SOFTWARE REQUIREMENTS SPECIFICATION

for

LabTracker

Automating Lab Management using GitHub

Version 1.0

Prepared by : *Mohd Saud*

23CAMSA107

GK0827

2024-2025

Department of Computer Science

Aligarh Muslim University

Date created : 20th October 2024

Table of Contents

1. Introduction

1.1 Purpose

The purpose of **LabTracker** is to automate the lab management process for the Department of Computer Science, AMU. The system enables students to submit programming problems via GitHub and helps teachers track their progress efficiently. By integrating with the GitHub API, LabTracker provides teachers with insights into students' commitment patterns and allows for the generation of various reports to monitor class performance. It streamlines the workflow for teachers by automating tasks like problem submission tracking and deadline management.

1.2 Document Convention

This document uses the following conventions:

- **Bold**: Indicates key terms or concepts that are important within the context of the document.

- ***Italic*:** Used for emphasis or to denote specific titles, names of documents, applications, or sections.

- **List Items**: Bullet points are used to enumerate features, requirements, or important notes for clarity and easy reference.

- **Headings**: Major sections are marked with headings of different levels to structure the document and aid in navigation.

- References: URLs and external resources are provided in a consistent format for easy access and further reading.

1.3 Intended Audience and Reading Sugesstions

This document is intended for:

- **Project Stakeholders**: This includes faculty members and department heads who will oversee the implementation of LabTracker.
- **End Users**: Students and teachers who will interact with the system.

Reading Suggestions:

- Stakeholders should focus on sections detailing the project scope and system overview.
- End users might find the user interface descriptions and functionalities most relevant.

1.4 Project Scope

LabTracker is a web-based application aimed at automating lab management for the Department of Computer Science at AMU. It enables students to submit programming assignments via GitHub, allowing teachers to efficiently track progress and generate performance reports.

Objectives:

- **Streamline Submission Processes**: Simplify how students submit assignments.
- **Enhance Teacher Efficiency**: Reduce administrative tasks through automation.
- **Provide Analytics**: Offer insights into student performance to inform teaching strategies.

Benefits:

- **Time Savings**: Automation saves time for both students and faculty.
- **Improved Accountability**: Students can easily track deadlines and submissions.
- **Data Management**: Simplifies retrieval and analysis of performance data.

LabTracker aligns with the Computer Science Department's goals by integrating technology into education, preparing students for industry practices.

1.5 References

- IEEE 830-1998: IEEE Recommended Practice for Software Requirements Specifications.
- Django Documentation: https://docs.djangoproject.com
- GitHub API Documentation: https://docs.github.com/en/rest
- Project Management Institute (PMI): Standards and guidelines for project management practices.

2. Overall Description

2.1 Product Perspective

LabTracker is an enhancement to traditional lab management systems specifically tailored for the Department of Computer Science at AMU. While existing systems manage lab assignments and track submissions, they lack the ability to integrate directly with GitHub, which is a vital tool for modern software development.

Context and Origin:

- **Motivation**: With the rise of GitHub as a standard for code collaboration, a solution that automates submission tracking and assessment is essential.

- **Target Audience**: Designed for students and faculty in the Computer Science department, LabTracker streamlines the submission process and provides insights into student performance.
- **Unique Positioning**: LabTracker distinguishes itself by automating the retrieval of assignments and tracking progress via GitHub, improving efficiency and accuracy.

Relation to Larger Systems:

LabTracker serves as an integral component that enhances existing lab management systems through GitHub integration, connecting to:

- **GitHub**: For real-time submission tracking.
- **Existing Lab Management Systems**: Future versions may offer interoperability with other educational platforms.

2.2 Product Features

LabTracker provides several key features that enhance lab management by integrating with GitHub to automate tracking of student assignments. The major features are organized as follows:

1. Student Management

- **Student Registration**: Enables students to sign up and create profiles within the system.
- **GitHub Profile Validation**: Verifies the existence of a student's GitHub account and checks for the necessary repositories related to their courses.

2. Assignment Tracking

- **GitHub Submission Monitoring**: Automatically tracks student submissions by monitoring their GitHub repositories for commit dates and activity related to assigned problems.
- **Completion Status Tracking**: Updates the completion status of assignments based on the last commit date from GitHub.

3. Reporting and Analytics

- **Individual Student Reports**: Generates detailed reports for faculties on individual student performance, including submission dates and problem completion statuses.

- **Class Performance Reports**: Provides aggregate analytics for the entire class, highlighting trends in submissions, on-time completions.
- **Faculty Activity Logs**: Maintains records of faculty actions within the system, such as problem creation and deadline settings.

4. Faculty Management

- **Assignment Creation**: Allows faculty members to create and manage programming problems, including descriptions and deadlines.
- **Deadline Management**: Enables faculty to set and update deadlines for assignments linked to specific problems.

2.3 User Classes and Characteristics

1. Students

- *Usage*: Regular (during lab sessions/assignments)
- *Functions*: Submit problems via GitHub, view progress, download reports, generate problem indexes.
- *Skills*: Basic GitHub and dashboard usage.
- *Privileges*: Access only their own data.
- *Importance*: High (primary users).

2. Faculty

- *Usage*: Frequent (for monitoring and evaluations).
- *Functions*: Track progress, generate reports, set deadlines, add problems, view activity logs.
- *Skills*: Moderate GitHub and web platform knowledge.
- *Privileges*: Access to all student data and reports.
- *Importance*: High (key managers of student progress).

3. Administrators

- *Usage:* Infrequent (for setup and troubleshooting).
- *Functions:* Manage users, maintain the system, handle GitHub API integration.
- *Skills:* High technical expertise (Django, APIs, databases).
- *Privileges:* Full system access.

- *Importance:* Moderate (support role).

2.4 Operating Environment

1. Hardware Platform:

- Runs on standard web servers (cloud-based or on-premises).
- Accessible via desktops, laptops, or mobile devices for end-users.

2. Operating System:

- *Server*: Linux (Ubuntu, Debian) or Windows Server.
- *Client*: Any OS (Windows, macOS, Linux, Android, iOS) with browser access.

3. Software Components:

- *Backend*: Django (Python framework).
- *Database*: PostgreSQL or MySQL.
- *Frontend*: HTML, CSS, JavaScript.
- *GitHub API*: For tracking student submissions and repositories.

4. Dependencies:

- Python 3.x and Django libraries.
- GitHub API access (requires API tokens).

5. Browser Compatibility and Network Requirements:

- Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari.
- Reliable internet connection for GitHub API integration and web access.

LabTracker is designed to function smoothly in environments with minimal technical infrastructure, provided basic network access and web browser availability are ensured.

2.5 Design and Implementation Constraint

1. Technologies and Tools :

- Must use Django (backend) and GitHub API for submission tracking.
- PostgreSQL or MySQL as the database.

2. API Rate Limit : GitHub API limits real-time updates; only periodic tracking is possible.

3. Security Requirements :

- Authentication for students and teachers.

- Secure API tokens for GitHub integration.

4. Maintenance Responsibility : System must follow Django conventions to facilitate future maintenance by developers.

5. Resource Constraints : Efficient handling of GitHub data to avoid API and memory overloads.

2.6 User Documentation

1. User Manual:

- Detailed instructions for students and teachers on using the system.
- Includes screenshots and step-by-step guides.
- *Format*: PDF

2. Quick Start Guide:

- Summary of essential tasks (e.g., logging in, submitting problems, generating reports).
- *Format*: PDF and accessible via the homepage.

2.7 Assumption and Dependencies

1. Assumptions :

- Reliable internet access for users.
- GitHub API remains stable.
- Students understand GitHub usage.
- Faculty can use the web interface.
- Institution provides server and maintenance support.

2. Dependencies :

- GitHub API: For tracking submissions.
- Server Maintenance: Ensures system stability.

3. System Features

3.1 Student Features

3.1.1 Progress Tracking

Description : This feature allows students to track their progress on assignments by monitoring submissions made to their GitHub repositories.

Priority : High

Stimulus/Response Sequences :

- *Stimulus* : Student pushes code to GitHub.

  Response : System updates the student's progress on their dashboard.

Functional Requirements :

- *REQ-1* : Connect to GitHub API to track student submissions.
- *REQ-2* : Display completed, pending and late submissions on the student dashboard.

3.1.2 File and Index Generation

Description : Students can generate structured submission files and download indexes.

Priority : High

Stimulus/Response Sequences :

- *Stimulus* : Student requests index generation.

  Response : System generates a DOCX file with problem details and images.
- *Stimulus* : Student generates a file for a specific week's work.

  Response : System creates a document with relevant problem information.

Functional Requirements :

- *REQ-1* : Allow students to generate DOCX files for lab submissions.
- *REQ-2* : Include problem descriptions, solutions and output images in generated files.
- *REQ*-3 : Provide downloadable indexes with formatting for teacher signatures.

3.2 Faculty Features

3.2.1 Weekly Assignment Deadline Management

Description : This feature allows teachers to set and manage deadlines for assignments, updating the deadlines displayed on students' dashboards.

Priority : High

Stimulus/Response Sequences :

- *Stimulus* : Faculty sets or updates a deadline.

  Response : System logs the change and updates the deadline on the student dashboard.
- *Stimulus* : Deadline is reached.

Response : System flags unsubmitted tasks as late in the reports.

Functional Requirements :

- *REQ-1* : Allow faculty to set and modify deadlines for weekly assignments.
- *REQ-2* : Automatically update the deadlines on students' dashboards when changes are made.
- *REQ*-3 : Automatically mark overdue tasks as late and display them on the dashboard.

3.2.2 Class Performance Monitoring and Reporting

Description : Faculty can monitor class performance through detailed reports that track student progress and engagement. The system supports three types of reports:

- *Specific Student Report* : Provides insights into an individual student's progress and submission history.
- *Class Weekly Report* : Summarizes weekly performance trends across the entire class.
- *Whole Class Report* : Offers an overview of the overall progress of the class across all weeks.

Priority : High

Stimulus/Response Sequences :

- *Stimulus* : Faculty requests a Specific Student Report.

  Response : System generates a detailed report for the selected student.
- *Stimulus* : Faculty requests a Class Weekly Report

  Response : System generated a detailed report for the class for the specified week.
- *Stimulus* : Faculty requests a Whole Class Report

  Response : System generated a detailed report for the class across all weeks.

Functional Requirements :

- *REQ-1* : Generates Specific Student Reports showing individual student progress.
- *REQ-2* : Provide Class Weekly Reports summarizing weekly performance.
- *REQ*-3 : Generate Whole Class Reports showing overall class progress.
- *REQ*-4 : Ensure that all reports are secure and accessible only to authorized faculties.

3.2.3 Teacher Activity Logs

Description : The system tracks all actions performed by teachers, such as setting deadlines or adding problems, to maintain transparency and accountability.

Priority : Medium

Stimulus/Response Sequences :

- *Stimulus* : Faculty sets or modifies a deadline.

  Response : System logs the action in faculty activity log.
- *Stimulus* : Faculty adds or edits a problem.

  Response : System logs the action in faculty activity log.
- *Stimulus* : Faculty requests a report.

  Response : System logs the action in faculty activity log.
- *Stimulus* : Faculty starts a new semester / updates the student data from GitHub.

  Response : System logs the action in faculty activity log.

Functional Requirements :

- *REQ-1* : Log activities such as setting deadlines, adding, or editing problems.
- *REQ-2* : Record request for class-wide or weekly reports, starting a new semester, or updating the student data from GitHub.
- *REQ*-3 : Ensure only authorized teachers can access these logs.

3.2.4 Adding and Editing Problems

Description : Faculties can add new problems for students to solve and update existing ones. Problems are categorized by course, semester, and week to ensure proper tracking and accessibility.

Priority : High

Stimulus/Response Sequences :

- *Stimulus* : Faculty submits a new problem.

  Response : System saves the problem and logs the addition.
- *Stimulus* : Faculty edits an existing problem.

  Response : System updates the problem details and logs the addition.

Functional Requirements :

- *REQ-1* : Allow faculties to add new problems based on course, semester and week.
- *REQ-2* : Provide an interface for faculties to edit existing problems.
- *REQ*-3 : Ensure the updated problems are immediately reflected on student's dashboards.
- *REQ*-4 : Maintain access control to ensure only authorized faculties can edit problems.

3.2.5 Starting a New Semester

Description : Faculties can start a new semester to reset student data and prepare for new assignments. This feature ensures students' progress tracking starts fresh for the new term.

Priority : Medium

Stimulus/Response Sequences :

- *Stimulus* : Faculty initiates a new semester.

  Response : System clears old semester data and sets up a new one.

Functional Requirements :

- *REQ-1* : Allow faculties to start a new semester, resetting relevant student data.
- *REQ-2* : Provide confirmation to faculties before wiping old data.
- *REQ*-3 : Log the start of a new semester in the activity log.

3.2.6 Updating Student Data from GitHub

Description : The system automatically updates students' GitHub data at scheduled intervals to ensure accurate progress tracking. Faculties also have the option to manually trigger updates to minimize potential errors. Due to GitHub API rate limits, real-time updates are not enabled.

Priority : High

Stimulus/Response Sequences :

- *Stimulus* : Automatic scheduled data update runs.

  Response : System fetches the latest commit data from students' GitHub repositories and updates the dashboard.
- *Stimulus* : Faculty clicks the "Update Data" button.

  Response : System performs an immediate update and logs the action.

Functional Requirements :

- *REQ-1* : Schedule automatic updates for fetching the latest GitHub data.
- *REQ-2* : Allow faculties to manually trigger data updates to reduce errors.
- *REQ*-3 : Fetch the latest commits and submission data from student repositories.
- *REQ*-4 : Handle API rate limits gracefully by displaying status updates to faculties.

3.3 Admin Feature (Django Admin Interface)

Description : The Django Admin interface provides a centralized platform for managing the application's data and settings. Admins can oversee students, faculties, problems, deadlines, and logs efficiently. This interface is crucial for ensuring the integrity and smooth operation of the LabTracker system.

Priority : High

Stimulus/Response Sequences :

- *Stimulus* : Admin logs into the Django Admin interface.

  Response : System displays a dashboard with access to models like students, faculties, problems, and activity logs.
- *Stimulus* : Admin adds, modifies, or deletes any entry.

  Response : System updates the database accordingly and maintains data integrity.
- *Stimulus* : Admin views activity logs.

  Response : System provides detailed records of teacher and student actions for auditing purposes.

Functional Requirements :

- *REQ-1* : Provide an intuitive interface for managing students, teachers, problems, deadlines, and other models.
- *REQ-2* : Ensure secure access control, allowing only authorized admins to perform data operations.
- *REQ*-3 : Allow admins to view and manage teacher and student activity logs.
- *REQ*-4 : Provide validation and error checking to prevent data inconsistencies during operations.
- *REQ-5 :* Enable bulk updates, filtering, and search functionalities for efficient data management.

- *REQ-6* : Integrate with Django’s authentication and permissions system for robust security management.

4. External Requirements

4.1 User Interfaces

LabTracker’s user interface is simple and user-friendly, following the AMU flag color scheme for a consistent visual identity. It includes key sections for both students and teachers, designed with intuitive navigation and clear functionality.

Key Interface Elements:

- **Login Screens**:
    - Separate login pages for students and teachers with clear input fields.
    - Errors are displayed in red below the fields.
- **Student Dashboard**:
    - Shows problem assignments, deadlines, and completion status in a card-based layout.
    - Buttons for accessing problem details, generating submission files, and viewing deadlines.
- **Teacher Dashboard**:
    - Sections for problem management, setting deadlines, and generating reports.
    - Tabs for navigating student progress, activity logs, and report generation.
- **Problem Management**:
    - Teachers can add, edit, and view problems with dropdowns for courses and semesters.
    - Error messages appear in red for validation issues.
- **Reports and Notifications**:
    - Dropdowns and buttons for generating reports.
    - Notifications for deadlines and overdue tasks are highlighted on the student dashboard.

General Interface Characteristics:

- **Color Scheme**: Follows AMU flag colors for buttons and highlights.

- **Navigation**: Fixed top bar for main navigation; consistent buttons for actions like "Submit" and "Edit."
- **Error Messaging**: Inline error alerts in red.

The design ensures a clean, AMU-branded experience with focus on usability.

4.2 Hardware Interfaces

LabTracker is a web-based application, requiring minimal hardware-specific interactions. The system supports standard computing hardware with the following characteristics:

Supported Devices:

- **Desktop Computers**: Running modern browsers like Chrome, Firefox, Edge.
- **Laptops**: Compatible with standard web browsers.
- **Tablets and Mobile Devices**: Optimized for smaller screens through responsive web design.

Data and Control Interactions:

- **Client-Server Interaction**: User devices (client-side) communicate with the server over HTTPS, ensuring secure data transfer.
- **Server**: Hosted on a machine with sufficient capacity to handle database operations, GitHub API interactions, and user requests.

LabTracker does not require specialized hardware; any standard internet-enabled device with a browser is sufficient for access.

4.3 Software Interfaces

LabTracker integrates with several software components to ensure smooth functionality:

Operating System:

- **Server**: Runs on Linux-based OS (e.g., Ubuntu) for hosting the Django application.
- **Client**: Compatible with any OS that supports modern web browsers.

Web Framework:

- **Django (v4.x)**: Manages server-side operations, authentication, and database interactions.

Database:

- **PostgreSQL**: Stores user data, problems, submissions, deadlines, and logs.

  - **Data In**: User and GitHub details.
  - **Data Out**: Reports, deadlines, and performance summaries.

APIs:

- **GitHub API (v3)**: Retrieves student commit and repository data.
- **Communication**: Uses HTTPS with REST API calls.

Libraries:

- **python-docx**: Generates Word documents for problem submissions.

Secure communications are ensured via **HTTPS**, with data consistency maintained through Django's ORM.

4.4 Communications Interfaces

LabTracker requires the following communication protocols and standards:

- **Web Browser Access**: Communication between client and server happens over **HTTPS**, ensuring secure transmission of data.
- **GitHub API Integration**: Data is fetched via secure **HTTPS REST API** calls, using standard JSON formatting for data exchange.
- **Server Communication**: Internal communication between the web server and database uses standard TCP/IP protocols.

All data exchanges are encrypted using **SSL/TLS** to maintain security and privacy. Data synchronization with GitHub adheres to API rate limits, ensuring stable updates.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

LabTracker must meet the following performance criteria:

- **Response Time**: Pages should load within **2 seconds** for standard requests (e.g., dashboard view, problem listing). Data-intensive operations, like report generation, should complete within **5 seconds**.
- **Data Update**: GitHub data updates triggered manually or scheduled should not exceed **10 minutes** to complete, accounting for GitHub API rate limits.
- **Simultaneous Users**: The system should support up to **100 concurrent users** without performance degradation.

- **Database Queries**: Should be optimized to handle complex report generation with minimal latency.
- **Scalability**: The system should maintain performance standards as the number of users and data volume increases, allowing for easy scaling of server resources.

Performance goals ensure a smooth user experience for both students and faculties, even with increasing usage.

5.2 Safety Requirements

LabTracker incorporates the following safety measures:

- **Data Integrity**: The system relies on GitHub as the primary source for student data, enabling easy data retrieval whenever needed.
- **Access Control**: Role-based access limits features based on user roles (e.g., student, faculty) to prevent unauthorized actions.
- **Error Handling**: Graceful error handling prevents data corruption and unauthorized access during unexpected situations or failures.
- **Regulations Compliance**: System design follows university policies on data privacy and security, ensuring that sensitive student information is protected.

These safeguards are in place to prevent data loss and unauthorized access while ensuring reliable and secure system operations.

5.3 Security Requirements

- **User Authentication** : Utilize Django's authentication system with hashed passwords (PBKDF2).
- **Data Protection** :
    - Use HTTPs for data in transit.
    - Encrypt sensitive data at rest and restrict database access.
- **Privacy Compilance** :
    - Comply with GDPR and FERPA regulations.
    - Obtain user consent for data collection.
- **Access Control** : Implement role-based access for students and faculties using Django permissions.

- **Security Certifications** : Aim for relevant certifications and conduct regular security audits.
- **Incident Response** : Create a breach response plan and notify users as required.
- **Monitoring and Logging** : Track user activities with Django's logging framework and maintain audit trials.
- **User Training** : Provide training on security best practices.

5.4 Software Quality Attributes

- **Usability** :
    - Intuitive UI allowing common tasks to be completed in 3 clicks or less.
    - Aim for 80% user satisfaction in post-interaction surveys.
- **Correctness** : Achieve 99% accuracy in tracking progress and generating reports.
- **Reliability** : Ensure 99.9% uptime, with scheduled maintenance communicated in advance.
- **Maintainability** :
    - Modular code allowing updates within two weeks for minor changes.
    - Up-to-date documentation.
- **Adaptability** : Easy integration of new features of maximum of 20% code changes.
- **Testability** : 80% code coverage through automated tests.
- **Flexibility** : Role-based access control configurable via the admin interface.
- **Interoperability** : Seamless integration with GitHub's API, with response times under 500 ms.
- **Robustness** : 90% of error cases should provide user-friendly error messages.
- **Portability** : Deployable on various platforms with setup time under 2 hours.
- **Reusability** : Code designed for reuse in future projects.

6. Other Requirements

1. **Database Requirements:**

- LabTracker will use PostgreSQL for database management due to its reliability, scalability, and support for complex queries. The database will store user data, course details, problem sets, submission records, and activity logs.

- Regular backups are required to ensure data recovery, with daily backups stored in a secure location.

2. **Internationalization Requirements:**

- LabTracker's initial version will be developed in English. However, the codebase and database should be structured to support additional languages if needed in future versions.

3. **Legal Requirements:**

- LabTracker must comply with relevant data privacy laws, such as GDPR and FERPA, ensuring the secure handling and storage of student information.

- User consent for data collection will be obtained during the registration process and documented accordingly.

4. **Reuse Objectives:**

- The application's modules, especially user authentication, GitHub integration, and report generation components, should be designed to allow reuse in similar educational management systems.

- Code documentation and modular design will support future updates and integration into other departmental systems, if necessary.

6. **Security and Access Control:**

- Data access control will restrict sensitive information based on user roles (e.g., faculties, students, admins), ensuring data integrity and confidentiality.

- Django's authentication framework will manage access, enforcing secure password requirements and session management.

7. **Documentation and Training:**

- Comprehensive user documentation, including setup guides for teachers and students, must be provided. In-app guides or video tutorials will also be considered for ease of use.

- Administrators and teachers will receive training on report generation and activity logging.

APPENDIX A : GLOSSARY LIST

- **API (Application Programming Interface):** A set of functions and protocols that allows different software applications to communicate with each other. LabTracker uses the GitHub API to track student activities.

- **Django:** A Python-based web framework used to build the LabTracker application, offering a structured approach to handling the back end.

- **GitHub:** A web-based platform used for version control and collaborative development. In LabTracker, GitHub repositories are used to track student submissions and progress.

- **Commit:** An action in GitHub where students submit code changes or updates. LabTracker tracks the frequency and timing of commits to monitor student progress and adherence to deadlines.

- **GitHub API Rate Limitation:** A restriction imposed by GitHub on the number of API requests allowed per hour. LabTracker's data tracking processes are optimized to work within these rate limits, avoiding real-time data retrieval.

- **Index Generation:** A feature in LabTracker that creates a downloadable Word document listing problem sets for a student's course and semester, used as a lab submission document.

- **Student Dashboard:** A personalized interface in LabTracker where students can view their progress, track deadlines, and monitor completion statuses for lab problems.

- **Faculty Dashboard:** An interface where faculties can view student performance, generate reports, and manage problem sets and deadlines.

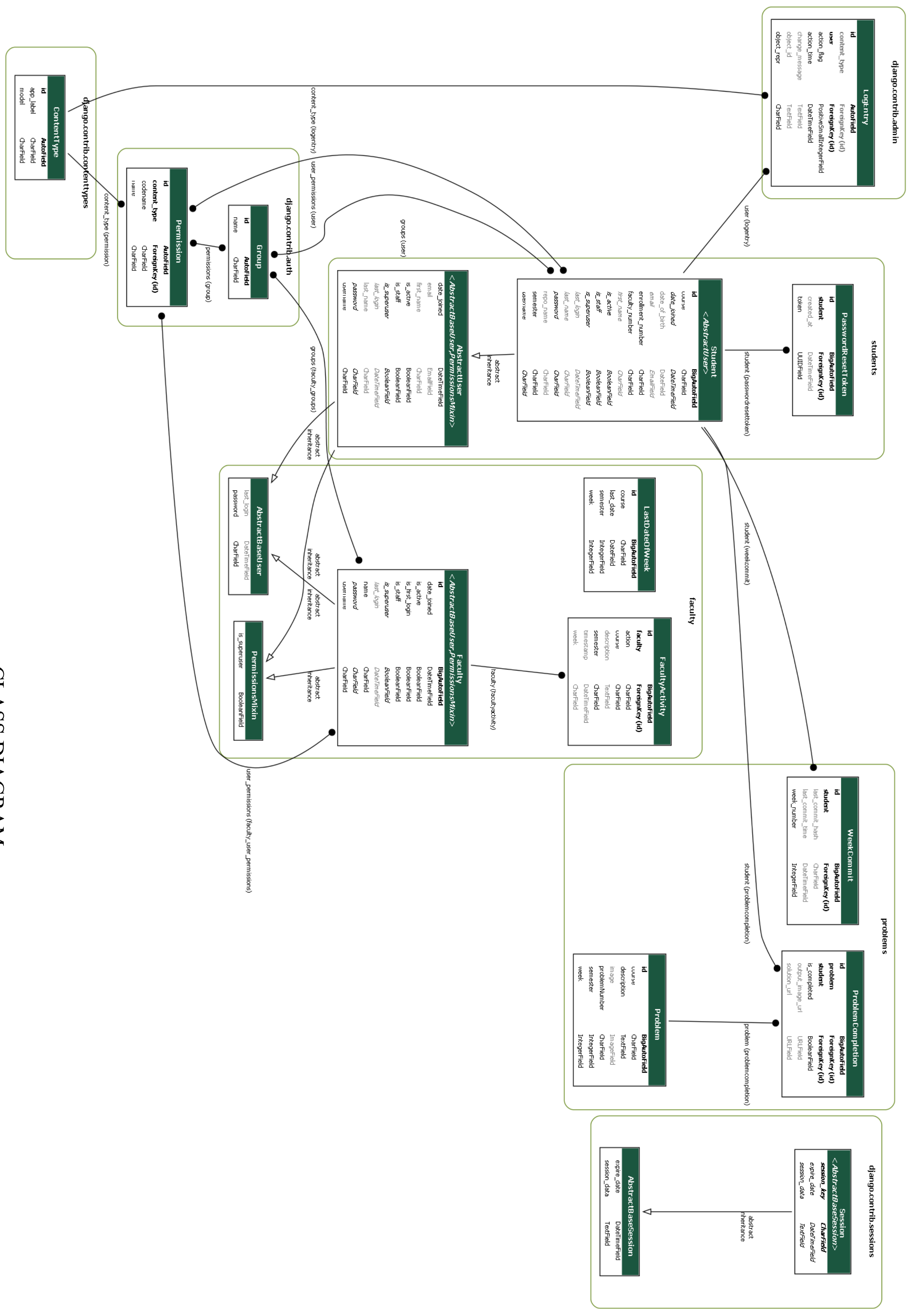
- **User Authentication:** The process of verifying a user's identity. LabTracker uses Django's authentication framework to ensure secure logins for students and teachers.

- **File Generation:** A feature in LabTracker that allows students to generate Microsoft Word documents containing lab problems, solutions, and outputs for specific weeks. This document is formatted for lab submission and includes relevant details.

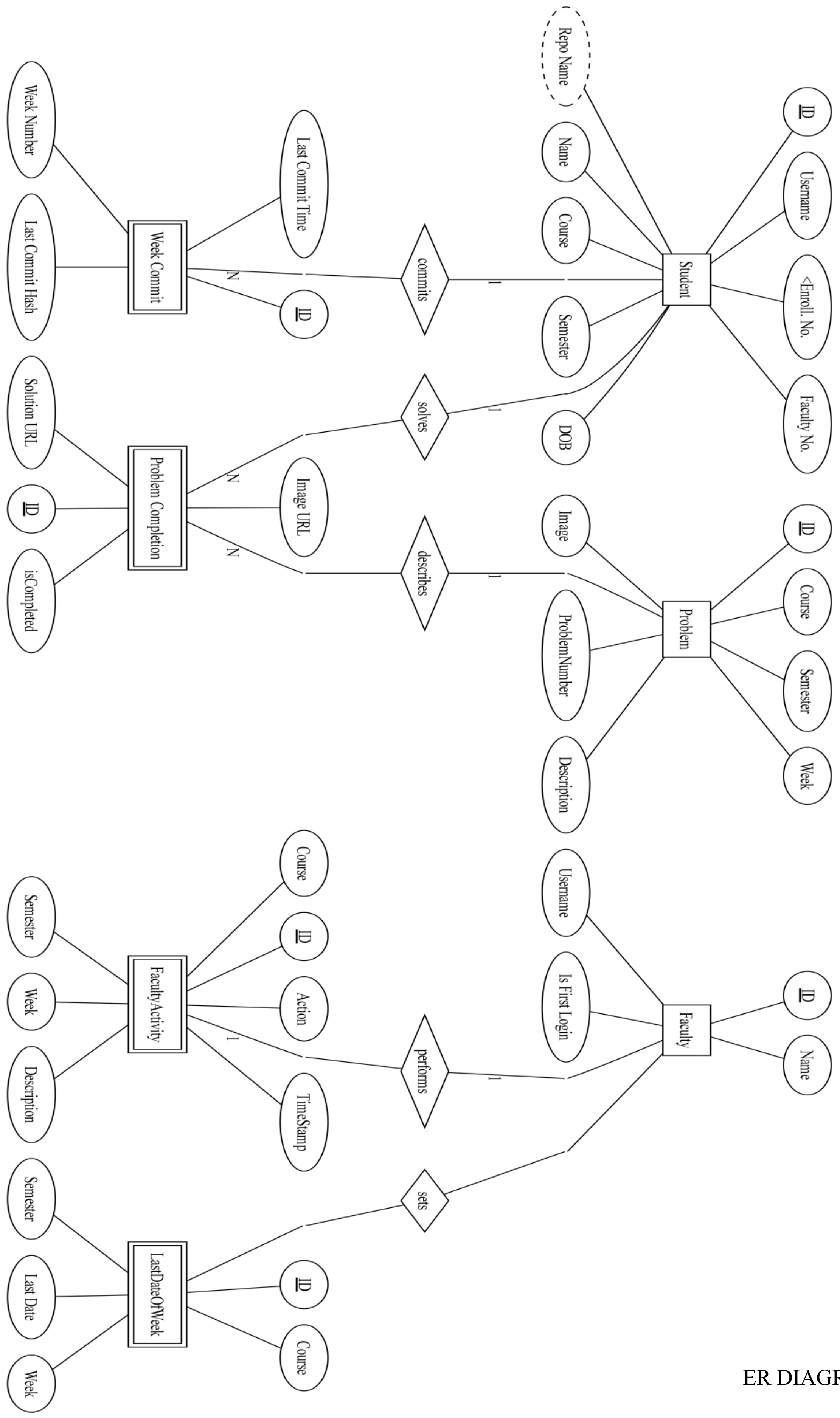
- **Report Generation:** A feature in LabTracker that enables faculties to create downloadable reports for tracking student progress. Reports include options like specific student reports, weekly class reports, and overall class reports, offering insights into lab completion rates and commit times.

- **GitHub Repository:** A storage location on GitHub where students' code and solutions are saved and version-controlled. LabTracker monitors these repositories to track submission timelines and completion.

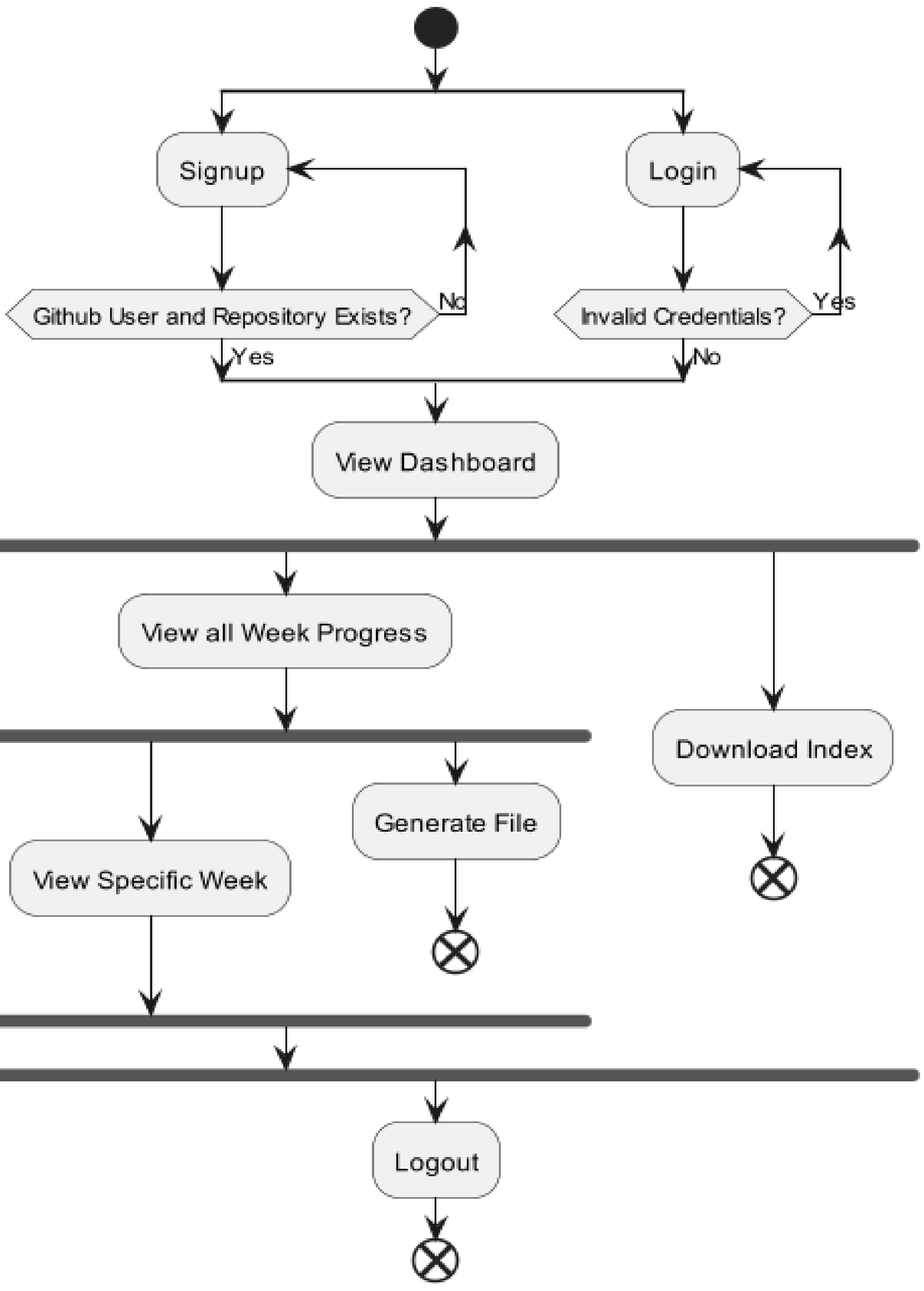
APPENDIX B : ANALYSIS MODELS



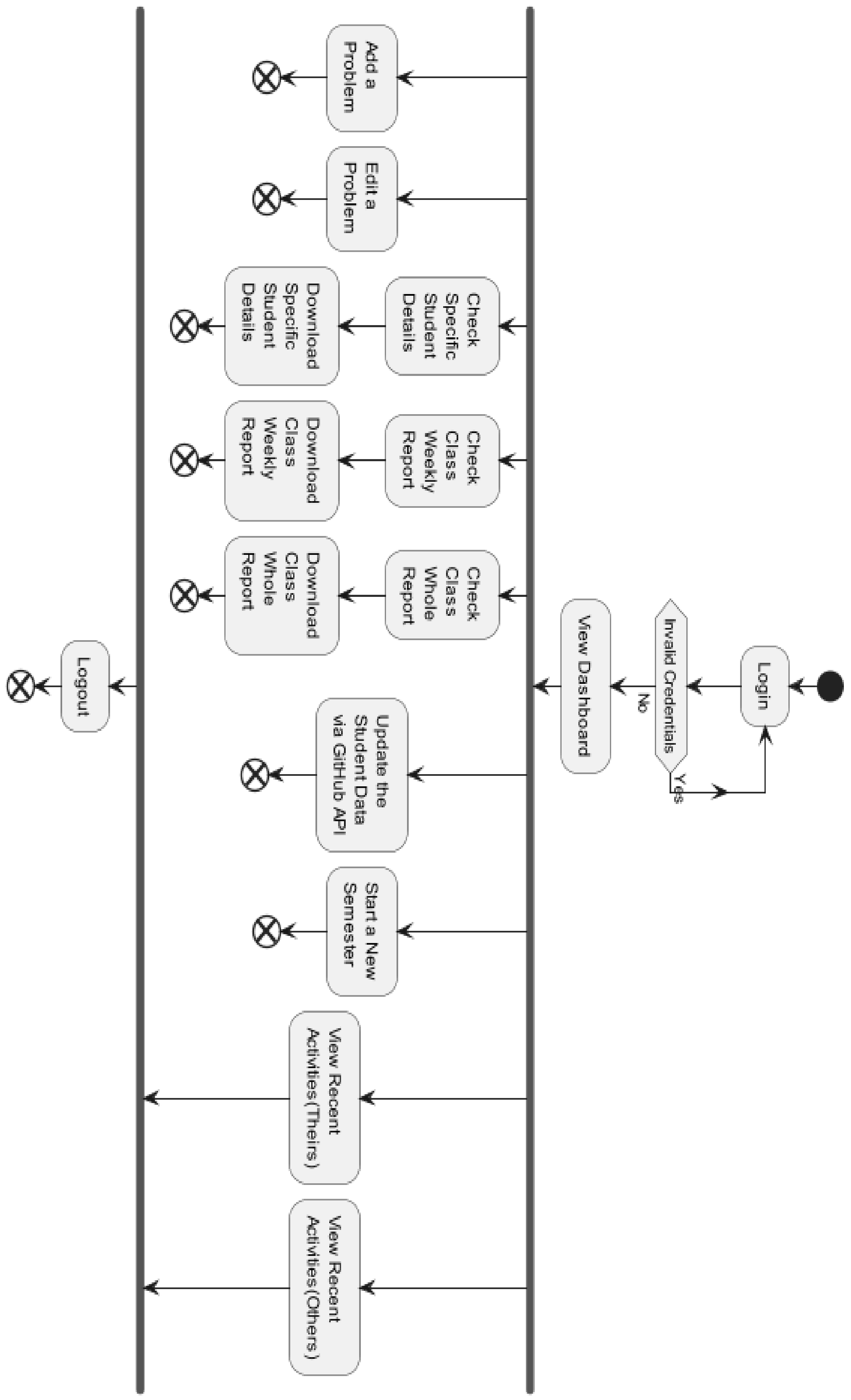
CLASS DIAGRAM


Student
ID
Username
<Enroll. No.
Faculty No.
Repo Name
Name
Course
Semester
DOB
commits
Week Commit
Week Number
Last Commit Hash
Last Commit Time
ID
solves
Problem Completion
Solution URL
ID
isCompleted
Image URL
describes
Problem
ID
Course
Semester
Week
Image
ProblemNumber
Description
Faculty
ID
Name
Username
Is First Login
performs
FacultyActivity
Semester
Week
Description
Course
ID
Action
TimeStamp
sets
LastDateOfWeek
Semester
Last Date
Week
ID
Course
1
N
1
N
1
N
1
1

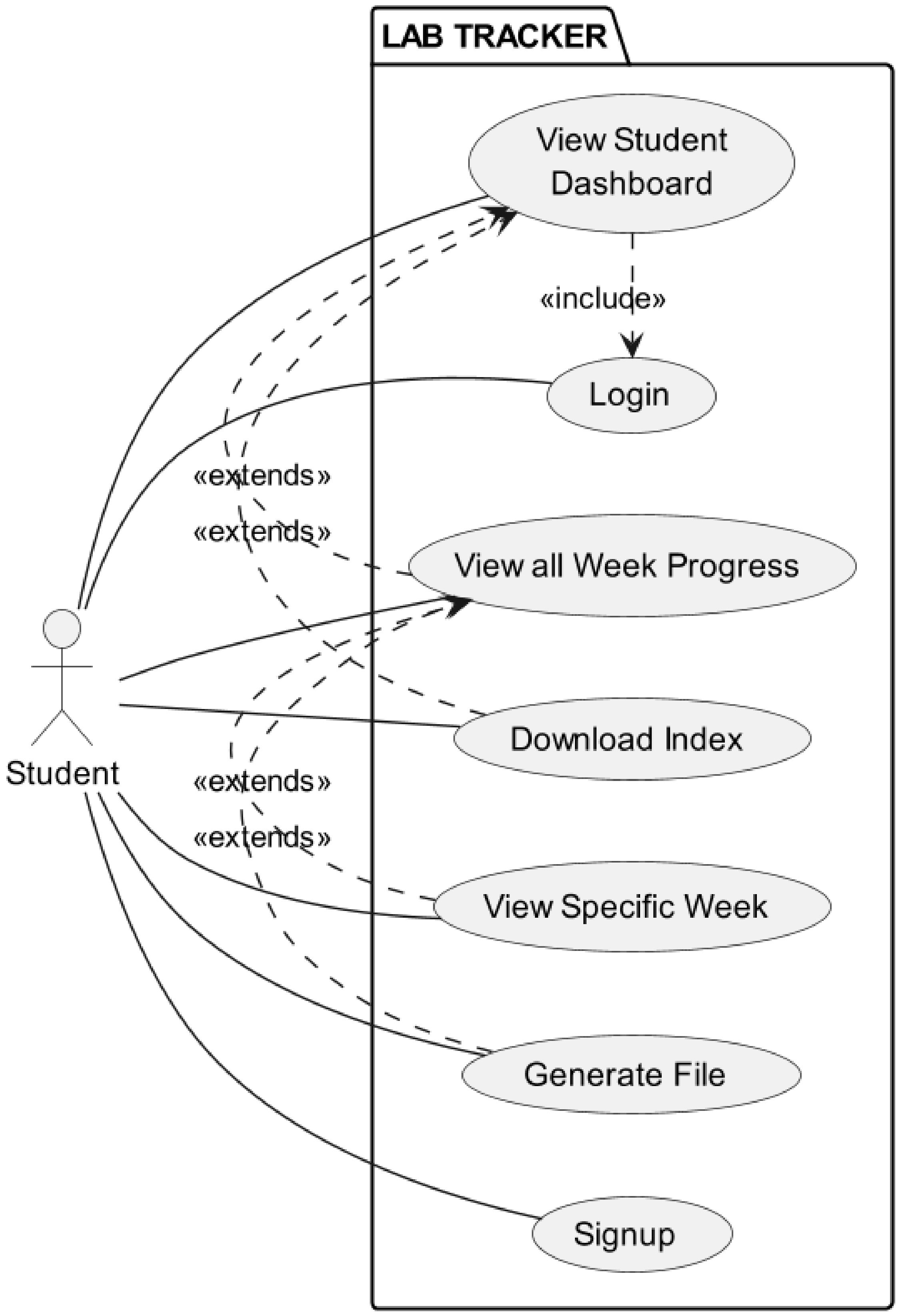
ER DIAGRAM



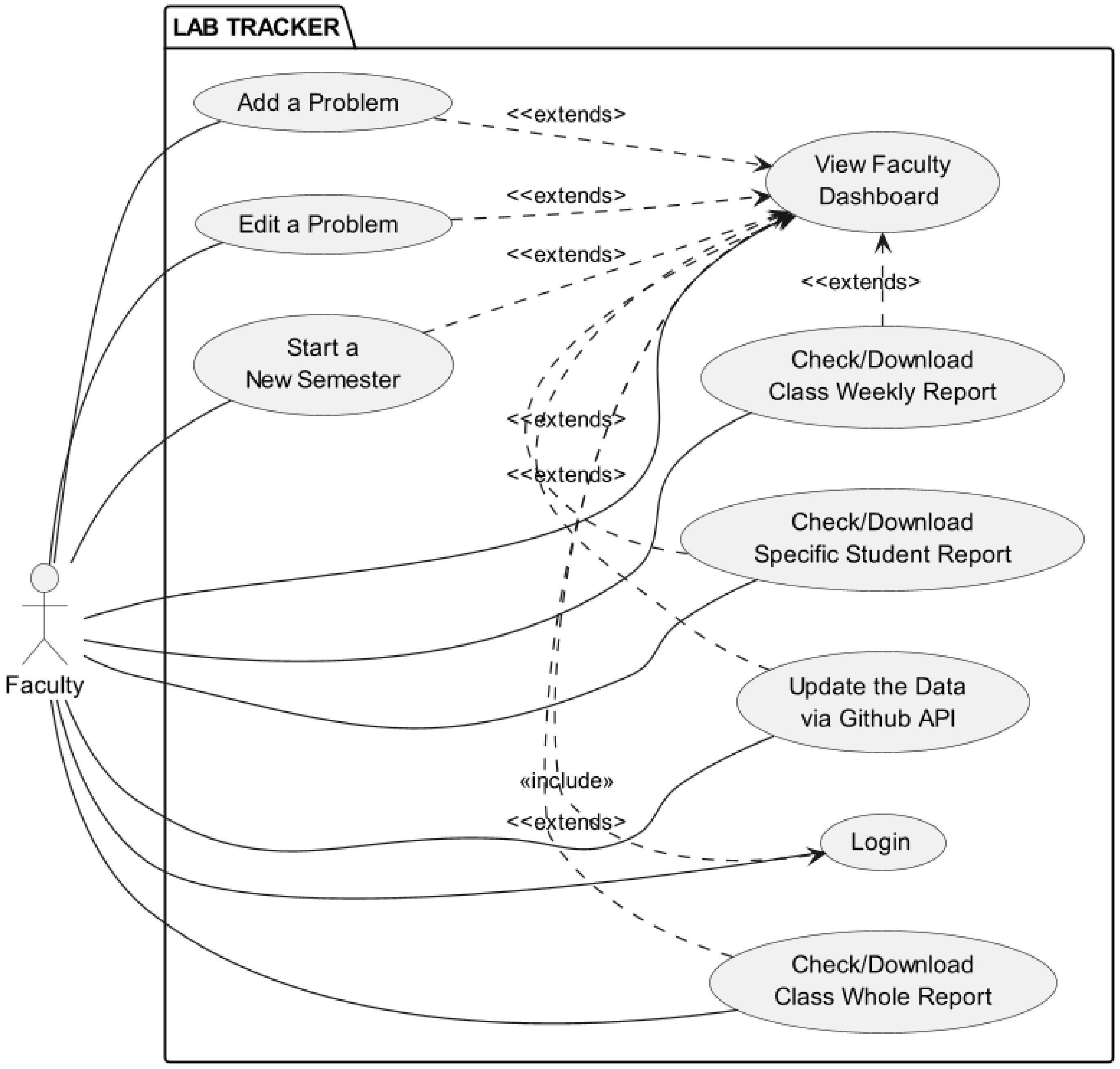
ACTIVITY DIAGRAM (STUDENTS)


Login
Invalid Credentials
Yes
No
View Dashboard
Check Class Whole Report
Download Class Whole Report
Check Class Weekly Report
Download Class Weekly Report
Check Specific Student Details
Download Specific Student Details
Edit a Problem
Add a Problem
Update the Student Data via GitHub API
Start a New Semester
View Recent Activities(Theirs)
View Recent Activities(Others)
Logout

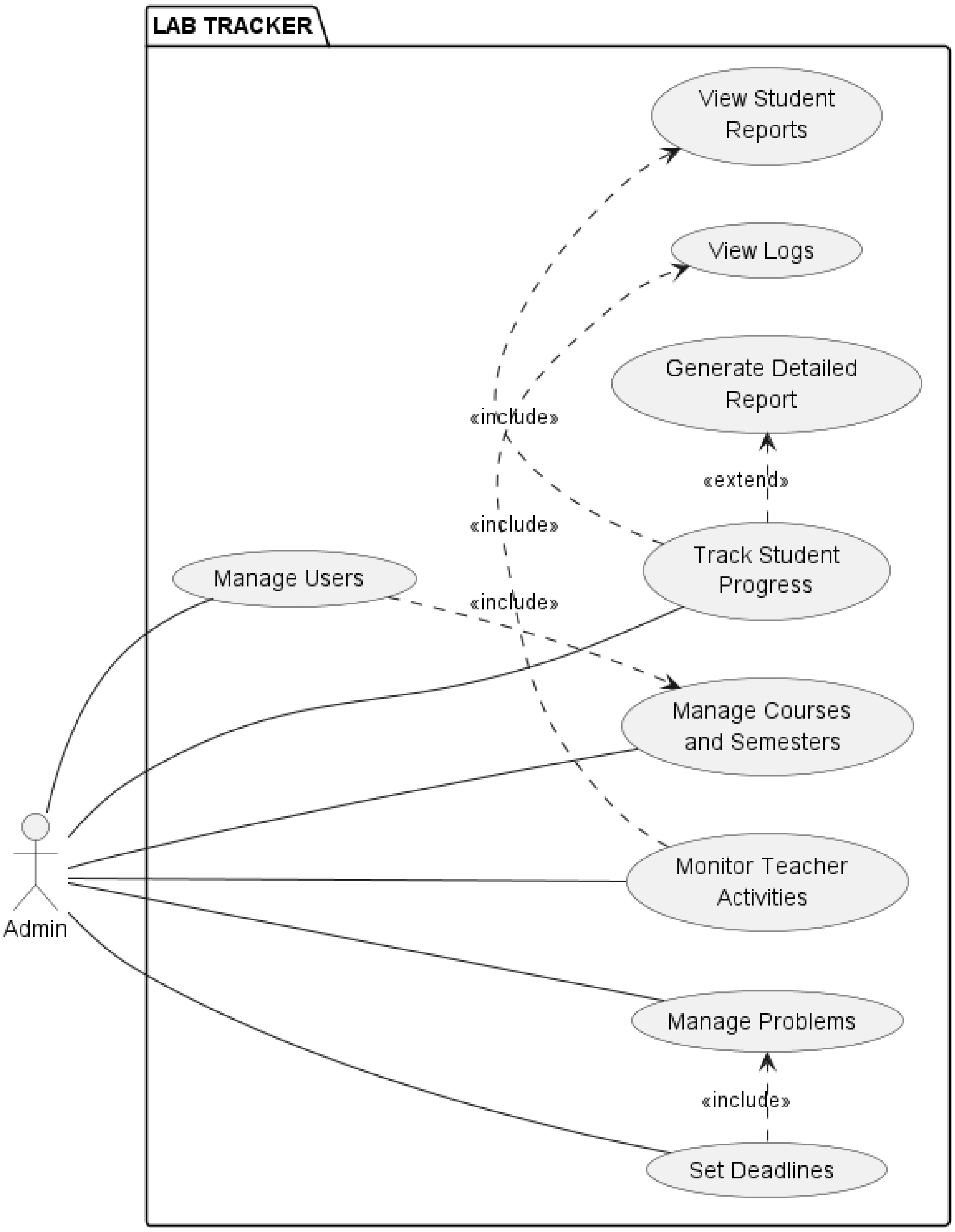
ACTIVITY DIAGRAM (FACULTY)



Use Case Diagram (Student)

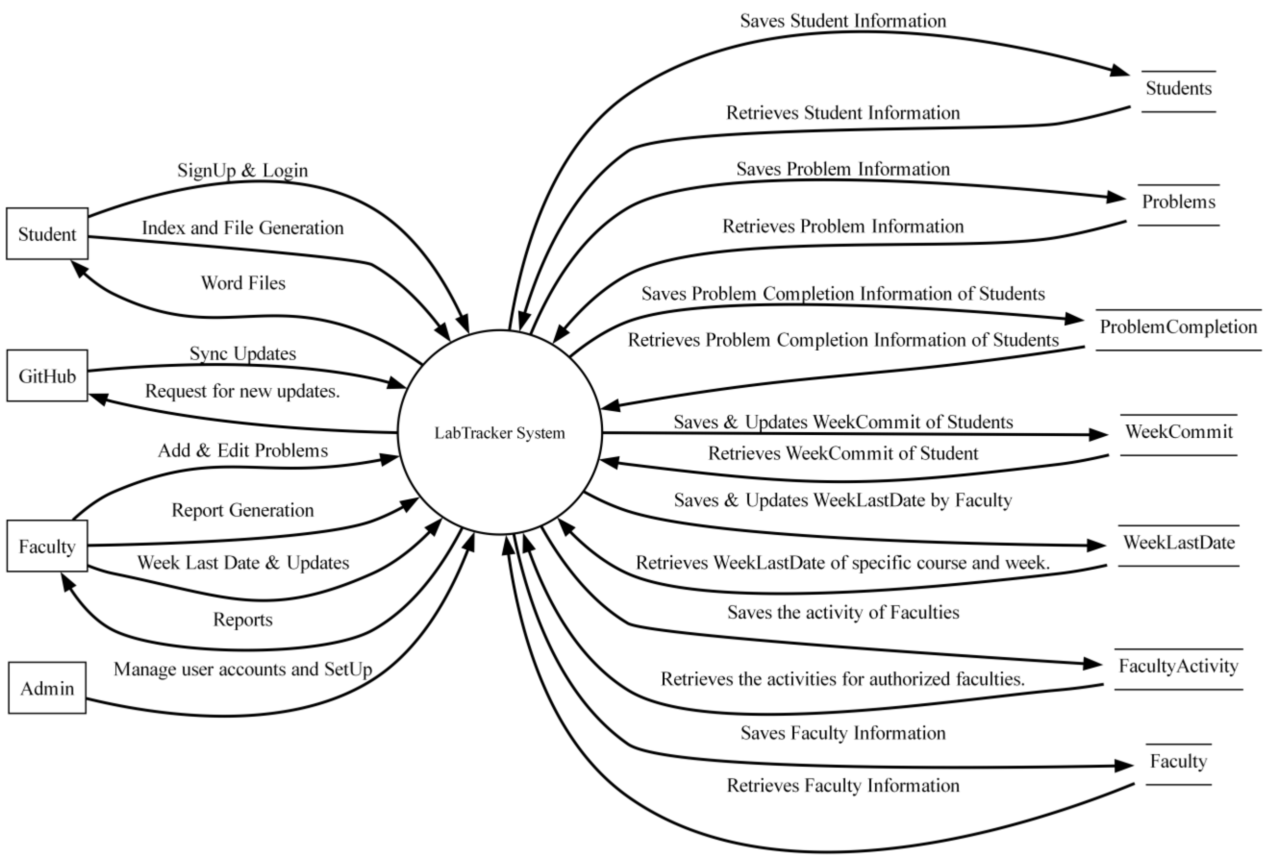

LAB TRACKER
Add a Problem
Edit a Problem
Start a New Semester
View Faculty Dashboard
Check/Download Class Weekly Report
Check/Download Specific Student Report
Update the Data via Github API
Login
Check/Download Class Whole Report
Faculty
<<extends>>
«include»

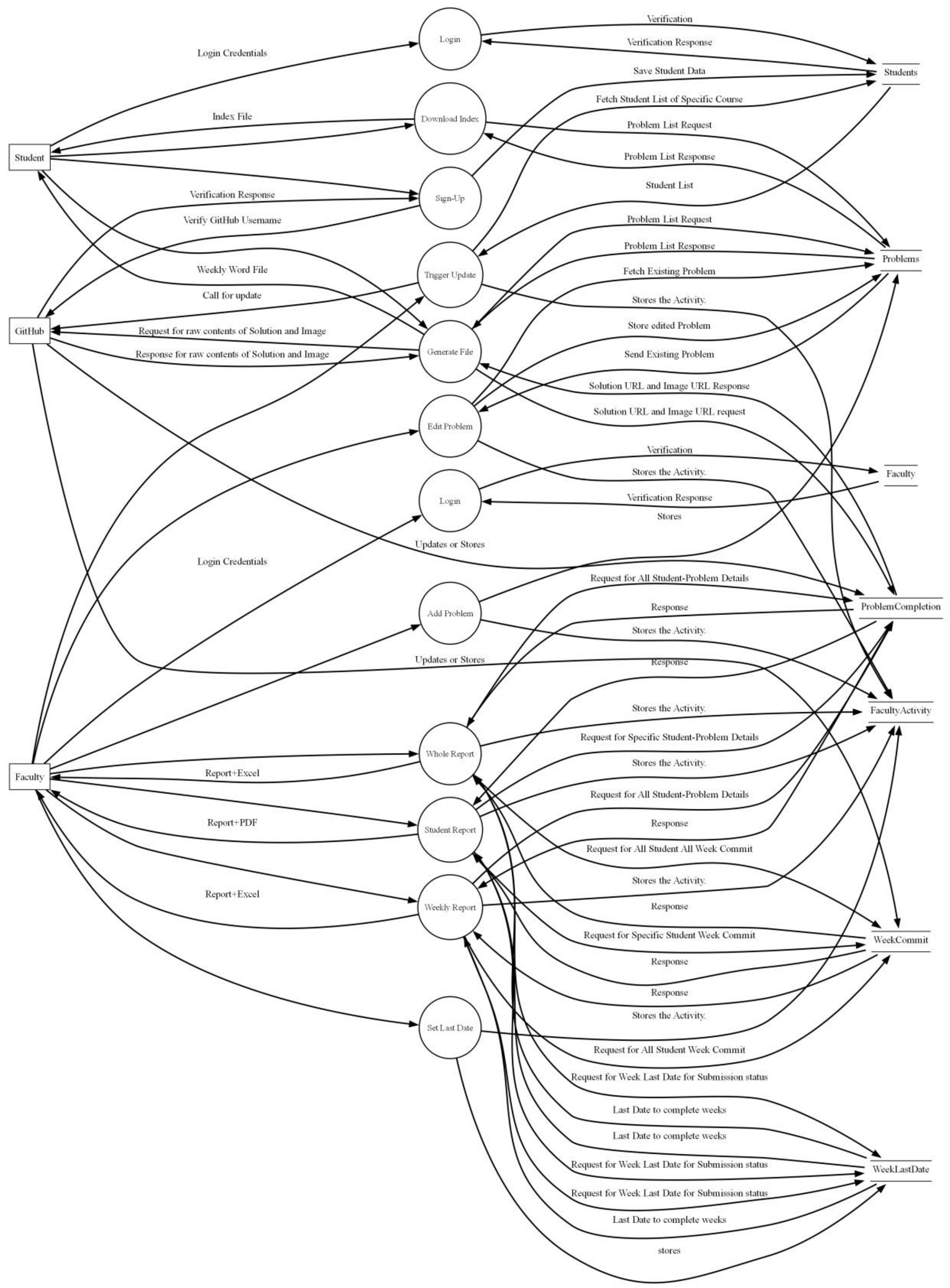
Use Case Diagram (Faculty)



Use Case Diagram (Admin)

DFD-Level0


Student
GitHub
Faculty
Admin
LabTracker System
SignUp & Login
Index and File Generation
Word Files
Sync Updates
Request for new updates.
Add & Edit Problems
Report Generation
Week Last Date & Updates
Reports
Manage user accounts and SetUp
Saves Student Information
Students
Retrieves Student Information
Saves Problem Information
Problems
Retrieves Problem Information
Saves Problem Completion Information of Students
ProblemCompletion
Retrieves Problem Completion Information of Students
Saves & Updates WeekCommit of Students
WeekCommit
Retrieves WeekCommit of Student
Saves & Updates WeekLastDate by Faculty
WeekLastDate
Retrieves WeekLastDate of specific course and week.
Saves the activity of Faculties
FacultyActivity
Retrieves the activities for authorized faculties.
Saves Faculty Information
Faculty
Retrieves Faculty Information



DFD-Level1

SDLC Model :

Model Used : Agile Model

The Agile Model is chosen for the LabTracker project to accommodate flexibility,iterative progress and continous feedback.

APPENDIX C : ISSUES LIST

1. **Real-Time Updates**: We need to determine if real-time tracking of student commits through the GitHub API is feasible, given the rate limitations. This is a high-priority item, as it affects the project's core functionality.

2. **Data Privacy Policy:** Finalizing and documenting privacy policies for handling student data is essential to ensure compliance with regulations like GDPR and FERPA. This issue is marked TBD and is a high priority.

3. **Localization Requirements:** A decision needs to be made on which languages should be supported in the initial release to better serve the user base. This is a medium-priority issue, as it impacts usability and accessibility.

4. **User Feedback Mechanism:** It's yet to be decided if an in-app tool should be provided for students and faculties to give feedback or suggest improvements. This is low priority but could enhance user engagement. The development team will assess feasibility and implementation requirements.

5. **Security Certifications:** We need to decide if pursuing certifications like ISO/IEC 27001 or SOC 2 Type II is feasible for LabTracker. This is pending a decision and is a medium-priority issue, as it depends on resource availability and timelines.

6. **Backup Frequency:** The frequency and retention policy for database backups must be confirmed to meet reliability requirements. This high-priority item is still TBD.