SAMS 2023 Design Documentation

Team 3

Prathamesh Patwekar
Jagrat Rao
Sushant Borse
Parinay Karande

Overview

The following system is designed for "Software Architecture Mining", and is proposed for an International Software Engineering Conference to be held in June 2023. We developed a web-based paper submission and review system for the workshop. The system will be referred to as "SAM2023" hereafter.

SAM2023 supported the following roles: A submitter who is allowed to submit papers. They may submit multiple papers and multiple versions of the same paper. A submitter can only access the papers submitted by him/her.

A Program Committee Chair (PCC) is responsible for assigning the papers to Program Committee members (PCM). A PCC has access to all content for each of the submissions. Each paper is assigned to three PCMs for review by the PCC.
A PCM can choose an initial list of papers that he/she wants to review from the submissions. Only the title of the paper and author names should be accessible to the PCM.

The PCMs can choose which papers he/she wants to review. He/She reviews the assigned artifacts and rates the artifacts. A PCC is then able to generate reports on the reviewed papers. A PCM has content access only to the artifact that is assigned to him/her.

A last role which is an administrator is responsible for setting up review and notification templates, setting up paper submission deadlines, notifications and reminders, and for managing user accounts (PCC, PCM and Submitter/Author). An administrator has access to all the content in SAM2023.

Requirements

The following list of diagrams attempt to capture our interpretation of the requirements and behavior of the system described in the overview. The highlighted use case will be the submitter submitting a paper. Highlighted means that this use was modeled, implemented and tested. The rest of the use cases were partially implemented, their behavior and models are still described.

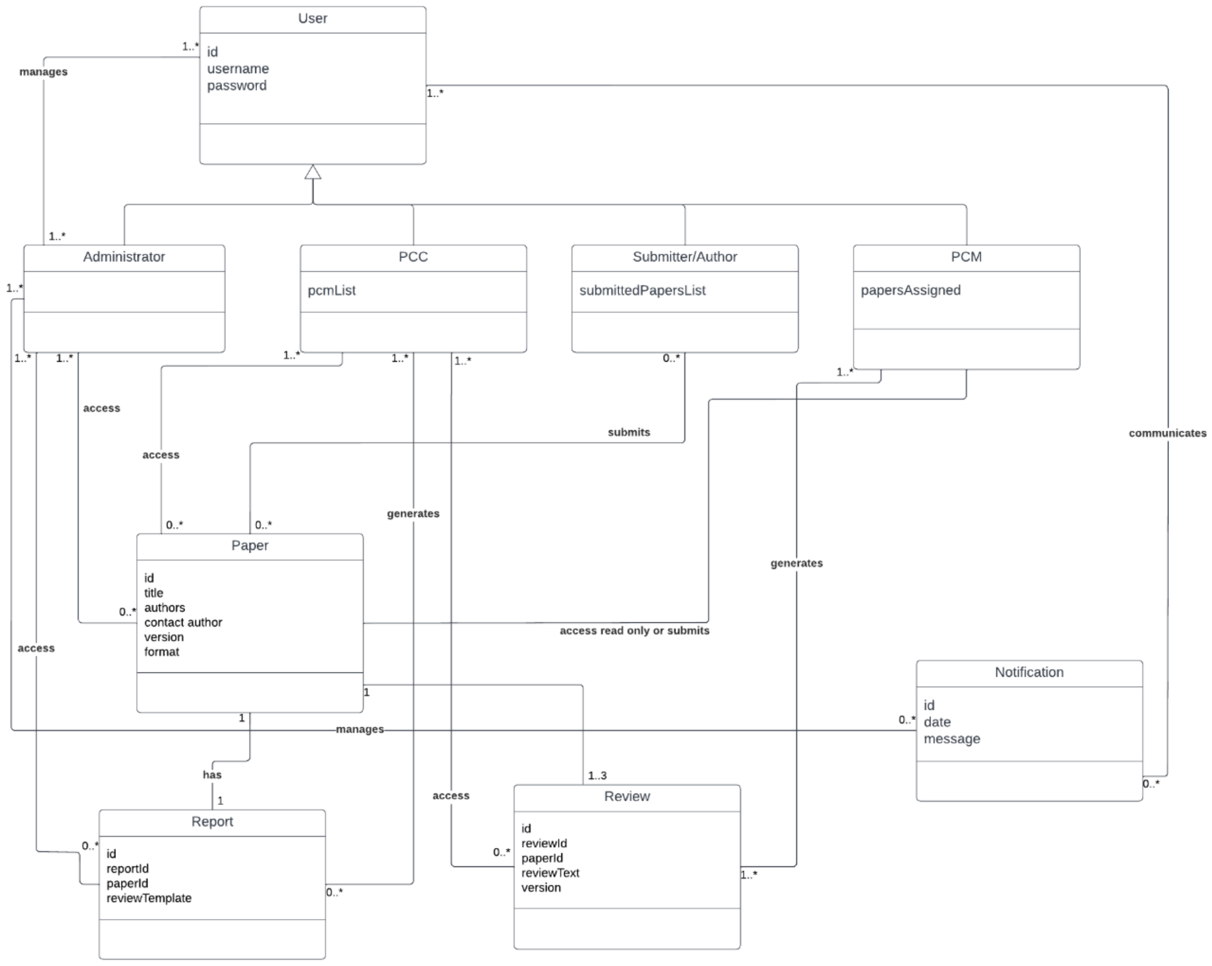


Figure 1 Domain model for the system.

As seen in figure 1 the domain model attempts to capture a high level view of the main entities for the system and how they intend to communicate with each other. Figure 2 further decomposes the system by providing all possible use cases in a use diagram. Figure 2 is the first step in describing the behavior of the system.

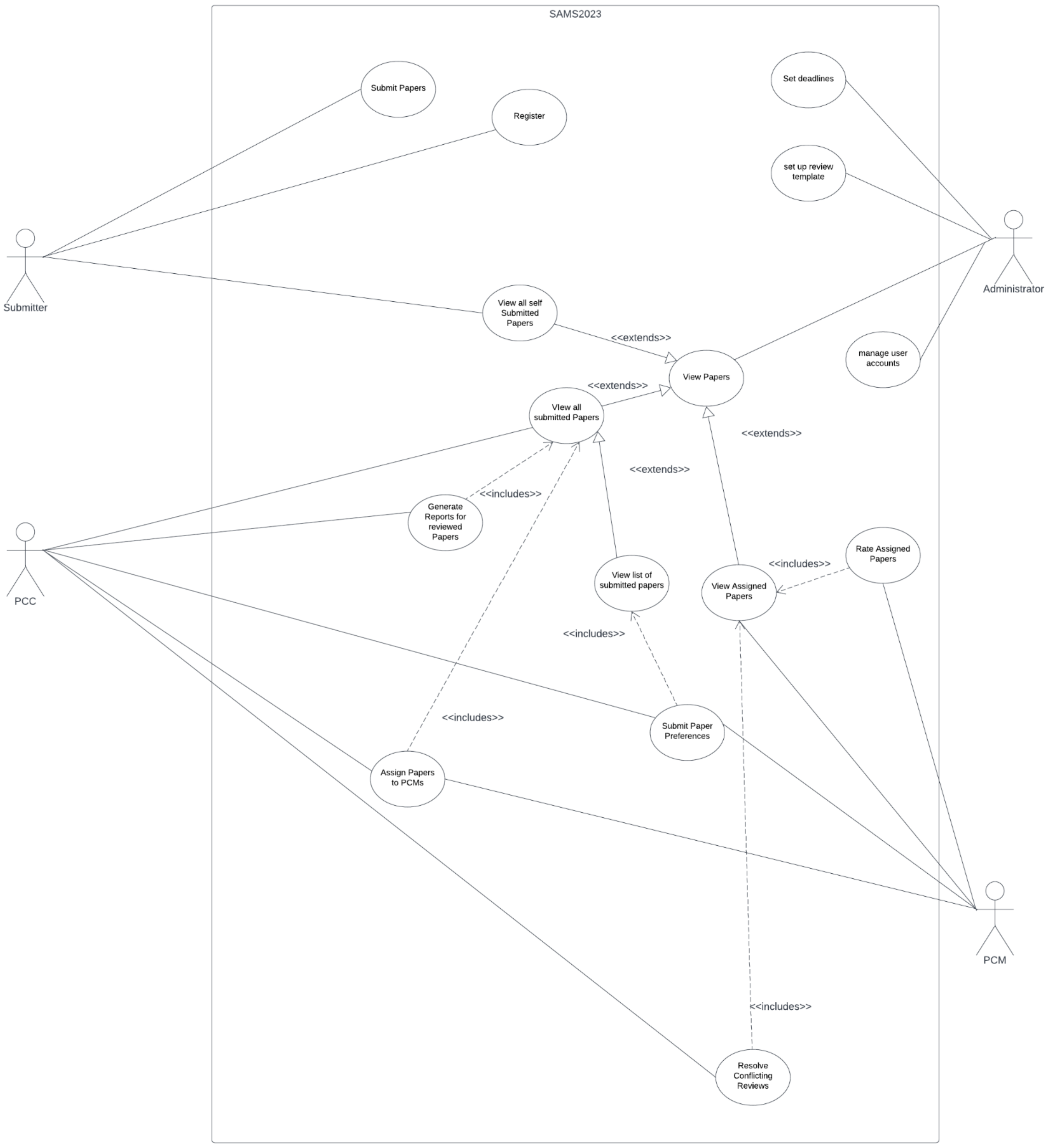


Figure 2 Use Case diagram for the system.

The next series of diagrams further describe the main use cases of the system by illustrating them in system sequence diagrams and detailed use case descriptions. For this iteration of the system we are mainly concerned with the happy day scenarios.

Highlighted Use case - Submitter submits paper

Presented below is the system sequence diagram for our highlighted Use case. The diagram took input both before and after implementation to get an accurate representation of the system behavior. This use case is also meant to help introduce the system behavior as we explore the rest of the system.

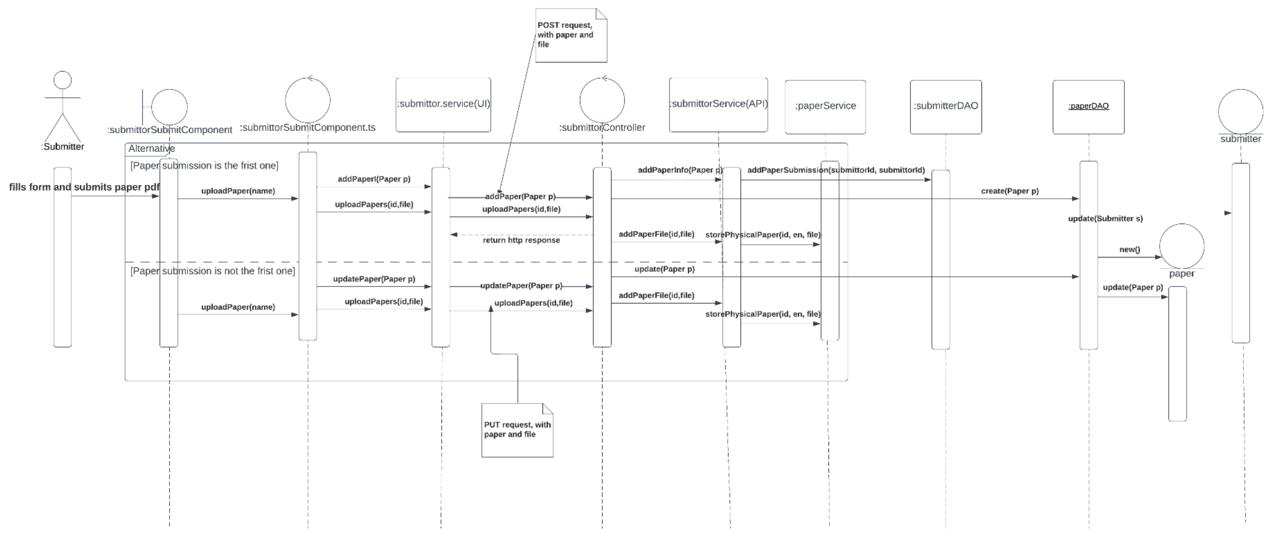


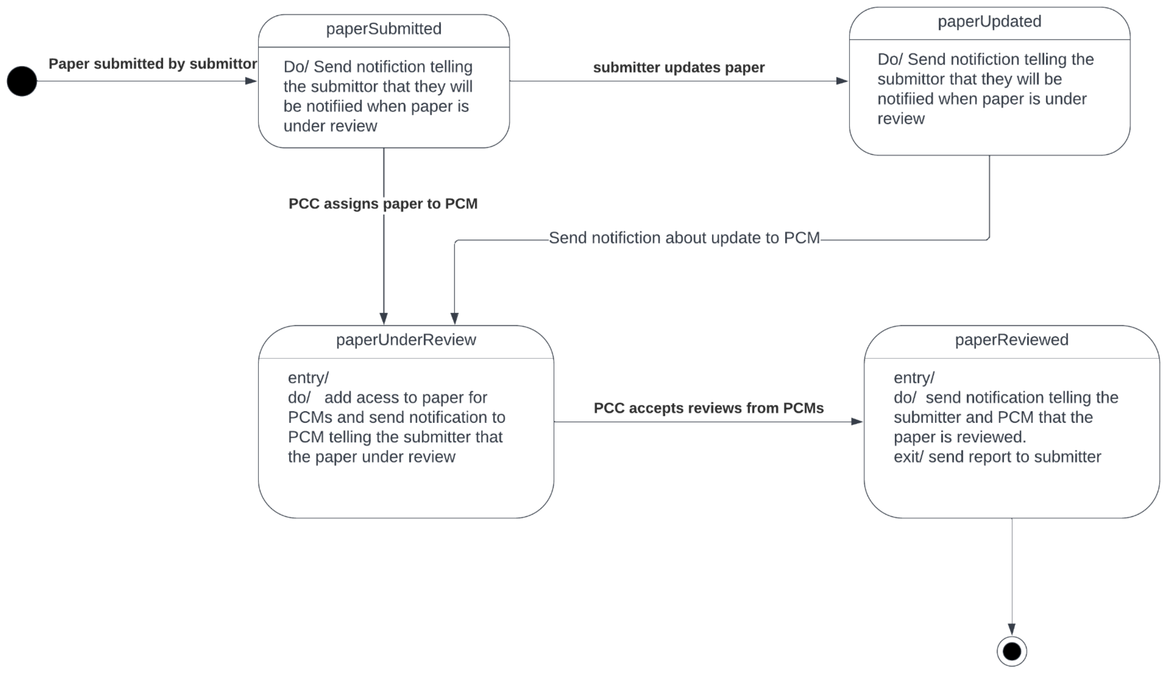
Figure 3 sequence diagram for submitter submitting a paper (hyperlink to lucidchart attached).

Use Case Number	1
Use Case Name	Submit papers
Overview	Submittor/Author submits papers in the submitter portal.
Type	Primary
Actors	Submittor
Pre-condition	Submitter is logged in
Main flow	1. SAM2023 displays the submit paper UI. 2. Submittor selects on the Submit a Paper 3. SAM2023 displays a paper information field and an upload document button. 4. Submittor enters the information and uploads the Paper. 5. submittor saves changes. 6. SAM2023 displays the Paper Submission confirmation

Alternate Flows	
Cross Reference	

Figure 4 use case description for submitter submitting a paper.

The next diagram represents the states of a paper to better understand the states through which a paper goes through. It also helps clarify the next few use cases' complexities.



Paper submitted by submittor
paperSubmitted
Do/ Send notifiction telling the submittor that they will be notifiied when paper is under review
submitter updates paper
paperUpdated
Do/ Send notifiction telling the submittor that they will be notifiied when paper is under review
PCC assigns paper to PCM
Send notifiction about update to PCM
paperUnderReview
entry/
do/ add acess to paper for PCMs and send notification to PCM telling the submitter that the paper under review
PCC accepts reviews from PCMs
paperReviewed
entry/
do/ send notification telling the submitter and PCM that the paper is reviewed.
exit/ send report to submitter

Figure 5 State diagram for a paper.

PCC- assigning papers to PCMs

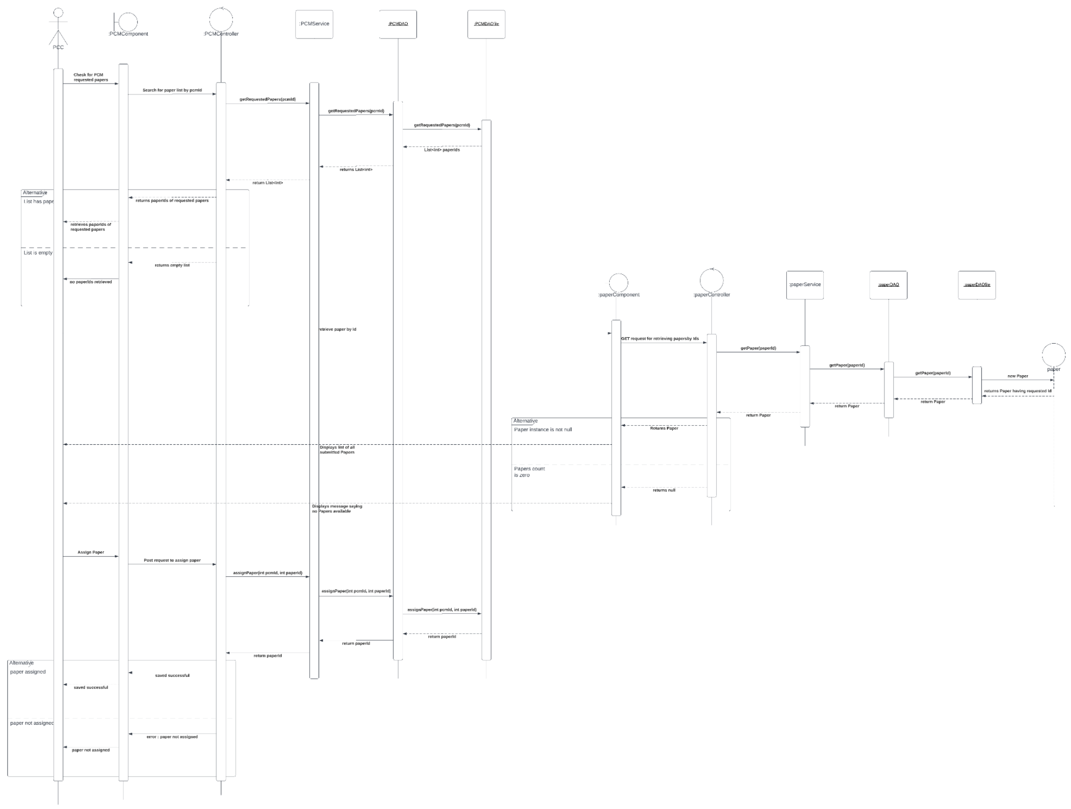


Figure 6 sequence diagram for PCC assigning papers to PCMs.

Use Case Number	2
Use Case Name	Assign Papers to PCMs
Overview	PCC assigns papers to PCMs
Type	Primary
Actors	PCC, PCM
Pre-condition	PCMs request for papers to review from the list of available papers.
Main flow	The PCC checks if any PCM has requested a specific paper to review. If the PCM is not the author of the requested paper, the PCC assigns the requested paper to the PCM for review. The PCC assigns a single paper to not more than 3 PCMs.
Alternate Flows	If any PCMs request their own paper for review, the PCC does not assign the paper for review.
Cross Reference	Submit Paper Preferences

Figure 6 use case description for PCC assigning papers to PCMs.

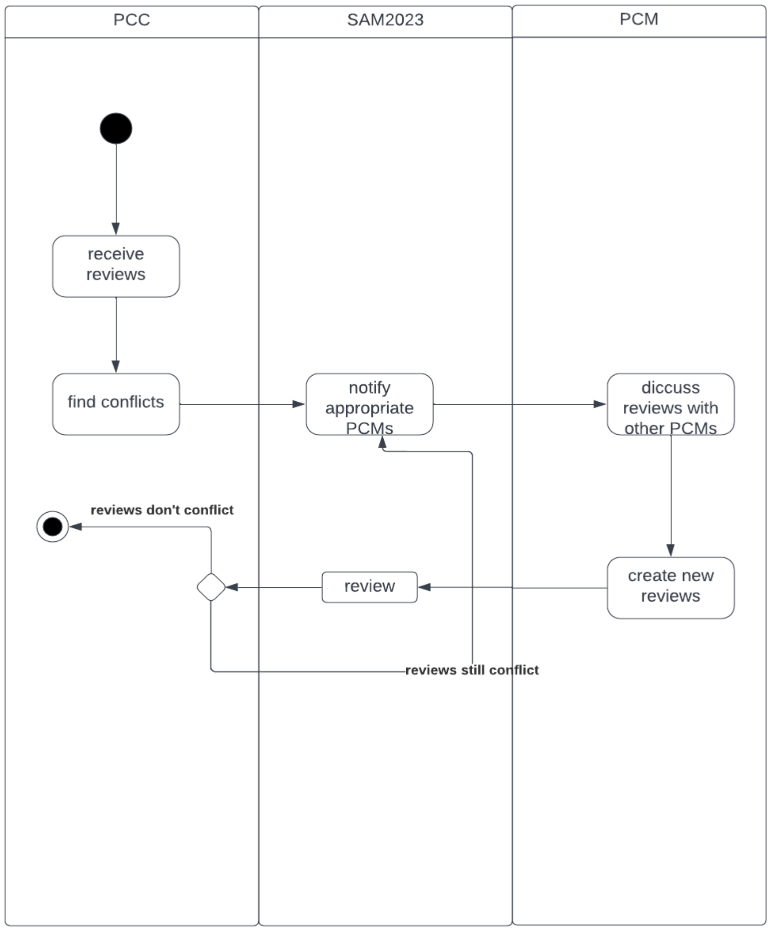


Figure 7 Activity diagram when there are conflicting reviews on a paper

Administrator - setting a template for paper review process

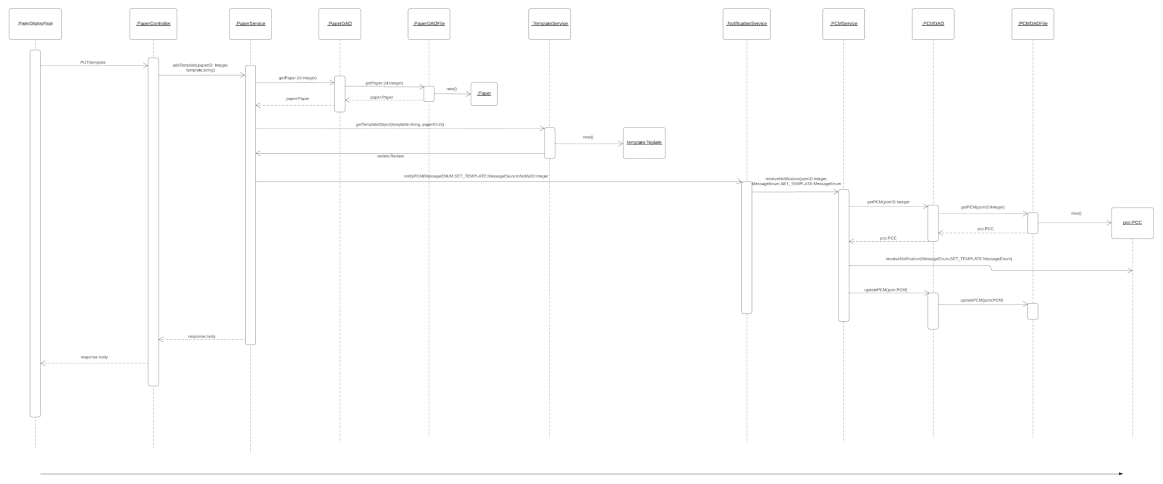


Figure 8 sequence diagram for Administrator for setting a template for paper review process.

Use Case Number	3
Use Case Name	Set paper review template
Overview	Administrator sets template for rate and review of papers.
Type	Primary
Actors	Administrator
Pre-condition	Login
Main flow	Admin clicks on the set template button. SAM2023 displays enter template box. Administrator enters the template information. Administrator enters the submit button. SAM2023 displays a confirmation message..
Alternate Flows	If any PCMs request their own paper for review, the PCC does not assign the paper for review.
Cross Reference	Sets template for review process

Figure 9 use case description for Administrator for setting a template for paper review process

PCM - viewing papers

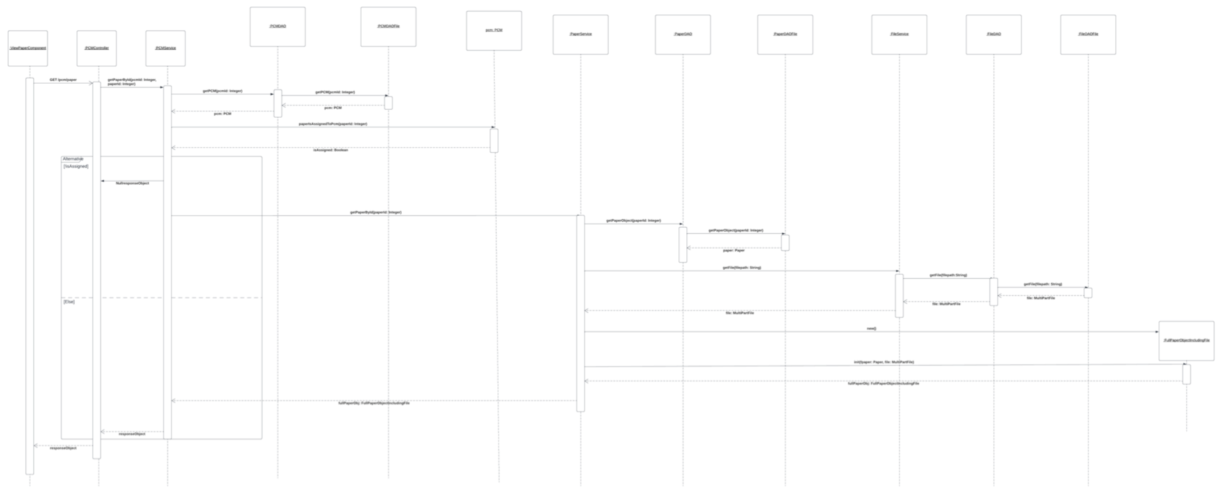


Figure 10 PCM viewing papers

View Papers Use Case

Use Case Number	015
Use Case Name	View Assigned Papers
Overview	PCM views a list of Assigned Papers
Type	Primary
Actors	PCM
Pre-condition	Login
Main flow	1. SAM2023 is displaying the Home Page for PCM 2. PCM clicks on ‘View Assigned Papers’ button 3. If the PCM has been assigned papers, SAM2023 displays the Assigned Papers page
Alternate Flows	3. If the PCM has not been assigned any Papers, SAM2023 displays the Papers Not Assigned Page
Cross Reference	

Figure 11 PCM viewing papers

Architectural Design

The system uses the MVVM architecture using Spring boot and angular framework. The diagram below shows the layers and how they are dependent on each other. The storage is planned to be implemented in a json file and the Persistence layer is described further on.

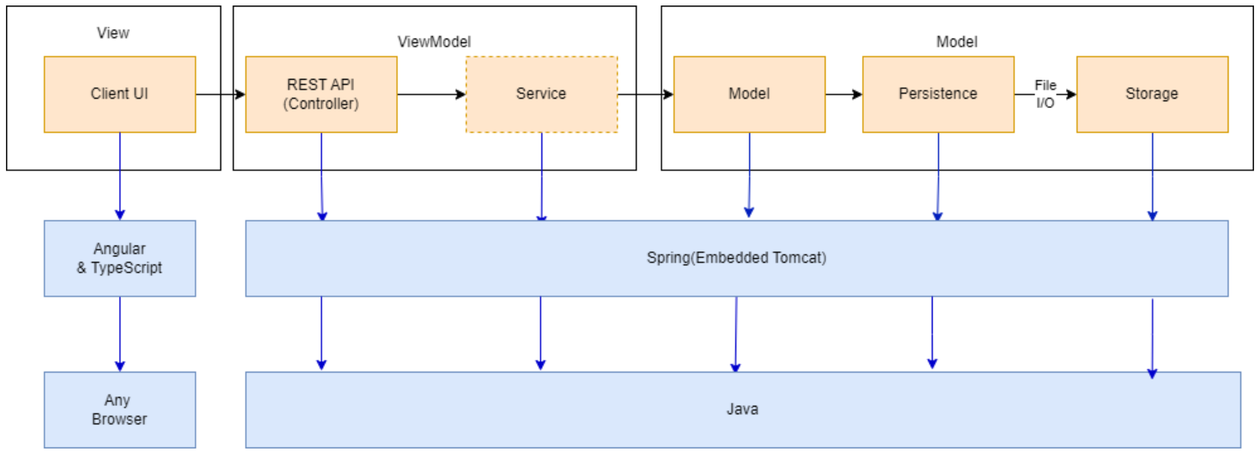


Figure 12 The following is the layer diagram for our system.

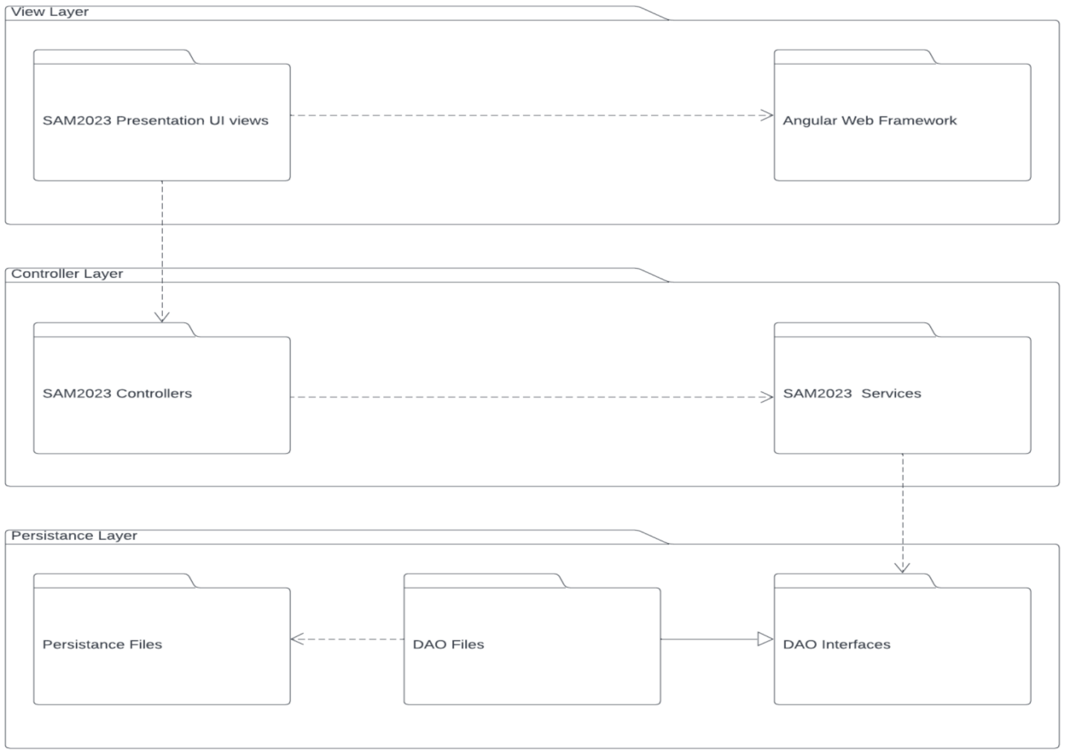


Figure 14 Package diagram for our system

Persistence Classes Design

Figure 15 below shows how each entity in the persistence layer is designed. The notes justify each association and their use. The generic DAO interface allows for CRUD operations. Since this generic interface defines all operations, the EntityDAO allows us to define more complex queries to the data store for specific types of data. The generic CRUD operations in DAO are implemented in AbstractDAOFile. Some of the model classes represent a specific type of user. Due to this, we include user authentication and its implementation in a separate abstract class. All user entity classes extend this class to obtain implementations of CRUD operations as well as user authentication. Finally, the EntityDAOFile implements any newly defined methods in EntityDAO.

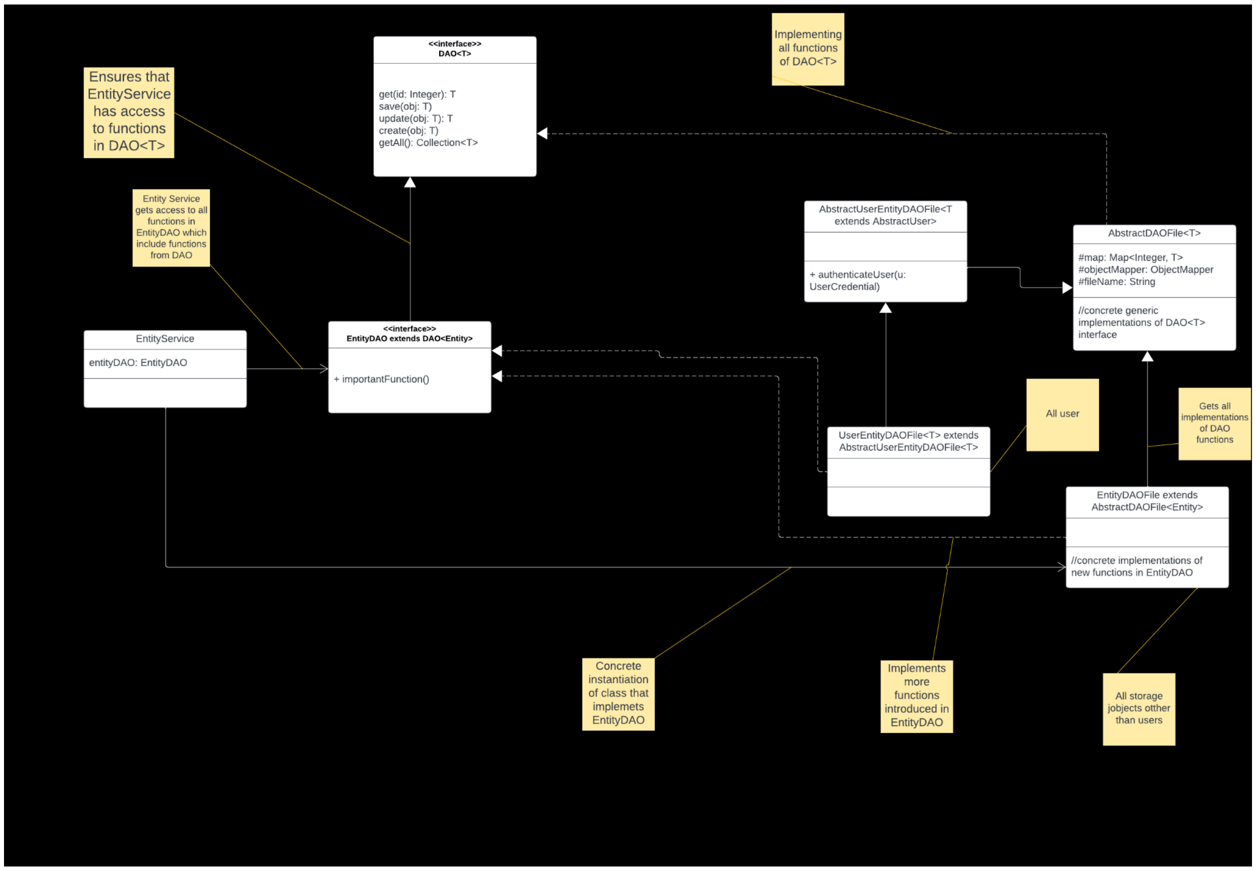

Implementing all functions of DAO<T>
<<interface>> DAO<T>
get(id: Integer): T
save(obj: T)
update(obj: T): T
create(obj: T)
getAll(): Collection<T>
Ensures that EntityService has access to functions in DAO<T>
Entity Service gets access to all functions in EntityDAO which include functions from DAO
AbstractUserEntityDAOFile<T extends AbstractUser>
+ authenticateUser(u: UserCredential)
AbstractDAOFile<T>
#map: Map<Integer, T>
#objectMapper: ObjectMapper
#fileName: String
//concrete generic implementations of DAO<T> interface
EntityService
entityDAO: EntityDAO
<<interface>> EntityDAO extends DAO<Entity>
+ importantFunction()
All user
Gets all implementations of DAO functions
UserEntityDAOFile<T> extends AbstractUserEntityDAOFile<T>
EntityDAOFile extends AbstractDAOFile<Entity>
//concrete implementations of new functions in EntityDAO
Concrete instantiation of class that implemets EntityDAO
Implements more functions introduced in EntityDAO
All storage jobjects otther than users

Figure 15 persistence layer

Class diagram

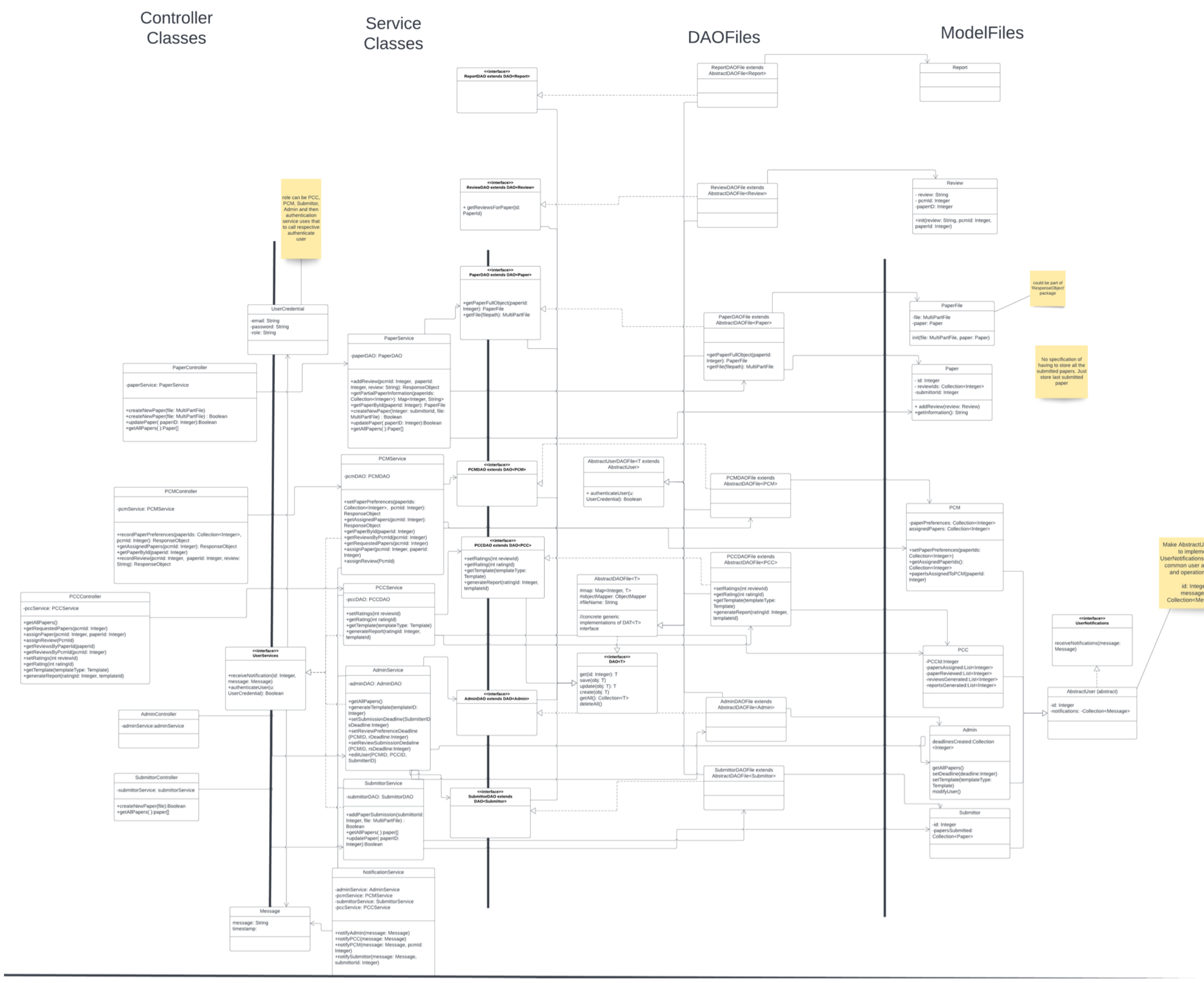


Figure 16 class diagram

Figure 16 covers all classes and their associations among each other. The project is structured in 5 separate layers namely: Controller (routers for HTTP request), Service classes (implements business logic of the project), DAO files (Implementation of concrete methods in the DAO interfaces), Model files (Consists of all concrete entities in the project)

Highlighted use case class diagram

Submittor subsystem

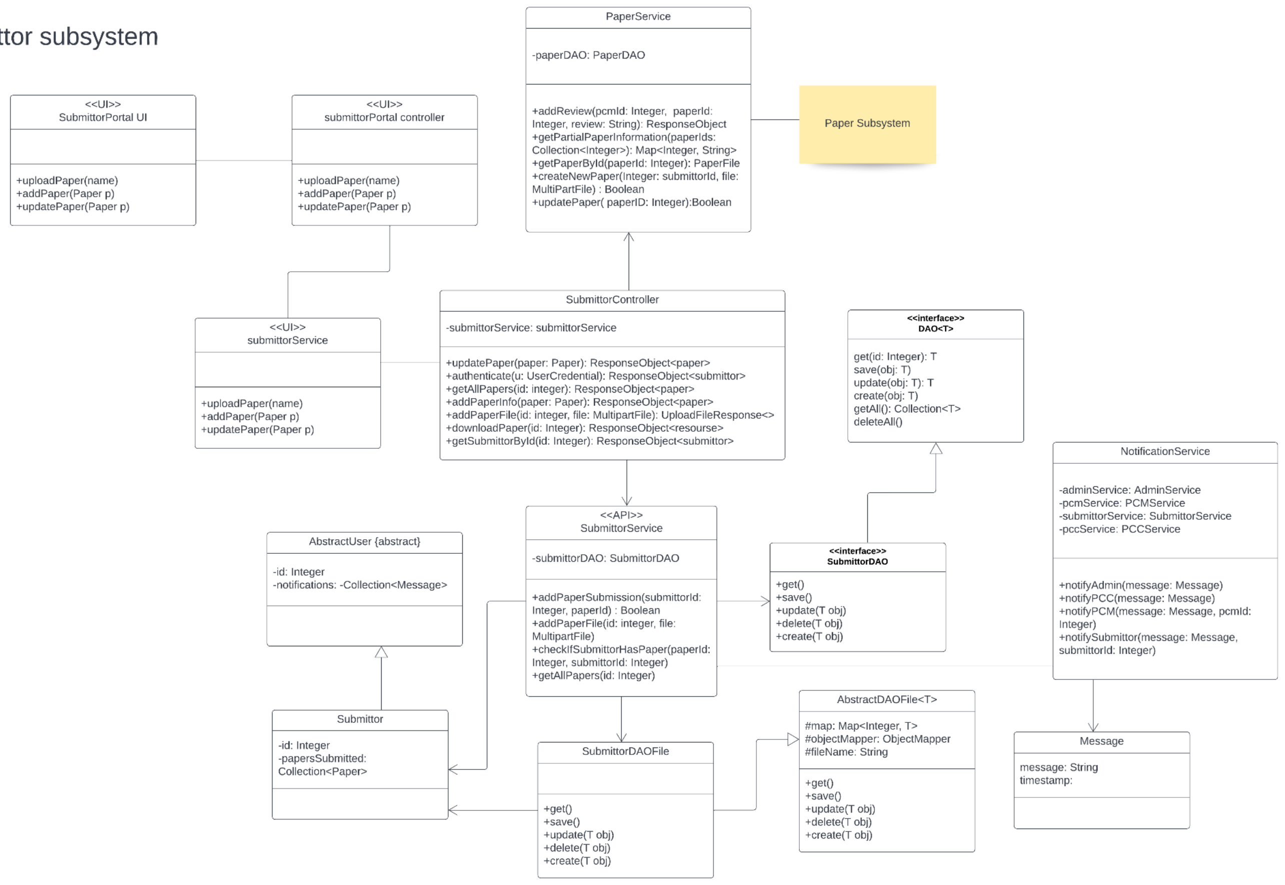

PaperService
-paperDAO: PaperDAO
+addReview(pcmId: Integer, paperId: Integer, review: String): ResponseObject
+getPartialPaperInformation(paperIds: Collection<Integer>): Map<Integer, String>
+getPaperById(paperId: Integer): PaperFile
+createNewPaper(Integer: submittorId, file: MultiPartFile) : Boolean
+updatePaper(paperID: Integer):Boolean
Paper Subsystem
<<UI>>
SubmittorPortal UI
+uploadPaper(name)
+addPaper(Paper p)
+updatePaper(Paper p)
<<UI>>
submittorPortal controller
+uploadPaper(name)
+addPaper(Paper p)
+updatePaper(Paper p)
<<UI>>
submittorService
+uploadPaper(name)
+addPaper(Paper p)
+updatePaper(Paper p)
SubmittorController
-submittorService: submittorService
+updatePaper(paper: Paper): ResponseObject<paper>
+authenticate(u: UserCredential): ResponseObject<submittor>
+getAllPapers(id: integer): ResponseObject<paper>
+addPaperInfo(paper: Paper): ResponseObject<paper>
+addPaperFile(id: integer, file: MultipartFile): UploadFileResponse<>
+downloadPaper(id: Integer): ResponseObject<resourse>
+getSubmittorById(id: Integer): ResponseObject<submittor>
<<interface>>
DAO<T>
get(id: Integer): T
save(obj: T)
update(obj: T): T
create(obj: T)
getAll(): Collection<T>
deleteAll()
NotificationService
-adminService: AdminService
-pcmService: PCMService
-submittorService: SubmittorService
-pccService: PCCService
+notifyAdmin(message: Message)
+notifyPCC(message: Message)
+notifyPCM(message: Message, pcmId: Integer)
+notifySubmittor(message: Message, submittorId: Integer)
AbstractUser {abstract}
-id: Integer
-notifications: -Collection<Message>
<<API>>
SubmittorService
-submittorDAO: SubmittorDAO
+addPaperSubmission(submittorId: Integer, paperId) : Boolean
+addPaperFile(id: integer, file: MultipartFile)
+checkIfSubmittorHasPaper(paperId: Integer, submittorId: Integer)
+getAllPapers(id: Integer)
<<interface>>
SubmittorDAO
+get()
+save()
+update(T obj)
+delete(T obj)
+create(T obj)
Submittor
-id: Integer
-papersSubmitted: Collection<Paper>
SubmittorDAOFile
+get()
+save()
+update(T obj)
+delete(T obj)
+create(T obj)
AbstractDAOFile<T>
#map: Map<Integer, T>
#objectMapper: ObjectMapper
#fileName: String
+get()
+save()
+update(T obj)
+delete(T obj)
+create(T obj)
Message
message: String
timestamp:

Figure 17 Highlighted class diagram for submitter use case

Other use case class diagram

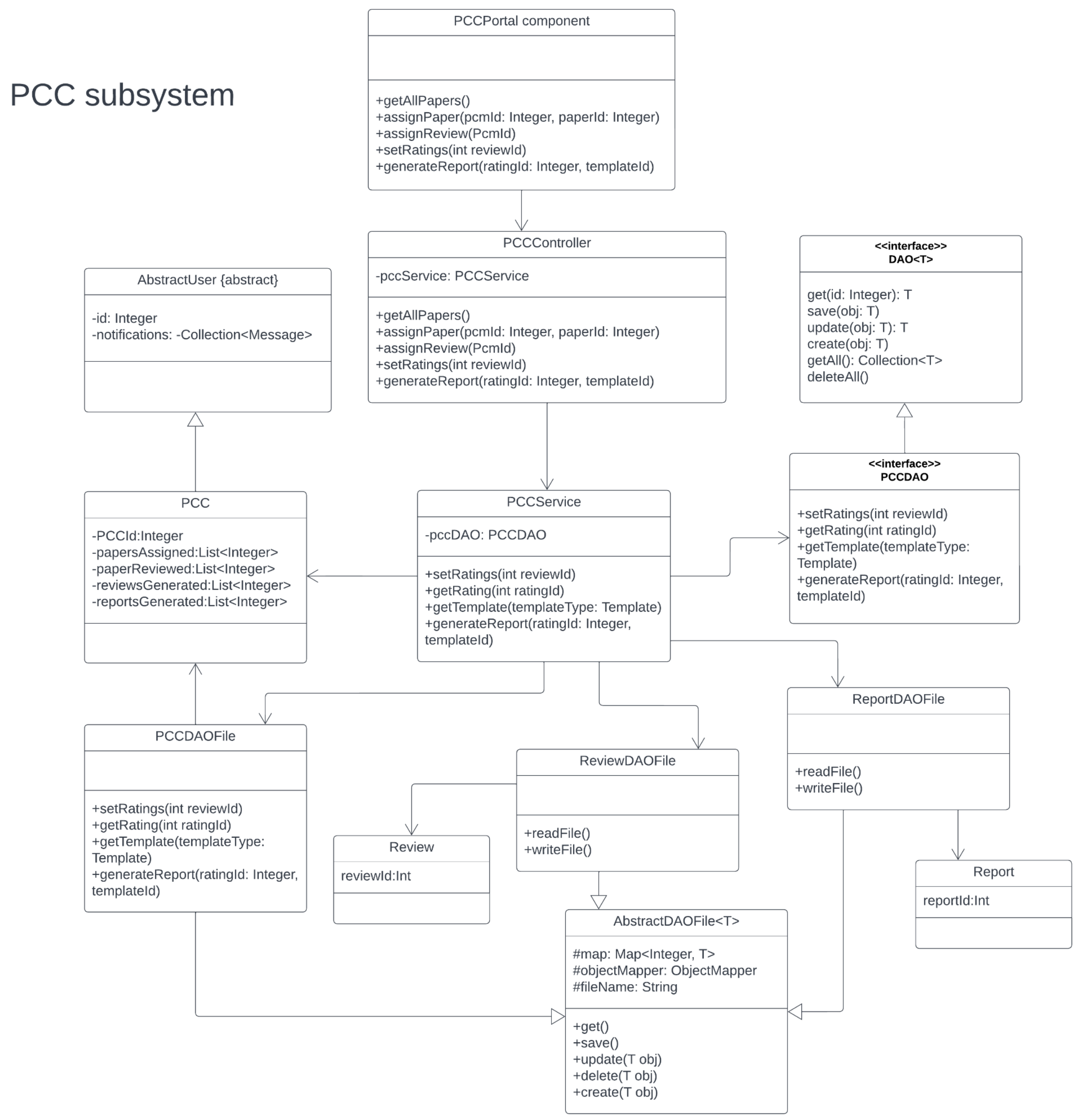

PCC subsystem
PCCPortal component
+getAllPapers()
+assignPaper(pcmId: Integer, paperId: Integer)
+assignReview(PcmId)
+setRatings(int reviewId)
+generateReport(ratingId: Integer, templateId)
PCCController
-pccService: PCCService
+getAllPapers()
+assignPaper(pcmId: Integer, paperId: Integer)
+assignReview(PcmId)
+setRatings(int reviewId)
+generateReport(ratingId: Integer, templateId)
<<interface>> DAO<T>
get(id: Integer): T
save(obj: T)
update(obj: T): T
create(obj: T)
getAll(): Collection<T>
deleteAll()
AbstractUser {abstract}
-id: Integer
-notifications: -Collection<Message>
PCC
-PCCId:Integer
-papersAssigned:List<Integer>
-paperReviewed:List<Integer>
-reviewsGenerated:List<Integer>
-reportsGenerated:List<Integer>
PCCService
-pccDAO: PCCDAO
+setRatings(int reviewId)
+getRating(int ratingId)
+getTemplate(templateType: Template)
+generateReport(ratingId: Integer, templateId)
<<interface>> PCCDAO
+setRatings(int reviewId)
+getRating(int ratingId)
+getTemplate(templateType: Template)
+generateReport(ratingId: Integer, templateId)
ReportDAOFile
+readFile()
+writeFile()
PCCDAOFile
+setRatings(int reviewId)
+getRating(int ratingId)
+getTemplate(templateType: Template)
+generateReport(ratingId: Integer, templateId)
ReviewDAOFile
+readFile()
+writeFile()
Review
reviewId:Int
Report
reportId:Int
AbstractDAOFile<T>
#map: Map<Integer, T>
#objectMapper: ObjectMapper
#fileName: String
+get()
+save()
+update(T obj)
+delete(T obj)
+create(T obj)

Figure 18 Class diagram for PCC subsystem

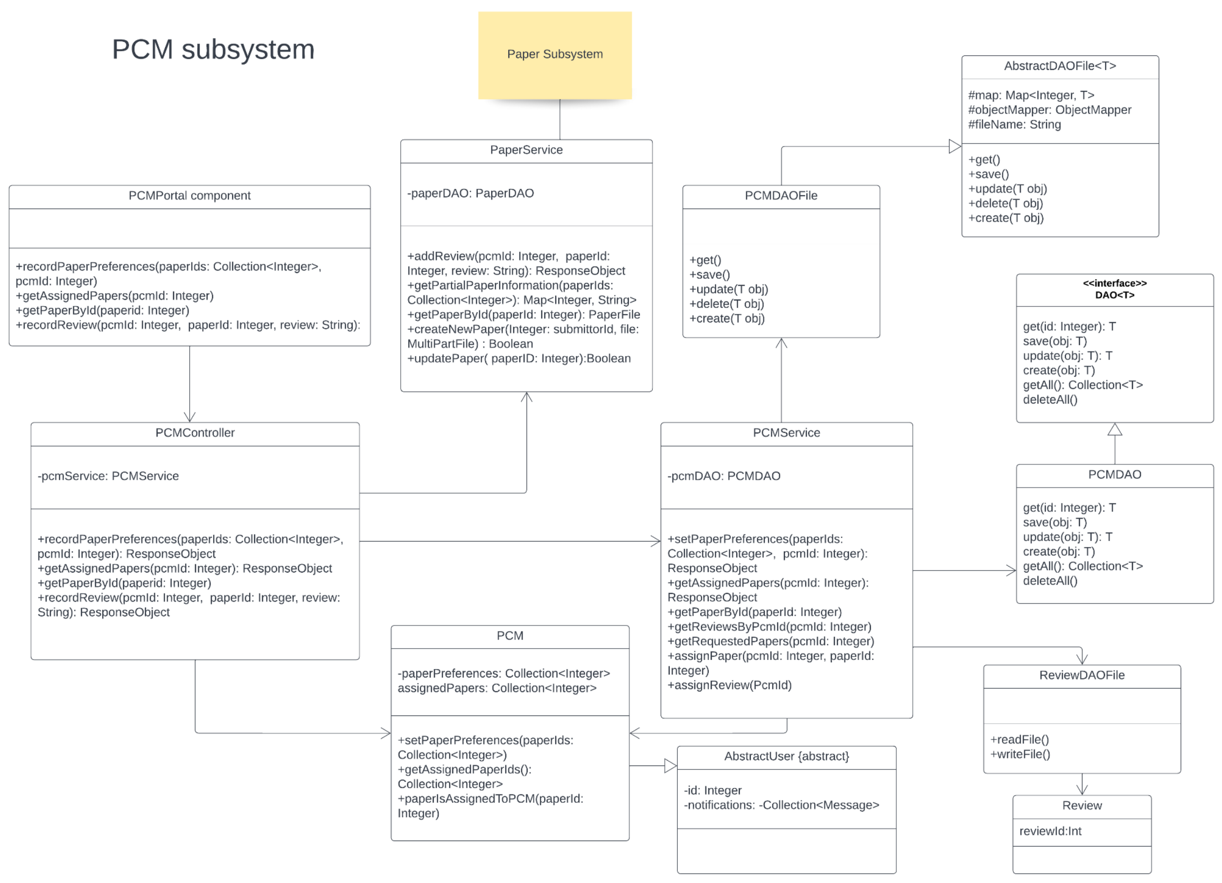

PCM subsystem
Paper Subsystem
PaperService
-paperDAO: PaperDAO
+addReview(pcmId: Integer, paperId: Integer, review: String): ResponseObject
+getPartialPaperInformation(paperIds: Collection<Integer>): Map<Integer, String>
+getPaperById(paperId: Integer): PaperFile
+createNewPaper(Integer: submittorId, file: MultiPartFile) : Boolean
+updatePaper(paperID: Integer):Boolean
AbstractDAOFile<T>
#map: Map<Integer, T>
#objectMapper: ObjectMapper
#fileName: String
+get()
+save()
+update(T obj)
+delete(T obj)
+create(T obj)
PCMDAOFile
+get()
+save()
+update(T obj)
+delete(T obj)
+create(T obj)
PCMPortal component
+recordPaperPreferences(paperIds: Collection<Integer>, pcmId: Integer)
+getAssignedPapers(pcmId: Integer)
+getPaperById(paperid: Integer)
+recordReview(pcmId: Integer, paperId: Integer, review: String):
<<interface>>
DAO<T>
get(id: Integer): T
save(obj: T)
update(obj: T): T
create(obj: T)
getAll(): Collection<T>
deleteAll()
PCMController
-pcmService: PCMService
+recordPaperPreferences(paperIds: Collection<Integer>, pcmId: Integer): ResponseObject
+getAssignedPapers(pcmId: Integer): ResponseObject
+getPaperById(paperid: Integer)
+recordReview(pcmId: Integer, paperId: Integer, review: String): ResponseObject
PCMService
-pcmDAO: PCMDAO
+setPaperPreferences(paperIds: Collection<Integer>, pcmId: Integer): ResponseObject
+getAssignedPapers(pcmId: Integer): ResponseObject
+getPaperById(paperId: Integer)
+getReviewsByPcmId(pcmId: Integer)
+getRequestedPapers(pcmId: Integer)
+assignPaper(pcmId: Integer, paperId: Integer)
+assignReview(PcmId)
PCMDAO
get(id: Integer): T
save(obj: T)
update(obj: T): T
create(obj: T)
getAll(): Collection<T>
deleteAll()
PCM
-paperPreferences: Collection<Integer>
assignedPapers: Collection<Integer>
+setPaperPreferences(paperIds: Collection<Integer>)
+getAssignedPaperIds(): Collection<Integer>
+paperIsAssignedToPCM(paperId: Integer)
ReviewDAOFile
+readFile()
+writeFile()
AbstractUser {abstract}
-id: Integer
-notifications: -Collection<Message>
Review
reviewId:Int

Figure 19 Class diagram for PCM subsystem

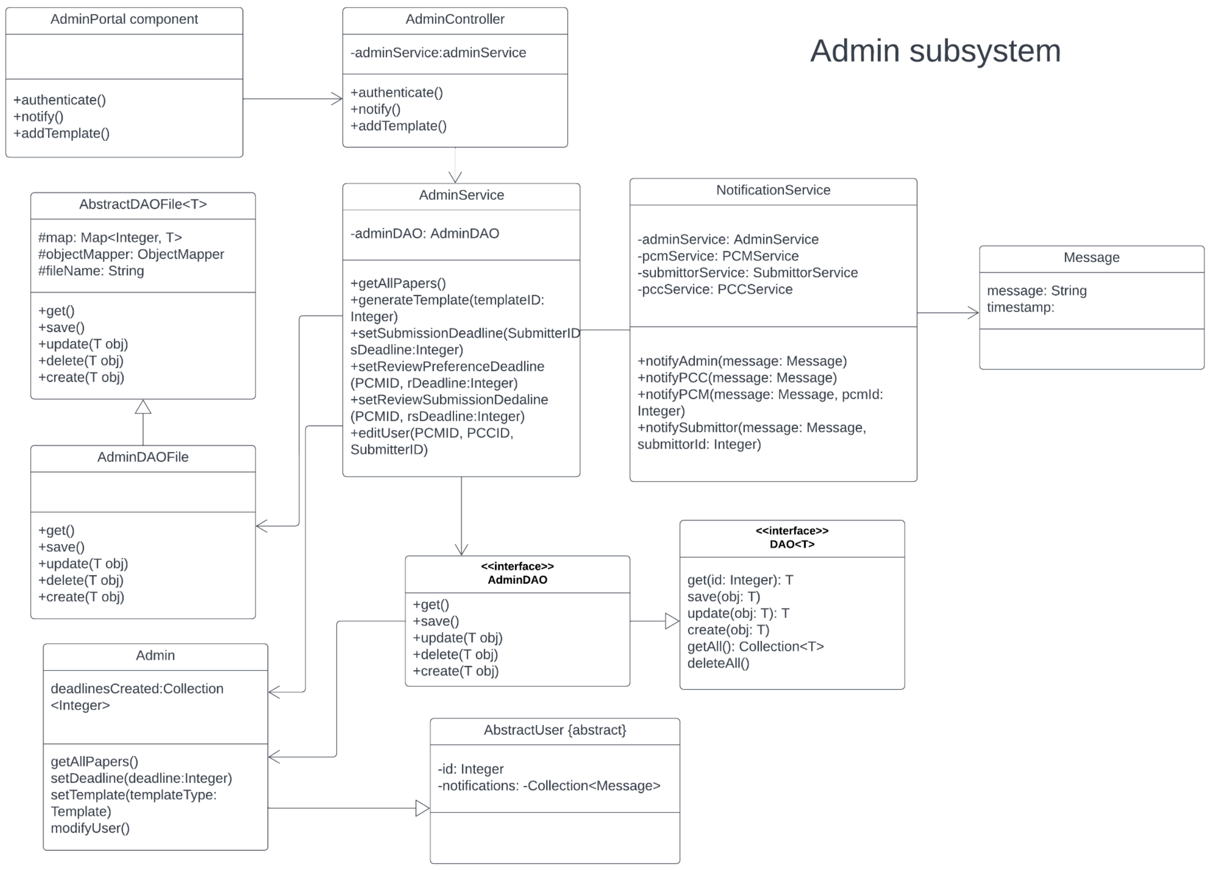


Figure 20 Class diagram for Admin subsystem

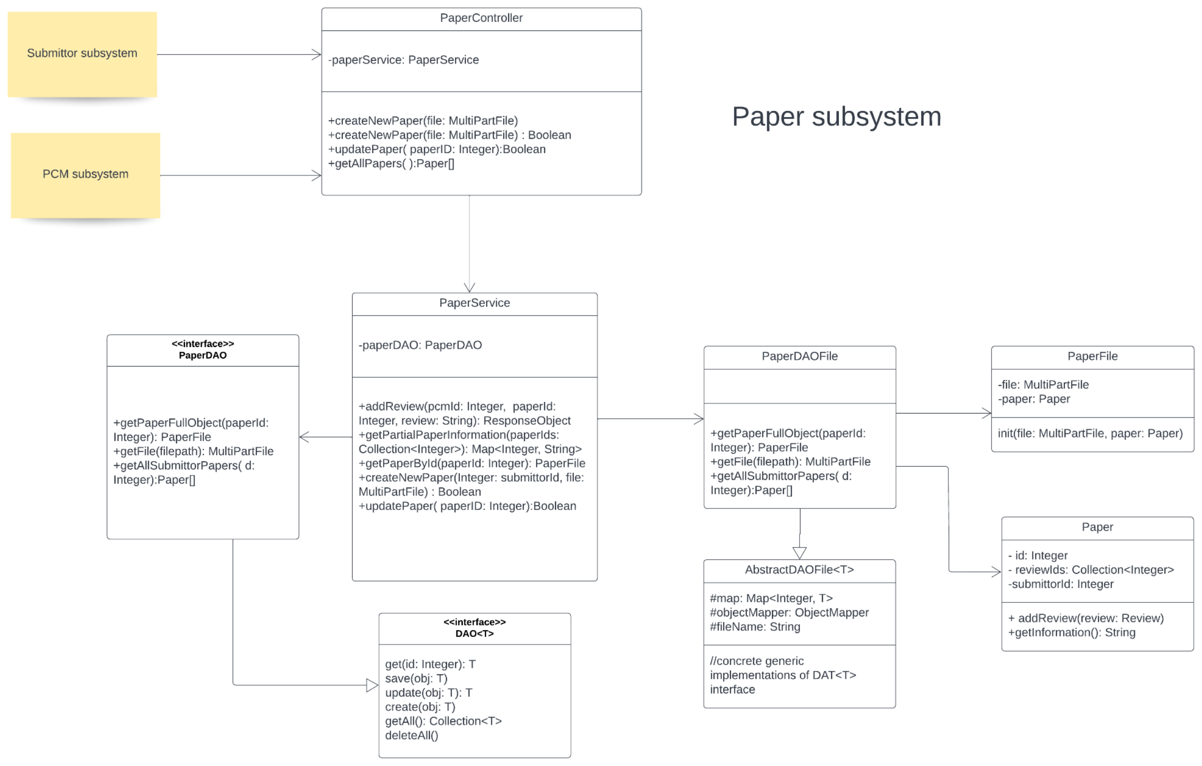

Submittor subsystem
PCM subsystem
PaperController
-paperService: PaperService
+createNewPaper(file: MultiPartFile)
+createNewPaper(file: MultiPartFile) : Boolean
+updatePaper(paperID: Integer):Boolean
+getAllPapers( ):Paper[]
Paper subsystem
PaperService
-paperDAO: PaperDAO
+addReview(pcmId: Integer, paperId: Integer, review: String): ResponseObject
+getPartialPaperInformation(paperIds: Collection<Integer>): Map<Integer, String>
+getPaperById(paperId: Integer): PaperFile
+createNewPaper(Integer: submittorId, file: MultiPartFile) : Boolean
+updatePaper(paperID: Integer):Boolean
<<interface>> PaperDAO
+getPaperFullObject(paperId: Integer): PaperFile
+getFile(filepath): MultiPartFile
+getAllSubmittorPapers(d: Integer):Paper[]
PaperDAOFile
+getPaperFullObject(paperId: Integer): PaperFile
+getFile(filepath): MultiPartFile
+getAllSubmittorPapers(d: Integer):Paper[]
PaperFile
-file: MultiPartFile
-paper: Paper
init(file: MultiPartFile, paper: Paper)
Paper
- id: Integer
- reviewIds: Collection<Integer>
-submittorId: Integer
+ addReview(review: Review)
+getInformation(): String
AbstractDAOFile<T>
#map: Map<Integer, T>
#objectMapper: ObjectMapper
#fileName: String
//concrete generic implementations of DAT<T> interface
<<interface>> DAO<T>
get(id: Integer): T
save(obj: T)
update(obj: T): T
create(obj: T)
getAll(): Collection<T>
deleteAll()

Figure 21 Class diagram for Paper subsystem

Test plan

For testing we plan to use JUnit5 and Mockito to test the MVC classes. The tests will be used to analyze the code coverage. For the code coverage, we plan on using Jacoco to obtain static code coverage and bugs.

For the Submittor, all the methods in the controller layer are tested with 100% test coverage for the SubmittorController. This indicates that the API calls for the Submittor are working correctly and as expected. The tests were performed with mock objects and mock data to verify and validate their behavior. The tests for the SubmittorController can be found in the sam-api/src/test/java/com/example/sam2023/controller/SubmittorController.java file.

Implementation view

The following is the file structure for our src directory in our spring api application, showcasing the structure described in the architectural view and the class objects that are represented in the class diagram.

```
📦src
┣ 📂data
┃ ┗ 📂files
┃ ┃ ┣ 📂jsonfiles
┃ ┃ ┃ ┣ 📜admin.json
┃ ┃ ┃ ┣ 📜papers.json
┃ ┃ ┃ ┣ 📜pcc.json
┃ ┃ ┃ ┣ 📜pcm.json
┃ ┃ ┃ ┣ 📜reports.json
┃ ┃ ┃ ┣ 📜reviews.json
┃ ┃ ┃ ┗ 📜Submittor.json
┃ ┃ ┗ 📂papers
┃ ┃ ┃ ┣ 📂pcm_papers
┃ ┃ ┃ ┃ ┗ 📜Paper.pdf
┃ ┃ ┃ ┗ 📂submittor_papers
┃ ┃ ┃ ┃ ┗ Paper.pdf
┣ 📂main
┃ ┣ 📂java
┃ ┃ ┗ 📂com
┃ ┃ ┃ ┗ 📂example
┃ ┃ ┃ ┃ ┗ 📂sam2023
┃ ┃ ┃ ┃ ┃ ┣ 📂controller
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜AdminController.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜PaperController.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜PCCController.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜PCMController.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜SubmittorController.java
┃ ┃ ┃ ┃ ┃ ┃ ┗ 📜TestController.java
┃ ┃ ┃ ┃ ┃ ┣ 📂model
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜AbstractIdFile.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜AbstractUser.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜Admin.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜Message.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜Paper.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜PaperFile.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜PCC.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜PCM.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜Report.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜Review.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜Submittor.java
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜UserCredential.java
┃ ┃ ┃ ┃ ┃ ┃ ┗ 📜UserNotifications.java
┃ ┃ ┃ ┃ ┃ ┣ 📂persistance
┃ ┃ ┃ ┃ ┃ ┃ ┣ 📂dao
┃ ┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜AdminDAO.java
┃ ┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜DAO.java
┃ ┃ ┃ ┃ ┃ ┃ ┃ ┣ 📜PaperDAO.java
```

```
|  |  |  |  |  |  |  ├📜PCCDAO.java
|  |  |  |  |  |  |  ├📜PCMDAO.java
|  |  |  |  |  |  |  ├📜ReportDAO.java
|  |  |  |  |  |  |  ├📜ReviewDAO.java
|  |  |  |  |  |  |  └📜SubmittorDAO.java
|  |  |  |  |  |  ├📂daoFiles
|  |  |  |  |  |  |  ├📜AbstractDAOFile.java
|  |  |  |  |  |  |  ├📜AbstractUserDAOFile.java
|  |  |  |  |  |  |  ├📜AdminDAOFile.java
|  |  |  |  |  |  |  ├📜PaperDAOFile.java
|  |  |  |  |  |  |  ├📜PCCDAOFile.java
|  |  |  |  |  |  |  ├📜PCMDAOFile.java
|  |  |  |  |  |  |  ├📜ReportDAOFile.java
|  |  |  |  |  |  |  ├📜ReviewDAOFile.java
|  |  |  |  |  |  |  └📜SubmittorDAOFile.java
|  |  |  |  |  |  └📂filestorageSystem
|  |  |  |  |  |  |  ├📜DirectoryEnum.java
|  |  |  |  |  |  |  ├📜FileStorage.java
|  |  |  |  |  |  |  └📜FileStorageFile.java
|  |  |  |  |  ├📂service
|  |  |  |  |  |  ├📜AdminService.java
|  |  |  |  |  |  ├📜NotificationService.java
|  |  |  |  |  |  ├📜PaperService.java
|  |  |  |  |  |  ├📜PCCService.java
|  |  |  |  |  |  ├📜PCMService.java
|  |  |  |  |  |  ├📜SubmittorService.java
|  |  |  |  |  |  └📜UserServices.java
|  |  |  |  |  └📜Sam2023Application.java
|  └📂resources
|  |  └📜application.properties
└📂test
|  └📂java
|  |  └📂com
|  |  |  └📂example
|  |  |  |  └📂sam2023
|  |  |  |  |  └📜Sam2023ApplicationTests.java
```

The next diagram shows screenshots from the web implementation. Starting with login and then going into the submitter portal and then into submitting a paper. Notifications were not yet implemented.

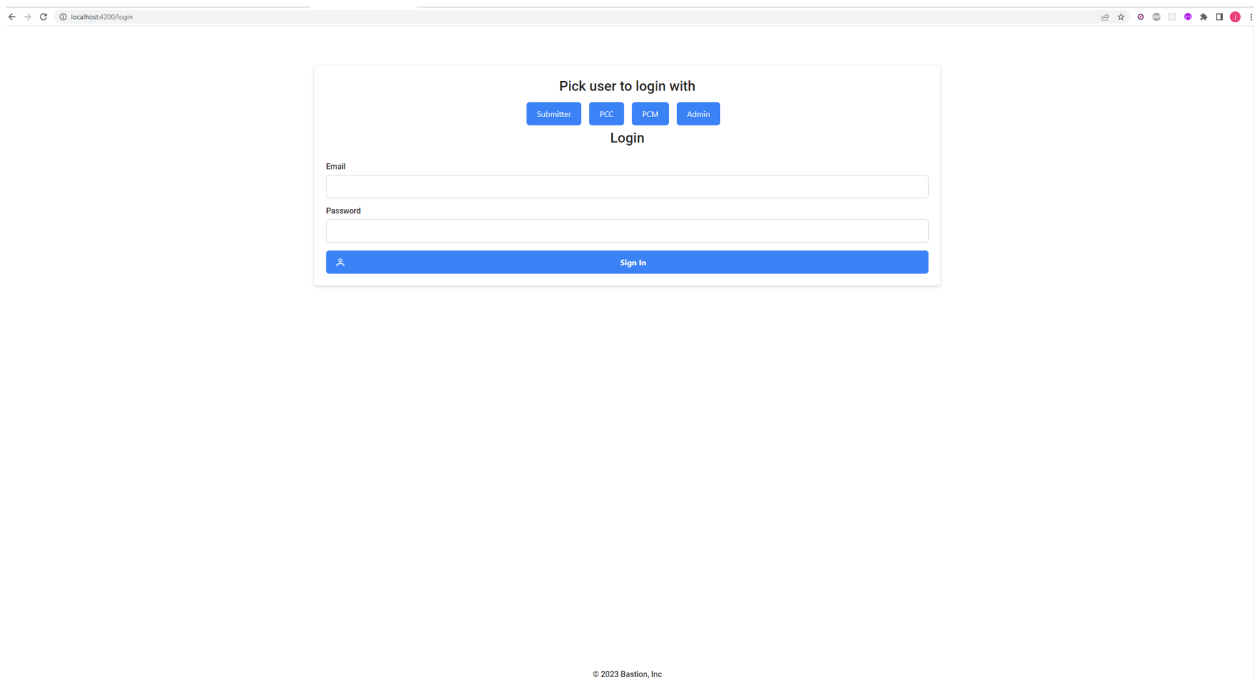
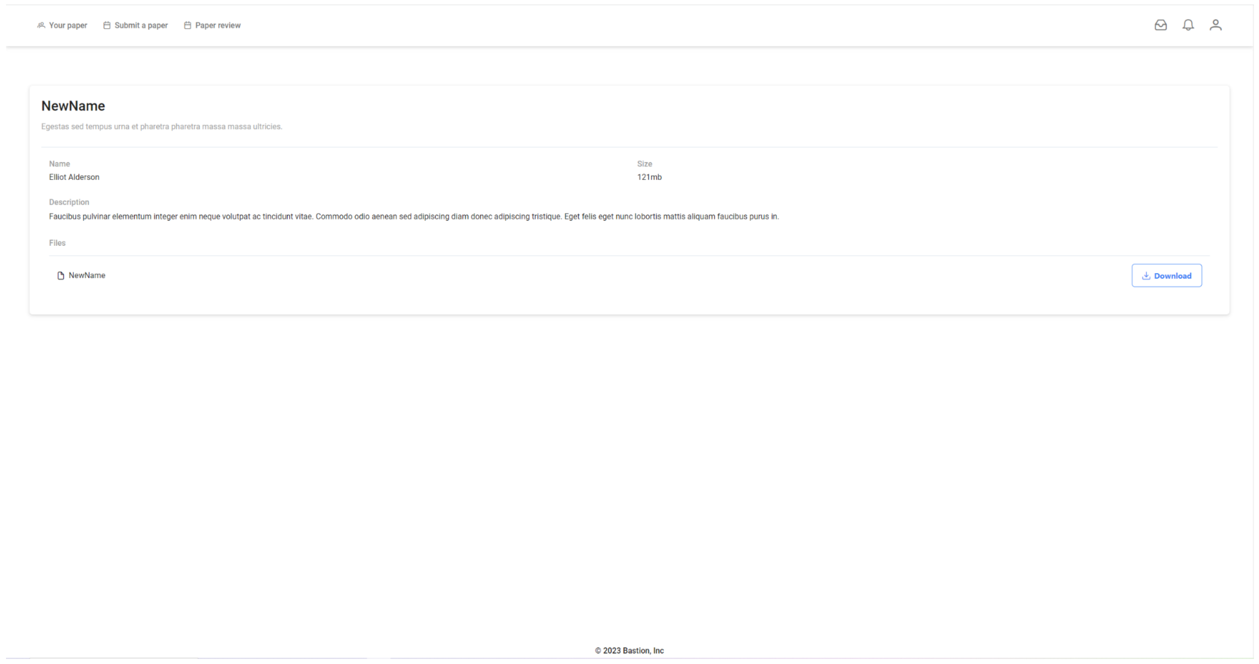


Figure 22 login page



NewName

Egestas sed tempus urna et pharetra pharetra massa massa ultricies.

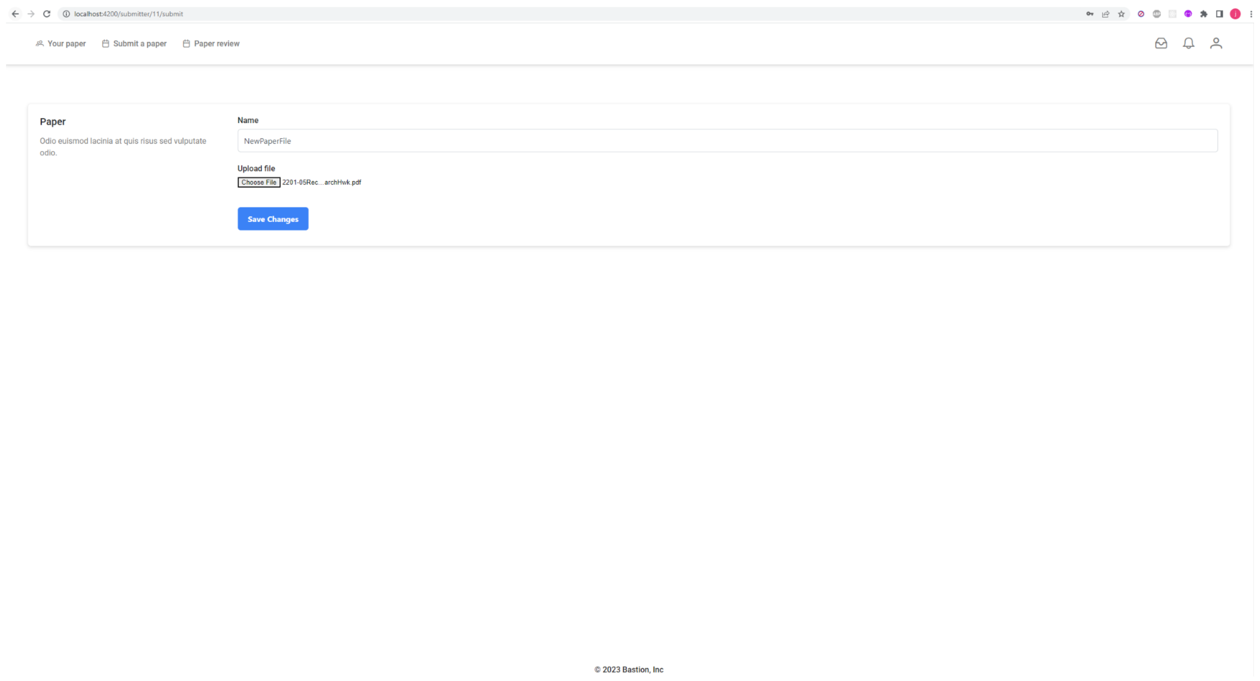
Name
Elliot Alderson

Size
121mb

Description
Faucibus pulvinar elementum integer enim neque volutpat ac tincidunt vitae. Commodo odio aenean sed adipiscing diam donec adipiscing tristique. Eget felis eget nunc lobortis mattis aliquam faucibus purus in.

Files
NewName

Figure 23 submitter portal page



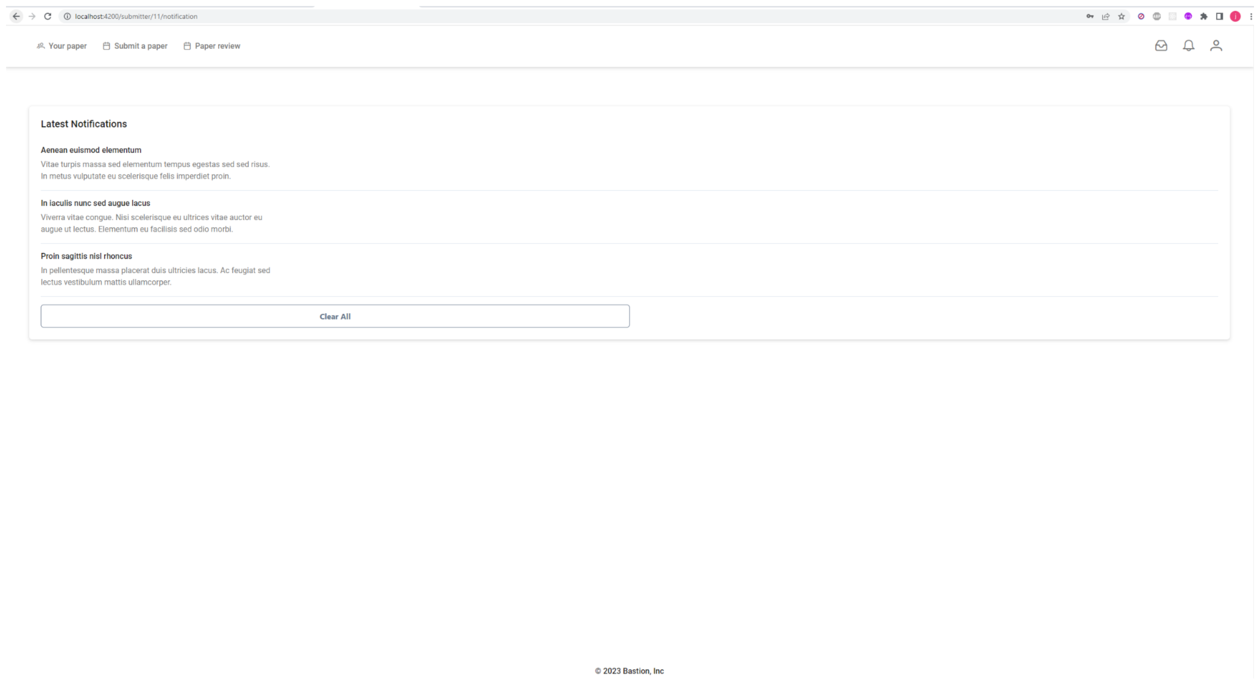
Paper

Odio euismod lacinia at quis risus sed vulputate odio.

Name: NewPaperFile

Upload file: Choose File 2201-05Rec...archHwk.pdf

Figure 24 submit paper page



Latest Notifications

Aenean euismod elementum
Vitae turpis massa sed elementum tempus egestas sed sed risus. In metus vulputate eu scelerisque felis imperdiet proin.

In iaculis nunc sed augue lacus
Viverra vitae congue. Nisi scelerisque eu ultrices vitae auctor eu augue ut lectus. Elementum eu facilisis sed odio morbi.

Proin sagittis nisl rhoncus
In pellentesque massa placerat duis ultricies lacus. Ac feugiat sed lectus vestibulum mattis ullamcorper.

Figure 25 notification page