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

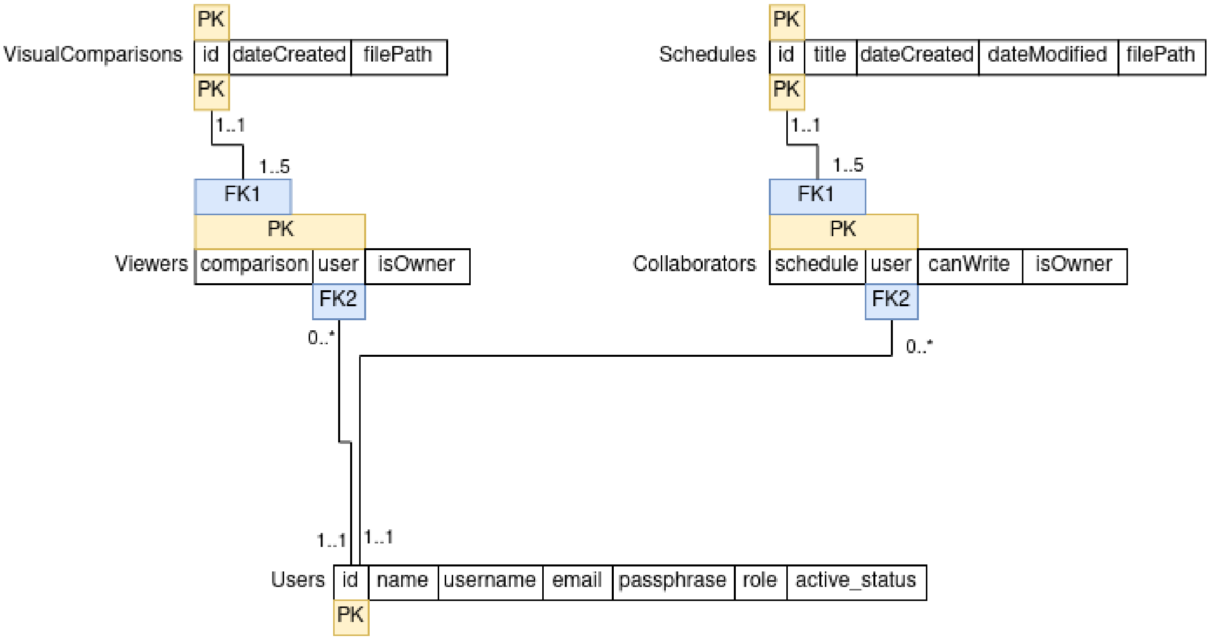
Technologies Used

- 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.

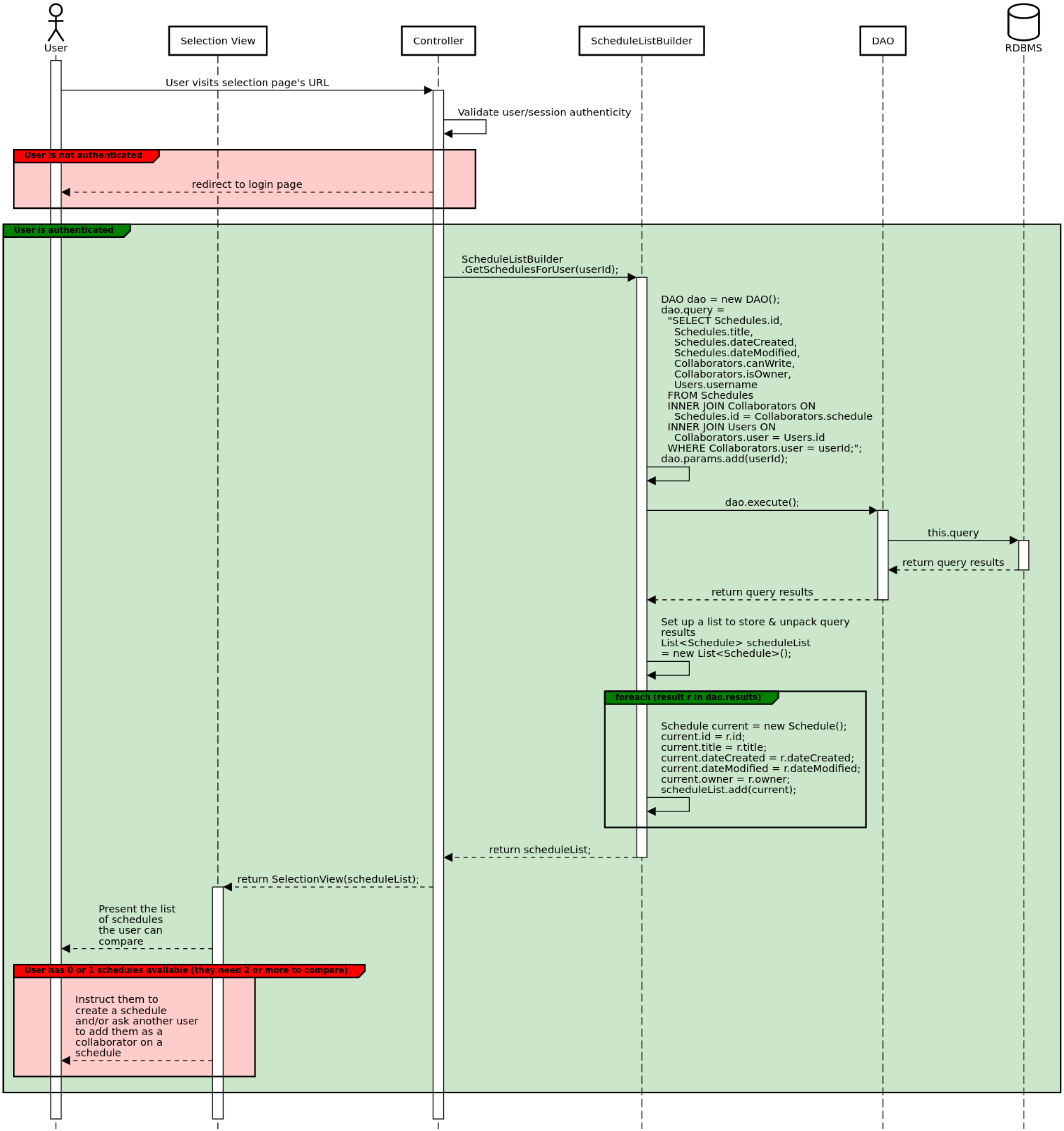
Users are associated with comparisons via the Viewers table. Presence of a comparison-user pair in the Viewers table implies the “view” permission. The owner of a comparison is simply the user who created the comparison, and they have the ability to delete the comparison.



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.

User Visits Schedule Selection View


User
Selection View
Controller
ScheduleListBuilder
DAO
RDBMS
User visits selection page's URL
Validate user/session authenticity
User is not authenticated
redirect to login page
User is authenticated
ScheduleListBuilder
.GetSchedulesForUser(userId);

```
DAO dao = new DAO();
dao.query =
  "SELECT Schedules.id,
    Schedules.title,
    Schedules.dateCreated,
    Schedules.dateModified,
    Collaborators.canWrite,
    Collaborators.isOwner,
    Users.username
  FROM Schedules
  INNER JOIN Collaborators ON
    Schedules.id = Collaborators.schedule
  INNER JOIN Users ON
    Collaborators.user = Users.id
  WHERE Collaborators.user = userId;";
dao.params.add(userId);
```

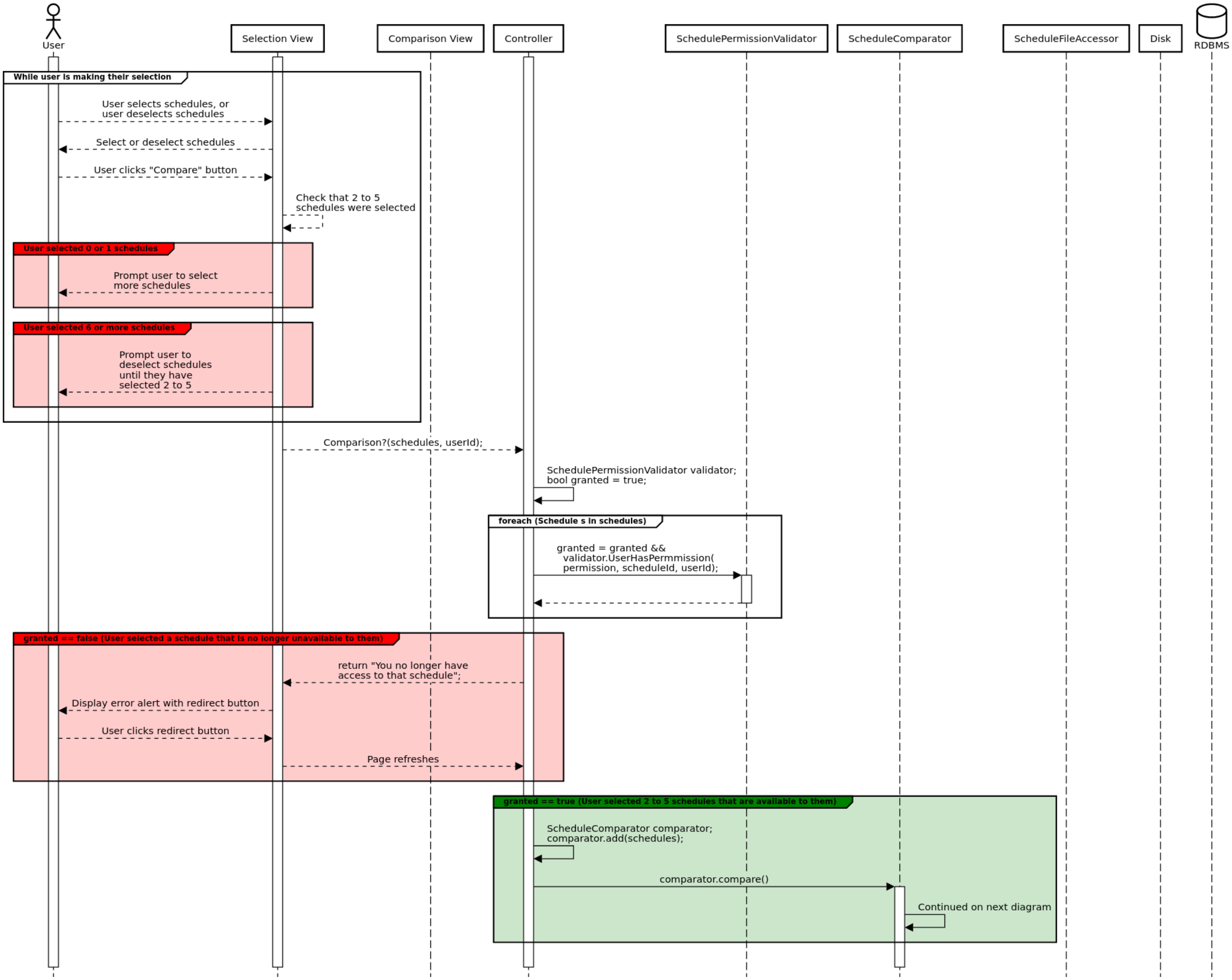
```
Set up a list to store & unpack query
results
List<Schedule> scheduleList
= new List<Schedule>();
```

```
Schedule current = new Schedule();
current.id = r.id;
current.title = r.title;
current.dateCreated = r.dateCreated;
current.dateModified = r.dateModified;
current.owner = r.owner;
scheduleList.add(current);
```

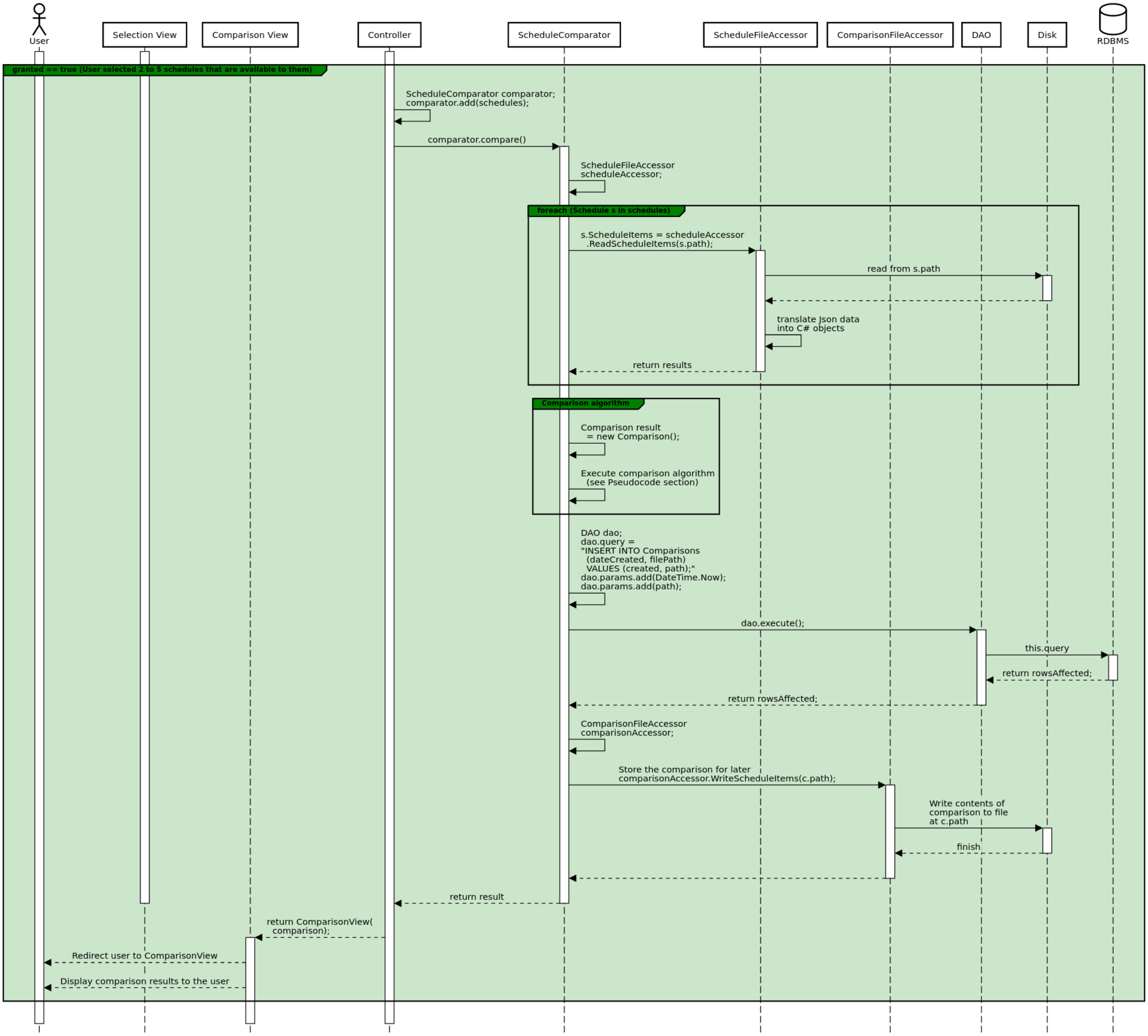
return scheduleList;
return SelectionView(scheduleList);
Present the list of schedules the user can compare
User has 0 or 1 schedules available (they need 2 or more to compare)
Instruct them to create a schedule and/or ask another user to add them as a collaborator on a schedule

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”.

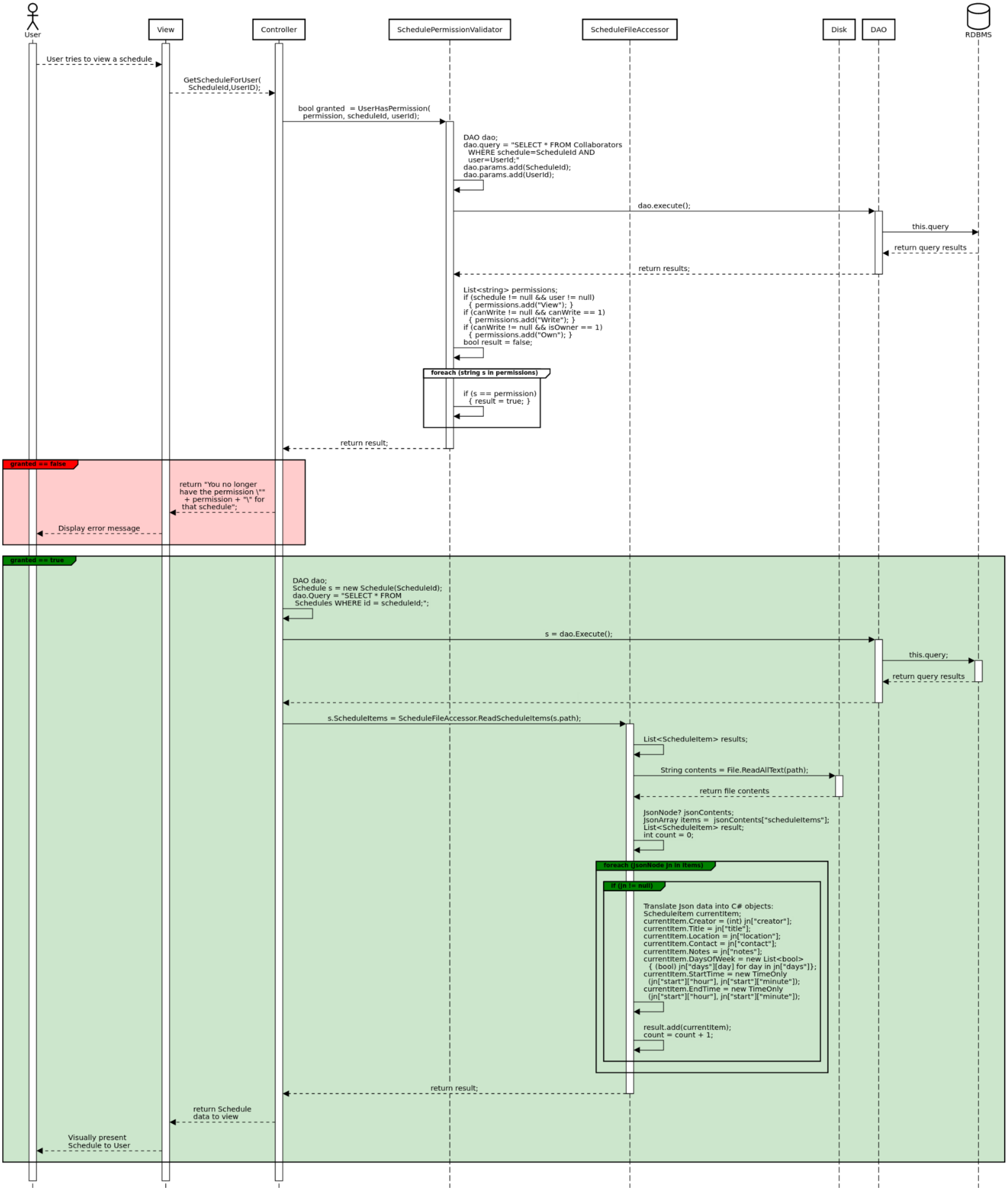
User Compares 2 to 5 Schedules (1 of 2)


User
Selection View
Comparison View
Controller
SchedulePermissionValidator
ScheduleComparator
ScheduleFileAccessor
Disk
RDBMS
While user is making their selection
User selects schedules, or
user deselects schedules
Select or deselect schedules
User clicks "Compare" button
Check that 2 to 5
schedules were selected
User selected 0 or 1 schedules
Prompt user to select
more schedules
User selected 6 or more schedules
Prompt user to
deselect schedules
until they have
selected 2 to 5
Comparison?(schedules, userId);
SchedulePermissionValidator validator;
bool granted = true;
foreach (Schedule s in schedules)
granted = granted &&
validator.UserHasPermmission(
permission, scheduleId, userId);
granted == false (User selected a schedule that is no longer unavailable to them)
return "You no longer have
access to that schedule";
Display error alert with redirect button
User clicks redirect button
Page refreshes
granted == true (User selected 2 to 5 schedules that are available to them)
ScheduleComparator comparator;
comparator.add(schedules);
comparator.compare()
Continued on next diagram

User Compares 2 to 5 Schedules (2 of 2)



User Loads a Schedule Into a View



```
GetScheduleForUser(
  ScheduleId,UserID);
```

```
bool granted  = UserHasPermission(
  permission, scheduleId, userId);
```

```
DAO dao;
dao.query = "SELECT * FROM Collaborators
  WHERE schedule=ScheduleId AND
  user=UserId;"
dao.params.add(ScheduleId);
dao.params.add(UserId);
```

```
dao.execute();
```

```
List<string> permissions;
if (schedule != null && user != null)
  { permissions.add("View"); }
if (canWrite != null && canWrite == 1)
  { permissions.add("Write"); }
if (canWrite != null && isOwner == 1)
  { permissions.add("Own"); }
bool result = false;
```

```
foreach (string s in permissions)
if (s == permission)
  { result = true; }
```

```
return "You no longer
have the permission \""
  + permission + "\" for
that schedule";
```

```
DAO dao;
Schedule s = new Schedule(ScheduleId);
dao.Query = "SELECT * FROM
 Schedules WHERE id = scheduleId;";
```

```
s = dao.Execute();
```

```
s.ScheduleItems = ScheduleFileAccessor.ReadScheduleItems(s.path);
```

```
List<ScheduleItem> results;
```

```
String contents = File.ReadAllText(path);
```

```
JsonNode? jsonContents;
JsonArray items =  jsonContents["scheduleItems"];
List<ScheduleItem> result;
int count = 0;
```

```
Translate Json data into C# objects:
ScheduleItem currentItem;
currentItem.Creator = (int) jn["creator"];
currentItem.Title = jn["title"];
currentItem.Location = jn["location"];
currentItem.Contact = jn["contact"];
currentItem.Notes = jn["notes"];
currentItem.DaysOfWeek = new List<bool>
  { (bool) jn["days"][day] for day in jn["days"]};
currentItem.StartTime = new TimeOnly
  (jn["start"]["hour"], jn["start"]["minute"]);
currentItem.EndTime = new TimeOnly
  (jn["start"]["hour"], jn["start"]["minute"]);
```

```
result.add(currentItem);
count = count + 1;
```

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
```