Software Analysis and Design Specification

for

Meeting Scheduler

Requirements for Version 1.0

Prepared by Shahrzad Masoumnia

University of Calgary

Instructor: Dr. M. Moussavi

Spring 2015

Table of Contents

1. Introduction

1.1 Purpose

This document includes software requirements for Meeting Scheduler, release number 1.0. Meeting Scheduler is a web-based software which organize professional meetings. The system gives resolution to optimum date and location for meetings. Its purpose is to consider all of the important users and most of regular users preference time and location for each meeting. The system is very small so it can be easily transferred from one computer to another. The database produced, is protected by a Master Password only known by its inventor with no backup if lost.

1.2 Intended Audience and Reading Suggestions

This requirement document contains general information about Meeting Scheduler, main classes and use cases, use case diagram, sequence diagram, class diagram, package diagram, deployment diagram, state transition diagram, functions, features and special technologies. It describes in detail all that Meeting Scheduler needs to work properly and with safety.

The rest of the document is divided into chapters for better understanding.

- In chapter 2 an overall description of Meeting Scheduler is provided. First product perspective is presented with product features and main actors. Then follow use cases and use case diagrams will give you more detailed insight about system functionality. At the end of the chapter non-functional requirements are investigated.
- In chapter 3 sequence diagrams for each use case scenario are described.
- In chapter 4 presents class diagrams.
- In chapter 5 is about the package diagram following by chapter 6 about deployment diagram and chapter 7 about state transition diagram.

This document is intended for

Developers: in order to be sure they are developing the right project that fulfills requirements provided in this document.
Testers: in order to have an exact list of the features and functions that have to respond according to requirements and provided diagrams.
Users: in order to get familiar with the idea of the project and suggest other features that would make it even more functional.
Documentation writers: to know what features and in what way they have to explain. How the system will response in each user's action etc.
Advanced end users, end users/desktop and system administrators: in order to know exactly what they have to expect from the system, right inputs and outputs and response in error situations.

1.3 Project Scope

Meeting Scheduler purpose is to solve a problem that really bothers many professionals today when they have to choose a time to attend an important meeting but there would be several emails back and force to find a good time for everyone which can be very frustrating. So this system provides you a very user friendly interface which let all users provide their preference times and locations and exclusion times. Anyone can send a meeting request with suggested time period and locations to specific users. Each user can have different priority ranking (important/non-important) for each meeting that would be assigned by meeting initiator to them.

Meeting Scheduler beside optimum meeting date also provides you with several functionalities in order to keep your meetings database organized and up to date. Those are analyzed in the following pages.

More about Meeting Scheduler you can find out at http://MeetingScheduler.info/

1.4 References

More about Meeting Scheduler can be found at

- *http://github.com/shahrzadmn/MeetingScheduler/*

  In this website you can find out more about the project and discuss any questions in the forums. You can look at code and problems that have been solved. There you can also find information about the developer as well as the project's main characteristics such as programming language and algorithms

- *http://MeetingScheduler.info/*

  This is project's official website where you can find links to all above and also find features available for downloading such as language translations and plug-ins.

2. Use Case Model (detailed version with lots of optional functionality)

First I will give a description of the actors that are going to use the system. Then use case diagrams and use cases text description are shown. The domain model is included in this chapter, and in the end of this chapter is my assumptions.

2.1 Actors Description

2.1.1 Meeting Initiator

Meeting initiator can be any of the company employees who wants to set a meeting. He is responsible to invite all potential participants and assign a value to each participant so they would know if it is mandatory for them to be in that meeting or not. In Addition, he is responsible to set the time frame for the meeting so everybody else should choose the preference and exclusion dates and time periods within that time frame. Location or locations can be suggested at this point and the importance degree of the meeting also will be determined by initiator. He also need to choose his own preference and exclusion dates.

2.1.2 Potential Participant

Potential participants are any employee in that organization who were invited to a meeting. They can accept or decline right away or set their own preference and exclusion date. Their label for participating is non-important which was set by initiator.

2.1.3 Potential Important Participant

These group of participants have to attend the meeting and their exclusion and preference date are more important than other participants. In worst case scenario, meeting scheduler can only set a meeting that important participant can attend.

2.1.4 System Administrator

System administrator is someone who takes care of technical and maintenance issues, checks the system databases and keeps them update.

2.1.5 Users Database

This database contain the users information which can differ from one organization to the other one but it should contain at least their user ID, password, contact information and their past and active meetings.

2.1.6 Locations Database

This database contain list of all locations that meeting can be held there with their room capacity and availability based on dates.

2.1.7 Meetings Database

This database holds meetings information such as its initiator, location, time, participants and importance.

2.2 Use Case Diagrams

To make the use case diagram easier to read, I break it down to partial use case diagrams at summary or user goal details level.

2.2.1 Partial Use Case diagram for Meeting Scheduler Application

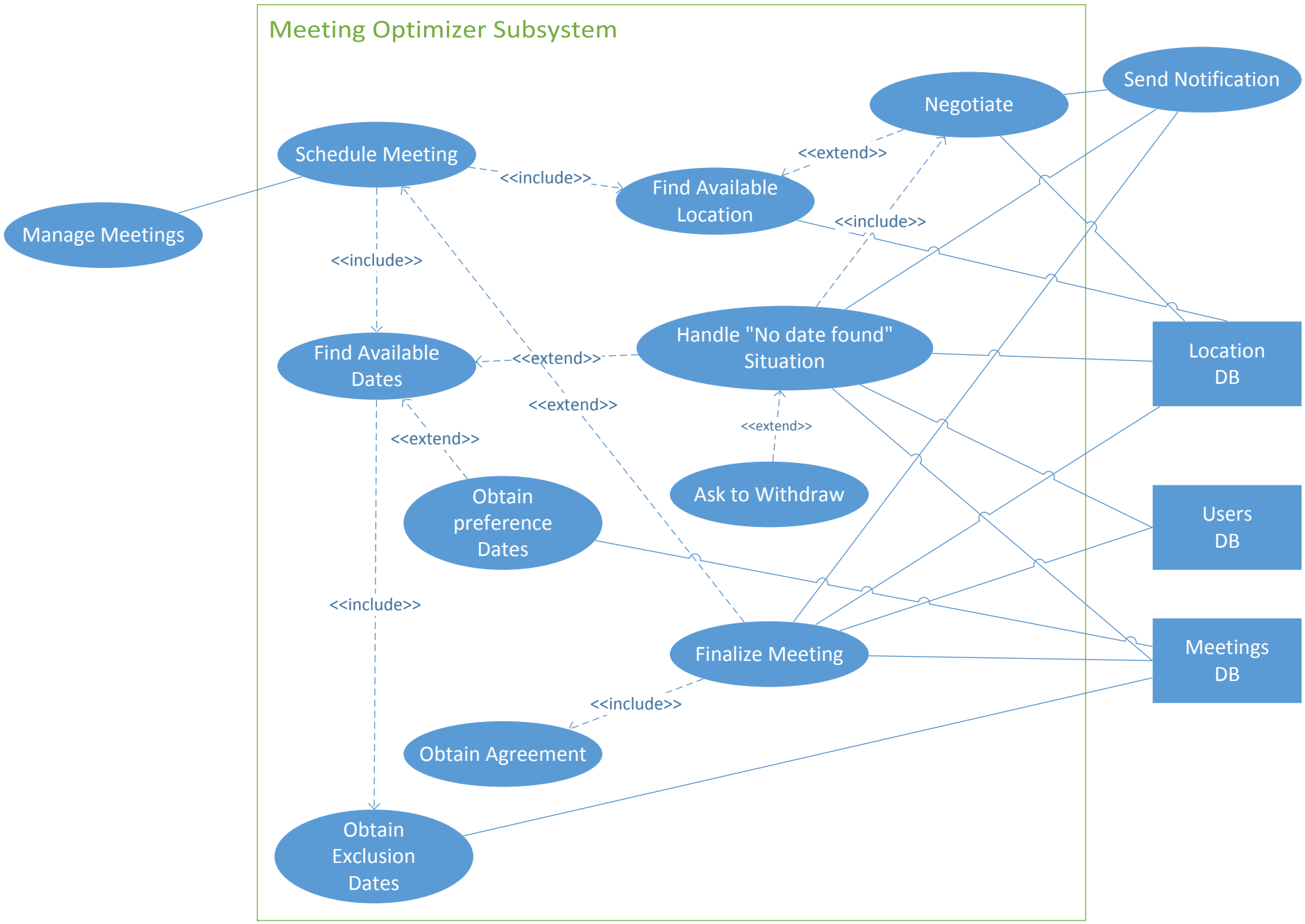
2.2.2 Meeting Optimizer Subsystems

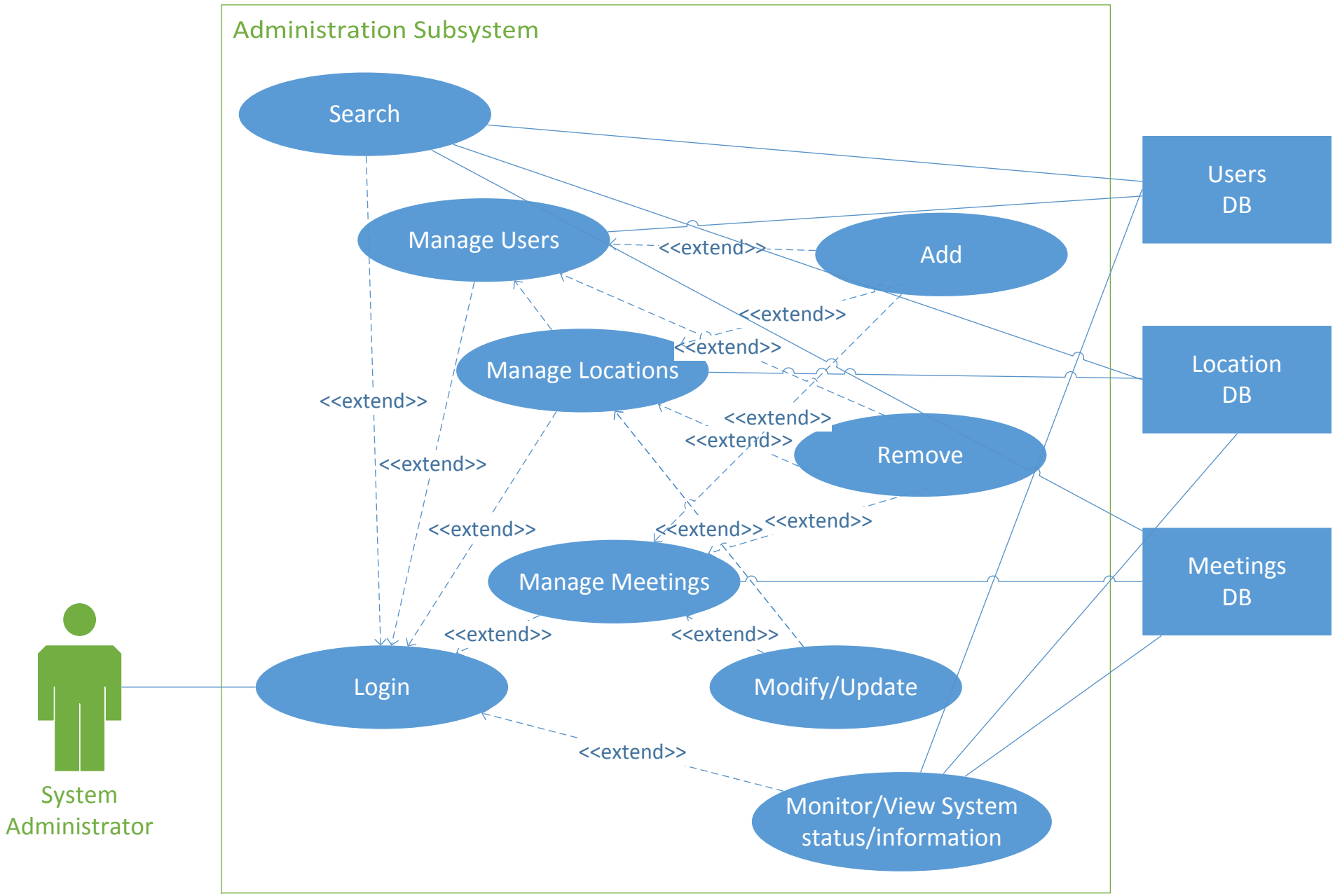
2.2.3 Administrator Subsystem

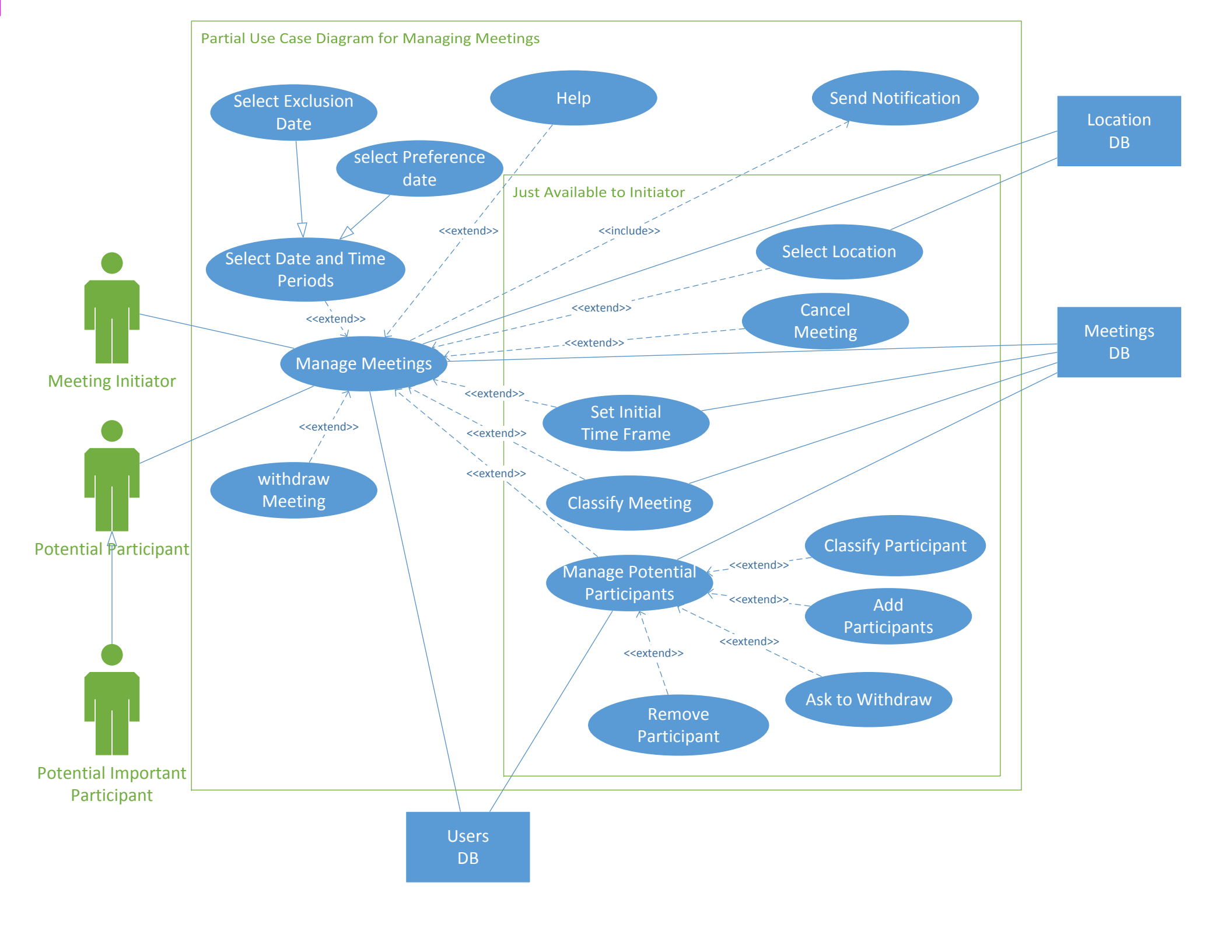
2.2.4 Partial Use Case Diagram for Managing Meetings

Partial Use Case Diagram at User Goal Level

Help
Search Meeting Category
Search
Authentication
Users DB
<<extend>>
<<include>>
Send notification
Search Locations
Search Participants
Login
<<include>>
<<include>>
Select Location
Search Calendar
<<extend>>
Request Meeting
<<include>>
Meetings DB
<<include>>
Set Initial Time Frame
<<include>>
Location DB
Meeting initiator
Choose Participants
Classify Meeting
<<include>>
<<include>>
View Scheduled Meetings
Classify Participants
<<include>>
Potential Participant
<<include>>
Accept Meeting
<<extend>>
Manage Meeting
<<extend>>
Withdraw Meeting
<<extend>>
Cancel Meeting
<<include>>
Release location
Potential participant
Potential important participant
Select dates and time periods
Select Preference set
Select Exclusion set
Users DB
Meetings DB
Location DB


Meeting Optimizer Subsystem
Manage Meetings
Schedule Meeting
Find Available Location
Negotiate
Send Notification
Find Available Dates
Handle "No date found" Situation
Location DB
Obtain preference Dates
Ask to Withdraw
Users DB
Finalize Meeting
Meetings DB
Obtain Agreement
Obtain Exclusion Dates
<<include>>
<<extend>>


Administration Subsystem
Search
Manage Users
Add
Manage Locations
Remove
Manage Meetings
Login
Modify/Update
Monitor/View System status/information
Users DB
Location DB
Meetings DB
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
System Administrator


Partial Use Case Diagram for Managing Meetings
Select Exclusion Date
Help
Send Notification
Location DB
select Preference date
Just Available to Initiator
<<extend>>
<<include>>
Select Date and Time Periods
Select Location
<<extend>>
Cancel Meeting
<<extend>>
Meeting Initiator
Manage Meetings
Meetings DB
<<extend>>
Set Initial Time Frame
<<extend>>
<<extend>>
withdraw Meeting
Classify Meeting
Potential Participant
Classify Participant
Manage Potential Participants
<<extend>>
Add Participants
<<extend>>
<<extend>>
Ask to Withdraw
Remove Participant
Potential Important Participant
Users DB

2.3 Use Cases Text Description

Main use cases described in detail which are at user goal level while sub function use cases in abstract.
For simplicity:
Db = database, user= Meeting initiator, potential participant and potential important participant, Initiator= Meeting initiator, participant= potential participant.

2.3.1 Request Meeting

Use Case ID:	UC-1.0.1
Use Case Name:	Request Meeting
Scope:	MeetingScheduler application
Level:	User goal
Primary Actors:	Meeting Initiator
Description:	This use case let users to ask for having a meeting with their colleagues
Preconditions:	**User is identified and authenticated (logged in).**
Success Guarantee (or Postconditions):	Proposed meeting is saved. Suggested location for that period of time is available. Suggested time frame and dates are from are chosen from calendar. Everyone who is invited to the meeting receives a notification email.
Main Success Scenario (or Normal Flow):	1. Meeting initiator requests to set a meeting. 2. System shows the meeting request session. 3. User chooses preferred date(s), meeting time frame, location(s), exclusion dates, potential participants for meeting and decides who is mandatory to attend and who is not (classifies participants). He also set the degree of meeting importance (3= very important (urgent), 2=important, 1=regular meeting) and submit the request. 4. Update meetings db and user's profile on user db for potential participants and location db for the booked location. 5. System send notifications via email to all potential participants.
Extensions (or Alternative Flows):	2a. System fails to load the page: 1. Initiator reloads/refreshes the page or he can go back to home page. 3a. For any part of the form: 1. Initiator can read help (?) section to fill it correctly. 3b. If any field was empty at the submission time: 1. System gives error to user to fill those empty field. 2. Initiator fills all the fields and submit it again. 3c. If there was no available location or date to select 1. Initiator should contact system admin to check if databases are updated or not. 4a. Any failure due to DB accessibility: 1. System should show the related error. 2. User should submit the request again 5a. If the system fails to send the email to a user: 1. System should retry to send it again and notify the user if it was successful or not. 2. If system couldn't sent the email, ask user to enter another

	contact information for that specific user. Use case: show error notification, help
Priority:	High
Includes:	Send notification, select location, add participants, classify participants, classify meeting, select dates and time period
Frequency of Use:	Very often (depend on size of the company)
Special Requirements:	
Assumptions:	
Notes and Issues:	-what customization is needed for different organizations? -what are different ways for notifying users about meetings? -what is the size of the company?

2.3.2 View Scheduled Meetings

Use Case ID:	UC-1.0.2
Use Case Name:	View Scheduled Meetings
Scope:	MeetingScheduler application
Level:	User goal
Primary Actors:	Meeting initiator, Potential Participant, Potential Important Participant
Description:	Users can view the meetings they initiated, are invited to or attended before.
Preconditions:	**User is logged in to the system.**
Success Guarantee (or Postconditions):	System shows all user meetings with related details and user can browse among them easily.
Main Success Scenario (or Normal Flow):	1. User enters the view session 2. System shows all the past and active meetings with related details
Extensions (or Alternative Flows):	1a. User can select filters to view specific meetings 1. Search based on participants ID 2. Search based on participants category (important/non-important) 3. Search based on location of meeting 4. Search based on dates 5. Search based on meeting category
Priority:	High
Includes:	
Frequency of Use:	Very often (depend on size of the company)
Special Requirements:	
Assumptions:	
Notes and Issues:	

2.3.3 Manage Meeting

Use Case ID:	UC-1.0.3
Use Case Name:	Manage Meeting
Scope:	MeetingScheduler application
Level:	User goal
Primary Actors:	Meeting initiator, Potential Participant, Potential Important Participant

Description:	Initiator can manage the meeting by modify the initial information such as extending the initial time frame for meeting, changing the suggested location(s) and so on. System need to update related databases to save the information and notify the potential participants about changes. Other users can manage their meetings too. They can change their preference and exclusion set or withdraw from a meeting. Any user can refer to help at any time. After initiation phase, potential participants would receive an email about the new meeting. By logging in to the system and referring to manage meetings section, they are able to see all their meetings. Response session would contain accept, withdraw, setting preference date and setting exclusion dates, and important participants can choose location too.
Preconditions:	**User is logged in and has invitation to a meeting or some scheduled meetings.**
Success Guarantee (or Postconditions):	System saves all the changes, update related databases and potential participants receive notification email about the meeting changes or system initiate the optimizer.
Main Success Scenario (or Normal Flow):	1. Initiator requests to manage the meeting 2. System show the meeting information 3. Initiator make the appropriate modifications which can be location, time frame, preference or exclusion time periods, potential participants and their role or meeting importance. 4. System will save the entered information to related databases and notify the potential users about the change.
Extensions (or Alternative Flows):	a* at any point if the system does not respond or show the appropriate session to user. 1. System should show the appropriate error message. 1.1 If it does not show the error, user need to reload the page. 2. User should take an action related to the message. 1a. Potential Participant can ask to manage his meetings. 3a. If user tagged as important: 1. He can choose the location of meeting too from the list (activated section). 3b. If meeting tagged as urgent or important, participant cannot withdraw from the meeting (deactivated section). 3c. Potential participant can modify the preference date and exclusion date or withdraw from the meetings.
Priority:	High
Includes:	Send notification
Frequency of Use:	sometimes
Special Requirements:	
Assumptions:	
Notes and Issues:	

2.3.4 Cancel Meeting

Initiator can cancel the meeting he initiated at any time. System will notify all potential participants about the cancelation and release the booked location and update related databases (meeting, user, location).
Primary actor: Meeting initiator
Includes: release location, send notification, updating meeting, user and location database.

2.3.5 Add Participants

Initiator should choose potential participants for each meeting he initiates. He also needs to determine if participant presence is mandatory or not.

2.3.6 Classify Participants

Initiator will do this task as part of selecting participants for the meeting.

2.3.7 Set Initial Time Frame

This would be done by initiator while creating the meeting request.

2.3.8 Select Location

This task can be done by initiator and potential important participants.

2.3.9 Send Notification

System is responsible to send notification when a meeting is initiated, changed, canceled or finalized.

2.3.10 Select Date and Time Period

Participants and Initiator can select date and time periods by responding to meeting invitation or managing their meetings. It can be setting exclusion set or preference set or both.

2.3.11 Select Exclusion Set

This list contains the dates and time periods that participant can not attend the meeting due to conflict with other meetings or another reason.

2.3.12 Select Preference Set

This list contain the dates that participant is willing to attend the meeting.

2.3.13 Login

All the users and system admin need to login to the system to be able to perform any task. System will authenticate users by searching the user database.
Active actors: Admin, Potential Participants, Potential Important Participants, Meeting Initiator
Include: Authentication (Access to user db)

2.3.14 Schedule Meeting

Use Case ID:	UC-1.0.14
Use Case Name:	Schedule Meeting
Scope:	Meeting Optimizer Subsystem
Level:	Summary
Primary Actors:	System
Description:	This use case is responsible to find the best date and location for meeting based on exclusion sets, preference sets and importance of

	participants.
Preconditions:	**All potential participants responded to invitation or if any one makes changes by using manage meeting section.**
Success Guarantee (or Postconditions):	Send notification which can be finalized date and location or a new round of negotiations to find new date or new location to potential participants.
Main Success Scenario (or Normal Flow):	1. Potential participant entered new information for the meeting 2. System initiate the schedule meeting process by considering new information (new time period or location) 3. System considers provided dates and locations by all potential participants for the meeting and finds available time and location which all important participants can come 4. System send notification about finalized date and location 5. Participants will receive an email about the finalized meeting
Extensions (or Alternative Flows):	1a. Potential participant changed previous selection for the meeting 1b. Potential participant withdraw from meeting 1c. Potential participant respond to new round of negotiation 3a. System cannot find any common date or location between important participants: 1. System investigate all possible options to set a meeting 2. Send some suggestions to participant to negotiate
Priority:	High
Includes:	Find available dates, finalize meeting, find available location
Frequency of Use:	Very often
Special Requirements:	
Assumptions:	
Notes and Issues:	

2.3.15 Finalize Meeting

Use Case ID:	UC-1.0.15
Use Case Name:	Finalize Meeting
Scope:	Meeting Optimizer Subsystem
Level:	Sub function
Primary Actors:	System
Description:	System was able to find the optimum date and location for meeting and notify participants to agree about that.
Preconditions:	**All participants provided necessary information for meeting date, time and location**
Success Guarantee (or Postconditions):	Agreement notification will be sent to potential participants
Main Success Scenario (or Normal Flow):	1. System find available dates and location meeting requirements 2. System finalize the time and location, update related databases and send agreement notification to all potential participants 3. Participants will receive the agreement notification and will respond to that.
Extensions (or Alternative Flows):	1a. if there was more than one available option with the same priority (same number of participants who can attend the meeting): 1. System will choose the first available option as the finalized

	option
Priority:	High
Includes:	
Frequency of Use:	Very often
Special Requirements:	
Assumptions:	
Notes and Issues:	

2.3.16 Obtain Agreement

As part of finalizing meeting, system need to make sure all important participants are agreed about the last time and location of meeting.

2.3.17 Find Available Location

As part of scheduling meeting process, scheduler need to consider initial suggested locations by initiator and important participants preferences for location and meet the maximum vote.

2.3.18 Find Available Date

As part of scheduling meeting process, the scheduler need to find available dates that all the important participants and most of the potential participants can attend. These job can be done by obtaining initial time frame, preference sets and exclusion sets from all participants.

2.3.19 Obtain Preference Dates

As part of scheduling meeting, scheduler needs to consider preference dates for all potential participants to decide about the time of meeting. The importance of considering remaining dates in time frame after taking out exclusion sets is higher. Then if there was any day left, we should consider the preference sets of potential important participants. At the end, if there was any choice, we can consider non-important potential participants preference dates.

2.3.20 Obtain Exclusion Dates

Schedule needs to obtain and exclude the exclusion time periods from initial time frame to find the best time for the meeting. If there was not any available date among all important participants,

2.3.21 Handle no date found situation

Use Case ID:	UC-1.0.21
Use Case Name:	Handle no date found situation
Scope:	Meeting Optimizer Subsystem
Level:	User goal
Primary Actors:	System
Description:	When meeting scheduler cannot find any available date which meets the minimum requirements, it would take some extra steps to make it possible to set a meeting by asking some non-important participants to withdraw, asking initiator to extend the initial time frame, asking a participant to withdraw from another meeting with lower significance, ask some participants to add dates to their preference set, or asking

	some participants to remove some dates from their exclusion set.
Preconditions:	**No dates was found for the meeting based on minimum requirements**
Success Guarantee (or Postconditions):	New round of negotiation (email) would be held by the scheduler and target participants should receive an email.
Main Success Scenario (or Normal Flow):	1. System could not find any available common date for participants 2. System suggests initiator to extend the initial time frame 3. Initiator receives the email, log in to system and change the initial time frame. 4. System schedule the meeting again considering new information 5. System find the common date and location for the meeting 6. System finalize the meeting and send notification to all participants 7. Participants receive the email and respond to agreement request
Extensions (or Alternative Flows):	2a. System suggests to specific participants to take out those conflicting exclusion dates and times from exclusion set 1. Recognized participant receive an email about the suggestion 2. Participants manage their meeting selection by logging in to the system 2b. System identify some non-important participants which their withdrawal lead to have a common dates for the meeting 2c. System suggests to specific participants to add some dates to their preference set 2d. System can suggest a participant to cancel another meeting which has conflict with this one and has lower importance 2e. some dates from exclusion set of non-important participants can be considered.
Priority:	High
Includes:	Negotiate
Frequency of Use:	Some times
Special Requirements:	
Assumptions:	
Notes and Issues:	

2.3.22 Release Location

Whenever the initiator cancel a meeting or scheduler finalize another location different from the previous booked one, the location database needs to be updated and change status of previous location to available.

2.3.23 Accept Meeting

When participant invited to a meeting and he does not have any restriction in suggested time frame.

2.3.24 Withdraw Meeting

When a potential participant does not want to participate in a meeting voluntary or as a request from system or initiator. Related dbs should be updated and meeting scheduler then need to schedule meeting with new condition.

2.3.25 Search

All users can search meetings, locations or other users based on id or keywords or label to view selected information from the system.

2.3.26 Search participants

Users can search other participants in the system, or in a specific meeting, with a specific label.

2.3.27 Search meeting category

Everyone can search among their own meetings by meeting label.

2.3.28 Search Calendar

All users can view calendar and limit it to specific dates or time periods.

2.3.29 Search Locations

Users can search among locations in general or available locations for the proposed meeting. However, the only ones who can set a location for meeting are initiator and important participants.

2.3.30 Classify Meeting

Initiator can classify the meeting by choosing one of these three labels: urgent, important or regular.

2.3.31 Negotiate

When the system could not find a common date or location among participants, it needs to send some specific suggestion to all participants or specific ones to resolve the problem.

2.3.32 Help

Users can ask for help (hint) form the system at any session they are to find out how they should work with the system.

2.3.33 Manage Potential Participants

Initiator can manage participants at the initiating stage or after that. He can change their label, add new participants, remove some participants or ask someone to withdraw from the meeting (depends on organization policy).

2.3.34 Ask to Withdraw

Initiator or system scheduler can ask some participants to withdraw from the meeting.

2.3.35 Remove Participants

Initiator can remove invited participants to the meeting.

2.3.36 Manage Meeting

Admin can manage the meetings to by accessing meetings database.

2.3.37 Manage Location

Admin can modify location's information by entering their name or ID and add, remove or change location information.

2.3.38 Manage User

Admin can modify user's information by entering their user ID and add, remove or change user information after he received the request for change.

2.3.39 Add

As part of managing db, admin can add location, meeting or users to related databases by receiving the request in order to update location and meeting list and register new users to the system.

2.3.40 Remove

As part of managing db, admin can remove users, locations and meetings from related databases upon request.

2.3.41 Modify

As part of managing db, admin can modify information about any user, location or meeting to keep the system accurate.

2.3.42 Monitor/View system

Admin can view the information which are saved to the database and monitor the system to see if the system works properly or not.

2.3.43 Search

Admin can search in system databases based on a keyword or date. Search can be performed on user, location and meeting databases.

2.4 Nonfunctional Requirements

2.4.1 Requirements for System

The user requires a menu-based system because it provide the easiest method of navigating the system. The maximum depth of menus are limited to three. The system will be accessed by various company employees with friendly user interface.

2.4.2 Requirements for Software

The user require that the system should be implemented by using Java technology. By using a graphical user interface (GUI), mouse support should also be available; this will further simplify the procedure. The system should have a familiar look and feel to any experienced computer user.

2.4.3 Requirement for response-time and process

As a customized system, the system should attempt to simplify the scheduling tasks as much as possible. The user require that he/she is permitted to arrange a time for a meeting by providing a minimal amount of information. System need to have acceptable response time to make decision about meeting.

2.4.4 Scalability

Immutable functions in implementation part prevent the system to have problem with concurrency.

2.4.5 Reliability

Providing unit tests for testing each function and using test driven development which means designing test cases before implementing. Moreover, for verifying overall system functionality we can use integration testing.

2.4.6 Availability

2.4.6.1 Clustering

2.4.6.2 Automated integration test: checking system status all the time

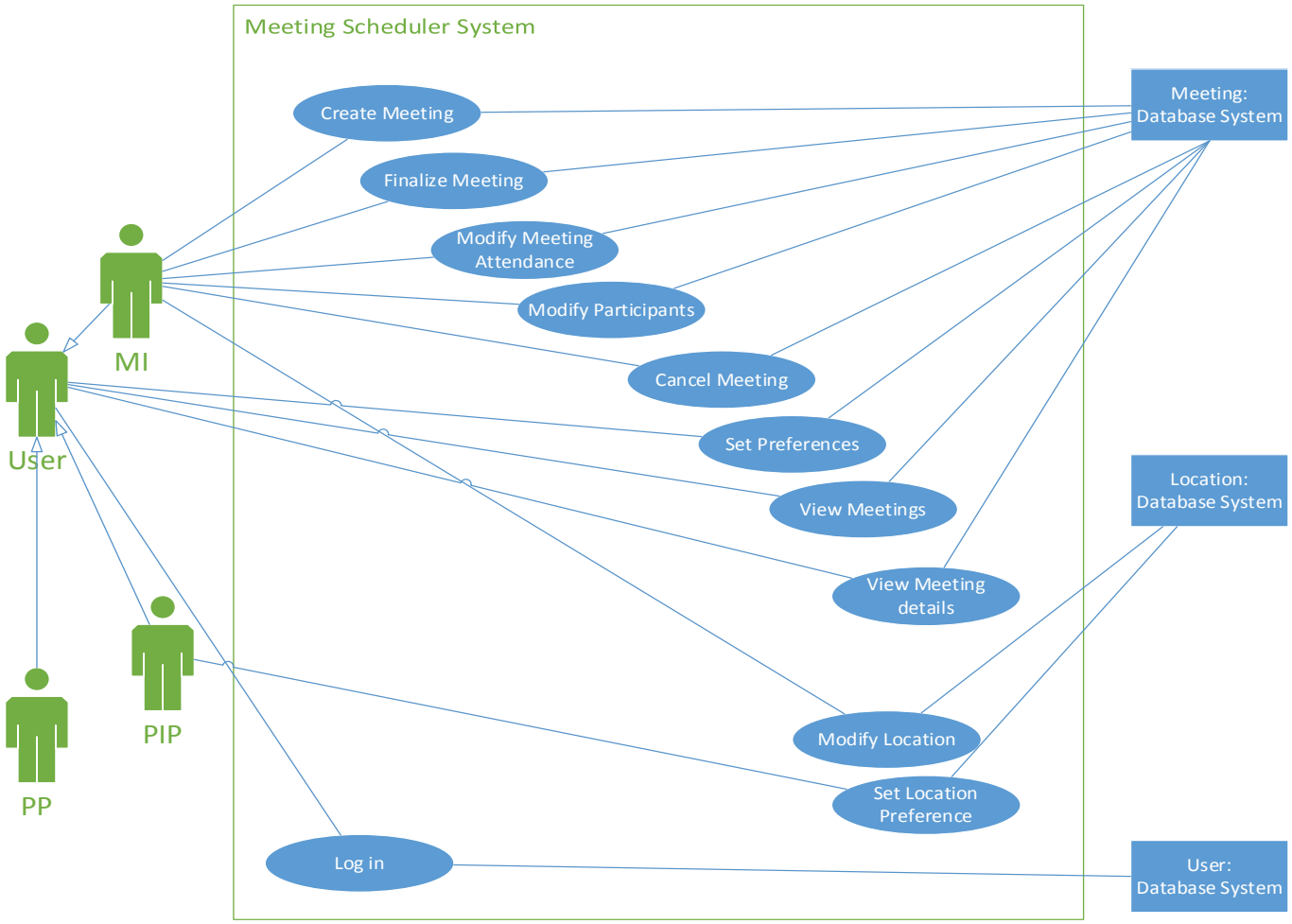
3. Use Case Model for reduced sized software

As part of this project, I need to implement the web-based meeting software with necessary functionality which turns the software into simpler, lower cost, with less maintainability problem software. In this part I will explain related use case and use case diagrams for first implemented version of the software. For the similar functions and actors description, please refer to chapter two as everything described with full details in that chapter.

3.1 Actors

MI = Meeting Initiator, PP = Potential Participant, PIP= Potential Important Participant
Databases: Location Db, User Db, Meeting Db (one database with three tables)

3.2 Use Case Diagram



3.3 Use Case Scenarios

3.3.1 Create Meeting

Use Case ID:	UC-1.0.1
Use Case Name:	Create Meeting
Scope:	MeetingScheduler application
Level:	User goal
Primary Actors:	MI
Description:	This use case let users to ask for having a meeting with their colleagues
Preconditions:	**User is identified and authenticated (logged in).**
Success Guarantee (or Postconditions):	Proposed meeting is saved to the db.
Main Success Scenario (or Normal Flow):	1. User request to create meeting. 2. System shows appropriate page 3. User enters meeting time period and location or locations 4. System save information successfully
Extensions (or Alternative Flows):	
Priority:	High
Includes:	
Frequency of Use:	Very often (depend on size of the company)
Special Requirements:	
Assumptions:	All locations are exist in db.
Notes and Issues:	-what customization is needed for different organizations? -what are different ways for notifying users about meetings? -what is the size of the company?

3.3.2 Modify Meeting Attendance

Use Case ID:	UC-1.0.2
Use Case Name:	Modify Meeting Attendance
Scope:	MeetingScheduler application
Level:	User goal
Primary Actors:	MI
Description:	MI can extend the time period of meeting
Preconditions:	**User is identified and authenticated as meeting initiator.**
Success Guarantee (or Postconditions):	New time sets are saved to the db.
Main Success Scenario (or Normal Flow):	1. MI asks to modify time period 2. System provide related page and field 3. MI append or prepend period by entering new start and end time 4. System save changes
Extensions (or Alternative Flows):	
Priority:	High
Includes:	

Frequency of Use:	sometimes
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.3.3 Modify Participants

Use Case ID:	UC-1.0.3
Use Case Name:	Modify Participants
Scope:	MeetingScheduler application
Level:	User goal
Primary Actors:	MI
Description:	MI can add and remove participants or change their labels as imp, non-imp for the meeting.
Preconditions:	**User recognized as MI for meeting**
Success Guarantee (or Postconditions):	
Main Success Scenario (or Normal Flow):	1. MI ask to modify participants 2. System shows related page 3. MI enter user ID and select remove 4. System save changes, update related db and notify related participants about changes.
Extensions (or Alternative Flows):	3a. MI choose participants label 1. Change Imp/non important label for existing participants 3b. MI add new participants and choose isImportant attribute value for each
Priority:	High
Includes:	Sending email
Frequency of Use:	Very often
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.3.4 Cancel Meeting

Use Case ID:	UC-1.0.4
Use Case Name:	Cancel Meeting
Scope:	MeetingScheduler application
Level:	User goal
Primary Actors:	MI
Description:	MI can cancel the meeting at any stage
Preconditions:	**User identified as MI**
Success Guarantee (or Postconditions):	System save changes and notify all participants
Main Success Scenario	1. MI ask to cancel the meeting

(or Normal Flow):	2. System remove the meeting and updates all related dbs and send email to participants.
Extensions (or Alternative Flows):	
Priority:	High
Includes:	send cancellation notification
Frequency of Use:	sometimes
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.3.5 Finalize Meeting

Use Case ID:	UC-1.0.5
Use Case Name:	Finalize Meeting
Scope:	MeetingScheduler application
Level:	User goal
Primary Actors:	MI
Description:	MI is the only one who can finalize the meeting
Preconditions:	**System could find a time and location for meeting**
Success Guarantee (or Postconditions):	Save changes to db, everyone receive notification and location get booked
Main Success Scenario (or Normal Flow):	1. Finalize section in MI profile become activated by system 2. MI log in and finalize the meeting which means he agrees to location and time. 3. System save changes, update meeting and location in db and send emails to participants.
Extensions (or Alternative Flows):	
Priority:	High
Includes:	Send notification, update location status
Frequency of Use:	Very often
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.3.6 Set Time Preferences

Use Case ID:	UC-1.0.6
Use Case Name:	Set Time Preferences
Scope:	MeetingScheduler application
Level:	User goal
Primary Actors:	User
Description:	Users can change their available and preference times
Preconditions:	**User log in to his own profile**

Success Guarantee (or Postconditions):	
Main Success Scenario (or Normal Flow):	1. user signs in to his profile and view meeting details 2.system shows set time section to all users 3. pp/pip user sets the right attendance time and exit 4. system save the changes
Extensions (or Alternative Flows):	3a. If all PIP users set their preference time 1. system should send an email to MI
Priority:	High
Includes:	
Frequency of Use:	Very often
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.3.7 View Meetings

Use Case ID:	UC-1.0.7
Use Case Name:	View Meetings
Scope:	MeetingScheduler application
Level:	User goal
Primary Actors:	User
Description:	Users can see all their meetings (past, active and invited) in their profile
Preconditions:	**User is logged in**
Success Guarantee (or Postconditions):	System loaded all user meetings
Main Success Scenario (or Normal Flow):	1. User logged in 2. System shows user session 3. User chooses to view all meetings related to him 4. System load all the meeting by getting user id
Extensions (or Alternative Flows):	
Priority:	High
Includes:	
Frequency of Use:	Very often
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.3.8 View Meeting Details

Use Case ID:	UC-1.0.8
Use Case Name:	View Meeting Details
Scope:	MeetingScheduler application

Level:	User goal
Primary Actors:	User
Description:	Users can see details of the meeting they are invited to
Preconditions:	**User logged in and select a meeting**
Success Guarantee (or Postconditions):	System loaded selected meeting details which can be different based on user role
Main Success Scenario (or Normal Flow):	1. User select a meeting to view the details 2. System get meeting and user id and load the data which shows initial time period, suggested locations and participants to pp user
Extensions (or Alternative Flows):	2a. System can load set locations preference for PIP users additionally 2b. System can load all participants with their submitted preferences for MI
Priority:	High
Includes:	
Frequency of Use:	Often
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.3.9 Modify Locations

Use Case ID:	UC-1.0.9
Use Case Name:	Modify Locations
Scope:	MeetingScheduler application
Level:	User goal
Primary Actors:	MI
Description:	MI can add or remove locations for the meeting
Preconditions:	**User has to logged in as MI and select meeting**
Success Guarantee (or Postconditions):	Save changes to system
Main Success Scenario (or Normal Flow):	
Extensions (or Alternative Flows):	
Priority:	High
Includes:	
Frequency of Use:	Often
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.3.10 Set Location Preference

Use Case ID:	UC-1.0.10
Use Case	Set Location Preference

Name:	
Scope:	MeetingScheduler application
Level:	User goal
Primary Actors:	PIP
Description:	Just PIP users can choose location among suggested locations by MI.
Preconditions:	**User is logged in and authenticated as pip for the meeting and asked to view meeting details.**
Success Guarantee (or Postconditions):	System saves location preferences to meeting db.
Main Success Scenario (or Normal Flow):	1. PIP select a meeting and ask to view details 2. System shows meeting detail with option of selecting preferred location 3. PIP can choose between previously suggested locations by MI 4. system save changes by getting location id
Extensions (or Alternative Flows):	4a. system cannot save the changes due to location status (booked): 1. System shows appropriate error and asks user to choose another location
Priority:	High
Includes:	
Frequency of Use:	Sometimes
Special Requirements:	
Assumptions:	
Notes and Issues:	

3.3.11 Log in

Use Case ID:	UC-1.0.11
Use Case Name:	Log in
Scope:	MeetingScheduler application
Level:	User goal
Primary Actors:	User
Description:	User can sign in to his own profile
Preconditions:	**User is connected to internet**
Success Guarantee (or Postconditions):	User successfully sing in to the system
Main Success Scenario (or Normal Flow):	3. User opens application log in page 4. User enters his id. 5. System connect to db and search for id. 6. System identify the user and let the uset to sign in and see the new page
Extensions (or Alternative Flows):	4a. system does not identify user id 1. System show appropriate error message 2. User need to enter correct id
Priority:	High
Includes:	Authentication
Frequency of Use:	Very often
Special Requirements:	
Assumptions:	

4. Sequence Diagrams

4.1 User log in

4.2 Meeting initiator create meeting

4.3 Meeting initiator finalize meeting

4.4 Meeting initiator modify meeting attendance

4.5 Meeting initiator modify participants

4.6 Meeting initiator modify location

4.7 Potential important participant set location preference

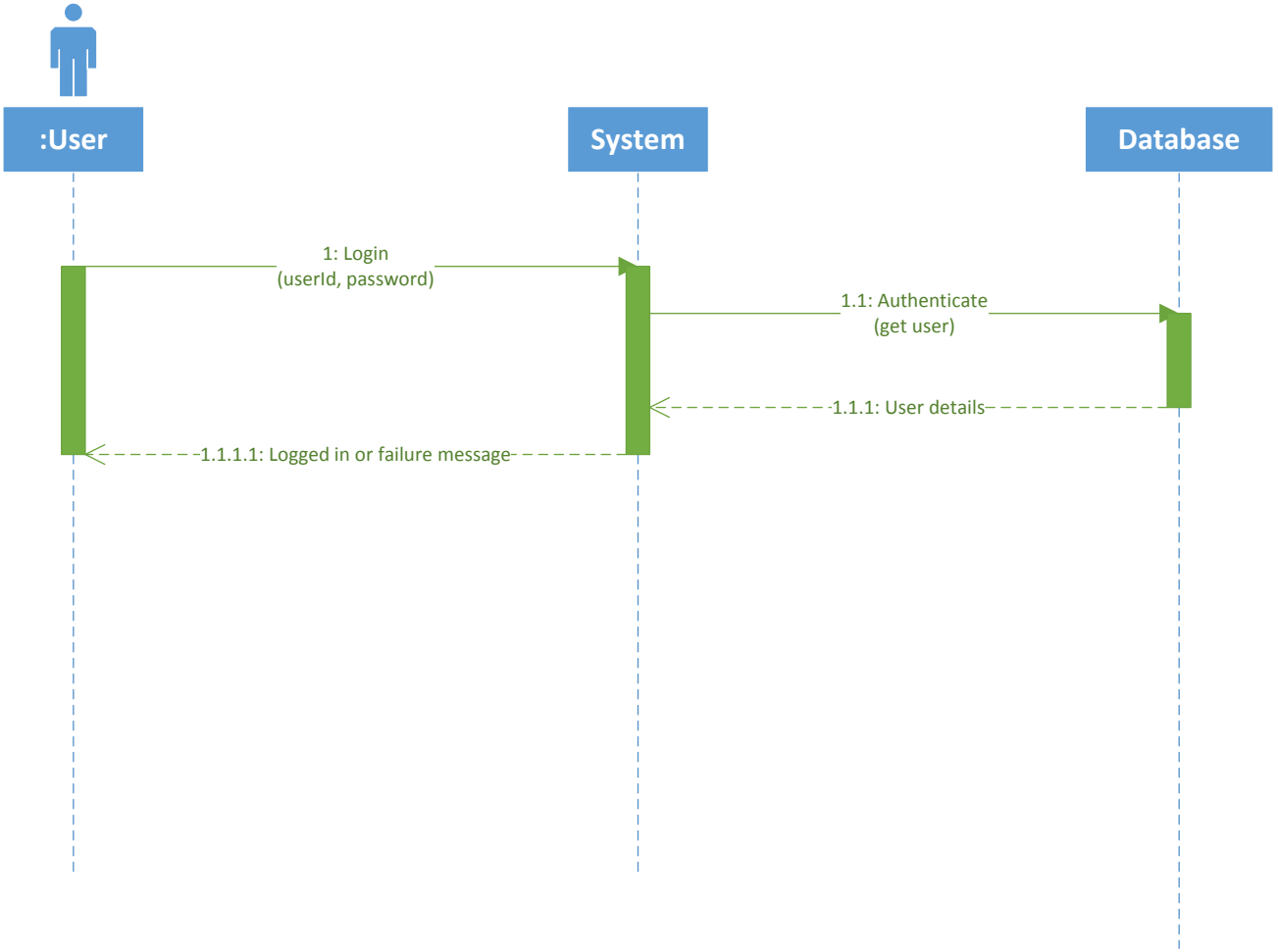
4.8 User view meetings

4.9 User view meeting details

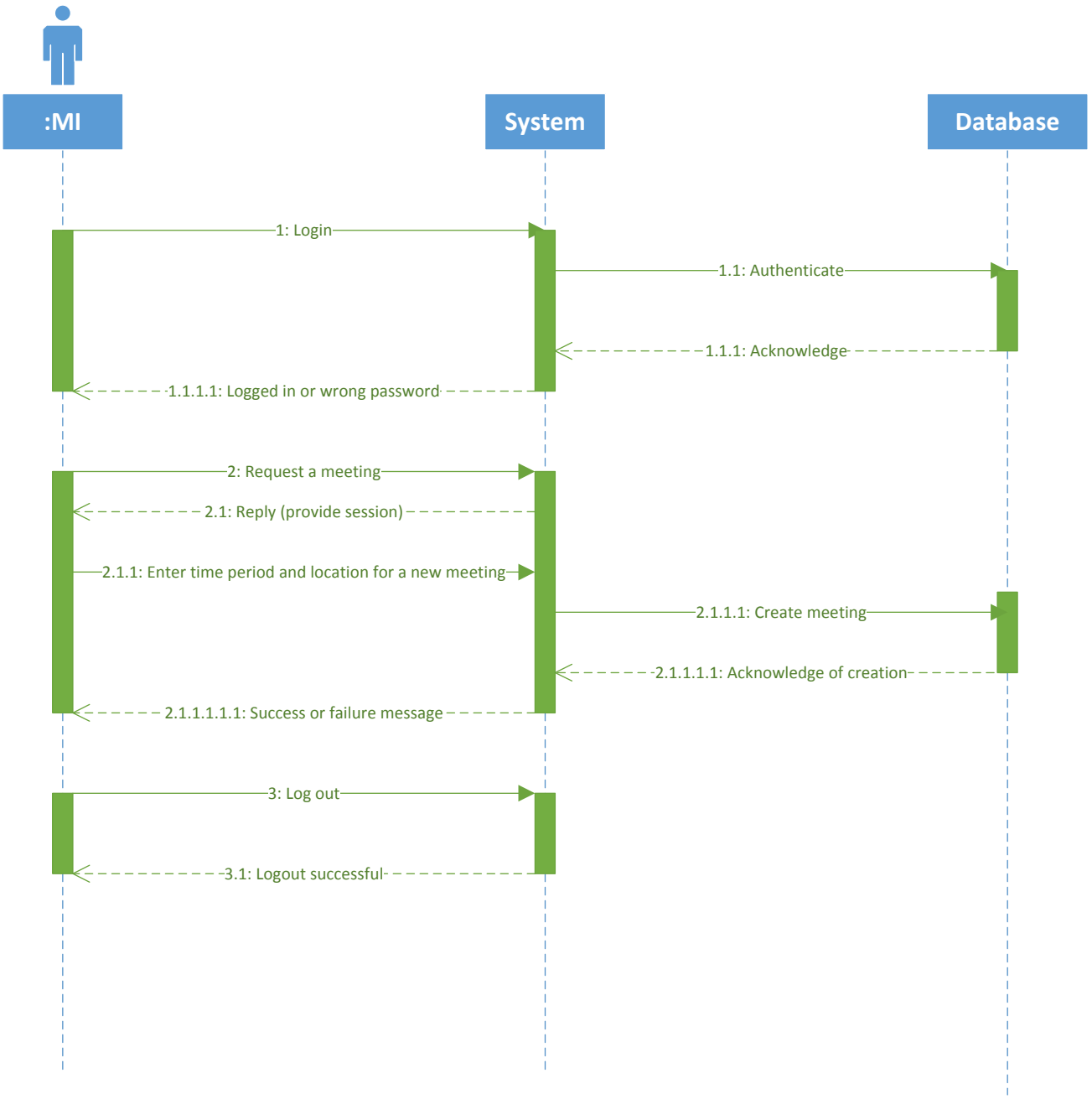
4.10 User set preference time

4.11 Meeting initiator cancel meeting

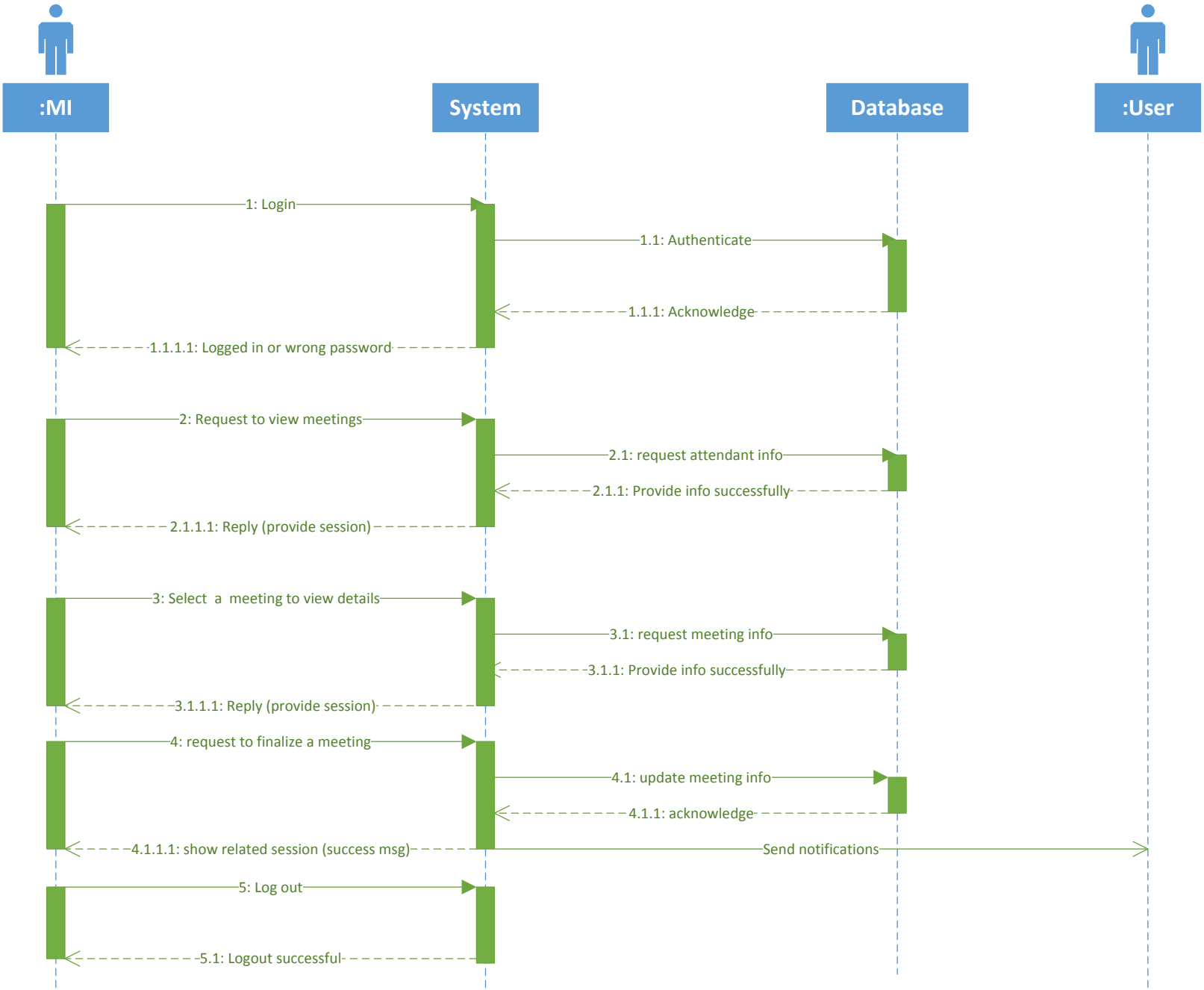
SD. User.Login()



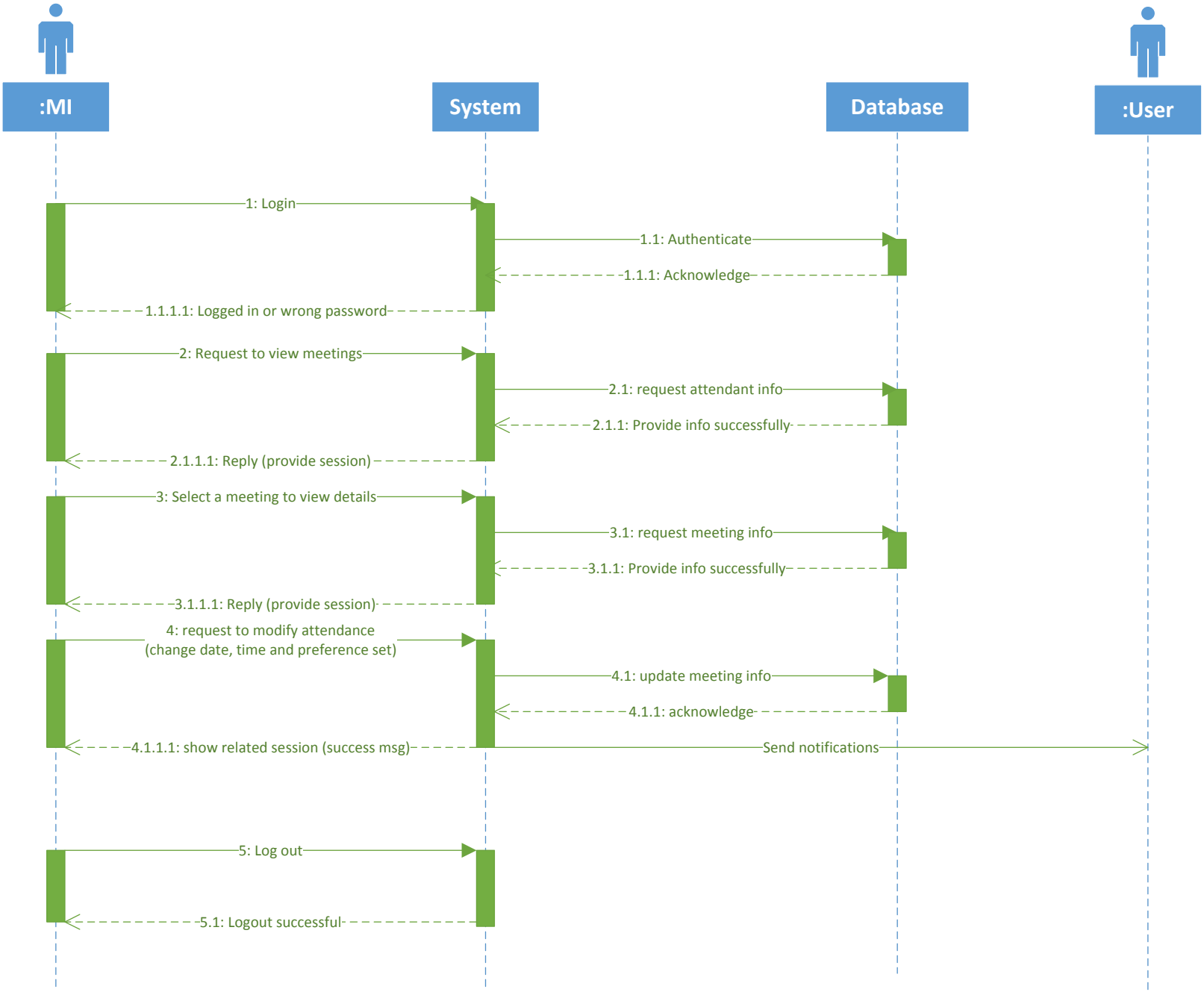
SD. MI.createMeeting()



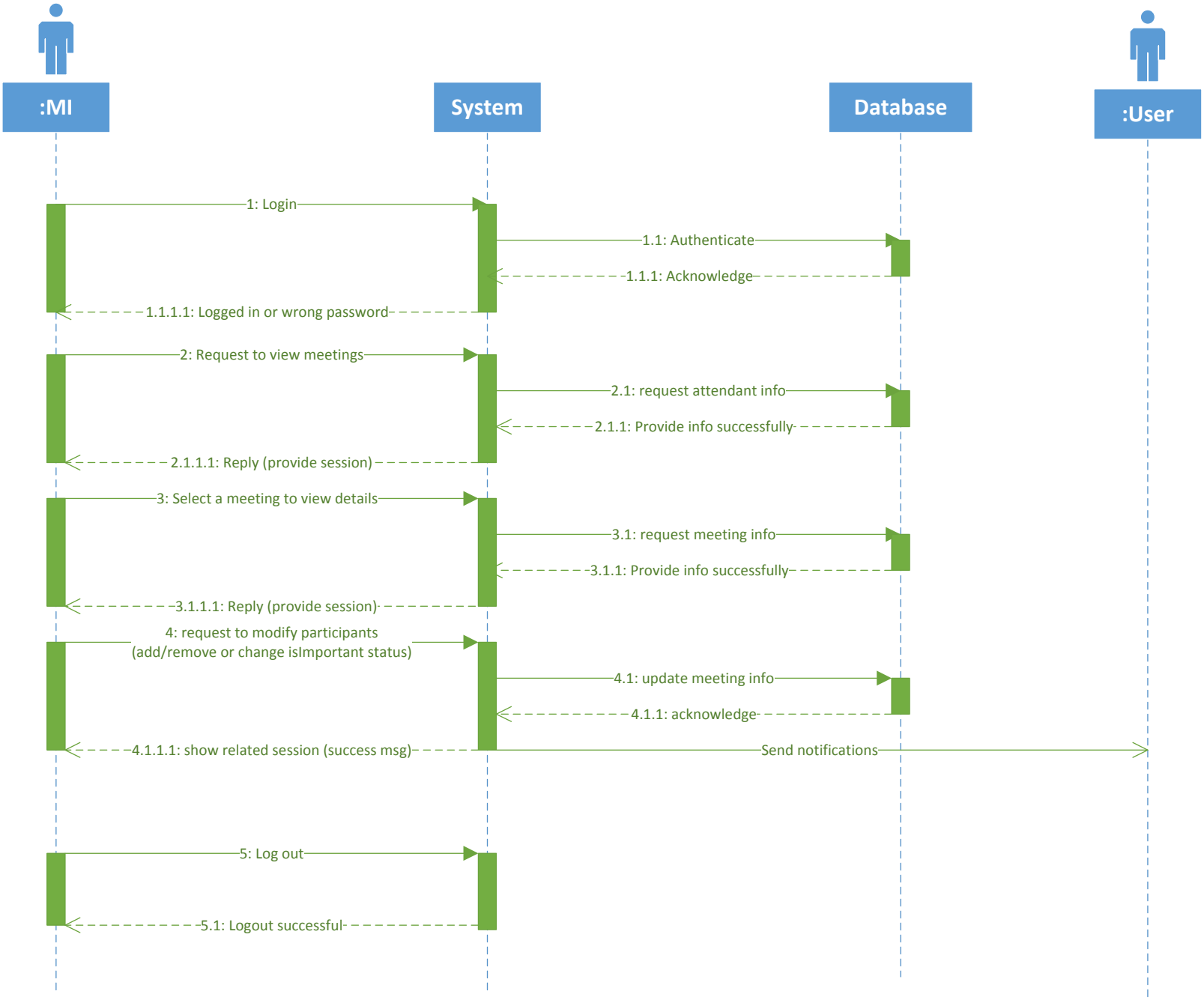
SD. MI.finalizeMeeting()



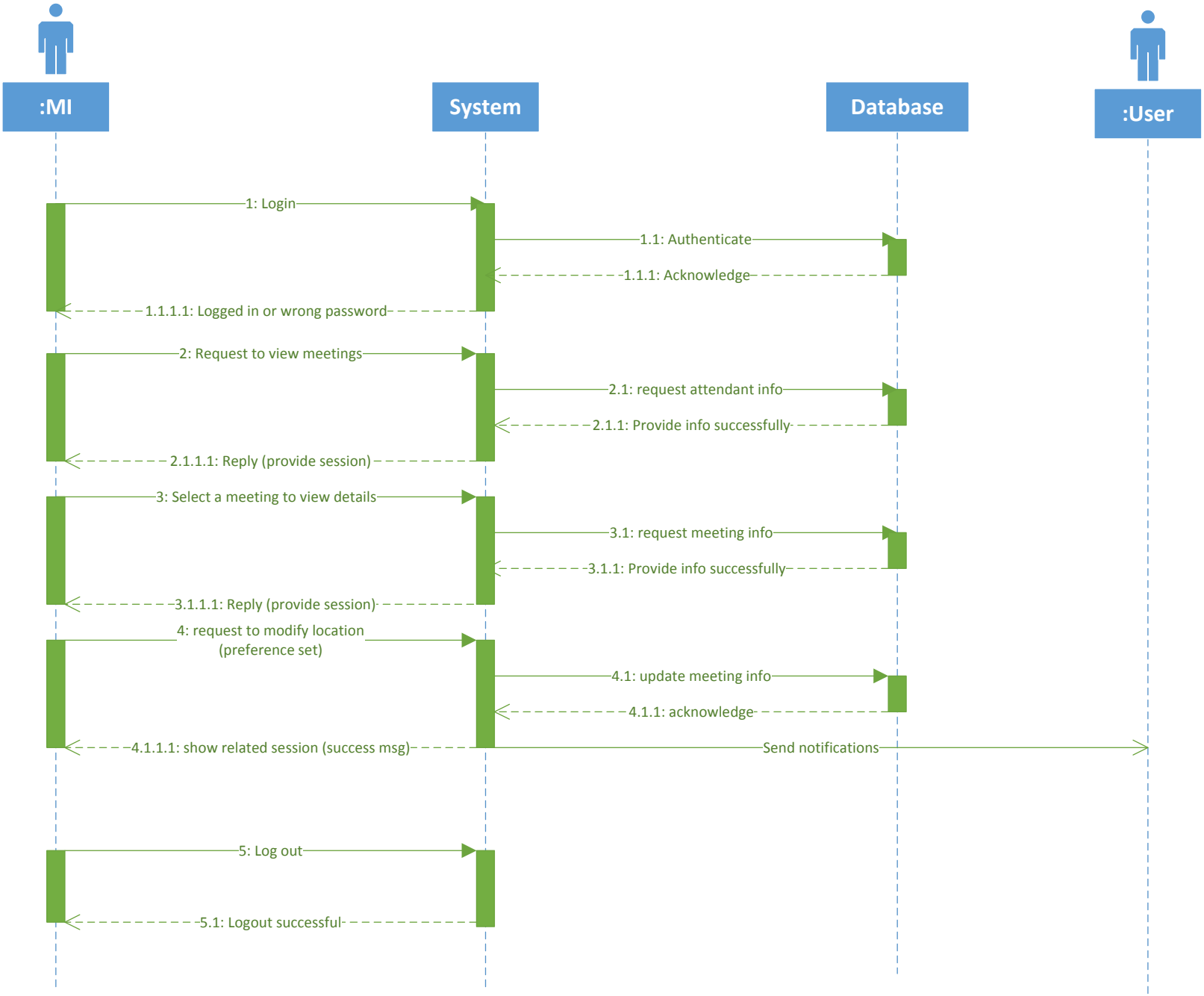
SD. MI.modifyMeetingAttendance()



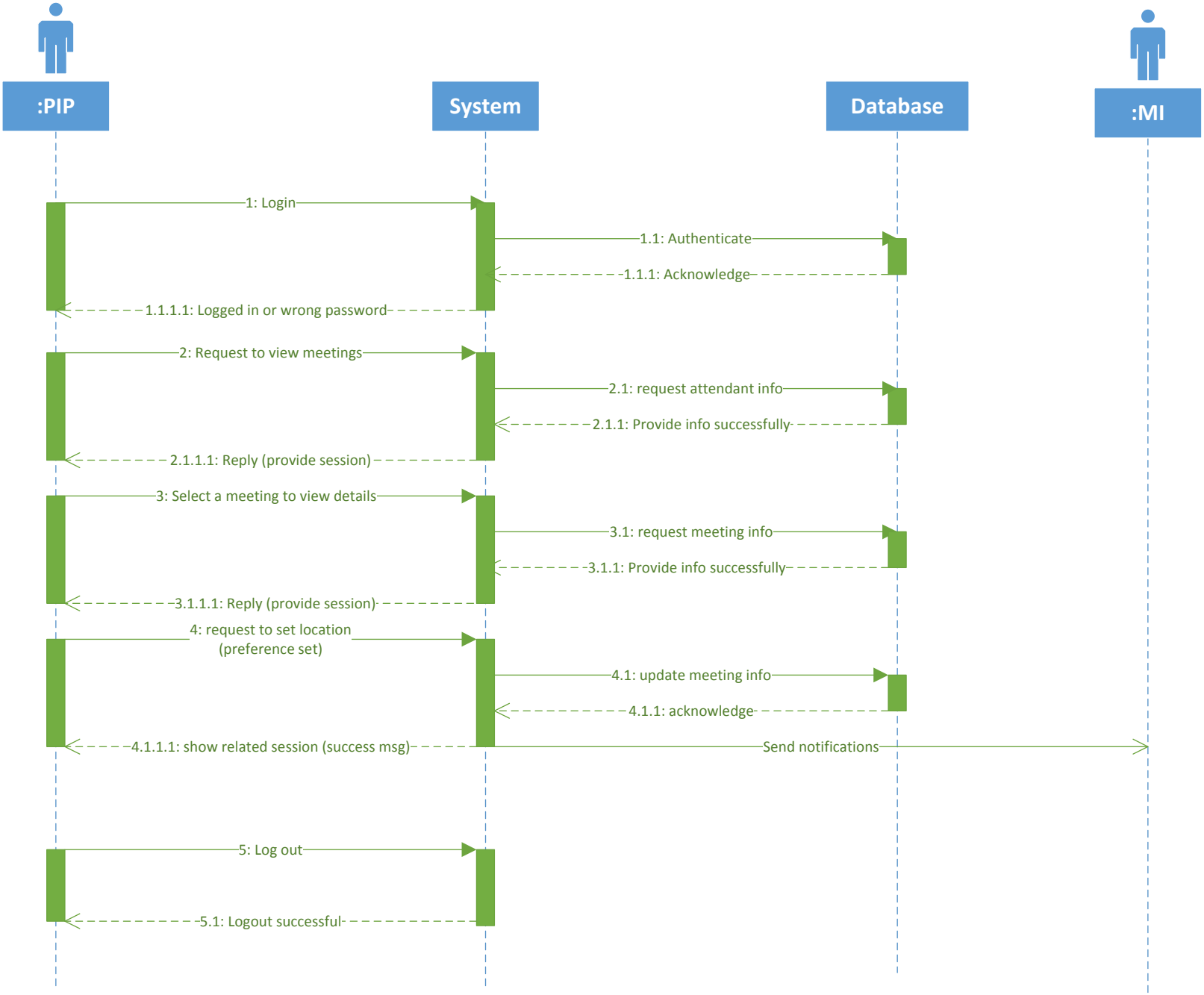
SD. MI.modifyParticipants()



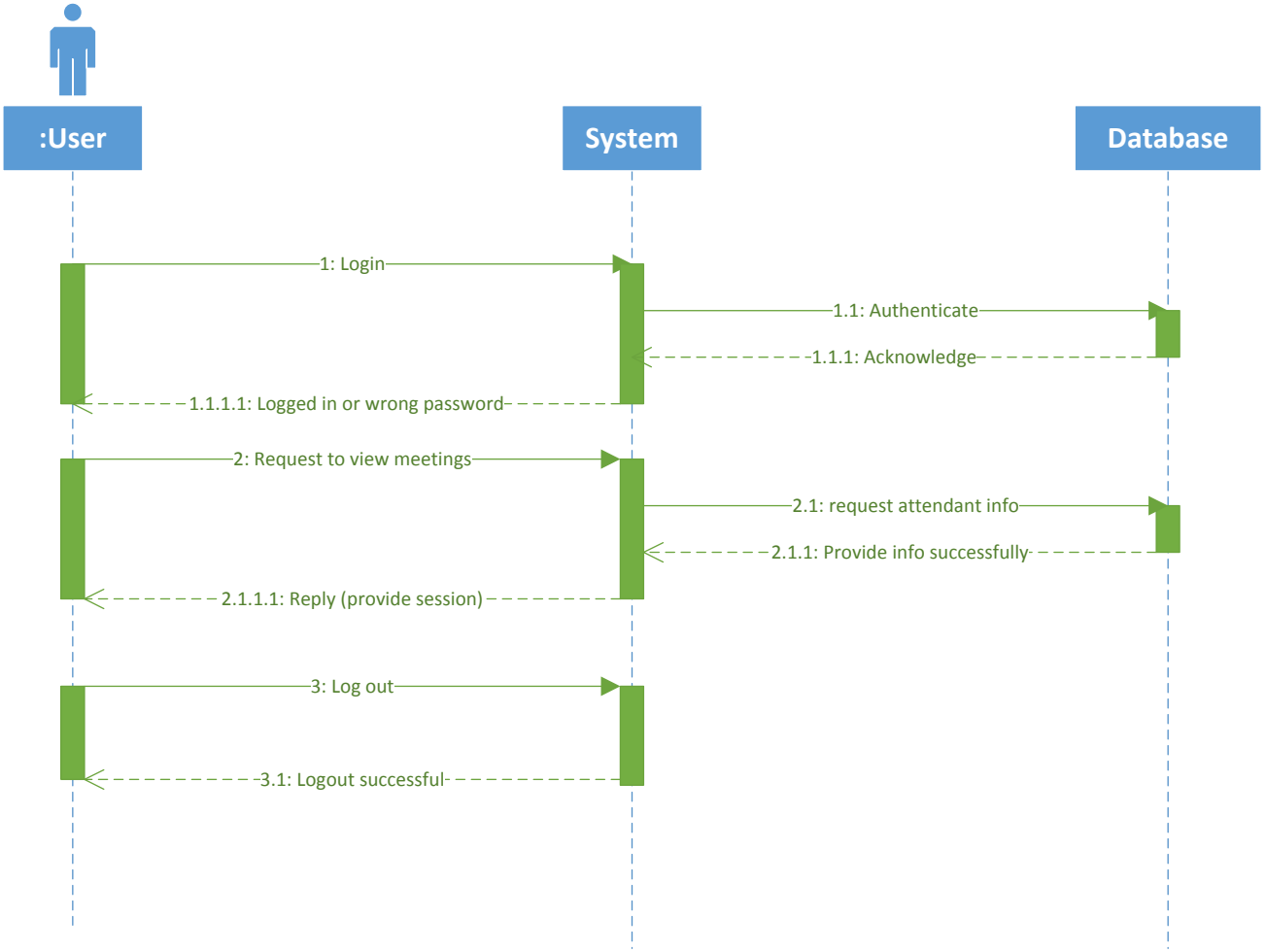
SD. MI.modifyLocation()



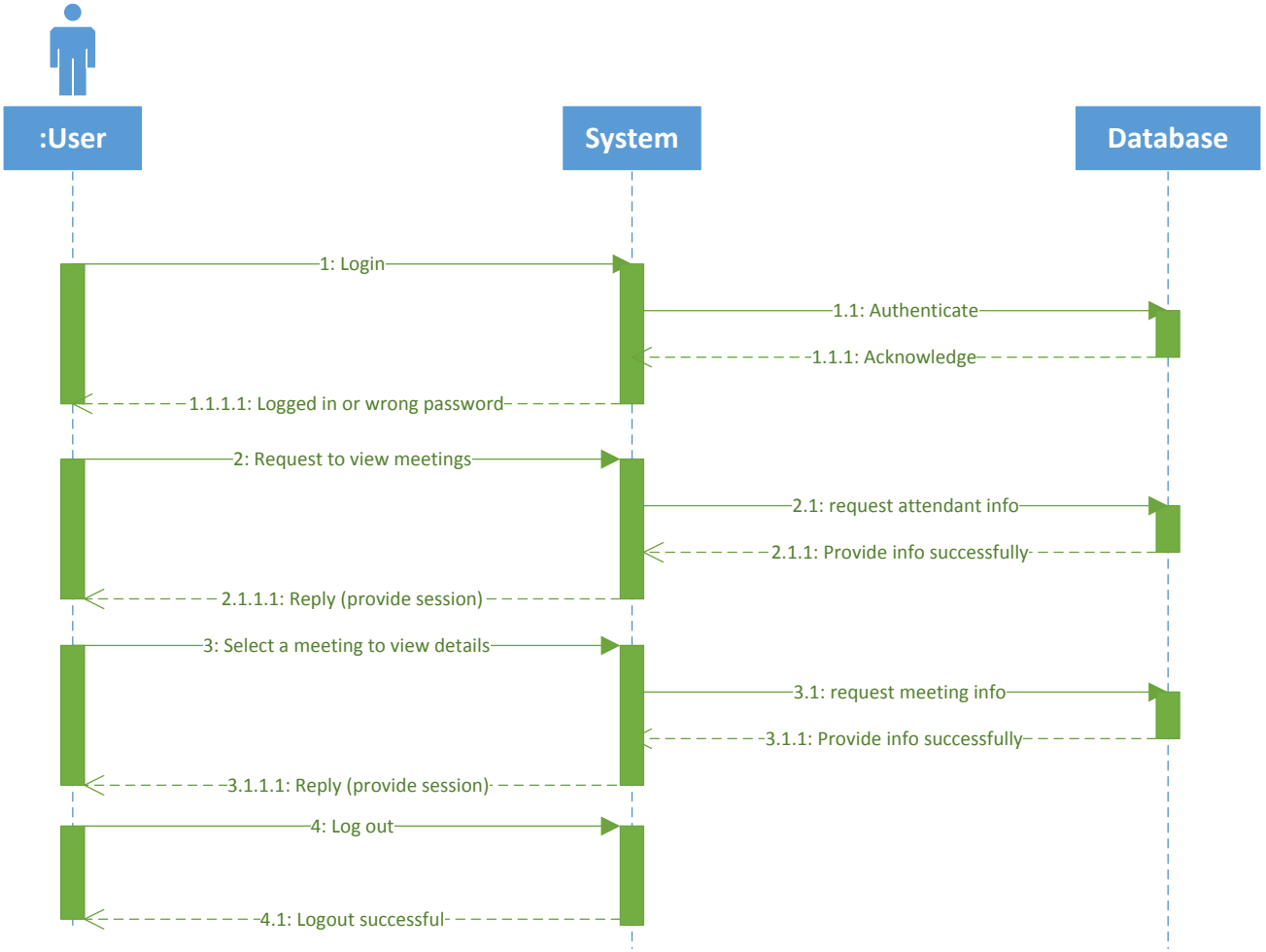
SD. PIP.setLocationPreference()



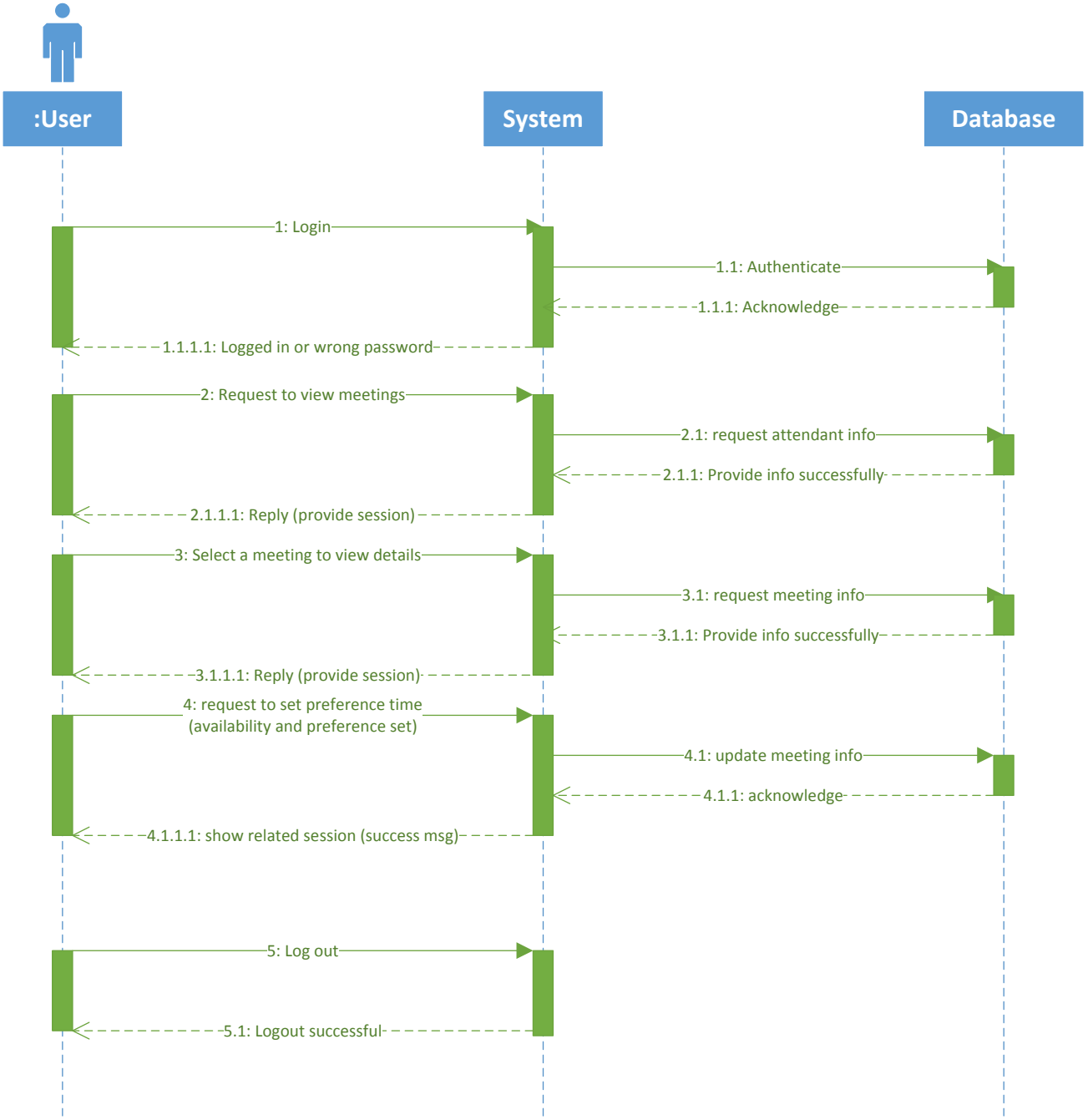
SD. User.viewMeeting()



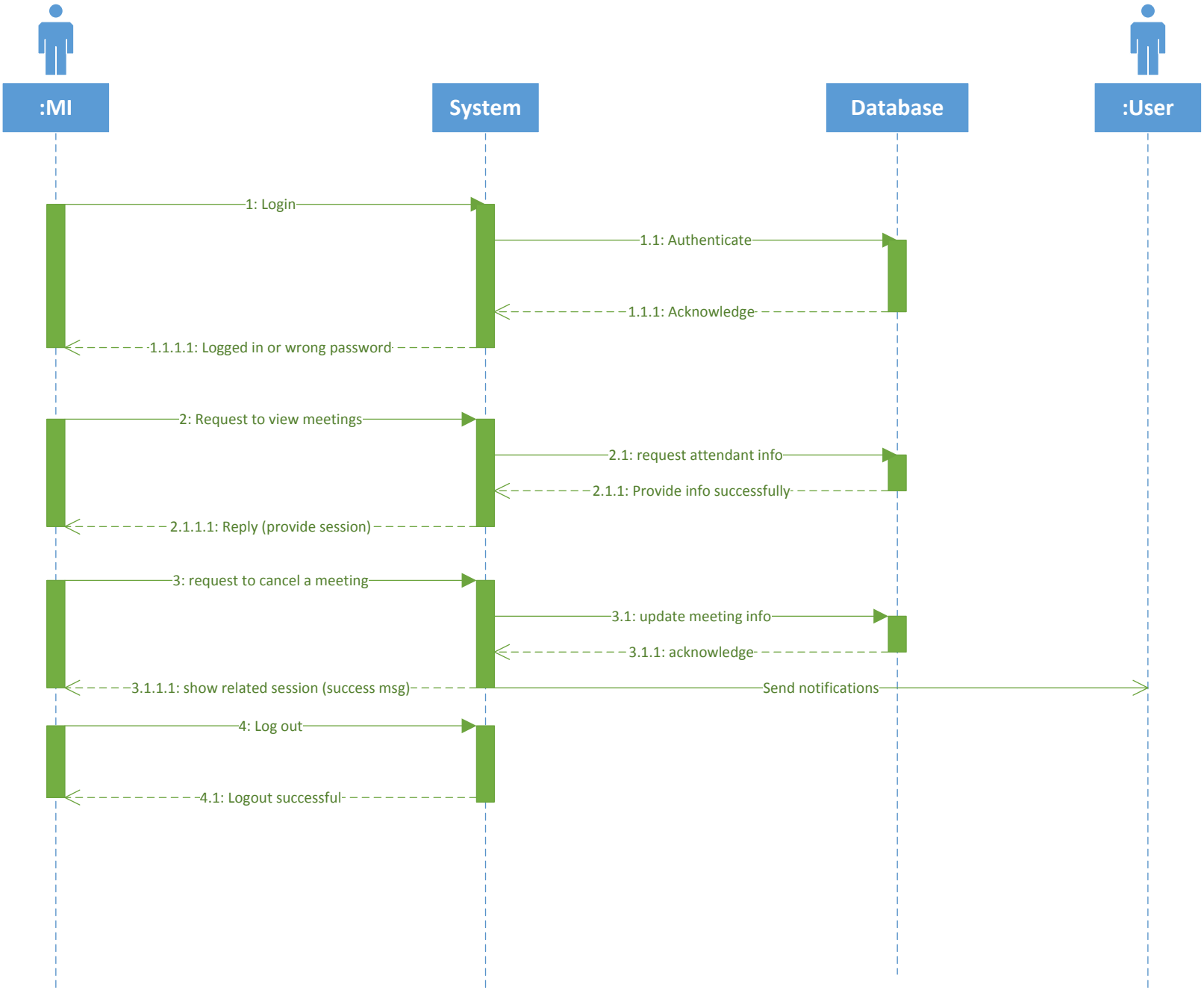
SD. User.viewMeetingDetails()



SD. User.setPreferenceTime()



SD. MI.cancelMeeting()



5. Class Model

5.1 Class Description

5.1.1 Availability Period

5.1.1.1 Attributes

start: Date
end: Date
isAvailable: boolean
preference : int

5.1.1.2 Functions

getStart(): Date
getEnd(): Date
isAvailable(): boolean
getPreference(): int

5.1.2 Availability Record

5.1.2.1 Attributes

elements []:AvailabilityPeriod

5.1.2.2 Functions

apply(AvailabilityPeriod): AvailabilityRecord
update(AvailabilityPeriod): AvailabilityRecord
isAvailable(): boolean
isAvailableBetween(start: Date, end: Date): boolean
getMostPreferredPeriod(): AvailabilityPeriod

5.1.3 Location

5.1.3.1 Attributes

id: int
name: String
availability: AvailabilityRecord

5.1.3.2 Functions

getId(): int
getName(): String
getAvailability():AvailabilityRecord
setAvailability(AvailabilityRecord)

5.1.4 Attendant

5.1.4.1 Attributes

availability: AvailabilityRecord
isImportant: boolean
preferredLocationId: int

5.1.4.2 Functions

getAvalability(): AvailabilityRecord
setAvailability(AvailabilityRecord)
isImportant(): boolean
getPreferredLocationId(): int
setPreferredLocationId(LocationId:int)

5.1.5 Meeting

5.1.5.1 Attributes

Id: int
name: String
attendants: Map <userId,attendant>
avalability: AvailabilityRecord
meetingScheduler: MeetingScheduler

5.1.5.2 Functions

getAvailability():AvailabilityRecord
setAvailability(AvailabilityRecord)
isFinalizable: boolean
finalize(start: Date, end: Date, LocationId: int)
getAttendant(UserId): Attendant
setAttendant(UserId, Attendant)
hasAttendant(UserId)

5.1.6 Meeting Scheduler (singleton object)

5.1.6.1 Attributes

allUsers []: User
allMeetings []: Meeting
allLocations []: Location
notificationService

5.1.6.2 Functions

reloadMeetings()
reloadLocations()
reloadMeetings(UserId)
getUserMeetings(UserId): UserMeetings []
getUserMeetingDetail(UserId,MeetingId): UserMeeting
createMeeting(start, end, locationIds[]): MeetingId:int
login(email:String, pass:String)
destroyMeeting(MeetingId)

5.1.7 Notification Service (interface)

5.1.7.1 Attributes
nothing

5.1.7.2 Functions
Send(email:String, message: String)

5.1.8 User

5.1.8.1 Attributes
Id: int
Email: String
Pass: String
meetingScheduler: MeetingScheduler

5.1.8.2 Functions
getId():int
getEmail: String
getPass(): String //Password
getMeetings() //to implement view meetings
getMeetingDetails()

5.1.9 User Meeting

5.1.9.1 Attributes
meeting: Meeting

5.1.9.2 Functions
getAvailability():AvailabilityRecord
setAvailability(AvailabilityRecord)

5.1.10 PIP Meeting

5.1.10.1 Attributes
Same as user meeting

5.1.10.2 Functions
getLocation(): locationId
setLocation(LocationId)

5.1.11 PP Meeting

5.1.11.1 Attributes
Same as user meeting

5.1.11.2 Functions
Same as user meeting

5.1.12 MI Meeting

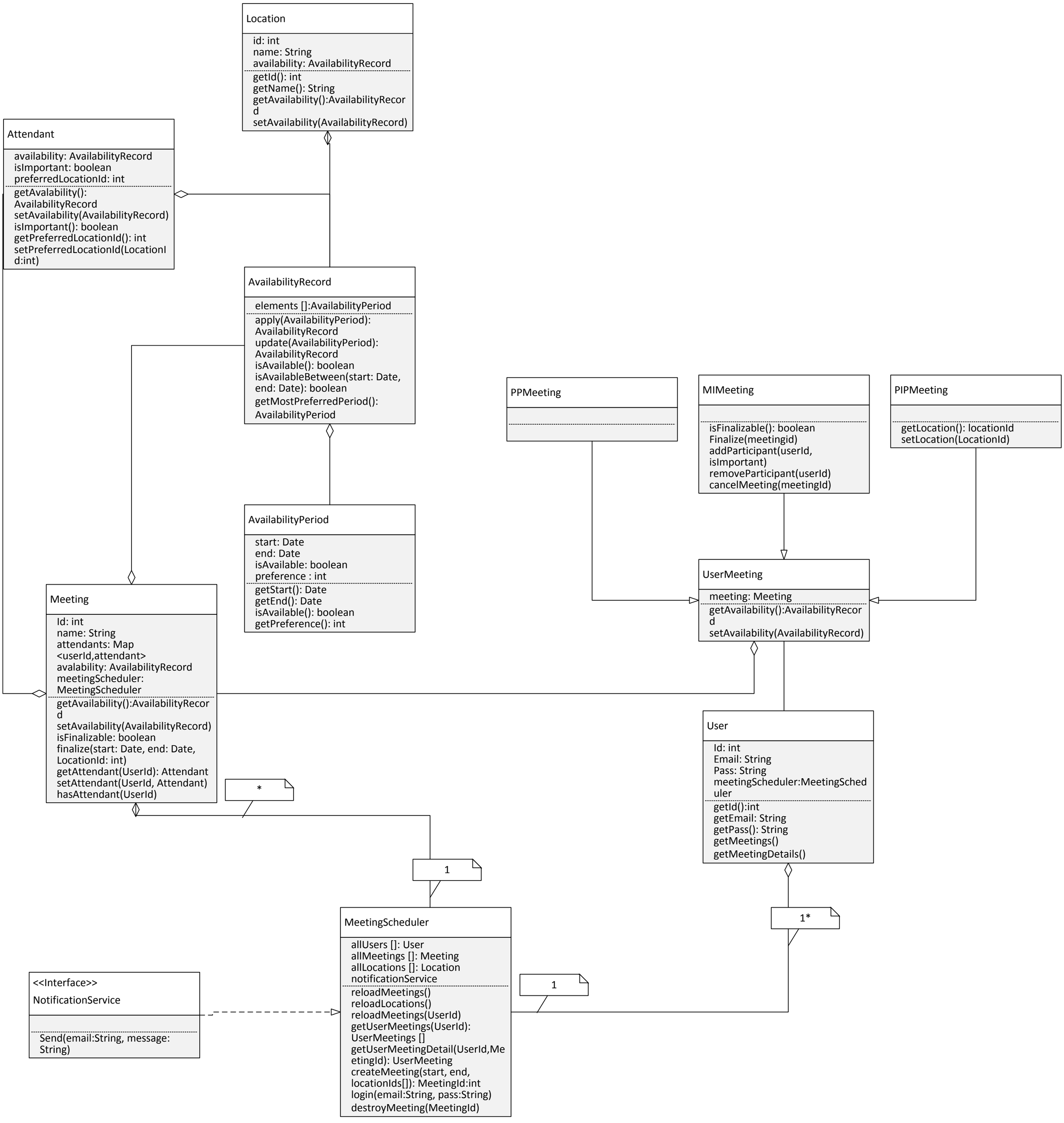
5.1.12.1 Attributes

Same as user meeting

5.1.12.2 Functions

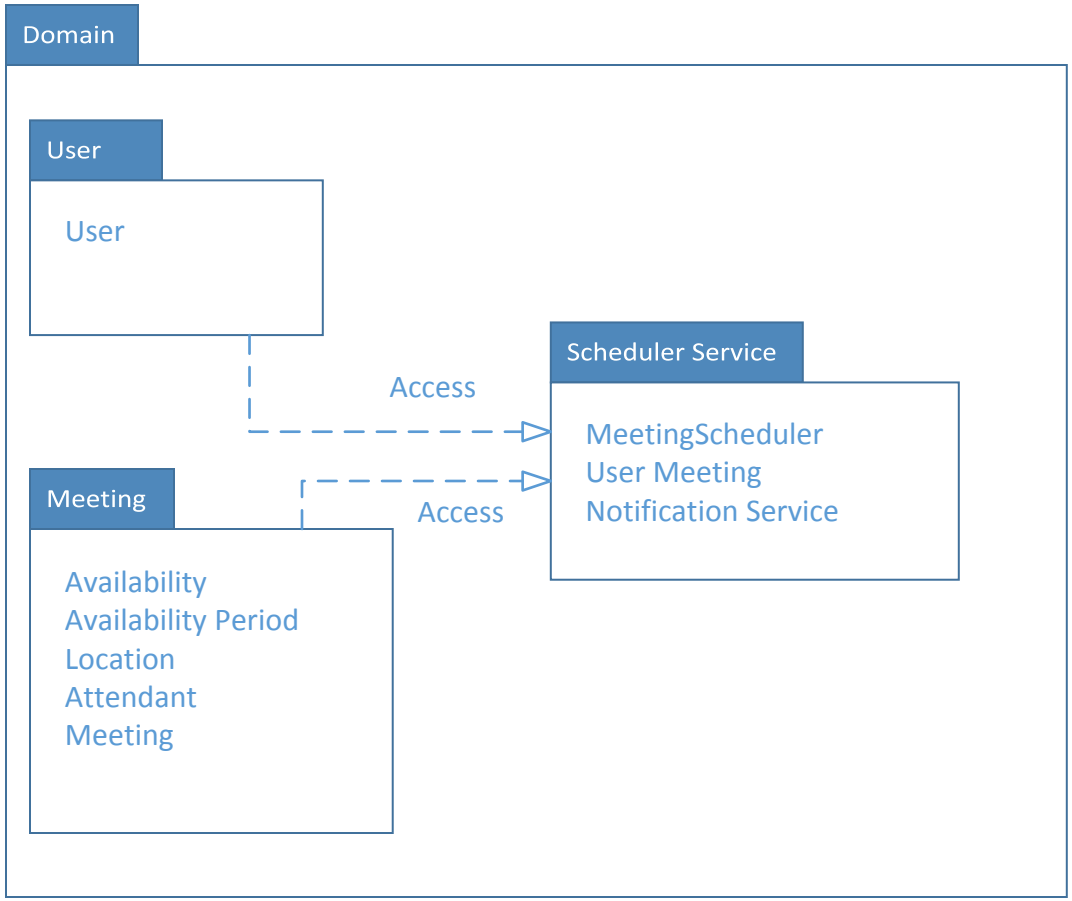
isFinalizable(): boolean
Finalize(meetingid)
addParticipant(userId, isImportant)
removeParticipant(userId)
cancelMeeting(meetingId)

5.2 Class Diagram


Location
id: int
name: String
availability: AvailabilityRecord
getId(): int
getName(): String
getAvailability():AvailabilityRecord
setAvailability(AvailabilityRecord)
Attendant
availability: AvailabilityRecord
isImportant: boolean
preferredLocationId: int
getAvalability(): AvailabilityRecord
setAvailability(AvailabilityRecord)
isImportant(): boolean
getPreferredLocationId(): int
setPreferredLocationId(LocationId:int)
AvailabilityRecord
elements []:AvailabilityPeriod
apply(AvailabilityPeriod): AvailabilityRecord
update(AvailabilityPeriod): AvailabilityRecord
isAvailable(): boolean
isAvailableBetween(start: Date, end: Date): boolean
getMostPreferredPeriod(): AvailabilityPeriod
AvailabilityPeriod
start: Date
end: Date
isAvailable: boolean
preference : int
getStart(): Date
getEnd(): Date
isAvailable(): boolean
getPreference(): int
PPMeeting
MIMeeting
isFinalizable(): boolean
Finalize(meetingid)
addParticipant(userId, isImportant)
removeParticipant(userId)
cancelMeeting(meetingId)
PIPMeeting
getLocation(): locationId
setLocation(LocationId)
UserMeeting
meeting: Meeting
getAvailability():AvailabilityRecord
setAvailability(AvailabilityRecord)
Meeting
Id: int
name: String
attendants: Map <userId,attendant>
avalability: AvailabilityRecord
meetingScheduler: MeetingScheduler
getAvailability():AvailabilityRecord
setAvailability(AvailabilityRecord)
isFinalizable: boolean
finalize(start: Date, end: Date, LocationId: int)
getAttendant(UserId): Attendant
setAttendant(UserId, Attendant)
hasAttendant(UserId)
User
Id: int
Email: String
Pass: String
meetingScheduler:MeetingScheduler
getId():int
getEmail: String
getPass(): String
getMeetings()
getMeetingDetails()
*
1
1*
MeetingScheduler
allUsers []: User
allMeetings []: Meeting
allLocations []: Location
notificationService
reloadMeetings()
reloadLocations()
reloadMeetings(UserId)
getUserMeetings(UserId): UserMeetings []
getUserMeetingDetail(UserId,MeetingId): UserMeeting
createMeeting(start, end, locationIds[]): MeetingId:int
login(email:String, pass:String)
destroyMeeting(MeetingId)
1
<<Interface>>
NotificationService
Send(email:String, message: String)

6. Package Diagram

Package diagrams can be used to show dependencies between different packages in a model. It used to semantically relate elements. We can use package diagrams with different goals. It can contain use cases to show the functionality, contain classes or represent different layers of software and how they communicate with each other.
I draw the package diagram to show the grouping between classes.


Domain
User
User
Scheduler Service
Access
MeetingScheduler
User Meeting
Notification Service
Meeting
Access
Availability
Availability Period
Location
Attendant
Meeting

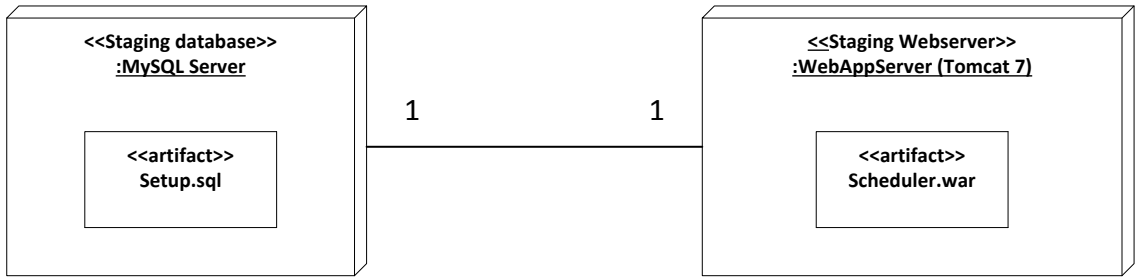
7. Deployment Diagram

A deployment diagram would show the physical deployment of artifacts (components) on nodes. For a web application, the diagram can show what hardware components (nodes) exist (e.g., a web server, an application server, and a database server), what software components run on each node (e.g., web application, database), and how the different pieces are connected (e.g. JDBC, REST, RMI).
Instance level deployment diagram exhibits deployment of instances of artifacts to specific instances of deployment targets. It used for example here to show differences in deployments to development, staging or production environments with the names/ids of specific deployment servers or devices.

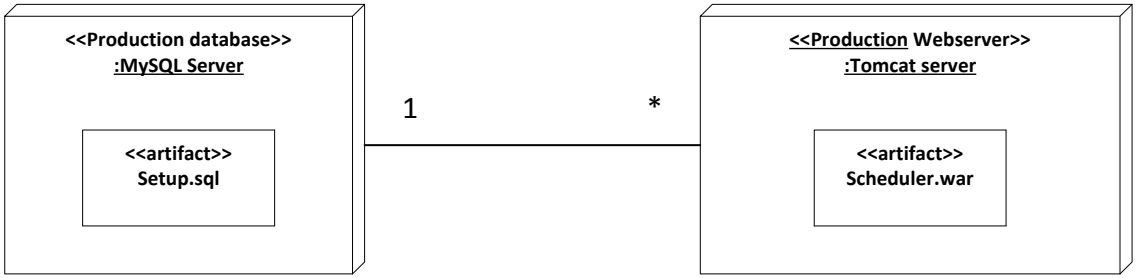
7.1 Staging deployment diagram

7.2 Production deployment diagram

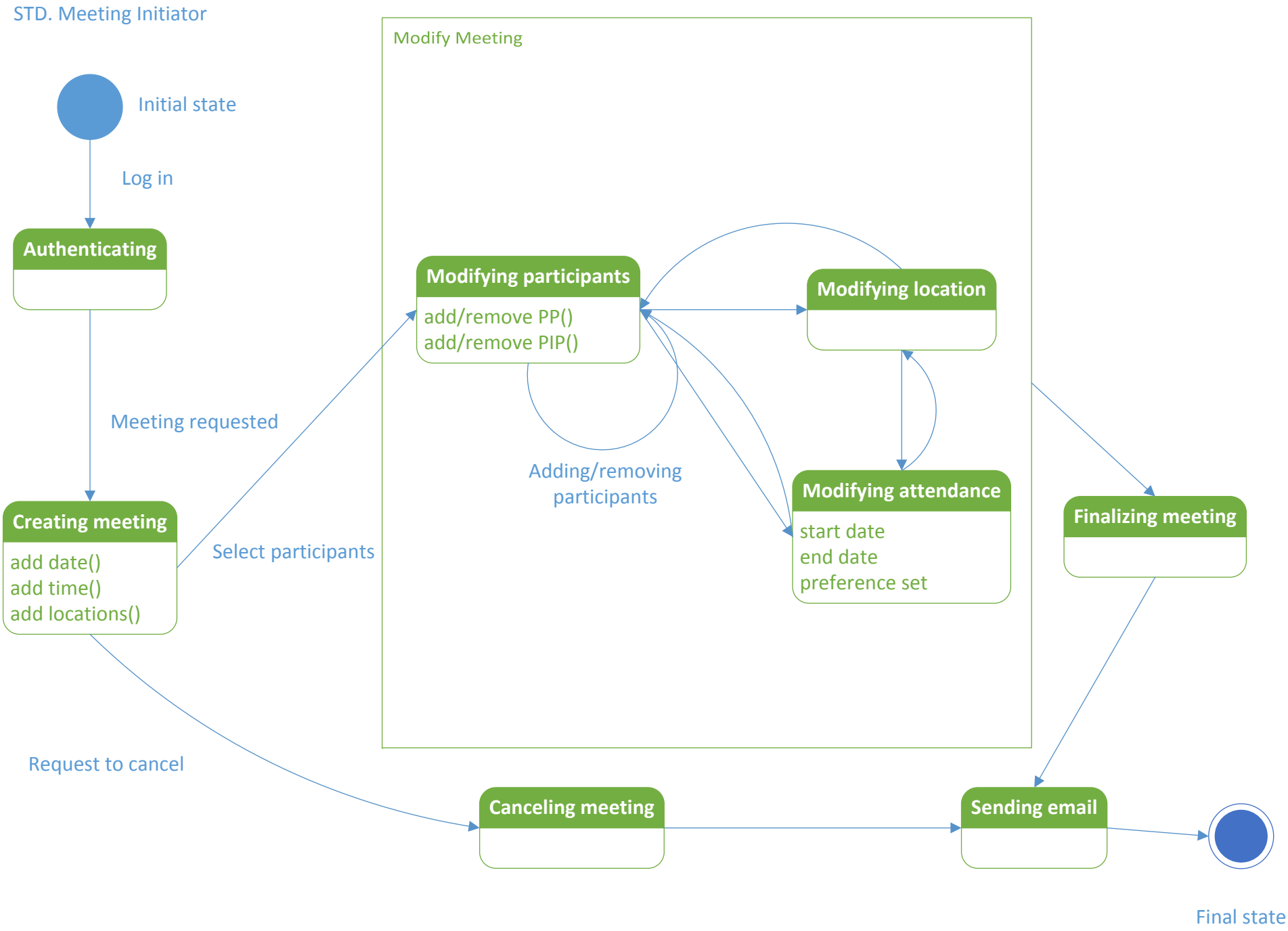
Staging Deployment Diagram



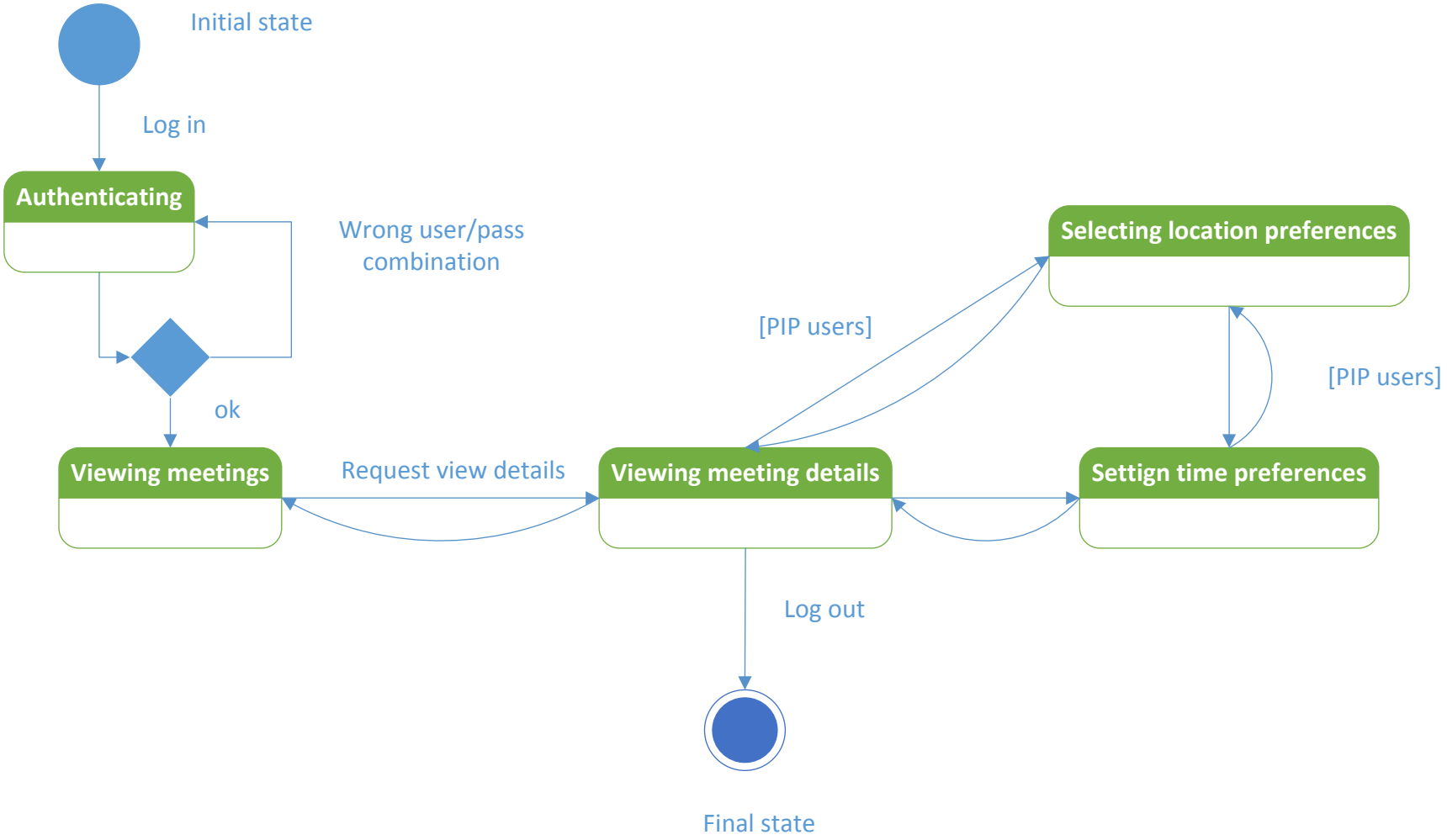
Production Deployment Diagram



8. State Transition Diagram


STD. Meeting Initiator
Modify Meeting
Initial state
Log in
Authenticating
Meeting requested
Creating meeting
add date()
add time()
add locations()
Select participants
Modifying participants
add/remove PP()
add/remove PIP()
Adding/removing participants
Modifying location
Modifying attendance
start date
end date
preference set
Finalizing meeting
Request to cancel
Canceling meeting
Sending email
Final state

STD. Users (PP,PIP,MI)

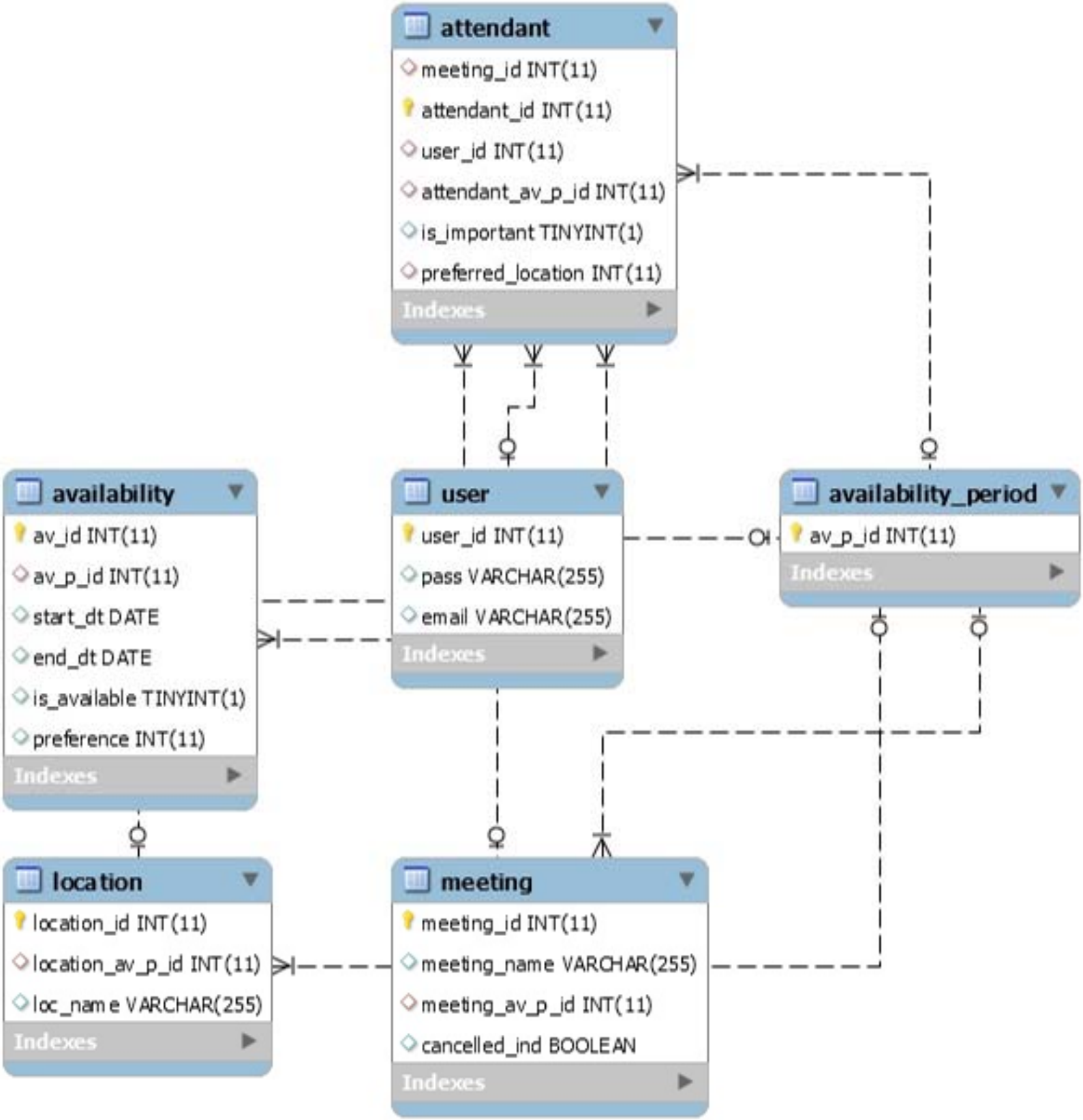


9. Entity Relationship Diagram

MySQL workbench 6.3 is used for this project.

9.1 ERD

This diagram shows the relationship between different database objects (tables).


attendant
meeting_id INT(11)
attendant_id INT(11)
user_id INT(11)
attendant_av_p_id INT(11)
is_important TINYINT(1)
preferred_location INT(11)
Indexes
availability
av_id INT(11)
av_p_id INT(11)
start_dt DATE
end_dt DATE
is_available TINYINT(1)
preference INT(11)
Indexes
user
user_id INT(11)
pass VARCHAR(255)
email VARCHAR(255)
Indexes
availability_period
av_p_id INT(11)
Indexes
location
location_id INT(11)
location_av_p_id INT(11)
loc_name VARCHAR(255)
Indexes
meeting
meeting_id INT(11)
meeting_name VARCHAR(255)
meeting_av_p_id INT(11)
cancelled_ind BOOLEAN
Indexes

9.2 Query scripts

9.2.1 Tables

```
#create database meeting;
CREATE TABLE User
(
user_id int AUTO_INCREMENT primary Key,
pass varchar(255),
email varchar(255)
);

CREATE TABLE Availability_period
(
av_p_id int AUTO_INCREMENT primary key
);

CREATE TABLE Availability
(
av_id int AUTO_INCREMENT primary key,
av_p_id int,
FOREIGN KEY (av_p_id) REFERENCES Availability_period(av_p_id),
start_dt date,
end_dt date,
is_available boolean,
preference int
);

Create table Location
(
location_id      int AUTO_INCREMENT primary key,
location_av_p_id int,
FOREIGN KEY (location_av_p_id) REFERENCES Availability_period(av_p_id),
loc_name varchar(255)
);


Create table Meeting
(
        meeting_id int auto_increment primary key,
        meeting_name varchar(255),
        meeting_av_p_id int,
        FOREIGN KEY (meeting_av_p_id) REFERENCES Availability_period(av_p_id),
   cancelled_ind boolean
);

Create table Attendant
(
meeting_id int,
FOREIGN KEY (meeting_id) REFERENCES Meeting(meeting_id),
attendant_id int primary key,
user_id int,
```

```
FOREIGN KEY (user_id) REFERENCES User(user_id),
attendant_av_p_id int,
FOREIGN KEY (attendant_av_p_id) REFERENCES Availability_period(av_p_id),
is_important boolean,
preferred_location int,
foreign key (preferred_location) references Location(location_id)
);
```