All User Use Case

Use Case: Login

- Primary Actor: User —> Student, Tutor, Administrator
- Scope: Login and Validating user
- Level: User
- Brief: The user logs into the application.
- Stakeholders: Not sure…
- Postconditions: If successful the user gets logged into the application
- Preconditions: User has account
- Triggers: User wants to login
- Basic flow:
  1. The system requests that the user enter their Point Park email and password.
  2. The user enters their email and password.
  3. The system validates the entered email and password and logs the user into the application.
- Extensions: Depending on type of user, privileges limit what the user will do.

API:

```
{
  “action”: “validate_user”,
  “email”: “some_user@pointpark.edu”,
  “password”: “password”
}
=>
{
  “success”: true,
  “message”: “user was logged in.”
}
or
{
 “success”: false
  “message”: “user was logged in.”
}
```

Student Use Cases

Use Case: Student schedules with tutor
• Primary Actor: User [Student]
• Scope: Tutor list of availability and course specialties.
• Level: Student goal
• Brief: Student selects course they need tutoring, student gets presented with list of available tutors and the days/hours of their availability.
• Stakeholders: Not sure…
• Postconditions: Student gets scheduled with tutor
• Preconditions: Login
• Triggers: Student request tutor in a given course
• Basic flow:

1. Student wants to find tutor in a course.
2. The student enters their course, day range they can attend the tutoring center, along with the hours.
3. Clicks find tutor
4. Student gets presented with list of tutors that meet criteria.
5. Student selects tutor and creates meeting with them.

• Extensions:
API:
```
{
  “action”: “find_tutor”,
  “course_id”: “CMPS-480”,
  “student_day_range”: [start_day, end_day],
  “student_hour_range”: [start_time, end_time]
}
=>
{
  “success”: true,
  “message”: “Tutor was found.”
  “tutor_id”: “…”,
  “tutor_name”: “…”,.
  “scheduled_day”: “…”,
  “scheduled_time”: “…”,
}
or
{
 “success”: false
  “message”: “No tutor was found in your entered course or are available in that date range.”
}
```

Use Case: Student Starts Tutoring Session

- Primary Actor: User [Student]
- Scope: Log student session with tutor
- Level: Student goal
- Brief: Create a timestamp of when tutoring session started, then logout when completed
- Stakeholders: Not sure..
- Postconditions: Tutoring session logged
- Preconditions: Student was schedule with a tutor
- Triggers: Student initiates tutoring session
- Basic flow:
  1. Student enters username or school ID.
  2. Gets prompted to select tutoring center —> Math, Writing, 1-1 Tutoring.
  3. Session timestamp gets created.
  4. After session Student re-enters username or ID to end session.
  5. Student gets prompted questionnaire about the tutor the listed.
  6. Session ending timestamp is created.
  7. Logic to ensure no visit that was not logged out is not added to DB for bad data.
- Extensions: Tutor and Admin will be to log a student in. Admin can edit the session info.

API:

```
{
  “action”: “start_session”,
  “student_id”: “…”,
  “start_datetime”: “YYYY-MM-DD HH:MM:SS”
}
=>
{
  “success”: true,
  “end_datetime”: “YYYY-MM-DD HH:MM:SS”,
  “tutor_id”: “…”,
  “course_id”: “…”,
  “session_message”: “…”
}
or
{
 “success”: false
 “message”: “Student was never logged out.”
}
```

<u>Tutor Use Cases</u>

Use Case: Adding a new tutor or editing tutor info.

- Primary Actor: User [Tutor]
- Scope: Tutor availability & course specialties
- Level: Tutor goal
- Brief: Tutor enters course they can tutor in and also set their schedule
- Stakeholders: Not sure…
- Postconditions: Tutor is registered and schedule is created
- Preconditions: New Tutor
- Triggers: New tutor is added
- Basic flow:
  1. New tutor is added.
  2. Tutor selects course specialties.
  3. Tutor selects reason specialties Math, Writing, HW help, Test help.
  4. Tutor selects days and hour they will be working.
  5. Tutor's info is added.
- Extensions: Tutor is able their info whenever, Admin will be able to edit all tutors schedule.

API:

```
{
  “action”: “create_tutor”,
  “tutor_id”: “id gets created”,
  “tutor_name”: “…”,
  “course_code_specialties_”: [“…”, “…”],
  “reason_specialties”: [“…”, “…”],
  “days_available”: “…”,
  “hours_available”: “..”
}
=>
{
  “success”: true,
  “message”: “Tutor has been successfully added.”
}
or
{
 “success”: false
 “message”: “Error adding new tutor”
}
```

Use Case: View log of sessions
• Primary Actor: User [Tutor]
• Scope: Log of tutoring sessions
• Level: Tutor goal
• Brief: Tutor can looks back at previous session with a student
• Stakeholders: Not sure…
• Postconditions: Log of session is created
• Preconditions: Student had session with tutor
• Triggers: Tutor or admin needs to review session history
• Basic flow:
  1. Tutor/Admin wants to view session logs.
  2. User enters search criteria.
  3. Table gets generated.
  4. Admin will be able results.
• Extensions: Admin will be able to view all tutors previous sessions.

API:
```
{
  “action”: “session_log”,
  “tutor_id”: “…”,
  “course_code”: “…”,
  “reason”: “…”,
  “student_name”: “…”,
  “day_of_session”: “…”,
  “time_of_session”: “..”,
}
=>
{
  “success”: true,
  “message”: “Student session table generation successful”,
  “session_id”: “…”,
  “student_id”: “..”,
  “student_name”: “..”,
  “date”: “..”,
  “time_in”: “..”,
  “time_out”: “..”,
  “duration”: “..”,
  “center”: “..”,
  “course”: “..”,
  “reason”: “..”,
  “tutor”: “..”
}
or
{
 “success”: false
 “message”: “Error creating session table.”
}
```

Admin Use Cases

Use Case: View list of tutors

- Primary Actor: Administrator
- Scope: Tutor list
- Level: Admin goal
- Brief: Admin can view the list of tutors and edit their schedule and course specialties
- Stakeholders: Not sure…
- Postconditions: List of tutors is generated
- Preconditions: Tutor has entered the info
- Triggers: Admin needs to view/edit list
- Basic flow:
  1. Admin enters specific query requirement.
  2. Sql sends back table view.
  3. Admin can edit the values returned.
- Extensions:

API:

```
{
  “action”: “tutor_list”,
  // List of search criteria is admin request one.
}
=>
{
  “success”: true,
  “tutor_id”: “…”,
  “tutor_name”: “..”,
  “tutor_scheduled_days”: [“…”,”…”],
  “tutor_scheduled_hours_range”: “…”,
  “course_code_specialties_”: [“…”, “…”],
  “reason_specialties”: [“…”, “…”],
}
or
{
  “success”: false,
  “message”: “Error view the list of students.”
}
```

Use Case: Admin gives privileges
- Primary Actor: Administrator
- Scope:
- Level: Admin Goal
- Brief: Admin will the ability to give accounts tutor and admin privileges and revoke them.
- Stakeholders: Not sure…
- Postconditions: User has new permissions
- Preconditions: Account has been created
- Triggers: Student becomes a tutor, new tutor added, or tutor gets admin privileges.
- Basic flow:
  1. User account is to be prompted to tutor or admin.
  2. Existing admin gives permission to the user account.
  3. Account now has the given privileges.
- Extensions:

API:
```
{
  “admin_id”: “…”,
  “user_id”: “…”,
  “is_admin”: true,
  “is_tutor”:  true

}
=>
{
  “success”: true,
  “message”: “User has been added as either tutor or admin”
}
or
{
  “success”: false,
  “message”: “Error handling request”
}
```