Software Design

Conference Management System

Professor Manish Singh

Vinta Reethu Havya Sree K Dheekshitha B Akash Tadwai

May 6, 2022

Contents

Chapter 1

Overview

Conference Management System is a web-based application that handles a variety of aspects of conference management, including user registration, conference registration, paper submissions by authors, reviewer registration, papers assignment for reviewers, review submission, conference notifications, paper acceptance, display of paper reviews.

1.1 Data Flow Diagram

DFD provides an overview of how this system processes data and meets various constraints. DFD provides an overview of how this system processes data and meets various constraints.

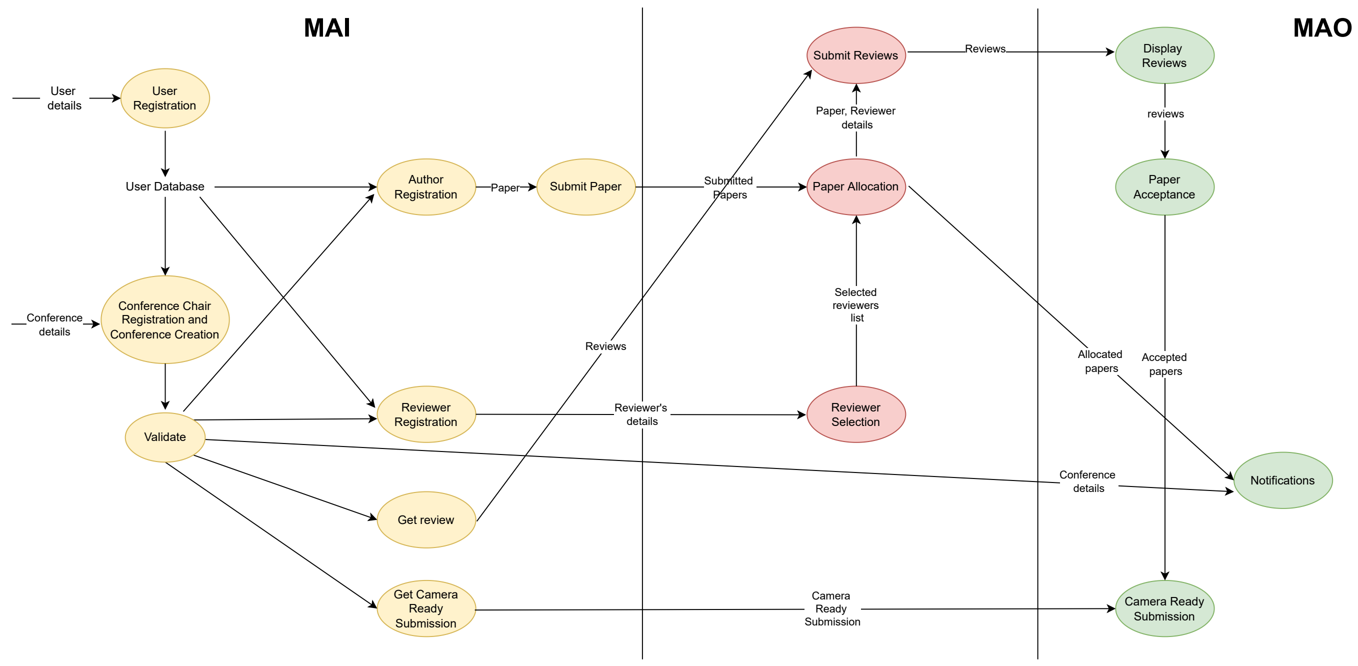

Figure 1.1: Data Flow Diagram

1.2 Structured Chart

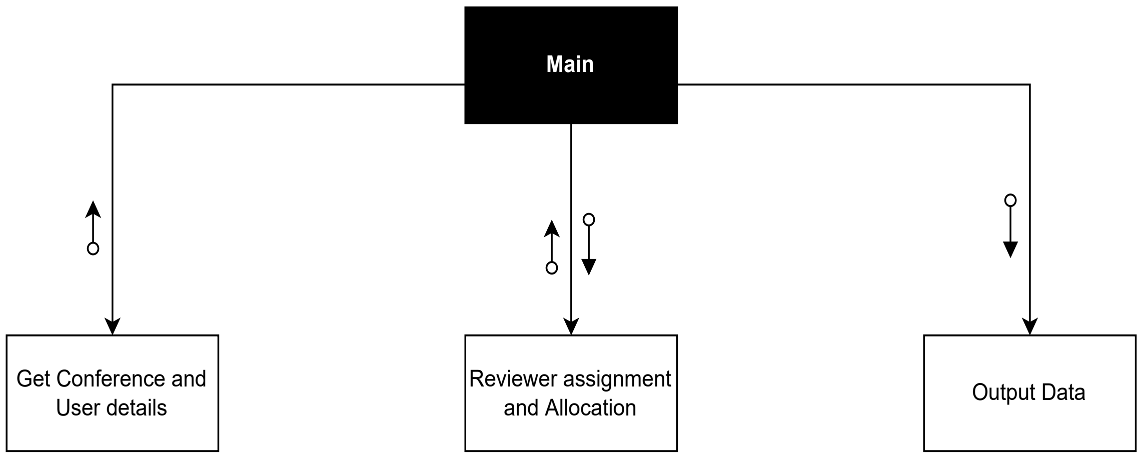
Main
Get Conference and User details
Reviewer assignment and Allocation
Output Data

Figure 1.2: First Level Factoring

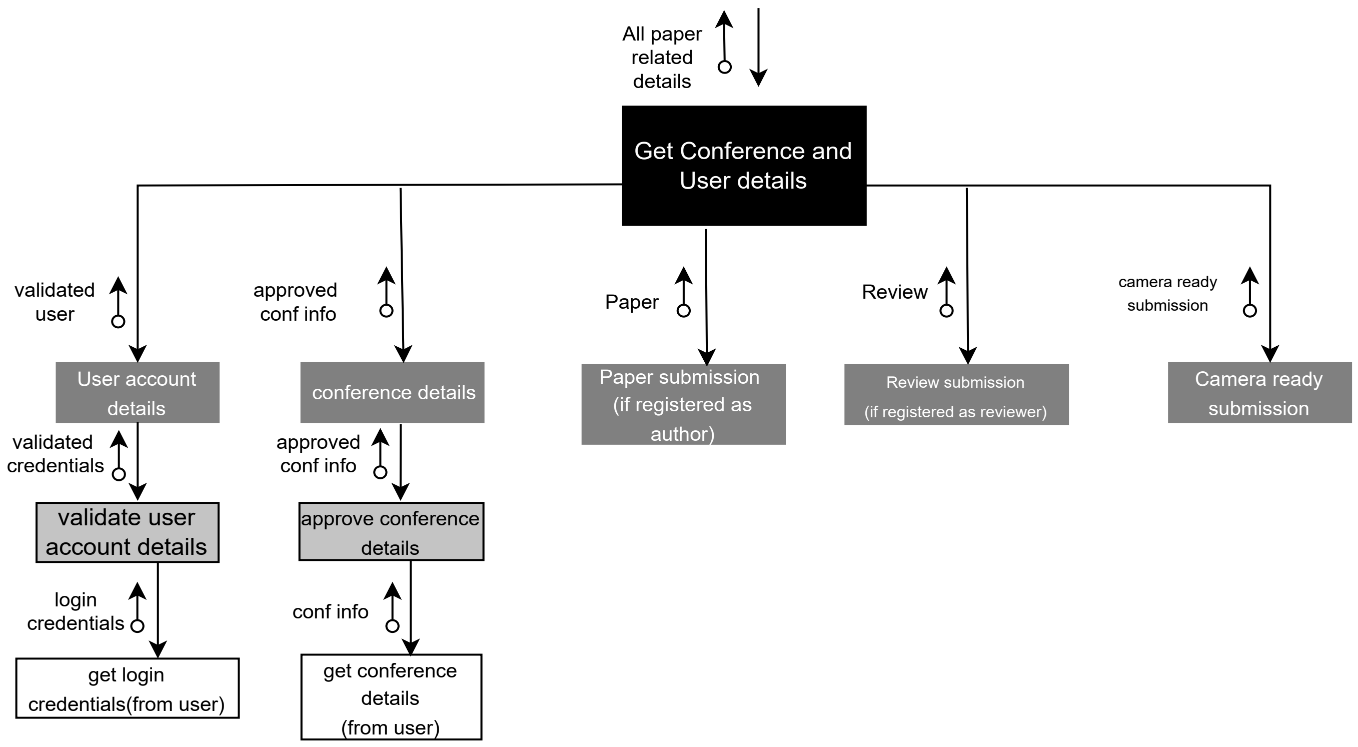
All paper related details
Get Conference and User details
validated user
approved conf info
Paper
Review
camera ready submission
User account details
conference details
Paper submission (if registered as author)
Review submission (if registered as reviewer)
Camera ready submission
validated credentials
approved conf info
validate user account details
approve conference details
login credentials
conf info
get login credentials(from user)
get conference details (from user)

Figure 1.3: Input Factoring

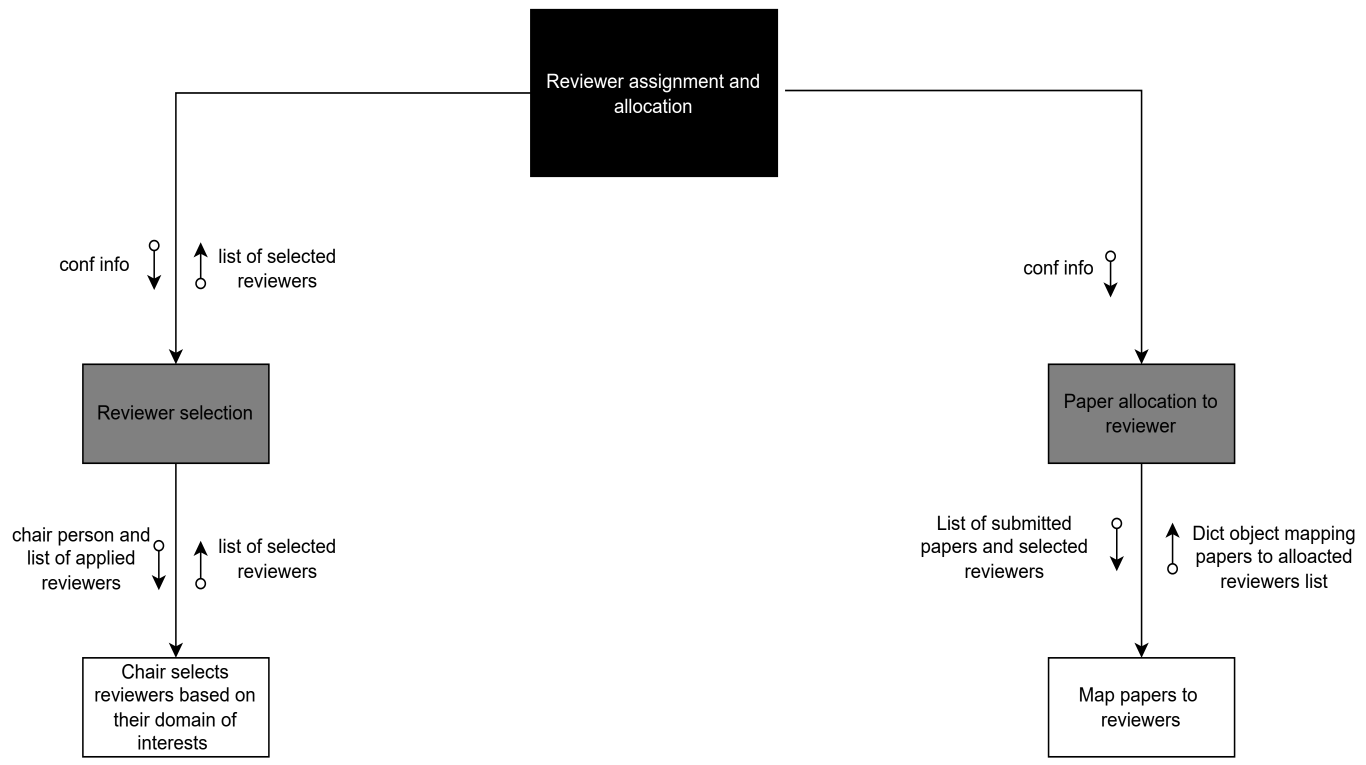
Reviewer assignment and allocation
conf info
list of selected reviewers
conf info
Reviewer selection
Paper allocation to reviewer
chair person and list of applied reviewers
list of selected reviewers
List of submitted papers and selected reviewers
Dict object mapping papers to alloacted reviewers list
Chair selects reviewers based on their domain of interests
Map papers to reviewers

Figure 1.4: Central Transform

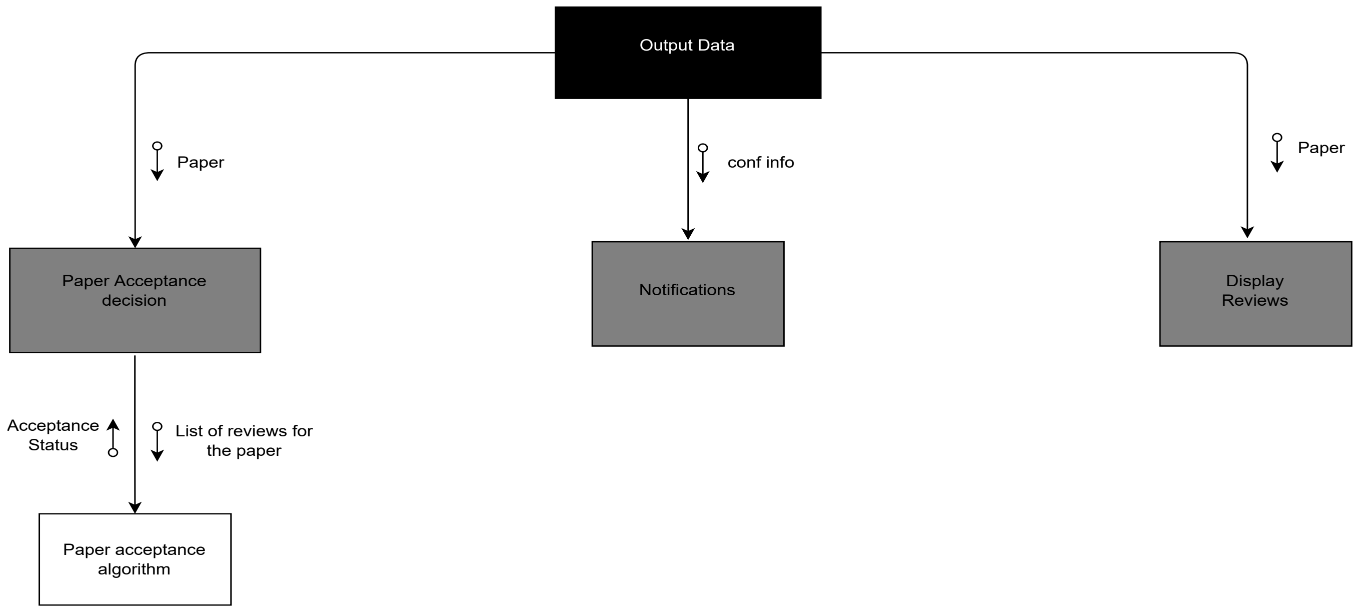
Output Data
Paper
conf info
Paper
Paper Acceptance decision
Notifications
Display Reviews
Acceptance Status
List of reviews for the paper
Paper acceptance algorithm

Figure 1.5: Output Factoring

1.3 Top Three Modules in terms of fan-in and fan-out

1. **Module**: get_conf_and_user_details
   fan-in: 1
   fan-out: 5

2. **Module**: reviewer_assignment_allocation
   fan-in: 1
   fan-out: 2

3. **Module**: output_data
   fan-in: 1
   fan-out: 3

1.4 Complex and Error Prone Modules

1. **get_conf_user_info** is the top-level module in input-factoring. since the number of subordinate modules is 5, the dependency of this module on its subordinates increases. It also makes the flow of information among all the subordinate modules difficult. Thus this is the most complex and error prone module.

2. **output_data**, which is the top-level module in output-factoring, Its fan-out is 3. It is calling different subordinate modules depending on the different outputs of the conference such as acceptance status of the papers, and all the events details for notifying the users. Thus outflow of this module is more as compared to others. Thus this module is complex relative to other modules.

1.5 Summary Table

Sl No.	**Module**	**LOC**
1	main	40
2	get_conf_and_user_details	100
3	user_account_details	10
4	validate_user_account_details	20
5	get_login_credentials	20
6	conference_details	30
7	approve_conference_details	30
8	get_conference_details	50
9	paper_submission	30
10	review_submission	20

11	camera_ready_submission	30
12	reviewer_assignment _allocation	40
13	reviewer_selection	50
14	paper_allocation_to_reviewer	30
15	map_papers_to_reviewers	150
16	output_data	40
17	paper_acceptance_decision	40
18	paper_acceptance_algorithm	70
19	Notifications	200
20	Display_Reviews	50

Table 1.1: Summary Table

Total there are **20** modules. Expected LOC of the software is **1050**. This LOC is for the core code. Determining LOC for the front-end is difficult. Rough estimate for the front-end would be **2000** LOC.

Chapter 2

Interfaces

```
class Contact:
    """
    Represents contact information of every user registered in the system.
    """

    email: str
    phone: str
    portifolio: str


class User:
    """
    Represents a user registered in the system.
    """

    userId: int
    firstName: str
    lastName: str
    hashed_password: str
    title: str
    affiliation: str
    contactInfo: Contact

    def reset_password(self, new_hashed_password: str) -> None:
        # resets password for a user
        self.hashed_password = new_hashed_password

    def getSubmittedPapers(self):
        # returns a list of papers submitted by a user
        query = "SELECT * FROM papers WHERE authorId = self.userId"
        return [Paper(**row) for row in self.db.query(query)]

    def getReviewedPapers(self):
        # returns a list of papers reviewed by a user
        query = "SELECT * FROM papers WHERE authorId = self.userId"
        return [Paper(**row) for row in self.db.query(query)]

    def registerAsAuthor(self, confid):
        # registers a user as an author
        query = "INSERT INTO authors (userId,confid) VALUES (self.userId,confid)"
        self.db.query(query)
        return Author(
            self.userId,
            self.firstName,
            self.lastName,
            self.title,
            self.affiliation,
            self.contactInfo,
            confid,
        )

    def registerAsReviewer(self, confid):
        # registers a user as a reviewer
        query = "INSERT INTO reviewers (userId,confid) VALUES (self.userId,confid)"
        self.db.query(query)
        return Reviewer(
            self.userId,
            self.firstName,
            self.lastName,
            self.title,
            self.affiliation,
            self.contactInfo,
            confid,
        )

    def registerAsChair(self, confid):
        # registers a user as a chair
        query = "INSERT INTO chairs (userId,confid) VALUES (self.userId,confid)"
        self.db.query(query)
        return Chair(
            self.userId,
            self.firstName,
            self.lastName,
            self.title,
            self.affiliation,
            self.contactInfo,
            confid,
        )


class Admin(User):
    """
    Represents an admin in the system.
    """

    def approveConference(self, conf: Conference):
        # approve and add conf to db
        return conf

```

```python

class Chair(User):
    """
    Represents user registered as a chair in a conference
    """

    conferenceId: int

    def selectReviewers(self) -> List[Reviewer]:
        # selects reviewers for a paper
        pass

    def assignReviewers(self) -> List[Reviewer]:
        # assigns reviewers to a paper
        pass

    def sendNotifications(self):
        # sends notifications to reviewers
        pass


class Author(User):
    """
    Represents users who are registered as authors in a conference.
    """

    conferenceId: int

    def submitPaper(Paper):
        # submits a paper for a conference
        pass

    def getSubmittedPapers(self):
        # returns submitted papers for this conference
        query = "SELECT * FROM papers WHERE authorId = self.userId AND conferenceId =
    self.conferenceId"
        return [Paper(**row) for row in self.db.query(query)]


class Reviewer(User):
    """
    Represents users who are registered as reviewers in a conference
    """

    conferenceId: int
    isSelected: bool = False

    def getAllocatedPapers(self):
        # returns a list of papers allocated to a reviewer
        query = "SELECT * FROM papers WHERE reviewerId = self.userId AND conferenceId =
self.conferenceId"
        return [Paper(**row) for row in self.db.query(query)]

    def submitReview(self, paperId):
        # submits a review for a paper
        allocated_papers = self.getAllocatedPapers()
        if paperId in allocated_papers:
            pass

    def getReview(paperid) -> str:
        # returns a review for a paper

        # check if paperid is in allocated papers of the conference
        query = "SELECT * FROM reviews WHERE paperId = paperId AND conferenceId = self.
    conferenceId AND reviewerId = self.userId"
        return query


class Paper:
    """
    Represents a paper submitted in a conference.
    """

    paperId: int
    Title: str
    Authors: List[Author]
    Abstract: str
    KeyWords: str
    Manuscript: str
    isAccepted: bool = False

    def getAllocatedReviewers(self) -> List[Reviewer]:
        # returns a list of reviewers allocated to a paper
        query = "SELECT * FROM reviewers WHERE paperId = self.paperId"
        return [Reviewer(**row) for row in self.db.query(query)]

    def getReviews(self) -> List[str]:
        # returns a list of reviews for a paper
        reviewers = self.getAllocatedReviewers(db)
        return [reviewer.getReview(self.paperId) for reviewer in reviewers]


class AcceptedPaper(Paper):
    """
    Represents a paper submitted in a conference and is accepted.
    """

    DOI: str
    FinalManuscript: str
    Copyright: str

    def submitCamReadySubmission(self):
        # submits a camera ready submission for a paper
        pass


class Deadlines:
    """
    Represents deadlines for a conference.
    """

    date: Date
    message: str
    isAuthor: bool  # if Reviewer False else True


class Conference:
    """
    Represents a conference entity in the system.
    """

    conferenceId: int
    chairPerson: Chair
    conferenceName: str
    conferenceURL: str
    ongoing: bool
    isApproved: bool
    conferenceDeadlines: List[Deadlines]

    def getRegisteredAuthors(self) -> List[int]:
        # returns a list of authors registered in a conference
        authorIds = (
            "SELECT authorId FROM authors WHERE conferenceId = self.conferenceId"
        )
        return authorIds

    def getRegisteredReviewers(db) -> List[int]:
        # returns a list of reviewers registered in a conference
        reviewerIds = (
            "SELECT reviewerId FROM reviewers WHERE conferenceId = self.conferenceId"
        )
        return db.query(reviewerIds)

    def getFinalReviewers(db) -> List[int]:
        # returns a list of reviewers who are selected as final reviewers
        finalReviewerIds = "SELECT finalReviewers FROM reviewers WHERE conferenceId = 
    self.conferenceId AND isSelected = True"
        return db.query(finalReviewerIds)

    def getSubmittedPapers(self) -> List[int]:
        # returns a list of papers submitted in a conference
        submittedPaperIds = (
            "SELECT submittedPapers FROM papers WHERE conferenceId = self.conferenceId"
        )
        return db.query(submittedPaperIds)

    def getAcceptedPapers(self) -> List[int]:
        # returns a list of papers accepted in a conference
        acceptedPaperIds = "SELECT acceptedPapers FROM papers WHERE conferenceId = self
.conferenceId AND isAccepted = True"
        return db.query(acceptedPaperIds)


def Main():
    conf_user_details = get_conf_and_user_details()
    reviewer_assignment_allocation(conf=None)
    output_data()


def get_conf_and_user_details():
    # get conference and user details
    validated_user = user_account_details()
    if is_registered_author(validated_user.userId, confId=None):
        author = Author(
            validated_user.userId,
            validated_user.firstName,
            validated_user.lastName,
            validated_user.title,
            validated_user.affiliation,
            validated_user.contactInfo,
            confId=None,
        )
        paper = paper_submission(author)
    if is_registered_reviewer(validated_user.userId, confId=None):
        reviewer = Reviewer(
            validated_user.userId,
            validated_user.firstName,
            validated_user.lastName,
            validated_user.title,
            validated_user.affiliation,
            validated_user.contactInfo,
            confId=None,
        )
        review = review_submission(reviewer)

    validated_conf_info = conference_details(admin=None)
    cam_ready = camera_ready_submission(acceptedPaper=None)
    return validated_user, validated_conf_info, paper, review, cam_ready


```

```python
def is_registered_author(userId, confId) -> bool:
    # check if user is registered as an author in the conference
    query = "SELECT * FROM authors WHERE userId = {userId} AND conferenceId = {confId}"
    return True if query else False


def is_registered_reviewer(userId, confId) -> bool:
    # check if user is registered as a reviewer in the conference
    query = (
        "SELECT * FROM reviewers WHERE userId = {userId} AND conferenceId = {confId}"
    )
    return True if query else False


def user_account_details() -> User:
    # get user details
    validated_credentials = validate_user_account_details()
    query = "..."
    return User(**query)


def validate_user_account_details():
    # validate user credentials
    credentials = get_login_credentials()
    # validate credentials using db query
    return validated_credentials


def get_login_credentials():
    # get login credentials

    # get credentials from user as input
    login_credentials = input()
    return login_credentials


def conference_details(admin: Admin):
    # get conference details
    return approve_conference_details(admin)


def approve_conference_details(admin: Admin):
    # asks admin to approve the conference
    conf_info = get_conference_details()
    # validate conference details using db query
    return admin.approveConference(conf_info)


def get_conference_details() -> Conference:
    # get conference details from conference object
    conference_info = input()
    return Conference(**conference_info)


def paper_submission(author: Author):
    # get paper details from author and submit paper
    return author.submitPaper(Paper)


def review_submission(review: Reviewer, paper: Paper):
    # get review details from reviewer and submit review
    return review.submitReview(paper.paperId)


def camera_ready_submission(paper: AcceptedPaper):
    # get camera ready details from author and submit camera ready submission
    return paper.submitCamReadySubmission()


def reviewer_assignment_allocation(conf: Conference):
    # get final reviewers and assign reviewers to papers
    selected_reviewers = reviewer_selection(conf.chairPerson)
    reviewer_paper_mapping = paper_allocation_to_reviewer(conf)


def reviewer_selection(chair: Chair):
    # chair selects the final reviewers
    return chair.selectReviewers()


def paper_allocation_to_reviewer(conf: Conference):
    # Algorithm maps papers to reviewers
    mapping = map_papers_to_reviewers(
        conf.getSubmittedPapers(), conf.getFinalReviewers()
    )
    # add mapping details to database


def map_papers_to_reviewers(
    submittedPapers: List[Paper], finalReviewers: List[Reviewer]
):
    # Algorithm maps papers to reviewers
    return {
        paper: reviewers
        for paper, reviewers in preference(submittedPapers, finalReviewers)
    }


def output_data():
    acceptanceStatus = paper_acceptance_decision(paper=None)
    display_reviews(paper=None)
    notifications(conf=None)


def paper_acceptance_decision(paper: Paper):
    # Algorithm decides whether paper is accepted or not based on reviews and other
    factors
    acceptanceStatus = paper_acceptance_algorithm(paper.getReviews())
    # Inserts the decison into the database
    return acceptanceStatus


def paper_acceptance_algorithm(reviews: List[str]):
    # Algorithm decides whether paper is accepted or not based on reviews and other
    factors
    return Union[True, False]


def display_reviews(paper: Paper):
    # Algorithm displays reviews for a paper
    reviews = paper.getReviews()
    # displays reviews on UI using UI framework


def notifications(conf: Conference):
    # Algorithm sends notifications to authors and reviewers

    authors = conf.getRegisteredAuthors()
    reviewers = conf.getFinalReviewers()
    # Algorithm sends notifications to authors and reviewers
    deadlines = conf.conferenceDeadlines()
    for (date, msg, isAuthor) in deadlines:
        if isAuthor:
            schedule_notification(date, msg, authors)
        else:
            schedule_notification(date, msg, reviewers)
```