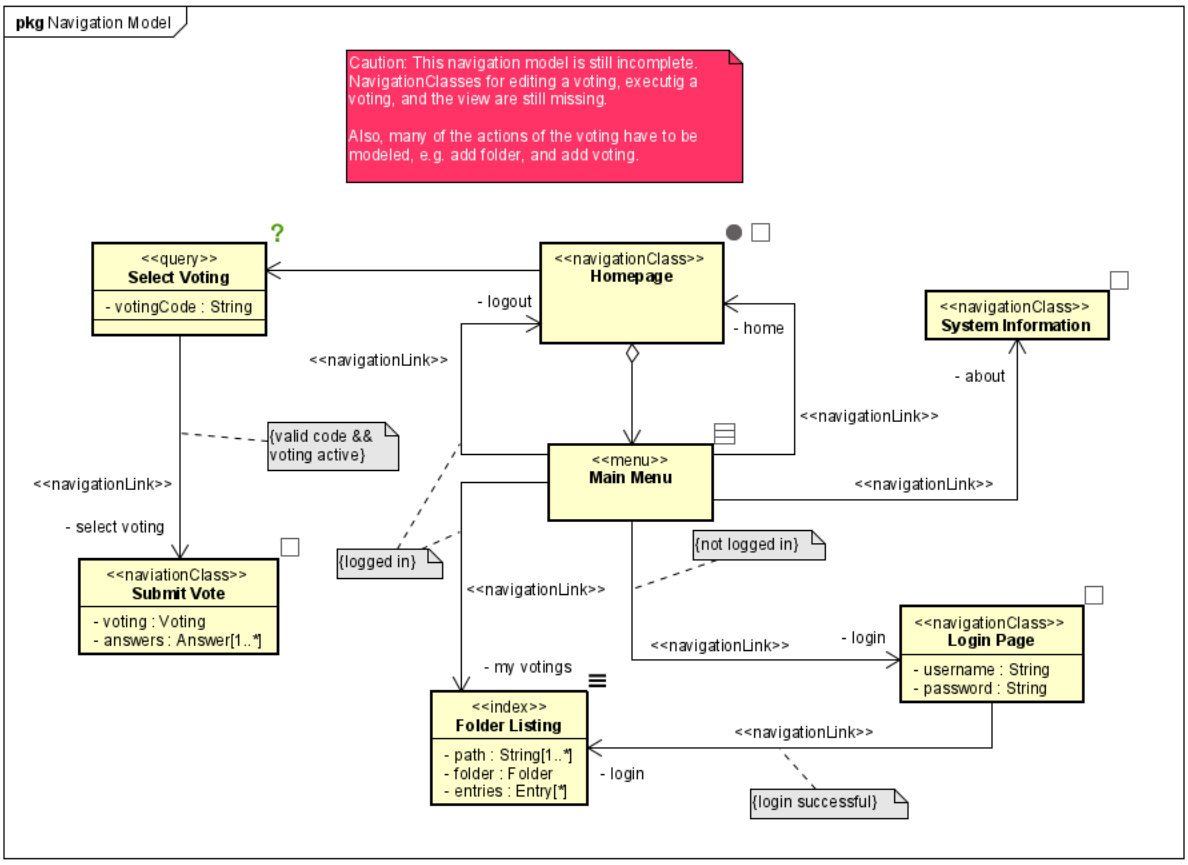

pkg
Web User
- logid : String
- pass : String
- state : UserState
Customer
- id : String
- address : String
- phone : int
- email : String
Payment
- id : String
- paid : Date
- total : double
- details : String
<<enum>>
UserState
- New : String
- Active : String
- Blocked : String
- Banned : String
Account
- id : String
- billingaddress : String
- isclosed : boolean
- open : Date
- closed : Date
ShoppingCart
- created : Date
Order
- number : String
- ordered : Date
- shipped : Date
- shipto : String
- attribute27 : int
LineItem
- quantity : int
- price : int
Product
- id : String
- name : String
- supplier : int
<<enum>>
OrderStatus
- New : String
- Hold : String
- Shipped : String
- Delivered : String
- Closed : String
- attribute0 : int
This class diagram is based on an excercise published on: www.uml-diagrams.org
0..1
1
0..*
*

Voting System

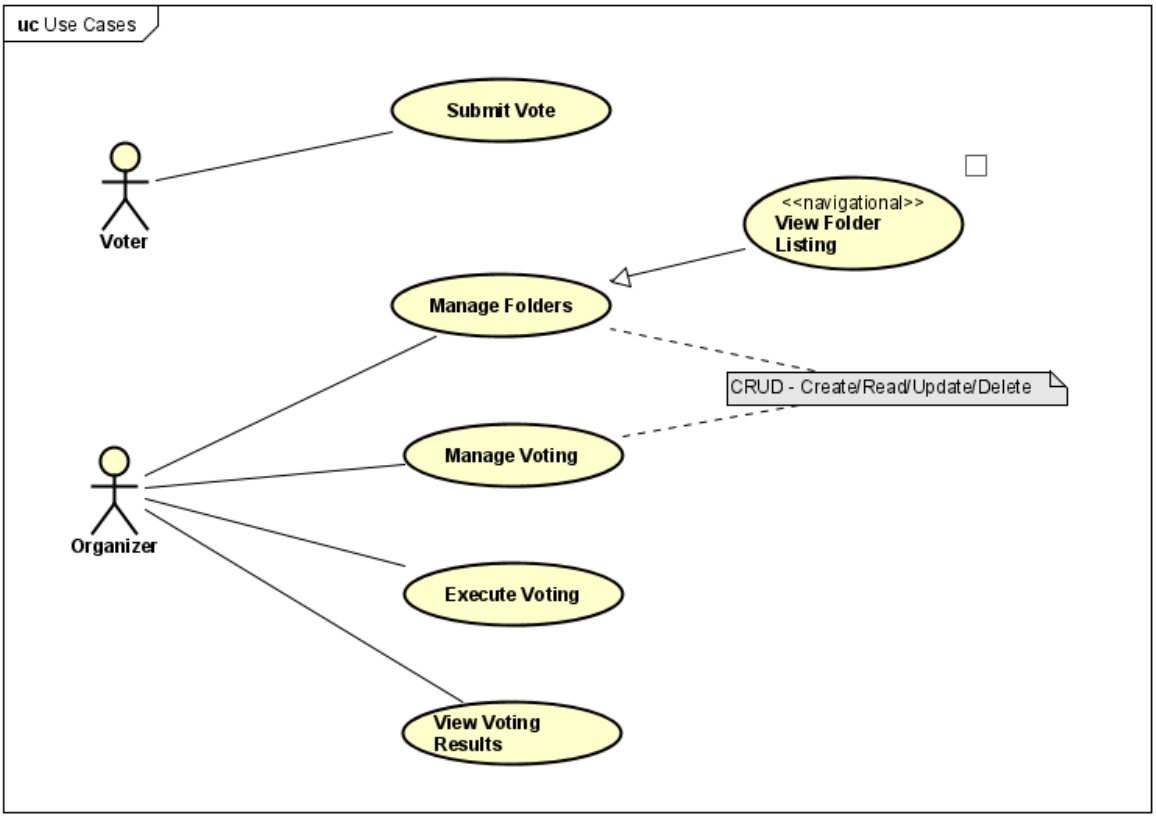
Domain Diagram



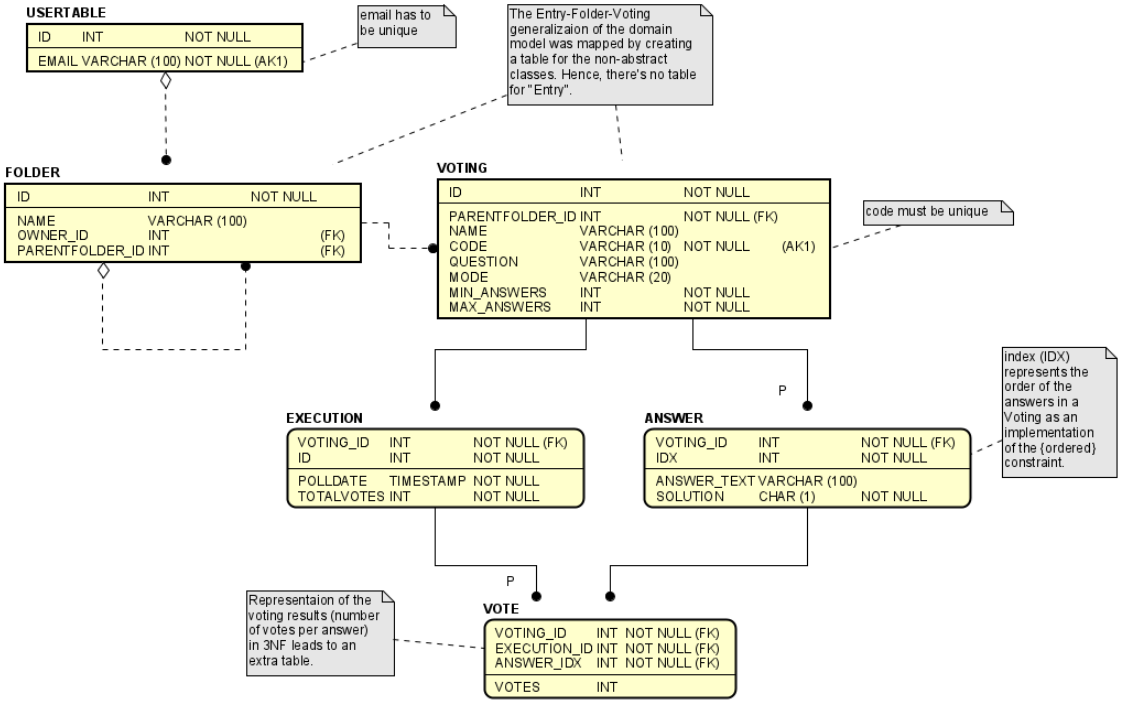
Navigation Diagram


pkg Navigation Model
Caution: This navigation model is still incomplete. NavigationClasses for editing a voting, executig a voting, and the view are still missing.
Also, many of the actions of the voting have to be modeled, e.g. add folder, and add voting.
<<query>>
Select Voting
- votingCode : String
<<navigationClass>>
Homepage
<<navigationClass>>
System Information
- logout
- home
- about
<<navigationLink>>
<<navigationLink>>
<<navigationLink>>
{valid code && voting active}
<<menu>>
Main Menu
<<navigationLink>>
- select voting
<<navigationLink>>
{logged in}
{not logged in}
<<navigationLink>>
<<navigationLink>>
- login
<<navigationClass>>
Login Page
- username : String
- password : String
- my votings
<<index>>
Folder Listing
- path : String[1..*]
- folder : Folder
- entries : Entry[*]
<<navigationLink>>
- login
{login successful}

Use Case


uc Use Cases
Submit Vote
Voter
<<navigational>>
View Folder Listing
Manage Folders
CRUD - Create/Read/Update/Delete
Manage Voting
Organizer
Execute Voting
View Voting Results

Relational DB Model



Food Article

CREATE

(u_1:User {name:"A"}),

(u_2:User {name:"B"}),

(u_3:User {name:"C"})

CREATE

(a_1:Article {name:"Butter"}),

(a_2:Article {name:"Sugar"}),

(a_3:Article {name:"cheese"}),

(a_4:Article {name:"milk"}),

(a_5:Article {name:"eggs"})

CREATE

(u_1) -[:buys]->(a_1),

(u_1) -[:buys]->(a_2),

(u_2) -[:buys]->(a_2),

(u_2) -[:buys]->(a_5),

(u_3) -[:buys]->(a_2),

(u_3) -[:buys]->(a_3),

(u_3) -[:buys]->(a_4),

(u_3) -[:buys]->(a_5)

match (u_s:User)-[r:buys]-(a_s:Article) return u_s,a_s

match (u_s:User)-[:buys]->(a_s:Article)<-[:buys]-(u_d:User)-[:buys]->(a_d:Article) where u_s.name <> u_d.name and NOT (a_d)<-[:buys]-(u_s) return distinct u_s.name,a_d.name order by u_s.name