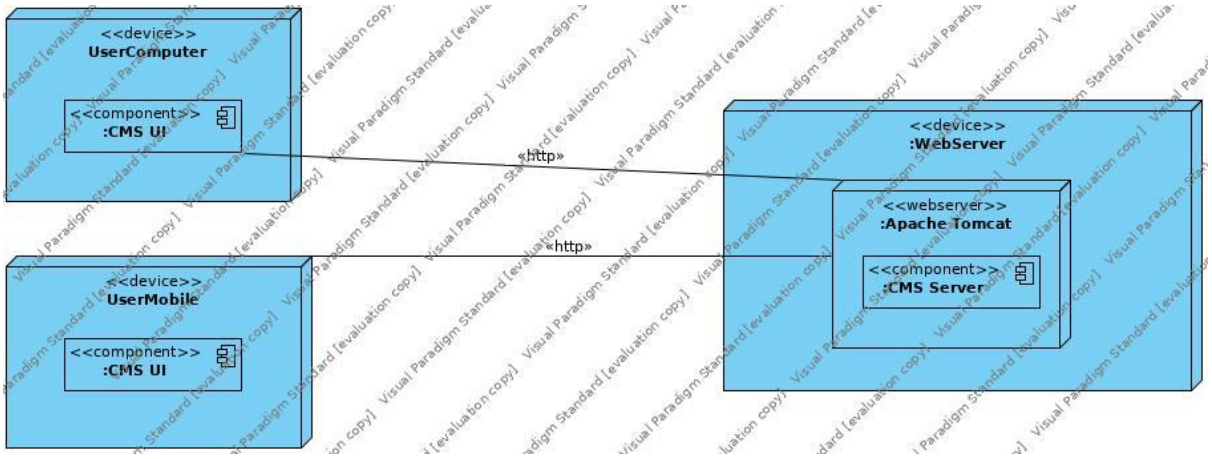
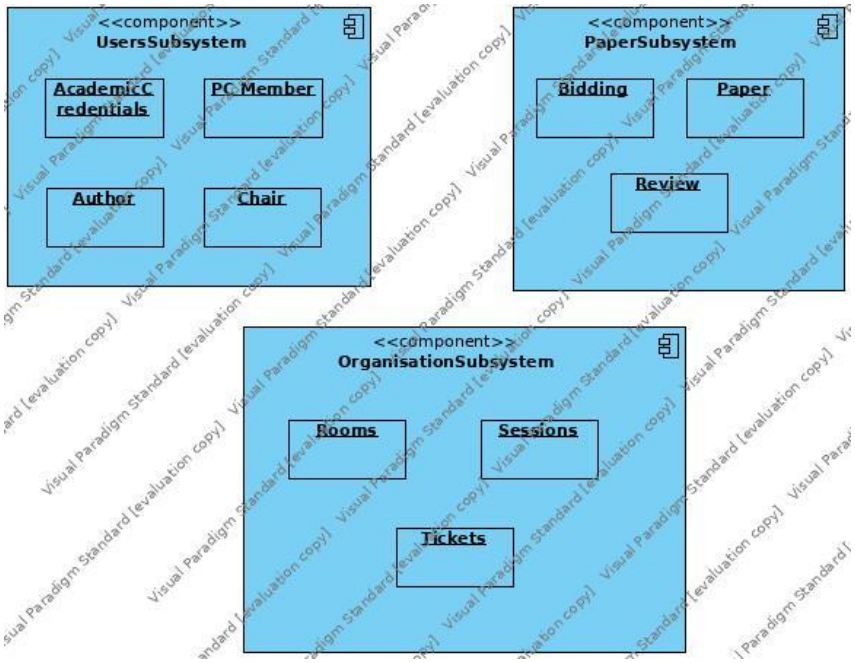
Design Model
CMS

1.Mapping Subsystems



The CMS has 2 generic components, one that has only one instance at any given moment, which is the **CMS Server** (the complexity of our application made it needless to use a distributed-server). It is a web-based service on an Internet host, deployed via Apache Tomcat. The host is a **Unix** machine (a wiser decision should be to have a Virtual Machine so snapshots could be regularly taken, or a **Docker** facility due to back up quality).
The **CMS UI** can have unlimited possible instances, on whatever device has an Internet connection and a **running web browser**, like Google Chrome or Mozilla Firefox.
Due to the needlessness of a distributed server, we did not parse it into subsystems. Nevertheless, the growth of the application should propose the apparition of this Subsystems (tentatively described):

1. **Users Subsystem**: responsible for handling the credentials of users, as much as for the validation of the academic credentials and other policy-related matters.
2. **Conference Organisations Subsystem**: responsible for the “hard” part of a conference, like organising it into sessions, on rooms, ticket selling.
3. **Paper Subsystem**: responsible for the main thrust of the application, handling paper, bidding and reviewing information.



2.Persistent data

Due to the fact that our data is not great in number as in “weight” (the abstracts, paper contents and presentations can be quite big files), we chose a 3-layered data management, succinctly described in the following lines.

1. **Session state**, i.e., the credentials of the user currently logged in in a client-machine is held in the browser session data/ **cookies**. Due to the EU regulations, it is not possible to store an infinite amount of user data in this manner, but nevertheless the access token should be persisted in this way. Being a semi-volatile type of data, it lives as long as the user is logged in, or, if he decides saving his credentials for further logging in, for a longer time (until the token expires and the user is automatically logged out).
2. **Conference soft data**, i.e., users information, paper metadata, review data and the like are persisted in a classic relational database (for us, **PostgreSQL**, but further inquiry could reveal better solutions). The management of the data in this section is done via **Hibernate**, and above, **Spring Data JPA**.
3. **Conference hard data**, i.e., abstracts, presentations, reviews is stored in a file-based system on the server machine. Knowing that this fact could drive the system into performance issues we are currently inquiring into finding a cloud facility to host this huge amount of data.

3.Access Control

Taking into consideration the highly-refined quality of the app requirements, the strategy we used when designing the access control could be, up to a point, considered “cheating” and a Product Owner could feel they were not thoroughly followed; in plain language that means that we parsed the multi-layered user hierarchy proposed in the requirements into only 5 basic kinds of users, with their according credentials.

1. **Non-registered users** can access only the homepage of the CMS and the ticket buying facility. No data besides theirs is available, and they can only do this kind of operation.
2. **Authors** can propose papers and further update their content. They also have access to the review of their work, but nothing more. They cannot get information about other paper or user-related matters.
3. **PC Members** can choose which paper they would like to review, that means they have full read credentials on paper related issues, but they cannot modify any data besides their review. A refining of this definition allows PCs to upload papers like authors, but at the expense of losing some of their review capacities, like the possibility of reviewing their own paper.
4. **Chair Members** are the total-rights users of the application. They can read any data and are allowed to modify other users’ rights. Nevertheless, they cannot update other users profiles or change the updated data.
5. **DB Administrators** are given full-blown rights directly on the DB, but should wisely-use them. One of the mandatory deeds of this actor is adding the organiser of the conference into the db before one could really enter the application and create a conference.

The data is secured at various levels against ill-meant people. The credentials are encrypted via **JWT** and the Server is secured with **Spring Security**. The DB, hosted by **Google**, is assured to be protected.

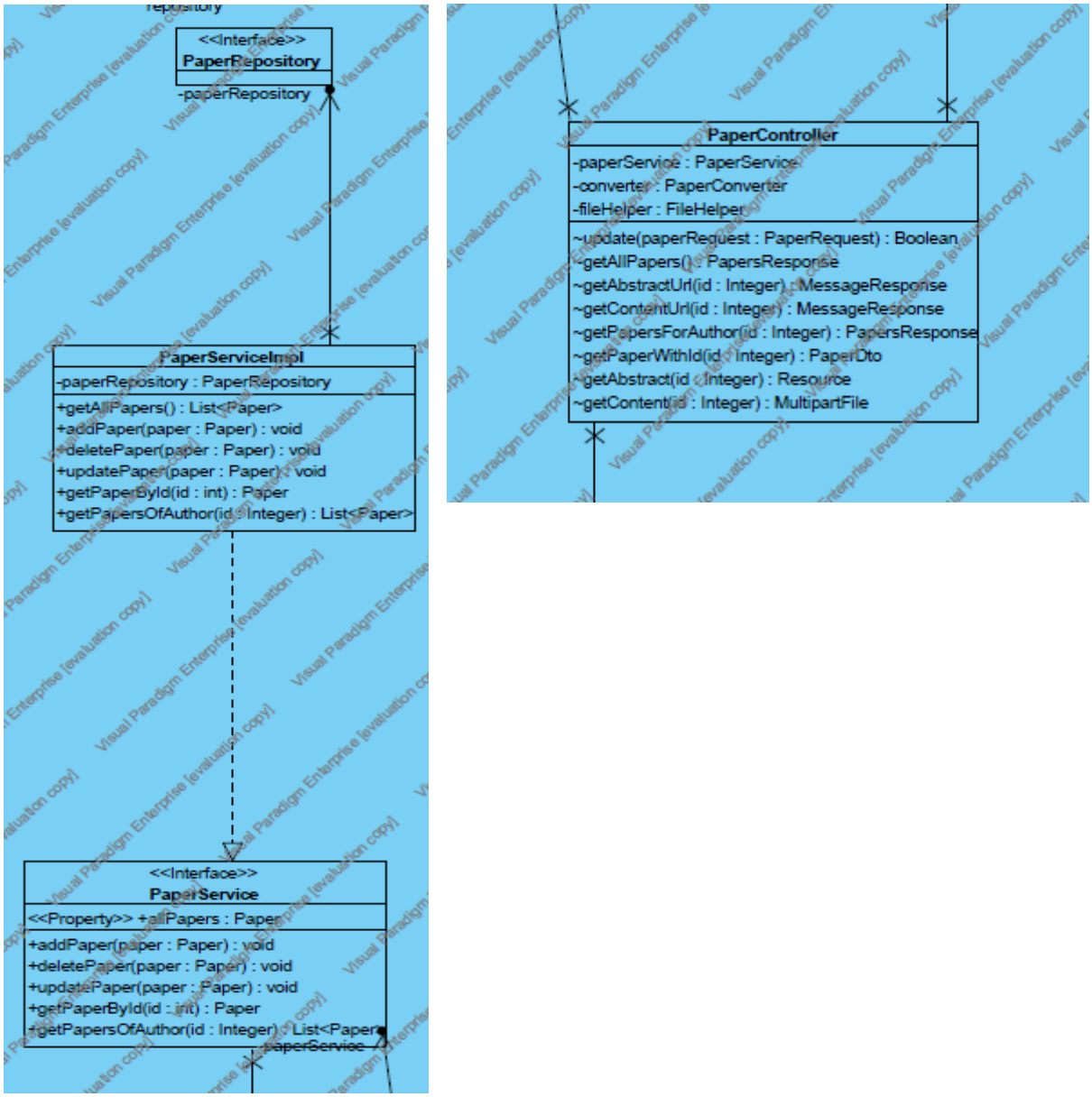
4.Designing the Global Control Flow

Due to the reliance that can be put upon the tools we are using (PostgreSQL hosted by Google, with Hibernate, hence **multithreading** support, Java Spring with **concurrency** support, Angular 9 on TypeScript with **asynchronous-programming** support) we decided that the wisest and most efficient in cost, reliability and user-feedback terms is using **Threads** as the control flow mechanism.
This means that theoretically an user can make an indefinite big concurrent number of http calls to the Server and they would be solved in paralel. Taking into consideration the limited diversity of possible operations available for an user at a given point of time, we can safely say that one can at a given time press as many buttons as he or she has available and the Server will heartily listen.

Now, the operation queue is definitely regulated by logic, security-related and policy related limits. None can do more than accessing the homepage and buy tickets without being registered, and every query to the server is validated to be in accordance with the user credentials and the current Conference phase (so even if the author has the URL of paper uploading, he couldn’t do it in the second phase of the conference).

5.Services

For our problem, we have designed a Service (interface and implementation) for each domain entity, such as *Paper* in order to ensure that the SOLID principles are fully respected by our application.
All the Services can be seen on the annexed improved Class Diagram, however we will now provide explanation for the fore-mentioned Paper Service, using a screenshot.



It can be seen that, on this example, the Paper Service has no interference whatsoever with any other unrelated Service such as Author Service, thus enforcing both high coherence and low coupling and making the system highly readable and maintainable.
In this way, we ensure that there exist minimum dependencies between the previously identified subsystems.
Furthermore, the Paper Controller acts as a mediator between the Server and the Client, being the one responsible with handling and processing the requests and answering with an appropriate response.

6.Boundary Conditions

When thinking about possible system failures, we made the decision to ensure that the external tools we would use are as reliable as possible, in order to handle system crashes, connection issues and so on. To exemplify, a power outage on the side of a potential Author would not affect his data, current state of the paper and so on, due to the fact that our database fully satisfies the ACID transaction properties, which are intended to guarantee validity even in the event of errors, power failures, etc.
For our application, an event which would cause an exception would be trying to review a paper before it was even submitted (more specifically, the paper does not exist). In this case, the error is handled using an Exception with a custom message which is propagated from the server to the client, and the respective reviewer is informed. This pattern is valid for each recurring problem entity, e.g. an author doesn’t exist.
Another problem we faced is when should our problem objects be created and destroyed, i.e. when do the objects change their state, and to exemplify the path we chose, we wrote below the extremely simplified “life” of different object types.

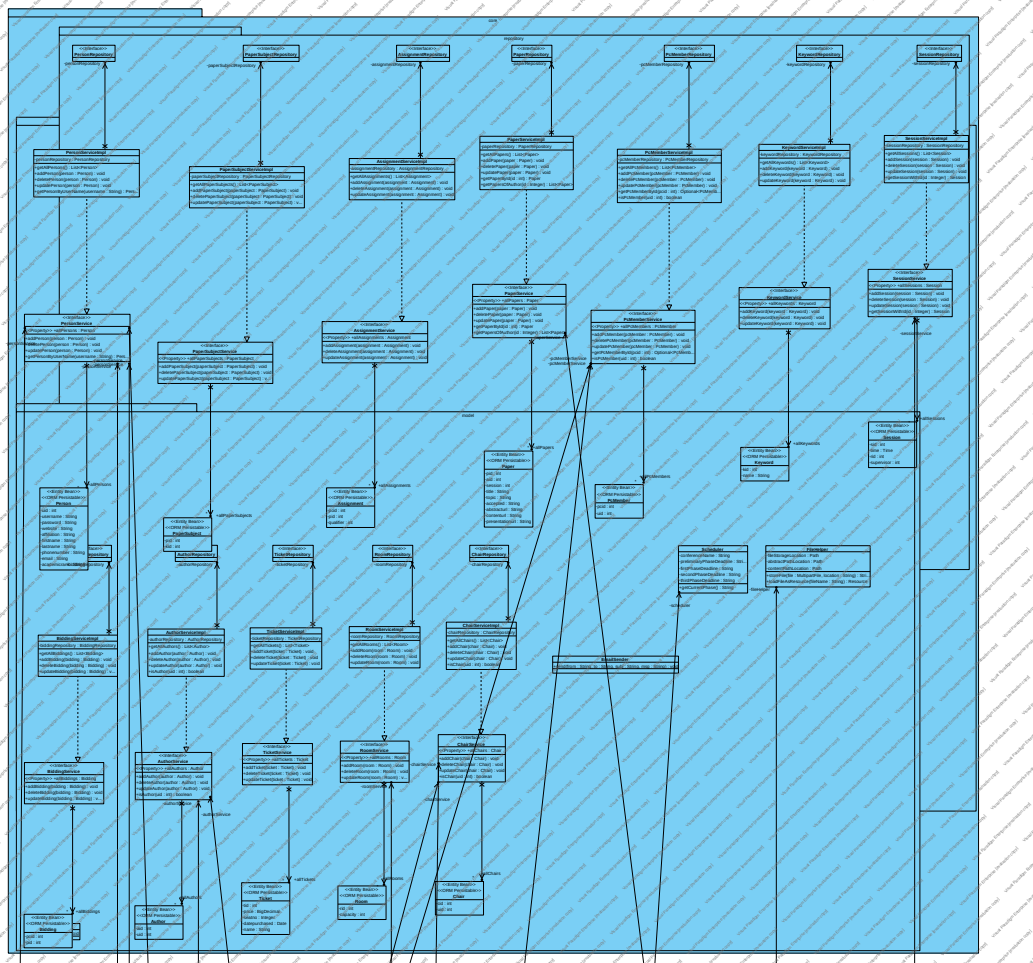
1. Services/ Controllers: created at server start, destroyed at server finish, unless a fatal error occurs
2. Converter Objects/DTOs: created only when needed by the program, destroyed afterwards, the occupied memory is freed.
3. Persistent Objects- always kept secure from any program harm, created before/ during the program, and destroyed only when the database itself ceases to exist.

7.Reviewing and Conclusions

Last but not least, when reviewing once more and comparing our implementation of the system design, we have drawn the following conclusions:

1. The system is **correct**, due to the fact that every decision made in the analysis model, i.e. use cases, nonfunctional requirements, actors access policy, is correlated with the features exemplified in the system design model.
2. The model is **consistent**, because it does not contain contradictions such as duplicate subsystem names and the objects’ exchange is done in orderly fashion.
3. The **realism** of the model is given by the fact that the robustness and reliability of the proposed external tools is undoubtedly good, and the concurrency issues are addressed by the multithreaded control-flow.

4. The **readability** of the model is proven by the subsystems being grouped by suggestive names, denoting similar concepts.
5. Because such problems as persistent storage, access control, boundary conditions, have been addressed, it can be concluded that out system is **complete.**


com
repository
«interface»
PersonRepository
personSubjectRepository
«interface»
PaperSubjectRepository
assignmentRepository
«interface»
AssignmentRepository
paperRepository
«interface»
PaperRepository
pcMemberRepository
«interface»
PcMemberRepository
keywordRepository
«interface»
KeywordRepository
sessionRepository
«interface»
SessionRepository
PersonServiceImpl
personRepository : PersonRepository
+getAllPersons() : List<Person>
+addPerson(person : Person) : void
+updatePerson(person : Person) : void
+getPersonById(personId : Integer) : Person
+getPersonByUserName(userName : String) : Person
PaperSubjectServiceImpl
paperSubjectRepository : PaperSubjectRepository
+getAllPaperSubjects() : List<PaperSubject>
+addPaperSubject(paperSubject : PaperSubject) : void
+deletePaperSubject(paperSubject : PaperSubject) : void
+updatePaperSubject(paperSubject : PaperSubject) : v...
AssignmentServiceImpl
assignmentRepository : AssignmentRepository
+getAllAssignments() : List<Assignment>
+addAssignment(assignment : Assignment) : void
+deleteAssignment(assignment : Assignment) : void
+updateAssignment(assignment : Assignment) : void
PaperServiceImpl
paperRepository : PaperRepository
+getAllPapers() : List<Paper>
+addPaper(paper : Paper) : void
+deletePaper(paper : Paper) : void
+updatePaper(paper : Paper) : void
+getPapersByAuthor(id : Integer) : List<Paper>
+getPapersByKeyword(id : Integer) : List<Paper>
PcMemberServiceImpl
pcMemberRepository : PcMemberRepository
+getAllPcMembers() : List<PcMember>
+addPcMember(pcMember : PcMember) : void
+deletePcMember(pcMember : PcMember) : void
+updatePcMember(pcMember : PcMember) : void
+getPcMemberById(pcId : int) : Optional<PcMem...
+isPcMember(pcId : int) : boolean
KeywordServiceImpl
keywordRepository : KeywordRepository
+getAllKeywords() : List<Keyword>
+addKeyword(keyword : Keyword) : void
+deleteKeyword(keyword : Keyword) : void
+updateKeyword(keyword : Keyword) : void
SessionServiceImpl
sessionRepository : SessionRepository
+getAllSessions() : List<Session>
+addSession(session : Session) : void
+deleteSession(session : Session) : void
+updateSession(session : Session) : void
+getSessionById(id : Integer) : Session
PersonService
+getPersons() : allPersons : Person
+addPerson(person : Person) : void
+deletePerson(person : Person) : void
+updatePerson(person : Person) : void
+getPersonByUserName(userName : String) : Pers...
+getPersonById(id : Integer) : Person
PaperSubjectService
+getPaperSubjects() : allPaperSubjects : PaperSubject
+addPaperSubject(paperSubject : PaperSubject) : void
+deletePaperSubject(paperSubject : PaperSubject) : void
+updatePaperSubject(paperSubject : PaperSubject) : v...
AssignmentService
+getAssignments() : allAssignments : Assignment
+addAssignment(assignment : Assignment) : void
+deleteAssignment(assignment : Assignment) : void
+updateAssignment(assignment : Assignment) : void
PaperService
+getPapers() : allPapers : Paper
+addPaper(paper : Paper) : void
+deletePaper(paper : Paper) : void
+updatePaper(paper : Paper) : void
+getPapersByAuthor(id : Integer) : List<Paper>
+getPapersByKeyword(id : Integer) : List<Paper>
PcMemberService
+getPcMembers() : allPcMembers : PcMember
+addPcMember(pcMember : PcMember) : void
+deletePcMember(pcMember : PcMember) : void
+updatePcMember(pcMember : PcMember) : void
+getPcMemberById(pcId : int) : Optional<PcMem...
+isPcMember(pcId : int) : boolean
KeywordService
+getKeywords() : allKeywords : Keyword
+addKeyword(keyword : Keyword) : void
+deleteKeyword(keyword : Keyword) : void
+updateKeyword(keyword : Keyword) : void
SessionService
+getSessions() : allSessions : Session
+addSession(session : Session) : void
+deleteSession(session : Session) : void
+updateSession(session : Session) : void
+getSessionById(id : Integer) : Session
model
«Entity Bean»
«ORM Persistable»
Person
id : int
username : String
password : String
salutation : String
affiliation : String
firstname : String
lastname : String
phonenumber : String
email : String
«Entity Bean»
«ORM Persistable»
Paper
id : int
session : int
title : String
topic : String
accepted : String
abstractText : String
orientationGrid : String
«Entity Bean»
«ORM Persistable»
PcMember
id : int
«Entity Bean»
«ORM Persistable»
Keyword
id : int
name : String
«Entity Bean»
«ORM Persistable»
Session
id : int
time : Time
hall : int
supervisor : int
«Entity Bean»
«ORM Persistable»
Assignment
id : int
«Entity Bean»
«ORM Persistable»
PaperSubject
id : int
«Entity Bean»
«ORM Persistable»
Review
«Entity Bean»
«ORM Persistable»
Author
«Entity Bean»
«ORM Persistable»
Booking
«Entity Bean»
«ORM Persistable»
Ticket
«Entity Bean»
«ORM Persistable»
Room
«Entity Bean»
«ORM Persistable»
Chair
id : int
hall : int
«Entity Bean»
«ORM Persistable»
Setting
id : int
hall : int
BookingRepository
AuthorRepository
TicketRepository
RoomRepository
ChairRepository
BookingService
AuthorService
TicketService
RoomService
ChairService
Scheduler
+schedule() : String
FileHelper
-fileStorageLocation : Path
-absoluteFileLocation : Path
-relativeFileLocation : Path
+storeFile(file : MultipartFile, location : String) : Bo...
+loadFileAsResource(fileName : String) : Resource
controller
«interface»
PersonController
+getPersons() : List<Person>
+addPerson(person : Person) : void
+updatePerson(person : Person) : void
+deletePerson(id : Integer) : void
+getPersonById(id : Integer) : Person
+getPersonByUserName(userName : String) : Person
PaperController
-paperService : PaperService
-converter : PaperConverter
-fileHelper : FileHelper
+uploadPaper(file : MultipartFile, paperRequest : PaperRequest) : ResponseEntity<String>
+getPapers() : List<PaperResponse>
+getPaperById(id : Integer) : PaperResponse
+getPapersByAuthor(id : Integer) : List<PaperResponse>
+getPapersByKeyword(id : Integer) : List<PaperResponse>
+updatePaper(id : Integer, paperRequest : PaperRequest) : PaperResponse
+deletePaper(id : Integer) : void
+getPaperFile(id : Integer) : ResponseEntity<Resource>
PcMemberController
-pcMemberService : PcMemberService
-converter : PcMemberConverter
+getPcMembers() : List<PcMemberResponse>
+addPcMember(pcMemberRequest : PcMemberRequest) : PcMemberResponse
+updatePcMember(id : Integer, pcMemberRequest : PcMemberRequest) : PcMemberResponse
+deletePcMember(id : Integer) : void
+getPcMemberById(id : Integer) : PcMemberResponse
+isPcMember(id : Integer) : boolean
KeywordController
-keywordService : KeywordService
-converter : KeywordConverter
+getKeywords() : List<KeywordResponse>
+addKeyword(keywordRequest : KeywordRequest) : KeywordResponse
+deleteKeyword(id : Integer) : void
+updateKeyword(id : Integer, keywordRequest : KeywordRequest) : KeywordResponse
SessionController
-sessionService : SessionService
-converter : SessionConverter
+getSessions() : List<SessionResponse>
+addSession(sessionRequest : SessionRequest) : SessionResponse
+deleteSession(id : Integer) : void
+updateSession(id : Integer, sessionRequest : SessionRequest) : SessionResponse
+getSessionById(id : Integer) : SessionResponse
BookingController
-bookingService : BookingService
-converter : BookingConverter
+getBookings() : List<BookingResponse>
+addBooking(bookingRequest : BookingRequest) : BookingResponse
+updateBooking(id : Integer, bookingRequest : BookingRequest) : BookingResponse
+deleteBooking(id : Integer) : void
+getBookingById(id : Integer) : BookingResponse
AuthorController
-authorService : AuthorService
-converter : AuthorConverter
+getAuthors() : List<AuthorResponse>
+addAuthor(authorRequest : AuthorRequest) : AuthorResponse
+updateAuthor(id : Integer, authorRequest : AuthorRequest) : AuthorResponse
+deleteAuthor(id : Integer) : void
+getAuthorById(id : Integer) : AuthorResponse
TicketController
-ticketService : TicketService
-converter : TicketConverter
+getTickets() : List<TicketResponse>
+addTicket(ticketRequest : TicketRequest) : TicketResponse
+updateTicket(id : Integer, ticketRequest : TicketRequest) : TicketResponse
+deleteTicket(id : Integer) : void
+getTicketById(id : Integer) : TicketResponse
RoomController
-roomService : RoomService
-converter : RoomConverter
+getRooms() : List<RoomResponse>
+addRoom(roomRequest : RoomRequest) : RoomResponse
+updateRoom(id : Integer, roomRequest : RoomRequest) : RoomResponse
+deleteRoom(id : Integer) : void
+getRoomById(id : Integer) : RoomResponse
ChairController
-chairService : ChairService
-converter : ChairConverter
+getChairs() : List<ChairResponse>
+addChair(chairRequest : ChairRequest) : ChairResponse
+updateChair(id : Integer, chairRequest : ChairRequest) : ChairResponse
+deleteChair(id : Integer) : void
+getChairById(id : Integer) : ChairResponse
LoginController
-personService : PersonService
-authorService : AuthorService
-pcMemberService : PcMemberService
-sessionService : SessionService
+login(loginRequest : LoginRequest) : String
ConferenceController
-scheduler : Scheduler
-conferenceConference : ConferenceRequest
+getCurrentPhase() : MessageResponse
ChairController
+chairPersonality(cmd : MessageRequest) : MessageResponse
PersonConverter
+toDto(model : Person) : PersonDto
+toModel(dto : PersonDto) : Person
PaperConverter
+toDto(model : Paper) : PaperDto
+toModel(dto : PaperDto) : Paper
PcMemberConverter
+toDto(model : PcMember) : PcMemberDto
+toModel(dto : PcMemberDto) : PcMember
KeywordConverter
+toDto(model : Keyword) : KeywordDto
+toModel(dto : KeywordDto) : Keyword
SessionConverter
+toDto(model : Session) : SessionDto
+toModel(dto : SessionDto) : Session
BookingConverter
+toDto(model : Booking) : BookingDto
+toModel(dto : BookingDto) : Booking
AuthorConverter
+toDto(model : Author) : AuthorDto
+toModel(dto : AuthorDto) : Author
RoomConverter
+toDto(model : Room) : RoomDto
+toModel(dto : RoomDto) : Room
ChairConverter
+toDto(model : Chair) : ChairDto
+toModel(dto : ChairDto) : Chair
AbstractConverter
+convertModelsToDtos(models : Collection<Model>) : Collection<Dto>
+convertDtosToModels(dtos : Collection<Dto>) : Collection<Model>
«interface»
Converter
«interface»
Converter
PersonDto
id : int
username : String
password : String
salutation : String
affiliation : String
firstname : String
lastname : String
phonenumber : String
email : String
academicrank : String
PaperDto
id : int
session : int
title : String
topic : String
accepted : String
abstractText : String
orientationGrid : String
RoomDto
id : int
capacity : int
SessionDto
id : int
time : Time
hall : int
supervisor : int
AuthorDto
id : int
name : String
ChairDto
id : int
hall : int
PcMemberDto
id : int