**Schedule Builder**

**Low Level Design Document**

Student Multi-Tool

---

03.29.2022

**Team Marvel**

Albert Toscano

Audrey Brio

Bradley Nickle

Devarsh Patel

Jacob Delgado (Team Leader)

Joseph Cutri

SzeMan Tang

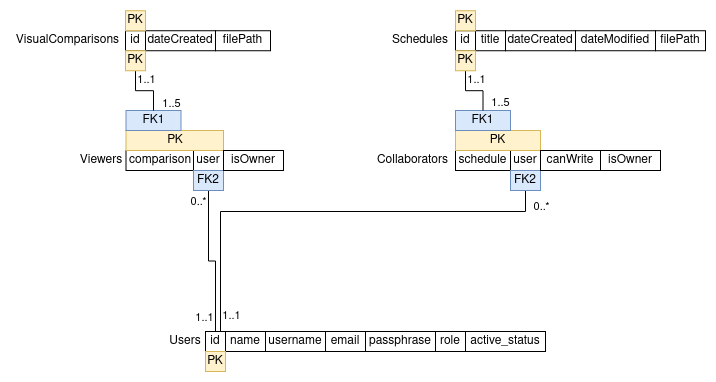
**Technologies Used**

* C#
* .Net
* Ecmascript
* SQL Server 2019 Express Edition
* SQL Server Management Studio
* Video Studio Code and Visual Studio Community
* jQuery
* Vue.js

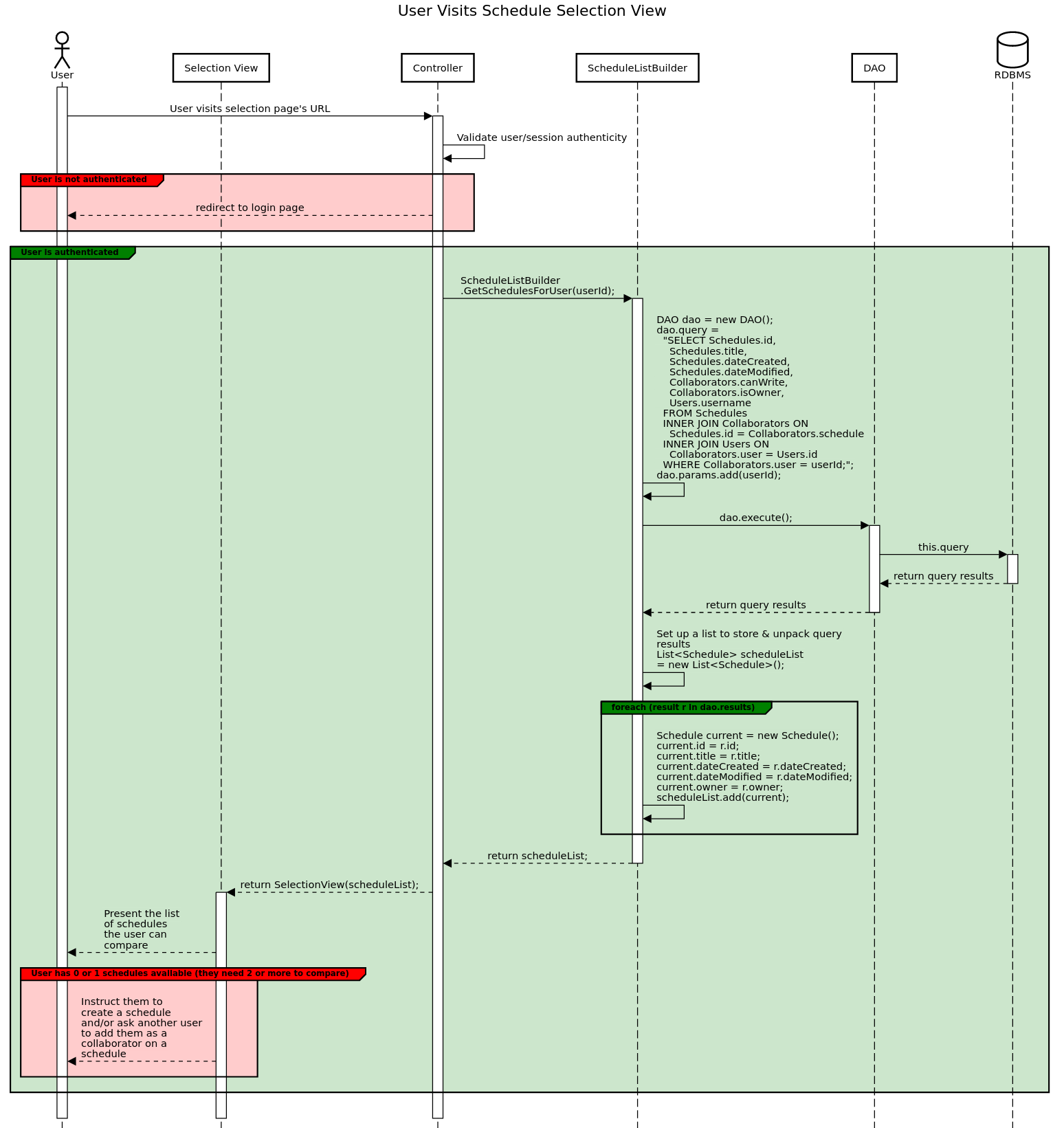
**Database Diagram**

Since schedule comparisons are visually represented to the user in a similar way to schedules, code for comparisons is intended to be similar to the code for schedules. Comparisons are stored in a similar way to schedules; the file path for a given comparison is stored in the database.

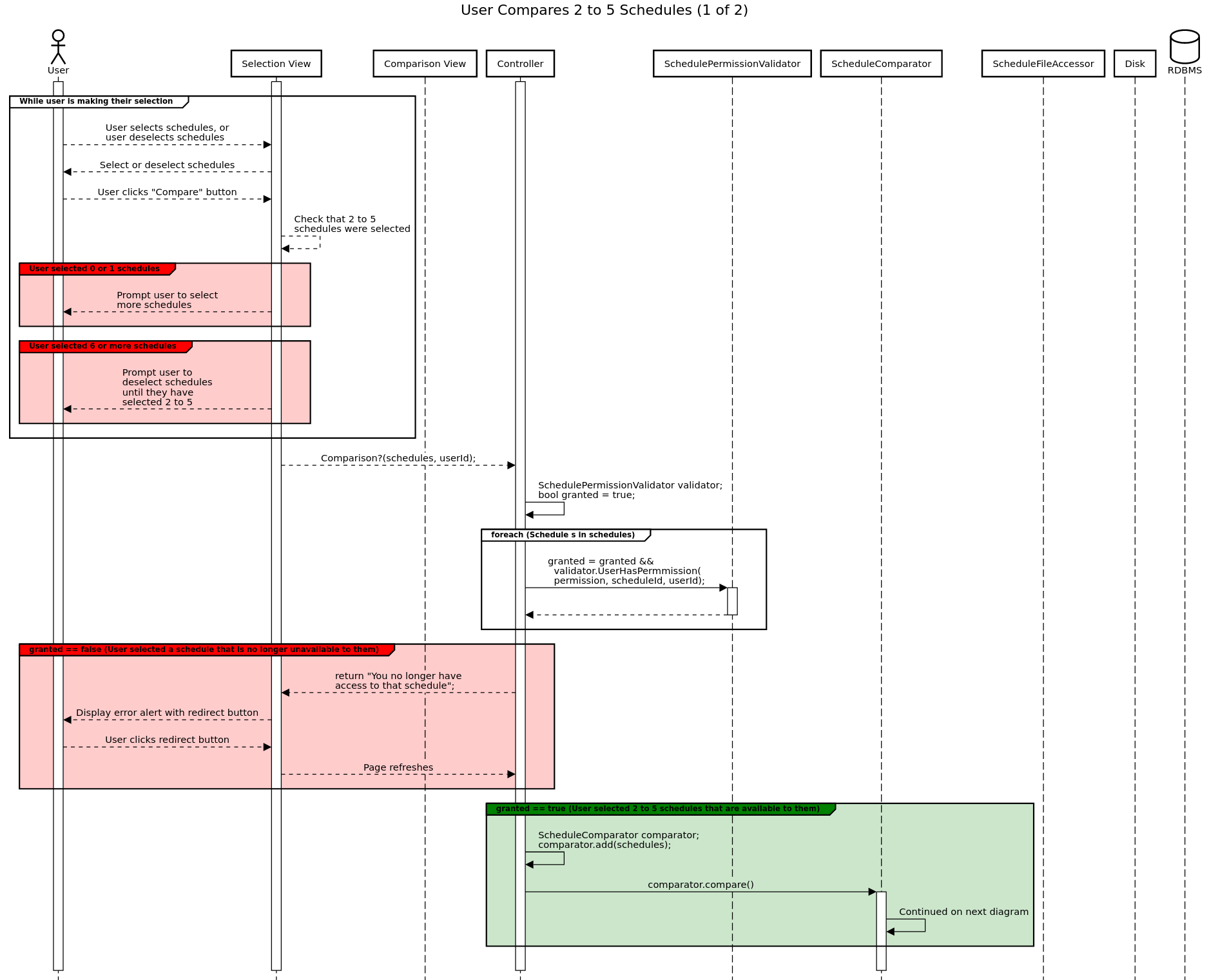
Users are associated with comparisons via the Viewers table. Presence of a comparison-user pair in the Viewers table implies the “view” permission. The owner of a comparison is simply the user who created the comparison, and they have the ability to delete the comparison.

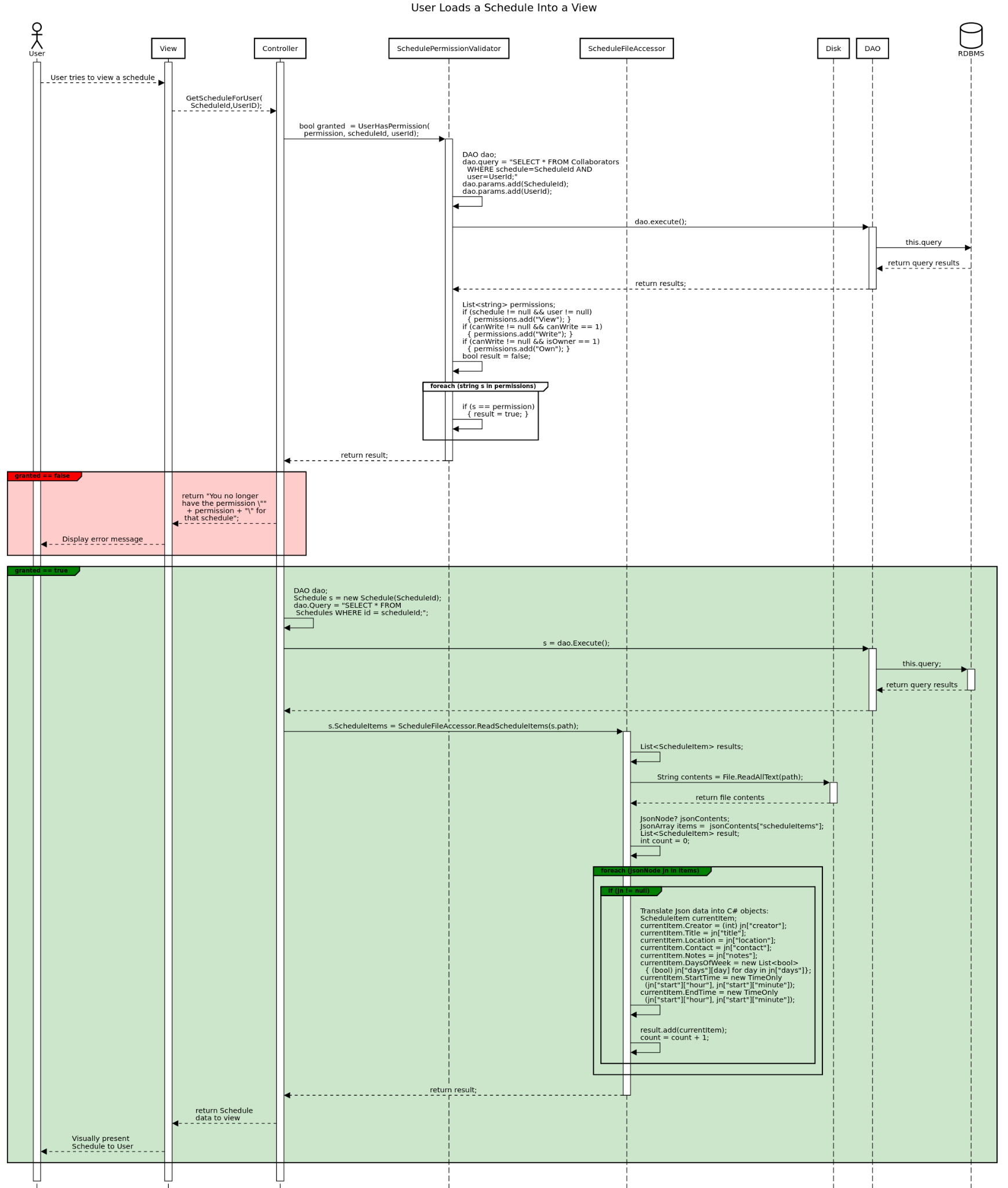
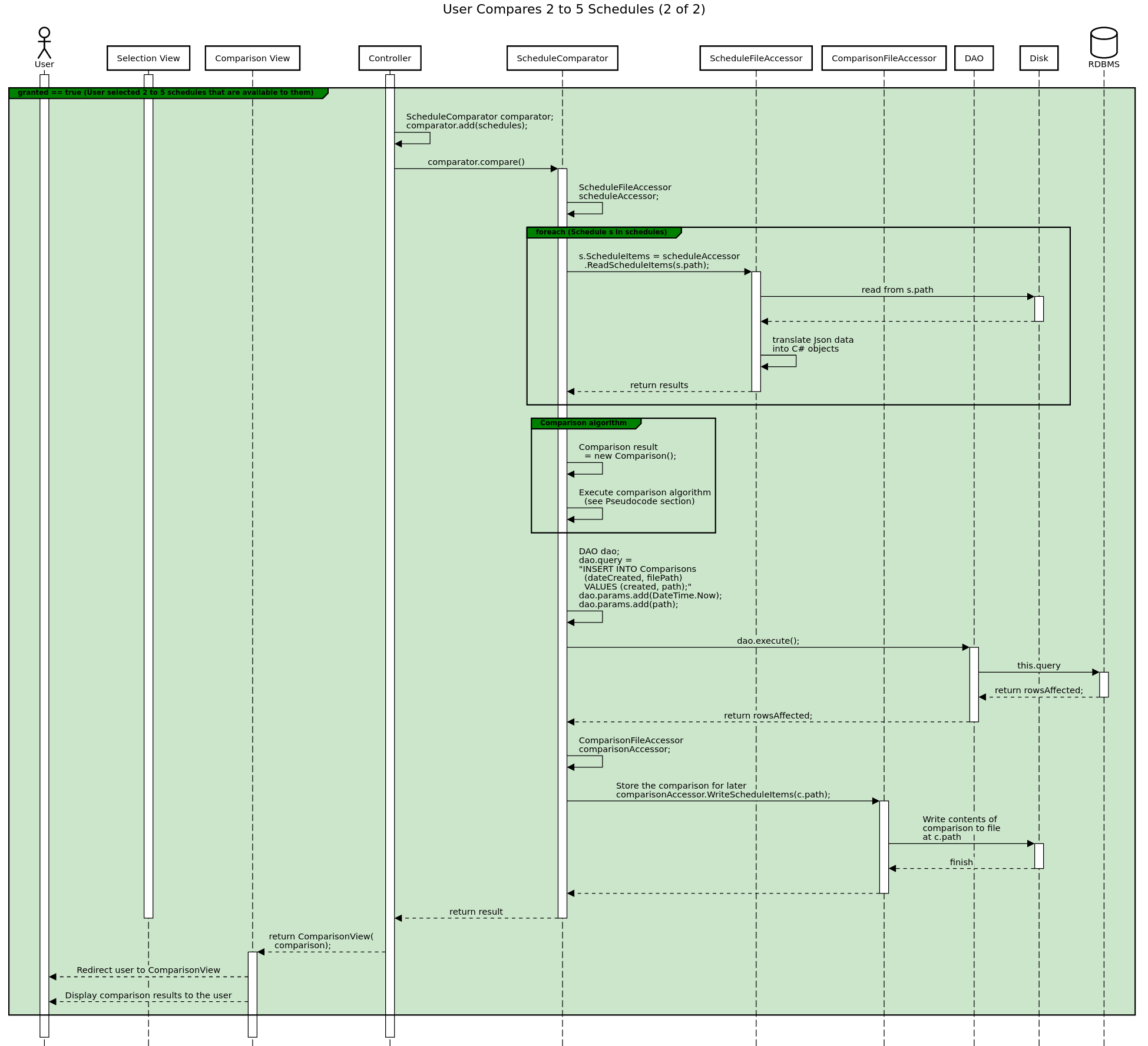


**Sequence Diagrams**

Comparison algorithms have been omitted from sequence diagrams to improve the readability of the diagrams. See the Pseudocode section for the comparison algorithms.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  


This diagram is continued on the next page. The assumption is made that this diagram is preceded by the sequence “User Visits Schedule Selection View”.





**Pseudocode**

The schedule comparison feature has two main goals: provide users with a visual representation of their shared free time, and enable the matching of users by shared free time. Both goals can be solved with very similar algorithms, differing mainly in the output. For “human-readable” comparisons, a schedule with “free time” schedule items is returned. For comparisons to be used in matching, the quantity of free time in minutes is returned.

The algorithms are based on “early bounds” and “late bounds”, which refer to the beginning and end of free time (“openings” or ”gaps”) in a schedule. Global bounds are used as starting points and ending points for the algorithm, and can be set individually for every comparison. Two more bounds are used while iterating over schedule items to determine when free time begins and ends.

Which schedule a schedule item belongs to is ignored, since the goal is not to look for Alice’s free time, or Bob’s; it is to look for free time that they both have.

Below is the algorithm for comparing schedules, to be used in visual comparisons.

**Schedule** *compareForVisualRepresentation*(List<Schedules> *l*)

configure *global early bound*

configure *global late bound*

*days* = each day of the week from Sunday to Saturday (inclusive)

**create a new** Schedule "*result*" // to store results in

**for each** *day* **in** *days*: // Store & sort all items for each day

**create a new** *heap* for ScheduleItems

**for each** Schedule *s* **in** *l*:

**for each** ScheduleItem *si* **in** *s*:

**if** *day*.*name* **in** *si*.*days*: // add each day’s items to its heap

*heap*.add(*si*)

// Comparison

*current early bound* = *global early bound*

**for each** ScheduleItem *si* **in** *heap*:

*current late bound* = *si.start*

*result*.add(**new** ScheduleItem(*start*=*current early bound,*

*end=current late bound*))

*current early bound* = *si.end*

*result*.add(**new** ScheduleItem(*start*=*current early bound,*

*end=global late bound*))

**delete** *heap*

// Repeat for next day

**return** *result*

Below is the algorithm for comparing schedules, to be used in matching. Since the items are sorted before comparison, late bounds are “greater” than early bounds, and therefore their difference will always be greater than or equal to zero. This can be guaranteed by subtracting the start and end time. The resulting TimeSpan object has a property, TotalMinutes, which can be used to obtain the duration of a ScheduleItem. Comparisons of schedules with more shared free time should produce larger integers.

**integer** *compareForMatching*(List<Schedules> *l*)

configure *global early bound*

configure *global late bound*

*days* = each day of the week from Sunday to Saturday (inclusive)

*result* = 0

**for each** *day* **in** *days*: // Store & sort all items for each day

**create a new** *heap* for ScheduleItems

**for each** Schedule *s* **in** *l*:

**for each** ScheduleItem *si* **in** *s*:

// check that si doesn’t overlap with anything in the heap

*found* = *false*

**for each** ScheduleItem *x* **in** *heap*:

**if** *x[day]* **==** *si[day]* **and** (*x.start* **is** *si.start* **or** *x.end* ***is*** *si.end)*:

*found* = *true*

// if an overlap was found, update the x in the heap to have the

*x.start* = **earliestTime**(*x.start*, *si.start*)

*x.end* = **latestTime**(*x.end, si.end*)

**break**

// if si doesn’t overlap with any item in the heap, insert it

**if not** *found* **and** *day*.*name* **in** *si*.*days*:

*heap*.add(*si*)

// Comparison

*current early bound* = *global early bound*

**for each** ScheduleItem *si* **in** *heap*:

*current late bound* = *si.start*

*Duration* =(*current late bound*) - (*current early bound*)

*result* += *duration*

*current early bound* = *si.end*

*Duration* = (*global late bound*) - (*current early bound*)

*result* += *duration*

**delete** *heap*

// Repeat for next day

**return** *result*