Algorithm 1: Authenticate Student Using PID

```
Input: PID (username), Password
Output: Authentication status (Success/Failure)
Display "Enter PID:";
Read PID;
Display "Enter Password:";
Read Password;
if PID or Password is empty then
    Display "Error: PID and Password cannot be empty.";
    return "Failure";
end
if CheckCredentials(PID, Password) then
    Display "Initiating 2-Factor Authentication...";
    if DuoPushAuthentication() then
        Display "Login Successful.";
        RedirectToDashboard();
        return "Success";
    end
    else
        Display "Error: Duo Push Failed.";
        return "Failure";
    end
end
else
    Display "Error: Invalid Credentials.";
    return "Failure";
end
```

Algorithm 2: Check Course Requirements

```
Input: StudentID
Output: List of Completed, Incomplete, and Suggested Courses
Validate StudentID;
if StudentID is invalid then
    Display "Error: Invalid Student ID.";
    return "Failure";
end
CompletedCourses ← FetchCompletedCourses(StudentID);
DegreeRequirements ← FetchDegreeRequirements(StudentID);
IncompleteCourses ← DegreeRequirements - CompletedCourses;
Display "Courses Completed:" CompletedCourses;
Display "Courses Outstanding:" IncompleteCourses;
foreach Course in IncompleteCourses do
    if PrerequisitesMet(Course) then
        Add Course to SuggestedCourses;
    end
end
Display "Suggested Courses for Next Semester:" SuggestedCourses;
```

Algorithm 3: View Financial Aid Balance and Status

```
Input: StudentID
Output: Financial Aid Status and Balance
Validate StudentID;
if StudentID is invalid then
    Display "Error: Invalid Student ID.";
    return "Failure";
end
FinancialAidData ← FetchFinancialAidData(StudentID);
if FinancialAidData is empty then
    Display "No financial aid data available.";
    return;
end
Display "Total Awarded:" FinancialAidData.TotalAwarded;
Display "Amount Used:" FinancialAidData.UsedAmount;
Display "Remaining Balance:" FinancialAidData.RemainingBalance;
```

Algorithm 4: Track Real-Time Waitlist Position

```
Input: StudentID, CourseID
Output: Real-Time Waitlist Position
Validate StudentID and CourseID;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
WaitlistData ← FetchWaitlistPosition(StudentID, CourseID);
if WaitlistData is empty then
    Display "No waitlist information available.";
    return;
end
Display "Course:" CourseID;
Display "Your Waitlist Position:" WaitlistData.Position;
Display "Estimated Wait Time:" WaitlistData.EstimatedTime;
```

Algorithm 5: Predict Final Grades Using AI

```
Input: StudentID, CourseID
Output: Predicted Final Grade
Validate StudentID and CourseID;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
PerformanceData ← FetchStudentPerformanceData(StudentID,
 CourseID);
if PerformanceData is empty then
    Display "Insufficient data to predict grade.";
    return;
end
PredictedGrade ← AIModel.Predict(PerformanceData);
Display "Predicted Final Grade:" PredictedGrade;
```

Algorithm 6: Retrieve Exam Schedule

```
Input: StudentID
Output: Personalized Exam Schedule
Validate StudentID;
if Invalid then
    Display "Error: Invalid Student ID.";
    return "Failure";
end
ExamSchedule ← FetchExamSchedule(StudentID);
if ExamSchedule is empty then
    Display "No exam schedule available.";
    return;
end
foreach Exam in ExamSchedule do
    Display Exam.CourseName, Exam.Date, Exam.Time,
      Exam.Location;
end
```

Algorithm 7: Notify Drop/Withdraw Deadlines

```
Input: StudentID
Output: Notification of Upcoming Deadlines
Validate StudentID;
if Invalid then
    Display "Error: Invalid Student ID.";
    return "Failure";
end
CurrentSchedule ← FetchCurrentSchedule(StudentID);
Deadlines ← FetchDropWithdrawDeadlines(CurrentSchedule);
foreach Deadline in Deadlines do
    SendNotification(StudentID, Deadline.CourseName, Deadline.Date);
end
```

Algorithm 8: Schedule Appointment with Advisor

```
Input: StudentID, AdvisorID
Output: Appointment Confirmation
Validate StudentID and AdvisorID;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
AvailableSlots ← FetchAdvisorAvailability(AdvisorID);
Display "Available Appointment Slots:" AvailableSlots;
Display "Select a Slot:";
Read SelectedSlot;
if BookAppointment(StudentID, AdvisorID, SelectedSlot) then
    Display "Appointment Confirmed for:" SelectedSlot;
end
else
    Display "Error: Unable to book appointment.";
end
```

Algorithm 9: Get Directions to Classroom Locations

```
Input: StudentID, ClassroomID
Output: Directions to Classroom Location
Validate StudentID and ClassroomID;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
ClassroomLocation ← FetchClassroomLocation(ClassroomID);
if ClassroomLocation is empty then
    Display "No location data available.";
    return;
end
Directions ← GenerateDirections(ClassroomLocation);
Display "Directions to Classroom:" Directions;
```

Algorithm 10: Match Scholarships to Student Profile

```
Input: StudentID, ProfileData
Output: List of Matching Scholarships
Validate StudentID and ProfileData;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
ScholarshipList ← FetchAvailableScholarships();
MatchingScholarships ← [];
foreach Scholarship in ScholarshipList do
    if ProfileMatches(Scholarship, ProfileData) then
        Add Scholarship to MatchingScholarships;
    end
end
Display "Matching Scholarships:" MatchingScholarships;
```

Algorithm 11: Sync Calendar with Academic Events

```
Input: StudentID, CalendarApp
Output: Synchronized Calendar with Academic Events
Validate StudentID and CalendarApp;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
AcademicEvents ← FetchAcademicEvents(StudentID);
if AcademicEvents is empty then
    Display "No academic events found.";
    return;
end
if SyncCalendar(AcademicEvents, CalendarApp) then
    Display "Calendar synchronized successfully.";
end
else
    Display "Error: Calendar synchronization failed.";
end
```

Algorithm 12: Retrieve Personalized Class Schedule

```
Input: StudentID
Output: Class Schedule for Current Semester
Validate StudentID;
if Invalid then
    Display "Error: Invalid Student ID.";
    return "Failure";
end
ClassSchedule ← FetchClassSchedule(StudentID);
if ClassSchedule is empty then
    Display "No class schedule available.";
    return;
end
Display "Class Schedule:" ClassSchedule;
```

Algorithm 13: Estimate Course Difficulty with AI

```
Input: StudentID, CourseID
Output: Estimated Course Difficulty
Validate StudentID and CourseID;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
HistoricalData ← FetchHistoricalData(CourseID);
if HistoricalData is empty then
    Display "No data available for this course.";
    return;
end
EstimatedDifficulty ← AIModel.PredictDifficulty(HistoricalData);
Display "Estimated Difficulty Level:" EstimatedDifficulty;
```

Algorithm 14: Receive Class Registration Reminders

```
Input: StudentID
Output: Registration Reminder Notifications
Validate StudentID;
if Invalid then
    Display "Error: Invalid Student ID.";
    return "Failure";
end
RegistrationWindow ← FetchRegistrationWindow(StudentID);
if CurrentDate near RegistrationWindow.StartDate then
    SendNotification(StudentID, "Registration begins soon. Prepare
      your course plan.");
end
```

Algorithm 15: Track Upcoming Assignments

```
Input: StudentID
Output: List of Upcoming Assignments with Deadlines
Validate StudentID;
if Invalid then
    Display "Error: Invalid Student ID.";
    return "Failure";
end
AssignmentList ← FetchUpcomingAssignments(StudentID);
if AssignmentList is empty then
    Display "No upcoming assignments.";
    return;
end
Display "Upcoming Assignments:";
foreach Assignment in AssignmentList do
    Display Assignment.Name, Assignment.DueDate;
end
```

Algorithm 16: Retrieve Exam Schedule

```
Input: StudentID
Output: Personalized Exam Schedule
Validate StudentID;
if Invalid then
    Display "Error: Invalid Student ID.";
    return "Failure";
end
ExamSchedule ← FetchExamSchedule(StudentID);
if ExamSchedule is empty then
    Display "No exam schedule available.";
    return;
end
Display "Exam Schedule:" ExamSchedule;
```

Algorithm 17: Read Professor Reviews and Ratings

```
Input: StudentID, ProfessorName or CourseID
Output: Aggregated Professor Reviews and Ratings
Validate StudentID and Input Parameters;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
Reviews ← FetchProfessorReviews(ProfessorName or CourseID);
if Reviews is empty then
    Display "No reviews available for the selected professor.";
    return;
end
Display "Aggregated Ratings and Reviews:";
foreach Review in Reviews do
    Display Review.Rating, Review.Comments;
end
```

Algorithm 18: Track Real-Time Waitlist Position

```
Input: StudentID, CourseID
Output: Real-Time Waitlist Position
Validate StudentID and CourseID;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
WaitlistData ← FetchWaitlistData(StudentID, CourseID);
if WaitlistData is empty then
    Display "No waitlist data available.";
    return;
end
Display "Course:" CourseID;
Display "Waitlist Position:" WaitlistData.Position;
Display "Estimated Wait Time:" WaitlistData.EstimatedTime;
```

Algorithm 19: Suggest Study Groups Based on Courses

```
Input: StudentID, CourseID
Output: List of Suggested Study Groups
Validate StudentID and CourseID;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
StudyGroups ← FetchStudyGroups(CourseID);
if StudyGroups is empty then
    Display "No study groups found for this course.";
    return;
end
Display "Suggested Study Groups:";
foreach Group in StudyGroups do
    Display Group.Name, Group.ContactDetails, Group.MeetingTimes;
end
```

Algorithm 20: Schedule Appointment with Advisor

```
Input: StudentID, AdvisorID
Output: Appointment Confirmation
Validate StudentID and AdvisorID;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
AvailableSlots ← FetchAdvisorAvailability(AdvisorID);
Display "Available Appointment Slots:" AvailableSlots;
Read SelectedSlot;
if BookAppointment(StudentID, AdvisorID, SelectedSlot) then
    Display "Appointment Confirmed for Slot:" SelectedSlot;
end
else
    Display "Error: Unable to book appointment.";
end
```

Algorithm 21: Get Directions to Classroom Locations

```
Input: StudentID, ClassroomID
Output: Directions to Classroom Location
Validate StudentID and ClassroomID;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
ClassroomLocation ← FetchClassroomLocation(ClassroomID);
if ClassroomLocation is empty then
    Display "No location data available.";
    return;
end
Directions ← GenerateDirections(ClassroomLocation);
Display "Directions to Classroom:" Directions;
```

Algorithm 22: Track Scholarship Application Status

```
Input: StudentID
Output: List of Scholarships and Application Statuses
Validate StudentID;
if Invalid then
    Display "Error: Invalid Student ID.";
    return "Failure";
end
ScholarshipApplications ← FetchScholarshipApplications(StudentID);
if ScholarshipApplications is empty then
    Display "No scholarship applications found.";
    return;
end
Display "Scholarship Applications:";
foreach Application in ScholarshipApplications do
    Display Application.ScholarshipName, Application.Status;
end
```

Algorithm 23: Predict Final Grades Using AI

```
Input: StudentID, CourseID
Output: Predicted Final Grade
Validate StudentID and CourseID;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
PerformanceData ← FetchStudentPerformanceData(StudentID,
  CourseID);
if PerformanceData is empty then
    Display "Insufficient data to predict grade.";
    return;
end
PredictedGrade ← AIModel.PredictGrade(PerformanceData);
Display "Predicted Final Grade:" PredictedGrade;
```

Algorithm 24: Receive AI-Powered Career Guidance

```
Input: StudentID, ProfileData
Output: Suggested Career Paths, Job Opportunities, or Internships
Validate StudentID and ProfileData;
if Invalid then
    Display "Error: Invalid Inputs.";
    return "Failure";
end
CareerSuggestions ← FetchCareerSuggestions(ProfileData);
if CareerSuggestions is empty then
    Display "No career suggestions available.";
    return;
end
Display "Career Suggestions:";
foreach Suggestion in CareerSuggestions do
    Display Suggestion.Title, Suggestion.Description, Suggestion.Link;
end
```