Bilkent University

Department of Computer Engineering

CS 319 Term Project Internship Management System

Project short-name: InternHub

Design Report

Hasan Ege Tunç 22003814, Anıl İlağa 22002044, Deniz Tuna Onguner 22001788, Alper Göçmen 22002948, Sarper Arda Bakır 21902781

Instructor: Eray Tüzün
Teaching Assistant(s): Yahya Elnouby & Muhammad Umair Ahmed

Design Report
May 21, 2023

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the Object Oriented Software Engineering course CS319 requirement

CONTENTS

1.0 Introduction

1.1 Purpose of The System

InternHub is a web-based application designed to simplify the process of managing student internship reports and evaluations. The system enables students to upload their internship reports to a central repository where they can be reviewed and evaluated by their respective instructors. In addition, students are able to fulfill company related procedures for their internships. Regarding instructors, they can see the list of internships they are assigned to, can open submissions for the reports, and see the uploaded internship reports to the system under the submissions to which they can give feedback.

The department secretaries are the key users in the internship management system. They have many responsibilities concerning students, companies and instructors. Firstly, she can assign internships to the instructors. Once the students' internship reports' evaluation processes are finished by the instructors, she can collect the grades of these reports. Also, she receives confidential company forms of the students from the supervisors and processes these forms into the system. Secondly, when students request a new company not in the company list, the department secretary handles the requests. If students submit their internship confirmations, she is the one in charge of approvals. She either refuses or accepts the internship requests.

The system facilitates the job of secretary by digitalizing the system and removing most of the paperwork. The dean and chairs can see the list of other users such as instructors and students, and see the statistics about them. The difference is that the dean can see all the users' statistics and lists whereas chairs can only see their department's lists and statistics. Lastly, the superuser is the one who handles the whole system such as registrations of users, managing companies in the database, and authorizing other super users. They are responsible for working the system smoothly. Compared to the current system, this system eliminates the use of other sources, and gathers all functionality in one platform. Thus, this system provides a convenient and efficient way to manage the entire process of internship evaluation.

1.2 Design Goals

1.2.1. Usability

The InternHub team offers a web-based application that facilitates the use of the current system both for the first-time users and staff. To achieve this, we provide a simple interface. For instance, there is a navigation bar that includes all the operations that can be done by users on the left of the website. Components of the navigation bar are self-explanatory such as "Contact" which navigates users to see the instructor and secretary e-mail or "Reports" which navigates related users to the internship reports. Moreover, InternHub provides a feature that displays a deadline date to inform students, and announcements that inform users about current events. InternHub offers its users to stay

and handle all the operations easily in just one place. Therefore, the first item of top two design goals is chosen as usability.

1.2.2. Reliability

The communication among students, instructors and secretary is a complex and long process. As the internship management process for a student lasts for one or two semesters, InternHub must be highly reliable for the entire duration. For instance, the internship reports uploaded by students and the evaluation forms written by instructors and secretaries should not be lost for a minimum of two semesters. To ensure that, InternHub provides sufficient data storage like 400 GB for all users. Also, the system should handle the huge data entrances. For example, in the deadline times, there could be lots of students, maximum 550 students, who upload their internship reports to the system. If their internship reports are approximately 20 MB, there could be data entrance up to 11 GB to the system. Thus, InternHub is designed to handle these kinds of huge data entrances at a specific time. Therefore, the second item of top two design goals is chosen as reliability.

2.0 High-Level Software Architecture

2.1 Subsystem Decomposition

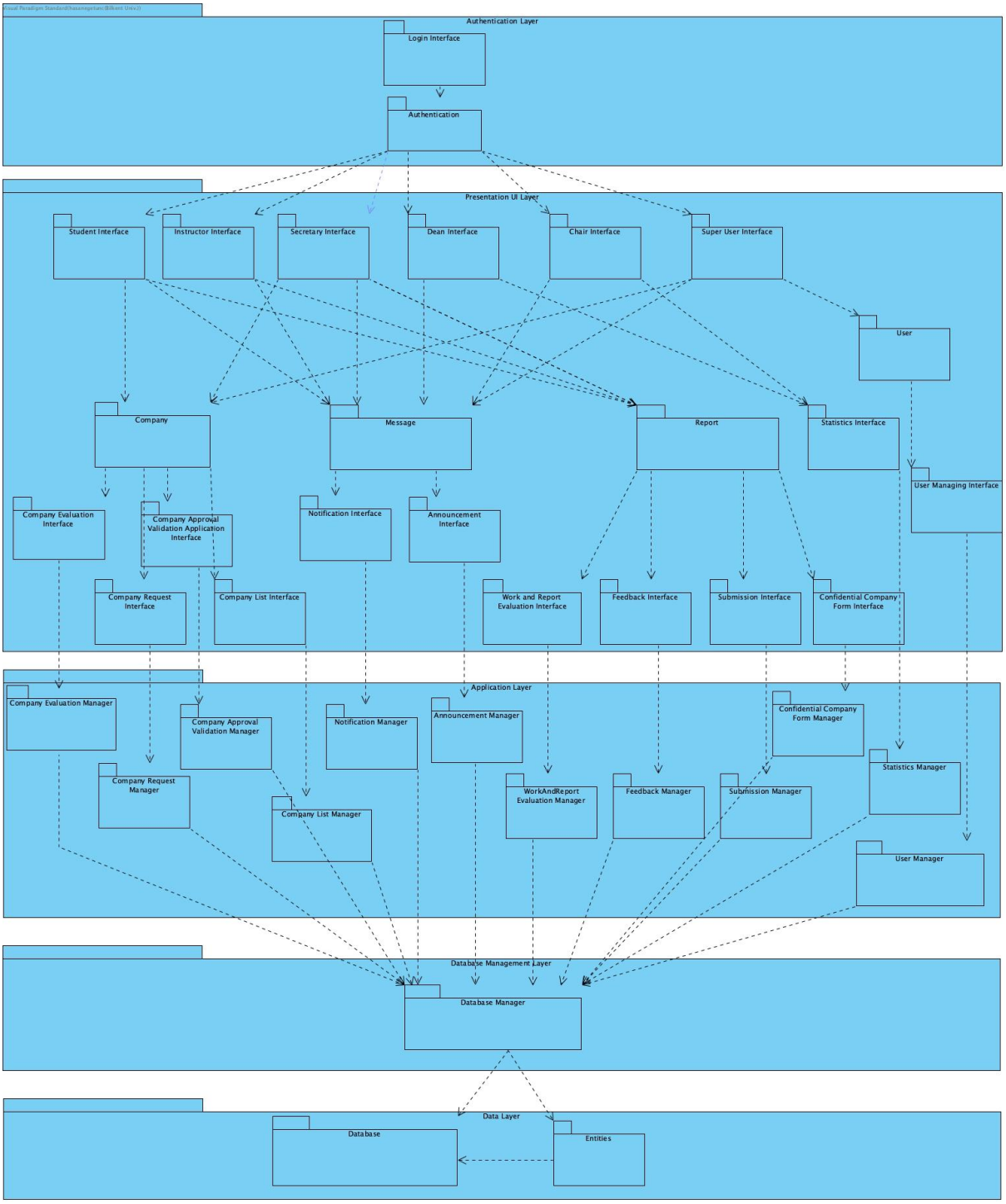

Figure-1: *Subsystem Decomposition Diagram of InternHub*

For high resolution please refer to the link below:
https://drive.google.com/file/d/16XdNalifSSlFuoRRawyom2DtTLWgpoP2/view?usp=share_link

For the internship management system, five layered architecture is implemented as seen above that decomposes the system into layers that are authentication layer, presentation layer, application layer, database management layer, and data Layer from top to bottom. This architectural approach divides the program structure into five distinct substructures, each possessing its own classes. We believe that employing this architecture enhances the security, functionality, maintainability, and usability of the system, all of the nonfunctional requirements will be mentioned below as the focus of layers concern them.

2.1.1 Authentication Layer

Authentication layer of InternHub consists of a login interface and authentication subsystem. InternHub users encounter with the login page, and depending on the data provided by the authentication subsystem directs either them to the proper interface regarding their roles; thereby, connecting the authentication layer to the presentation layer or not allowing authorization by displaying the right message (i.e. user does not exist). Hence, the authentication layer serves for security.

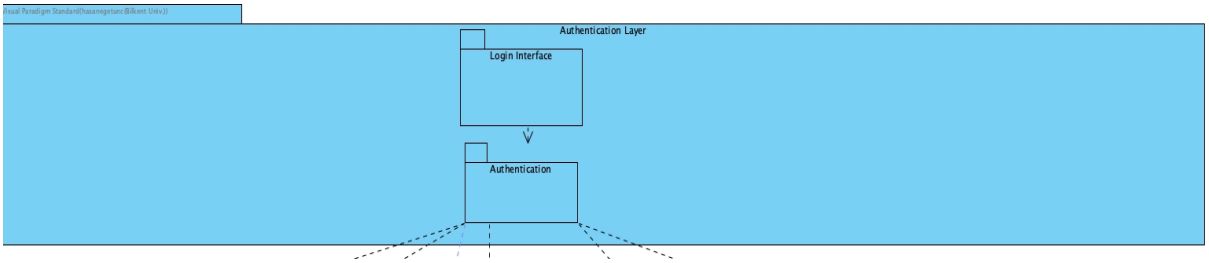

Figure-2: *Authentication Layer of InternHub*

2.1.2 Presentation UI Layer

Presentation layer of InternHub includes related user interfaces for all actors. User interfaces communicate with the subsystems -Company, Messages, Report, and User- depending on the actor type. See the dependency rows below for a better understanding. Each subsystem has its own interfaces, for example, company subsystem has company evaluation, company request, company list, and company approval validation application interfaces, which may alternate in functionality depending on the actor. To illustrate, the company approval validation application interface sends a form to the student to carry out the application; on the other hand, it displays the list of all applications made to the department secretary. Indeed, the two of them could be presented as independent interfaces, but since they belong to the same functionality of the same subsystem, it is preferred in this way. A similar argument holds for other subsystems as well. For instance, the feedback interface cannot be the same for the instructor and student. Finally, the user subsystem is where user entities are kept and manipulated (i.e. creation or deletion), which can only be performed by the superuser. To conclude, user interfaces communicate with other subsystems via interfaces provided by these subsystems' interfaces, so the presentation layer serves for functionality and usability.

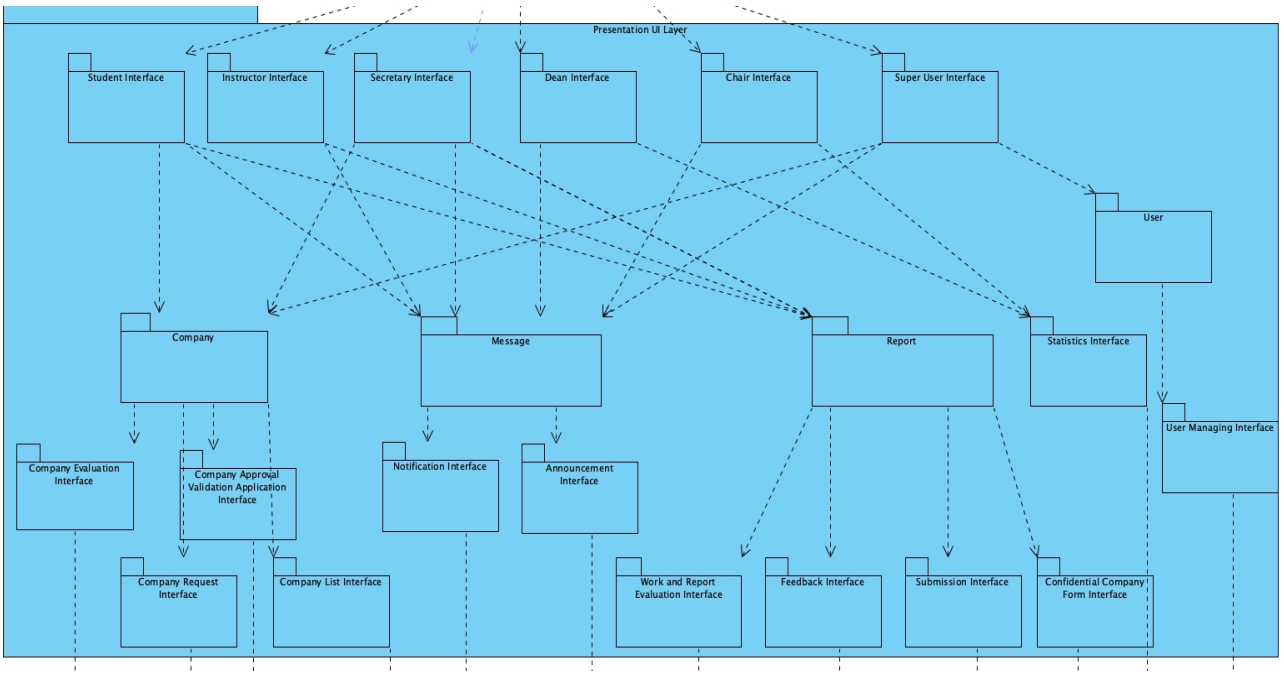

Figure-3: *Presentation Layer of InternHub*

2.1.3 Application Layer

Application layer is where the management of operations carried through interfaces in the presentation layer occurs. In other words, backend operations are performed in this layer. For instance, when an appropriate actor makes an announcement by clicking the related button, the announcement manager decides how to create this announcement, where to store it, and on which users' home page it should be displayed, with which data is included in it. In short, the application layer manages the operations performed on user interfaces, so the application layer serves for functionality

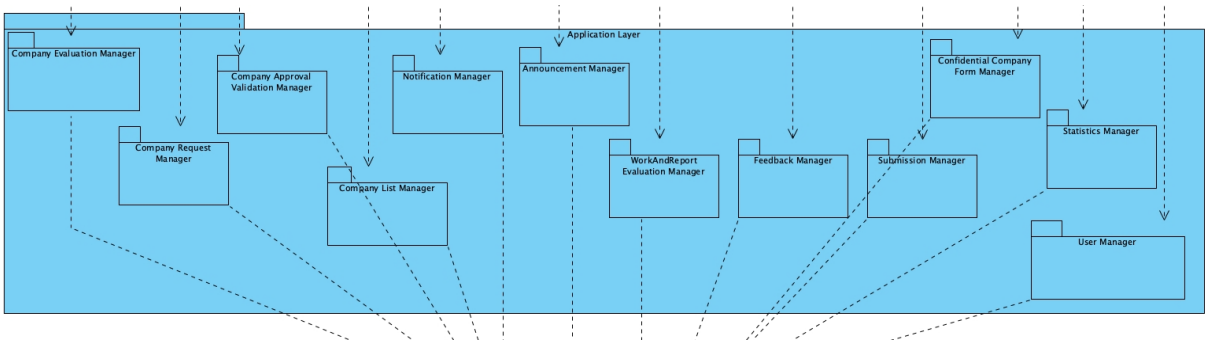

Figure-4: *Application Layer of InternHub*

2.1.4 Database Manager Layer

As illustrated in the previous section with an announcement example, the application layer may want to create new objects and store them properly. Obviously, these are database operations, so the application layer needs to communicate with the database regarding all its managers. Instead of communicating with each manager to the database separately, they can talk to a single database manager, which then performs the changes on

the database and entities. In other words, this layer acts as a transactional intermediary between the application and data layers, facilitating the exchange of data between the two layers. Indeed, this layer serves for maintainability because when an update is required on the database, making necessary changes on the database manager is sufficient, which prevents one from changing every single manager in the application layer.

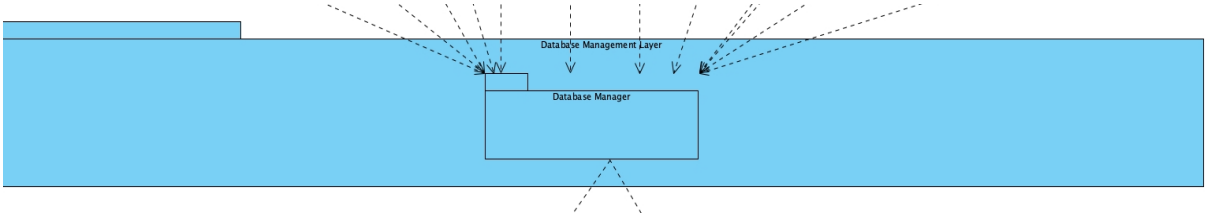

Figure-5: *Database Manager Layer of InternHub*

2.1.5 Data Layer

Data layer is the base layer on which all layers are constructed, and it performs changes on the existing database and entities based on the commands reaching it from the database manager.

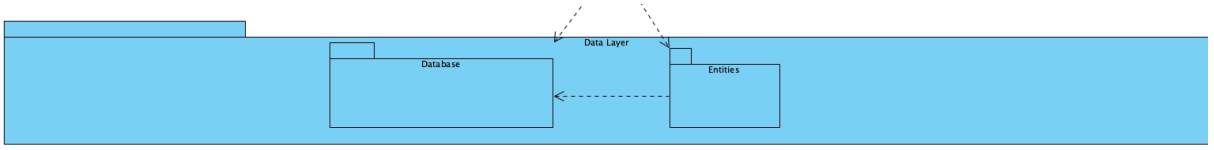

Figure-6: *Data Layer of InternHub*

2.2 Deployment Diagram

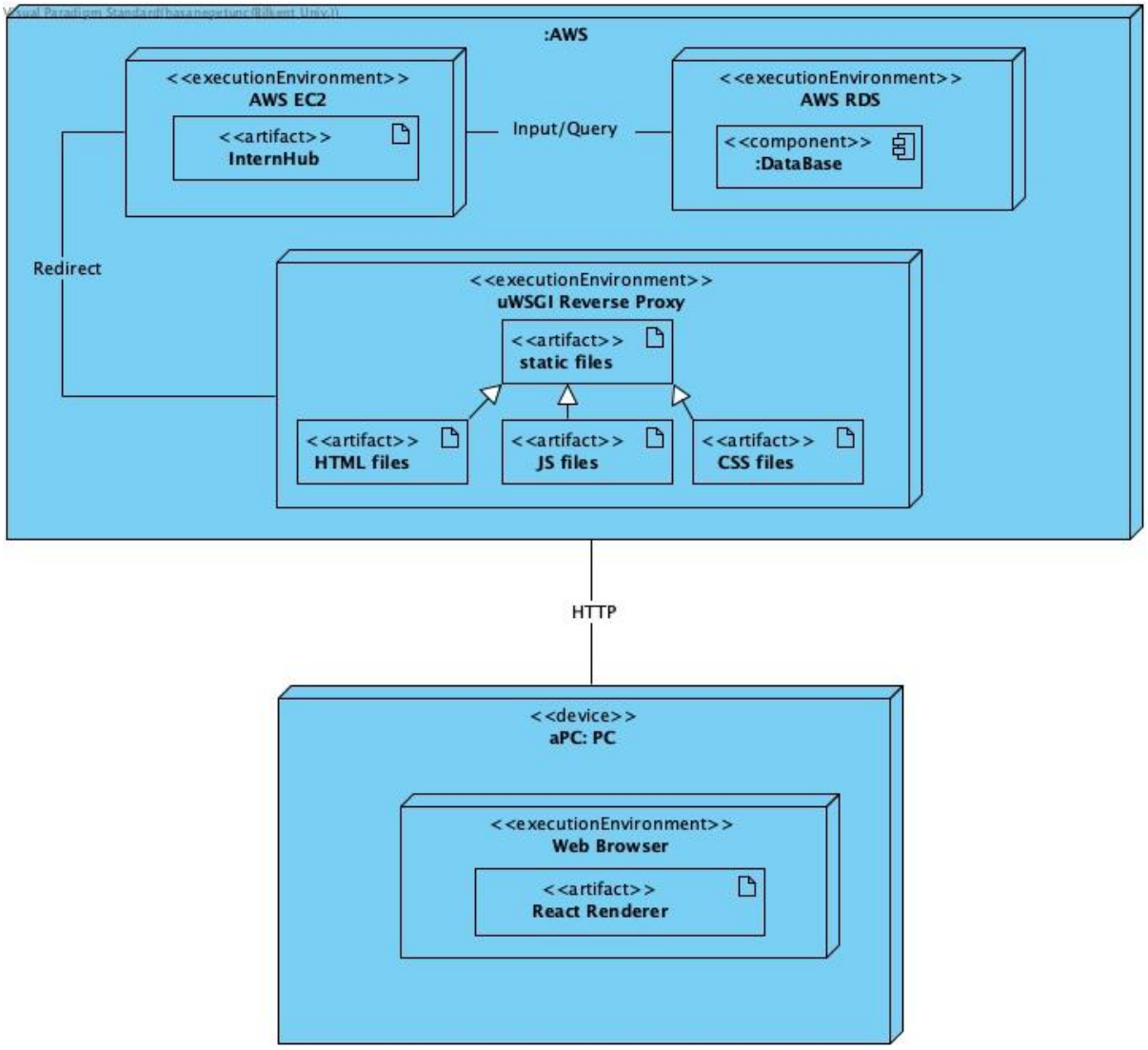

Figure-7: *Deployment Diagram of InternHub*

InternHub is hosted on AWS, while InternHub is executed on an AWS EC2 instance, the database is stored in AWS RDS. The existence of a device that is able to run ES5 or newer JavaScript versions on its web browsers is sufficient to use InternHub. The reverse proxy mechanism is beneficial to deliver the HTTP requests made by the user and also to deliver the response given by the system to the user again. In addition, reverse proxy presents required static files to users so that React Renderer can work.

2.3 Hardware/Software Mapping

InternHub is a small to medium-scaled web application that requires at least one server to host the backend of the InternHub and users must have a device with browser access/internet connection to use the application. As InternHub adopts the client-server model, it is optimized for modern browsers that support at least HTML5, CSS3, and ES5 on computers and mobile phones.

Since InternHub users frequently view reports and forms, the server should be powerful in terms of CPU and RAM. Moreover, InternHub is officially supported to run on a GNU/Linux environment, which can be another hardware specification constraint. Lastly, we should consider that the SQLite database runs on the same server as InternHub.

Considering the given requirements, we can pick an AWS EC2 instance with the following specifications:

m6g.xlarge:

● 4 vCPUs (AWS Graviton2 64-Core ARMv8 2.5 GHz)
● 16 GB RAM
● 500 GB SSD (For storing company-related documents,report submissions, forms, and logs.)

This setup is expected to handle up to 1650 concurrent users in a year. That is because the total number of students admitted to the engineering faculty is 550 in a year. Since 3rd and 4th-grade students receive internship-related courses, we need to double this number at least, and also, there must be some students retaking these courses, so we can triple 550 to set an upper bound. Each student is assumed to upload two internship reports, each of which is presumed to be 20MB in size, and other forms filled by the secretary and instructor for a particular student are also assumed to be 10MB in size. Then we need to store 50MB of data for 1650 students a year, so the calculation gives nearly 80GB of SSD requirement for annual data storage. Assuming Bilkent University wants to store the reports of a particular student for five years, we can determine SSD size as 400GB. Since 1GB of SSD costs 0.10$ per month, we have 40$ SSD cost and approximately $200 cost AWS EC2 instance cost, so the total cost of 240$. Even though AWS is scalable, since we made an upper approximation, we will not possibly need rescaling.

2.4 Persistent Data Management

SQLite is a popular choice for developers who need a lightweight and self-contained database management system. It is a relational database that uses SQL to manage and query data, making it a familiar choice for those who have worked with other SQL-based databases.

One advantage of using SQLite is that it does not require a separate server. Instead, the database is stored in a single file on the local machine, making it easy to manage and share among our team members. This also makes it a good choice for small to medium-sized projects that do not require high traffic. Considering this fact and that our project will encompass engineering students only, SQLite seems to be a plausible choice.

Another advantage of SQLite is its simplicity. It is easy to set up and does not require much configuration, making it a good choice for our team to increase progress speed.

In addition, SQLite also supports many of the same data types and commands as other SQL-based databases, making it a versatile choice for a wide range of applications.

And since it is an open-source database, many resources are available for developers who need help or guidance.

Finally, SQLite outstands among other databases because Django framework is being used in the project, and this framework is most popularly used with SQLite database by initializing a project connected to this database. In addition, Django reflects the changes made on object classes to existing databases with 2 to 3 lines of code. So, we decided to use SQLite as a database for the project.

2.5 Object Access Matrix

	Authentication	Profile	Notification	Announcement	Placement	CompanyForm	Submission	Feedback	View	Settings
InternHubUser	login()	-	-	-	-	-	-	-	-	-
Student	login() forgetPassword() 	add_contact() remove_contact()	receiveNotification() filter_last_five_notifications()	filter_last_five_announcements_in_department()	-	apply_for_company_approvalvalidation() request_company() evaluate_Company() filter_companies_by_department()	make_submission() request_extension()	get_given_feedbacks() get_status_of_last_feedback()	ViiewCompaniesList() view_notifications() view_submissions() view_feedbacks() view_announcementst()	changePassword()
Instructor	login() forgetPassword() 	add_contact() remove_contact()	receiveNotification() filter_last_five_notifications()	filter_last_five_announcements_in_department()	-	CreateWorkAndReportEvaluationForm UpdateWorkAndReportEvaluationForm	set_deadline() assign_submission()	give_feedback() set_feedback_status()	view_students() view_notification() view_submissions() view_WEFs() view_announcement()	changePassword()
DepartmentSecretary	login() forgetPassword() 	add_contact() remove_contact()	receiveNotification() filter_last_five_notifications()	makeAnnouncement() filter_last_five_announcements_in_department()	assign_instructor_to_internship() assign_instructors_to_internships_randomly()	add_company() remove_company() approve_company_request() reject_company_request() filter_company_requests_by_department() approveCAVA() rejectCAVA() filter_cava_by_department() mark_company_evaluation_as_considered() CreateConfidentialCompanyForm UpdateConfidentialCompanyForm	-	-	view_companies_list() view_students() view_instructor() view_Announcement() view_WREs() view_CCFs() view_final_PDFs()	changePassword()
Dean	login() forgetPassword() 	add_contact() remove_contact()	receiveNotification() filter_last_five_notifications()	makeAnnouncement() filter_last_five_announcements_in_department()	-	-	-	-	view_students() view_instructor() view_statistic() view_announcemenst()	changePassword()
Chair	login() forgetPassword() 	add_contact() remove_Contact()	receiveNotification() filter_last_five_notifications()	makeAnnouncement() filter_last_five_announcements_in_department()	-	-	-	-	view_companies_list() view_students() view_instructor() view_statistics() view_announcements()	changePassword()
SuperUser	login() forgetPassword() 	add_contact() remove_contact()	receiveNotification() filter_last_five_notifications()	makeAnnouncement() filter_last_five_announcements_in_department()	create_user() delete_user()	add_company() remove_company()	-	-	view_companies_list() view_students() view_instructor() view_announcements() view_users()	changePassword()

Figure-8: *Object Access Matrix of InternHub*

In the above diagram, parameters of methods are not shown, the application layer (3.4.2) and the presentation layer (3.4.3) can be referred to see the parameters. Camel case notations imply that Django classes are used to achieve related functionality.

2.6 Boundary Conditions

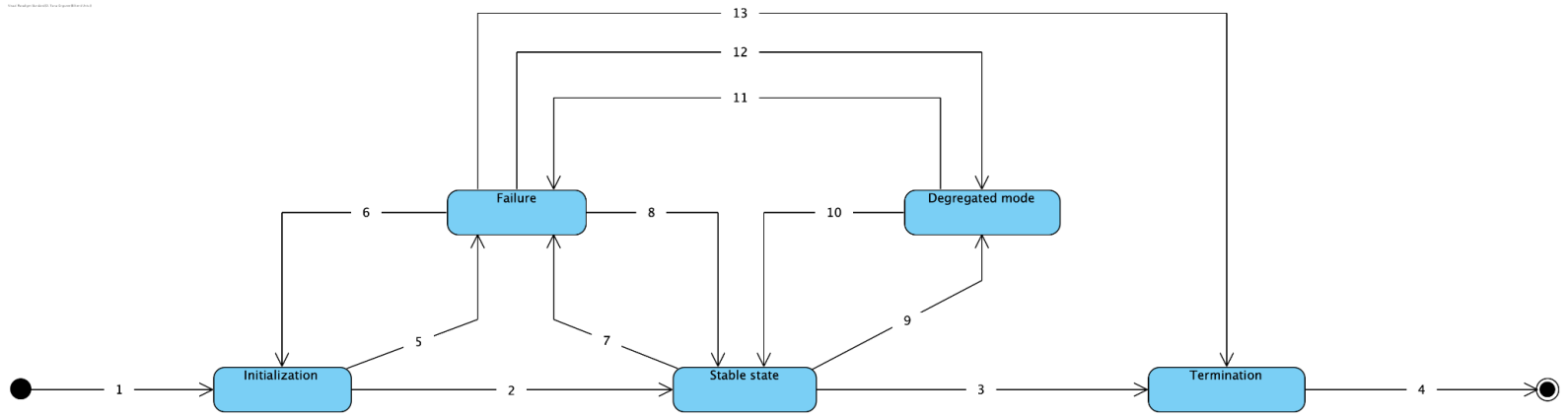

Figure-9: *System State Diagram*

Checklist for #1

1. Is the database connected to the server?
    - Boundary Condition: The system must establish a functional connection between the server (AWS) and the database stored in AWS RDS.
    - How to verify: Validate the connection status between the server and the database. Monitor system logs or employ database management tools to ensure successful connectivity. Perform database operations such as querying or data insertion to confirm the validity of the connection.
2. Is the run server command executed?
    - Boundary Condition: The system must successfully execute the command to initiate the server.
    - How to verify: Monitor system logs or observe console output to identify any errors or confirm the successful execution of the run server command. Additionally, access the designated URL or port associated with the server to ensure its availability and responsiveness.
3. Is the contact environment initialized?
    - Boundary Condition: The system must appropriately initialize the contact environment, encompassing the necessary configuration and setup of libraries, modules, or services pertaining to contact management.
    - How to verify: Inspect system logs or console output for initialization messages related to the contact environment. Perform basic contact management operations, such as creating, updating, or deleting contacts, to ensure the proper setup and functionality of the contact environment.
4. Is uWSGI deployed?
    - Boundary Condition: The system must deploy uWSGI, effectively enabling the communication between the web application and the server.
    - How to verify: Monitor uWSGI logs or observe console output for any indications of successful deployment. Test access to the web application

through the designated URL or port to confirm its functional and expected behavior.

Checklist for #2

1. Is the database connection maintained appropriately?
    - Boundary Condition: The system must maintain a stable and reliable connection with the database throughout its operation.
    - How to verify: Continuously monitor the status of the database connection. Regularly review the system logs for any connection errors or disruptions. Implement appropriate error handling mechanisms to gracefully address any database connection issues that may arise.
2. Can users utilize the functionality appointed to them from the system without any trouble?
    - Boundary Condition: Users should be able to access and utilize the designated functionality of the system seamlessly, without encountering any significant issues or disruptions.
    - How to verify: Conduct user testing and gather feedback to identify any usability issues or functional limitations. Monitor the system logs for any error messages related to user functionality. Maintain up-to-date user documentation or guides to assist users in effectively utilizing the system's functionality.
3. Can HTTP requests be handled conveniently?
    - Boundary Condition: The system must efficiently handle incoming HTTP requests without encountering major obstacles or performance bottlenecks.
    - How to verify: Utilize load testing tools or frameworks to simulate a high volume of concurrent HTTP requests. Monitor the system's response time and resource usage during the testing process. Analyze the system logs for any request handling errors or latency issues. Optimize the system's code and configuration to ensure the efficient handling of HTTP requests.
4. Can HTTP responses be sent back to the users as expected?
    - Boundary Condition: The system must accurately generate and send HTTP responses back to users within an acceptable time frame.
    - How to verify: Conduct integration testing by sending various types of requests to the system and verifying the generation of correct responses. Monitor the system logs for any errors or anomalies related to responses. Perform benchmarking to compare response times against defined performance targets.
5. Does the reverse proxy method work correctly?
    - Boundary Condition: If a reverse proxy is implemented, it must function correctly by appropriately routing incoming requests to the relevant backend servers.
    - How to verify: Configure and deploy the reverse proxy in accordance with the system's requirements. Monitor the reverse proxy logs for any errors or warnings related to request routing. Test different URL paths to ensure the reverse proxy correctly forwards requests to the backend servers. Conduct load testing to assess the reverse proxy's performance under high traffic conditions.

Checklist for #3

1. Is the database connection intentionally interrupted?
    - Boundary Condition: The system should not intentionally interrupt the database connection.
    - How to verify: Monitor system logs or employ database monitoring tools to identify any intentional disruptions or errors related to the database connection. Investigate the cause of any observed interruptions and rectify them accordingly.
2. Are the functionalities given to users intentionally hindered?
    - Boundary Condition: The system should not intentionally hinder or limit the functionalities provided to users.
    - How to verify: Conduct user testing, gather user feedback, or monitor system logs for any indications of intentional hindrance or limitations in user functionality. Ensure that all specified functionalities are accessible and available to users without intentional obstructions.
3. Is the system not receiving any HTTP requests intentionally?
    - Boundary Condition: The system should not intentionally prevent or reject incoming HTTP requests.
    - How to verify: Monitor the system logs or employ network monitoring tools to identify any intentional blocks or rejections of HTTP requests. Ensure that the system remains capable of receiving and processing HTTP requests as expected.
4. Is the reverse proxy method no longer functioning?
    - Boundary Condition: The reverse proxy method should continue to function as intended and effectively route incoming requests to the appropriate backend servers.
    - How to verify: Monitor the reverse proxy logs or observe system logs for any indications of intentional disruptions or errors related to the reverse proxy functionality. Test different URL paths to ensure that the reverse proxy correctly forwards requests to the intended backend servers.
5. Is uWSGI deliberately undeployed?
    - Boundary Condition: The system should not intentionally undeploy uWSGI.
    - How to verify: Monitor system logs or observe console output for any indications of intentional uWSGI undeployment. Ensure that uWSGI remains deployed and functioning as expected to handle the communication between the web application and the server.

Checklist for #4

1. Will the system be totally shut down?

- Boundary Condition: The system may be intentionally shut down for specific reasons or events.
- How to verify: Monitor system logs or receive notifications from administrators or maintenance personnel regarding planned shutdown activities. Verify system accessibility and functionality to confirm the complete shutdown state.

Checklist for #5

1. Did any sudden database interruption occur?
    - Boundary Condition: The system should not experience any sudden interruptions or failures in the database connection during its operation.
    - How to verify: Monitor system logs or employ database monitoring tools to identify any unexpected disruptions or errors related to the database connection. Implement proper error handling and recovery mechanisms to mitigate the impact of potential interruptions. Perform regular database maintenance and monitoring to minimize the occurrence of sudden interruptions.
2. Did any uWSGI deployment problem occur?
    - Boundary Condition: The uWSGI deployment should occur without any significant problems or failures during the system's initialization phase.
    - How to verify: Monitor uWSGI logs or observe system logs for any indications of deployment problems or errors related to uWSGI. Implement proper deployment procedures and configuration management to ensure the successful deployment of uWSGI. Conduct thorough testing and monitoring during the deployment process to detect and address any potential issues promptly.

Checklist for #6

1. Can any sudden database interruption be fixed?
    - Boundary Condition: If a sudden database interruption occurs, the system should be capable of resolving the issue and restoring the database connection to a functional state.
    - How to verify: Monitor system logs or employ database monitoring tools to identify the cause of the interruption. Implement appropriate error handling and recovery mechanisms to address the issue promptly. Perform troubleshooting steps, such as checking network connectivity, restarting services, or resolving any database-related errors. Verify the successful resolution of the interruption by confirming the reestablishment of a stable and reliable database connection.
2. Can uWSGI be redeployed again?
    - Boundary Condition: If there is a deployment problem or failure with uWSGI, the system should allow for redeployment to restore the proper functioning of uWSGI.
    - How to verify: Analyze uWSGI logs or observe system logs to identify the root cause of the deployment problem. Address any configuration or deployment errors that caused the failure. Implement proper deployment procedures and configuration management to ensure successful redeployment. Monitor the redeployment process to confirm that uWSGI is deployed correctly and operating as expected.

Checklist for #7

1. Were database connections lost?

- Boundary Condition: The system should not experience the loss of database connections, resulting in the inability to establish or maintain a stable connection with the database.
- How to verify: Monitor system logs or employ database monitoring tools to identify any errors or disruptions related to database connections. Implement proper error handling and connection management mechanisms to prevent and address any connection losses. Regularly test and monitor the database connection to ensure its stability and availability.

2. Were the server or host experiencing troubles?
    - Boundary Condition: The server or host on which the system is running should not encounter significant issues or disruptions that hinder its normal operation.
    - How to verify: Monitor system logs or employ server monitoring tools to identify any errors, crashes, or performance degradation related to the server or host. Implement proper server management practices, including monitoring resource usage, applying updates and patches, and conducting regular maintenance tasks. Resolve any server-related issues promptly to prevent system failures.
3. Did the storage reach its allocated amount?
    - Boundary Condition: The storage allocated for the system should not reach its capacity limit, resulting in the inability to store or retrieve data.
    - How to verify: Monitor the storage usage and available space on the storage system hosting the system's data. Implement proper storage capacity planning and scaling strategies to ensure sufficient storage space. Monitor system logs for any warnings or errors related to storage capacity. Implement alerts or automated processes to notify administrators when storage usage approaches its limit.

Checklist for #8

1. Can the database connection be fixed?
    - Boundary Condition: If the database connection is lost or disrupted, the system should allow for the resolution of the issue and the restoration of a stable connection with the database.
    - How to verify: Monitor system logs or employ database monitoring tools to identify the cause of the connection issue. Implement appropriate error handling and recovery mechanisms to address the problem. Perform troubleshooting steps, such as verifying network connectivity, restarting services, or resolving any database-related errors. Verify the successful resolution by confirming the reestablishment of a stable and reliable database connection.
2. Did the server or host fix its troubles?
    - Boundary Condition: If the server or host encounters issues or disruptions, the system should allow for the resolution of those problems and the restoration of normal operation.
    - How to verify: Monitor system logs or employ server monitoring tools to identify the root cause of the server or host troubles. Implement appropriate troubleshooting steps based on the nature of the problem, such as analyzing

system resource usage, applying updates or patches, or resolving hardware or software failures. Confirm the successful resolution by monitoring the system's stability and performance.

3. Did the storage expand from the host?
    - Boundary Condition: If the allocated storage reaches its capacity limit, the system should provide the capability to expand the storage to accommodate additional data.
    - How to verify: Monitor storage usage and available space on the host system. Implement appropriate storage management practices, such as adding additional storage devices, expanding existing storage capacity, or migrating data to larger storage systems. Verify the successful expansion by confirming the availability of sufficient storage space for data storage and retrieval.

Checklist for #9

1. Did malicious attacks happen (DDoS)?
    - Boundary Condition: The system may experience malicious Distributed Denial of Service (DDoS) attacks that disrupt the normal operation and availability of the system.
    - How to verify: Monitor network traffic, system logs, or employ security monitoring tools to identify patterns or signs of DDoS attacks. Implement proper DDoS mitigation techniques, such as rate limiting, traffic filtering, or deploying DDoS protection services. Validate the effectiveness of these measures by monitoring the system's response to the attacks and assessing its availability.
2. Did some plugin issues occur?
    - Boundary Condition: The system may encounter issues with plugins or extensions that impact its functionality and performance.
    - How to verify: Monitor system logs or employ plugin monitoring tools to identify any errors or compatibility issues with installed plugins or extensions. Implement proper plugin management practices, such as updating plugins to the latest versions, resolving conflicts, or disabling problematic plugins temporarily. Validate the resolution of plugin issues by testing the system's functionality and ensuring that the affected features are restored.
3. Did a lot of bandwidth data transfer occur at once?
    - Boundary Condition: The system may experience a sudden surge in bandwidth usage or high data transfer rates that can lead to performance degradation or limited availability.
    - How to verify: Monitor network traffic, system logs, or employ bandwidth monitoring tools to identify spikes or unusual patterns in bandwidth usage. Implement appropriate network traffic management techniques, such as traffic shaping, quality of service (QoS) policies, or load balancing. Verify the effectiveness of these measures by monitoring the system's performance during periods of high data transfer and ensuring that the system remains accessible to users.
4. Are users unable to make uploads to the system?
    - Boundary Condition: Users may encounter difficulties or restrictions in uploading files or data to the system.

- How to verify: Gather user feedback, monitor system logs, or conduct user testing to identify any issues or errors related to file uploads. Verify the availability of sufficient storage space, proper file size limits, and correct permissions for user uploads. Resolve any restrictions or issues to restore the ability for users to make uploads successfully.

Checklist for #10

1. Can the website be protected from malicious attacks?
    - Boundary Condition: The system should implement measures to protect the website from malicious attacks and ensure its availability and security.
    - How to verify: Implement robust security measures, such as firewalls, intrusion detection systems (IDS), or web application firewalls (WAF). Regularly update security patches and monitor system logs for any signs of malicious activity. Perform security audits and penetration testing to identify vulnerabilities and implement necessary fixes. Verify the effectiveness of these measures by monitoring the system's resilience against attacks and ensuring the website remains accessible to legitimate users.
2. Can plugin issues be fixed?
    - Boundary Condition: The system should allow for the resolution of plugin issues to restore full functionality and performance.
    - How to verify: Identify and prioritize plugin-related issues based on their impact on system functionality. Perform troubleshooting steps, such as updating plugins to the latest versions, resolving conflicts, or reaching out to plugin developers for support. Test the affected features to confirm that the issues have been resolved and that the system operates without any plugin-related limitations or errors.
3. Can the amount of bandwidth data transfer be reduced?
    - Boundary Condition: The system should implement measures to reduce excessive bandwidth usage or data transfer rates to ensure optimal performance and stability.
    - How to verify: Implement bandwidth management techniques, such as traffic shaping, compression, or caching mechanisms. Optimize file sizes, implement content delivery networks (CDNs), or implement rate limiting policies. Monitor the system's network traffic and bandwidth usage to ensure that the implemented measures effectively reduce the amount of data transfer. Verify the system's stability and performance after implementing these measures.
4. Is the system open to uploads from users?
    - Boundary Condition: The system should allow users to make uploads, ensuring that appropriate security measures and limitations are in place.
    - How to verify: Review and update security measures related to file uploads, such as implementing file type restrictions, size limits, and proper validation mechanisms. Test the upload functionality to ensure that users can upload files without encountering errors or restrictions. Monitor system logs for any potential issues or vulnerabilities related to user uploads and address them accordingly.

Checklist for #11

1. Did a malicious attack penetrate the system?
    - Boundary Condition: The system may experience a successful penetration by a malicious attack, leading to unauthorized access, data breaches, or other security compromises.
    - How to verify: Monitor system logs, security monitoring tools, or intrusion detection systems to identify any signs of a successful attack. Analyze network traffic patterns, system behavior, or unusual access attempts. Implement proper security measures, such as firewalls, intrusion prevention systems (IPS), or security incident response procedures, to detect and mitigate attacks. Validate the effectiveness of these measures by monitoring the system's security and response to potential attacks.
2. Did a plugin issue make the server inaccessible?
    - Boundary Condition: A critical plugin issue may lead to server inaccessibility, affecting the availability and functionality of the system.
    - How to verify: Monitor system logs, plugin logs, or employ monitoring tools to identify any errors, crashes, or conflicts caused by plugins. Analyze the impact of the plugin issue on server resources, performance, or critical system processes. Implement proper plugin management practices, such as disabling problematic plugins, updating to the latest versions, or seeking support from plugin developers. Verify the successful resolution of the issue by testing the system's accessibility and confirming that the server is operational.

Checklist for #12

1. Were malicious attacks solvable?
    - Boundary Condition: The system should provide the capability to identify and resolve malicious attacks, mitigating their impact and restoring the system's security.
    - How to verify: Analyze system logs, security monitoring tools, or intrusion detection systems to identify the extent of the malicious attack and its impact on the system. Implement incident response procedures, such as isolating affected components, patching vulnerabilities, or resetting compromised credentials. Deploy security patches or updates, strengthen authentication mechanisms, and conduct security audits to identify and fix any vulnerabilities. Verify the successful mitigation of the attack by monitoring system behavior and security indicators.
2. Can the server be made accessible again?
    - Boundary Condition: The system should allow for the restoration of server accessibility, ensuring the resumption of normal operation and user access.
    - How to verify: Identify the root cause of the server inaccessibility, whether it is related to hardware failures, software conflicts, or network issues. Perform appropriate troubleshooting steps, such as restarting services, restoring network connectivity, or resolving system errors. Implement disaster recovery or backup procedures to restore the system to a known good state if necessary. Validate the restoration of server accessibility by monitoring

system availability and ensuring that users can access the system without limitations.

Checklist for #13

1. Can the malicious attack not be solved?
    - Boundary Condition: The system may encounter a malicious attack that cannot be effectively mitigated, compromising the system's security and integrity beyond recovery.
    - How to verify: Analyze the extent and impact of the malicious attack on the system. Assess the effectiveness of security measures, incident response procedures, and attempts to mitigate the attack. If the attack cannot be effectively resolved, and the system remains vulnerable or compromised, it may lead to the decision to terminate the system.
2. Is the system unable to be made accessible to the users?
    - Boundary Condition: Despite attempts to resolve issues and restore server accessibility, the system remains inaccessible or unusable for users.
    - How to verify: Monitor system logs, conduct thorough troubleshooting, and perform diagnostic tests to identify the cause of the inaccessibility. Implement appropriate measures to address server or network issues, software conflicts, or other factors impacting system accessibility. If all efforts fail to restore user access to the system, termination may be considered.
3. Is the database connection unable to be established?
    - Boundary Condition: The system fails to establish a stable and reliable connection with the database, hindering essential operations and rendering the system unusable.
    - How to verify: Monitor database connection attempts, error logs, and network connectivity. Implement proper error handling and connection management mechanisms to address database connection issues. If all attempts to establish a functional database connection fail, and alternative solutions are not feasible, termination may be necessary.
4. Are the troubles experienced by users unsolvable?
    - Boundary Condition: Despite efforts to address user-related issues and provide satisfactory resolutions, persistent and unsolvable troubles are experienced by users, impacting their ability to utilize the system effectively.
    - How to verify: Gather user feedback, monitor support requests, and conduct user testing to identify and understand the issues faced by users. Implement troubleshooting steps, user support, and training initiatives to address user troubles. If the issues persist and cannot be satisfactorily resolved, and the system becomes unviable for users, termination may be considered.

3.0 Low-Level Design

3.1. Object Design Trade-Offs

I. **Usability** versus **Security:** No two gate authentication is preferred in InternHub; that is because, there is no private data requiring strict security checks is stored in the system. Absence of two gate authentication increases usability by allowing quick log of users to the system, however, reduces security.

II. **Functionality** versus **Usability:** InternHub allows students to not only upload their internship reports to the system, but also makes it feasible for students to request company related issues and process of report. Since there are too many functions that can be done by users on InternHub, this situation limits usability.

III. **Maintainability** versus **Performance:** Object oriented design of InternHub decreases performance of the system because managers in the application layer needs to interact with entities which reduces performance, but increases maintainability since the system is decomposed into sub- compartments.

IV. **Reliability** versus **Cost:** Using of AWS EC2 instance and allocating a big SSD in size to store multiple copies increase reliability, but also increases cost which is calculated as 240$.

V. **Functionality** versus **Performance:** Students capability to upload various reports and instructors ability to create multiple forms increase functionality, but reduces performance since created documents overload the database.

3.2 Final Object Design

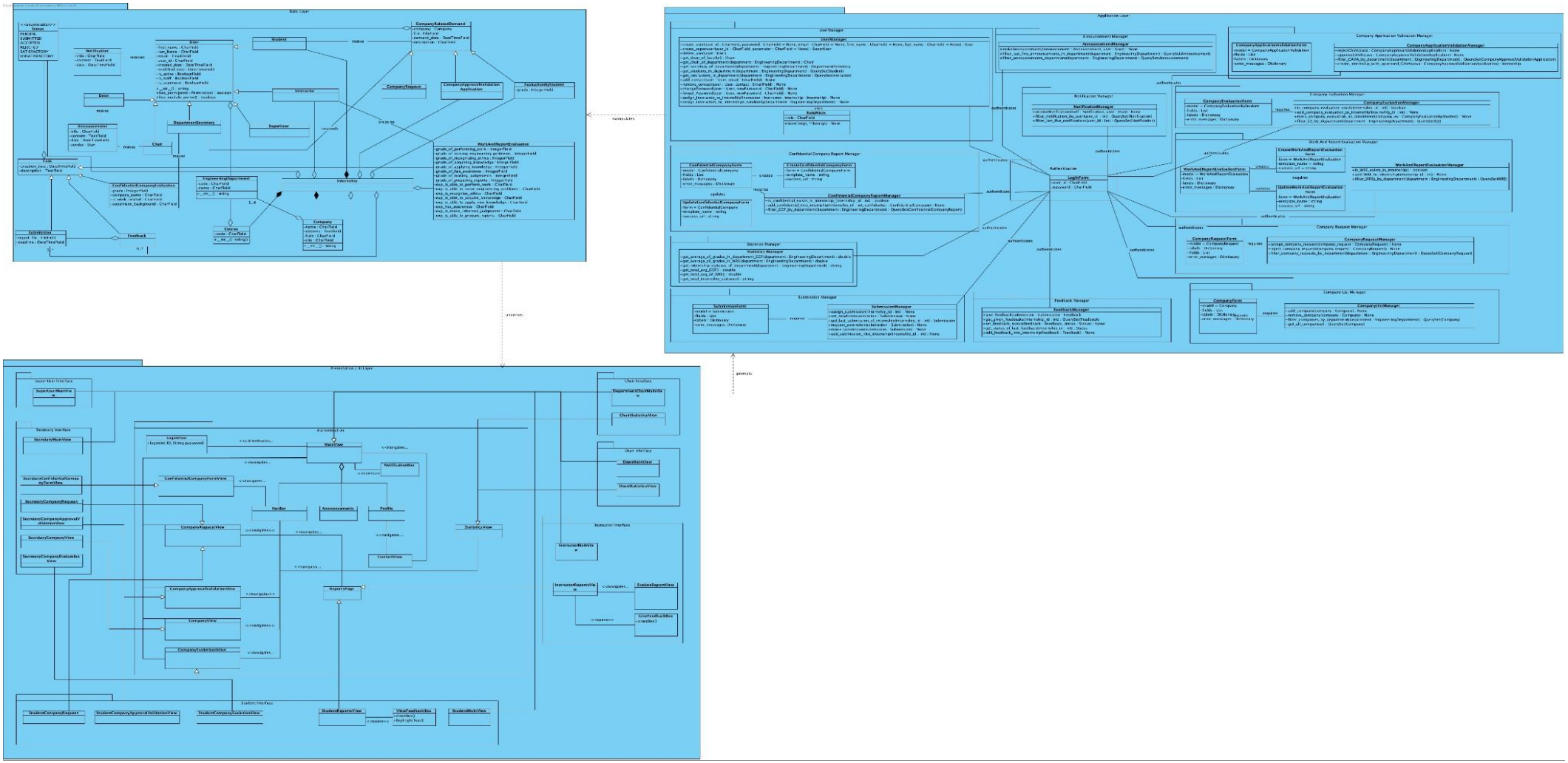

Figure-10: *Final Object Design of InternHub*

Note that the Presentation Layer classes does not involve any methods, because these methods are handled in related manager classes. Views are represented as classes because class based views can be used in Django. For the higher resolution, please refer to the link: https://drive.google.com/file/d/1gRF0YulVqLHNsnjdTRCfpIJmIDke_rqF/view?usp=share_lin

3.3 Packages

3.3.1 External Packages

The list of benefitted external package is as follows:

- **Django:** A high-level web framework for Python is provided by this package, allowing for the quick building of safe and dependable websites. To control backend logic and communicate with both the React frontend and the SQLite database, we have selected Django [1].
- **React:** This package is used to build the application's front end and provides a declarative and effective method for developing user interfaces [2].
- **SQLite:** We utilize this package as our preferred database option since it provides a compact and serverless solution for preserving data pertinent to the system's defined entities. Django and SQLite collaborate on data management [3].
- **Django Storage and Django Storages:** These packages offer an abstraction layer for handling files, support for various storage backends, and quick file uploading and downloading [4].
- **Axios:** This package provides a user-friendly and effective method for data collecting through HTTP calls to our Django backend, which is used in our React frontend [5].
- **React Router:** The package provides robust routing functionality for our React application, enabling it to manage navigation and render components based on specific URLs [6].
- **AWS:** The package is utilized to establish attachment properties between users and remote servers [7].
- **BCryptPasswordHasher:** Django includes several built-in password hashing algorithms, and one we've chosen to use is the BCryptPasswordHasher. This package is used for securely storing passwords in our database. Rather than storing user passwords in plain text, which would pose a significant security risk, this hasher stores a cryptographic hash of the password. When a user provides a password, the system hashes the input and compares it to the stored hash, allowing us to validate passwords without ever directly handling them. This hasher uses the bcrypt algorithm, which is currently considered to be one of the safest options for password hashing due to its resistance to brute-force and rainbow table attacks. This package plays a key role in our system, enhancing security and protecting user data [8].

3.3.2 Internal Packages

3.3.2.1. User Screens Package

This package includes all boundary classes, encompassing the system's user interfaces. It handles all interactions between the system and its users, facilitating tasks such as viewing notifications and announcements, and accessing various functions specific to each role.

3.3.2.1.1. Dean Screens Package

This package contains all boundary classes for the dean. It allows the dean to receive notifications, make announcements, view students, instructors, statistics, and announcements.

3.3.2.1.2. Students Screens Package

This package includes all boundary classes for students. It supports features like viewing notifications, applying for Company Approval Validation Applications (CAVA), accessing the company list, evaluating previous internships, making submissions, requesting report extensions, and viewing feedbacks.

3.3.2.1.3. Instructor Screens Package

This package involves all boundary classes related to the instructors. It enables instructors to view notifications and announcements, create and update the work and report evaluation form, set deadlines, assign submissions, provide feedback, set feedback status, view students, submissions, Work and Report Evaluation (WRE) forms, and announcements.

3.3.2.1.4. Secretary Screens Package

This package includes all boundary classes that manage the secretary's interaction with the system. It enables the secretary to receive notifications, make announcements, assign instructors, add and remove companies, approve or reject company requests, approve or reject CAVAs, filter CAVA by department, create and update Confidential Company Forms, mark company evaluations as considered, and view students, companies, instructors, announcements, and final PDFs.

3.3.2.1.5. Chair Screens Package

This package manages all boundary classes related to the chair role. It facilitates the chair's access to notifications, announcements, viewing companies, students, instructors, statistics, and announcements.

3.3.2.1.6. Superuser Screens Package

This package handles all boundary classes for superusers. Superusers are allowed to make announcements, create and delete users, view companies, students, instructors, announcements, and users.

3.3.2.2. Announcement Manager Package

This package manages the creation, distribution, and deletion of announcements. This package would be used by the roles that have permissions to create and disseminate announcements.

3.3.2.3. Notification Manager Package

This package handles all operations related to notifications, including creation, distribution, and removal. This would be applicable to all users as it covers the system's way of alerting users of updates or changes.

3.3.2.4. Company Evaluation Manager Package

This package facilitates student evaluations of companies they have interned with. It handles the input, storage, and retrieval of these evaluations.

3.3.2.5. Company Request Manager Package

This package manages student requests to add new companies to the system for internship opportunities. It handles the request submission process and ensures that these requests are seen by the appropriate authority for approval or rejection.

3.3.2.6. Company Approval Validation Manager Package

This package handles applications for company approval validation (CAVA). It ensures these applications are appropriately reviewed and responded to.

3.3.2.7. Company List Manager Package

This package is responsible for the listing, filtering, and retrieving details of companies in the system. It helps students to view and choose companies for their internships.

3.3.2.8. Work and Report Evaluation Manager Package

This package deals with the creation, update, and viewing of work and report evaluation forms. It is primarily used by instructors to assess student internship submissions.

3.3.2.9. Feedback Manager Package

This package manages the process of submitting and viewing feedback on internship reports. It provides interfaces for instructors to give feedback and for students to view it.

3.3.2.10. Submission Manager Package

This package manages the submission of internship reports by students. It helps in handling the submission process and ensures that all submissions are correctly stored and can be accessed by authorized users.

3.3.2.11. Confidential Company Form Manager Package

This package manages the process of filling out and updating Confidential Company Forms by the secretary.

3.3.2.12. Statistics Manager Package

This package handles the collection and presentation of statistical data related to completed/incomplete internships.

3.3.2.13.User Manager Package

This package manages all operations related to user profiles, including new user registration, user authentication, updating user profile information, and maintaining data integrity and security.

3.4 Class Diagrams

Diagrams will be represented in this section based on the separation of layers. Each layer will appear with the classes they involve. This section can be considered as partitioning of the final object design (3.2).

3.4.1 Application Layer

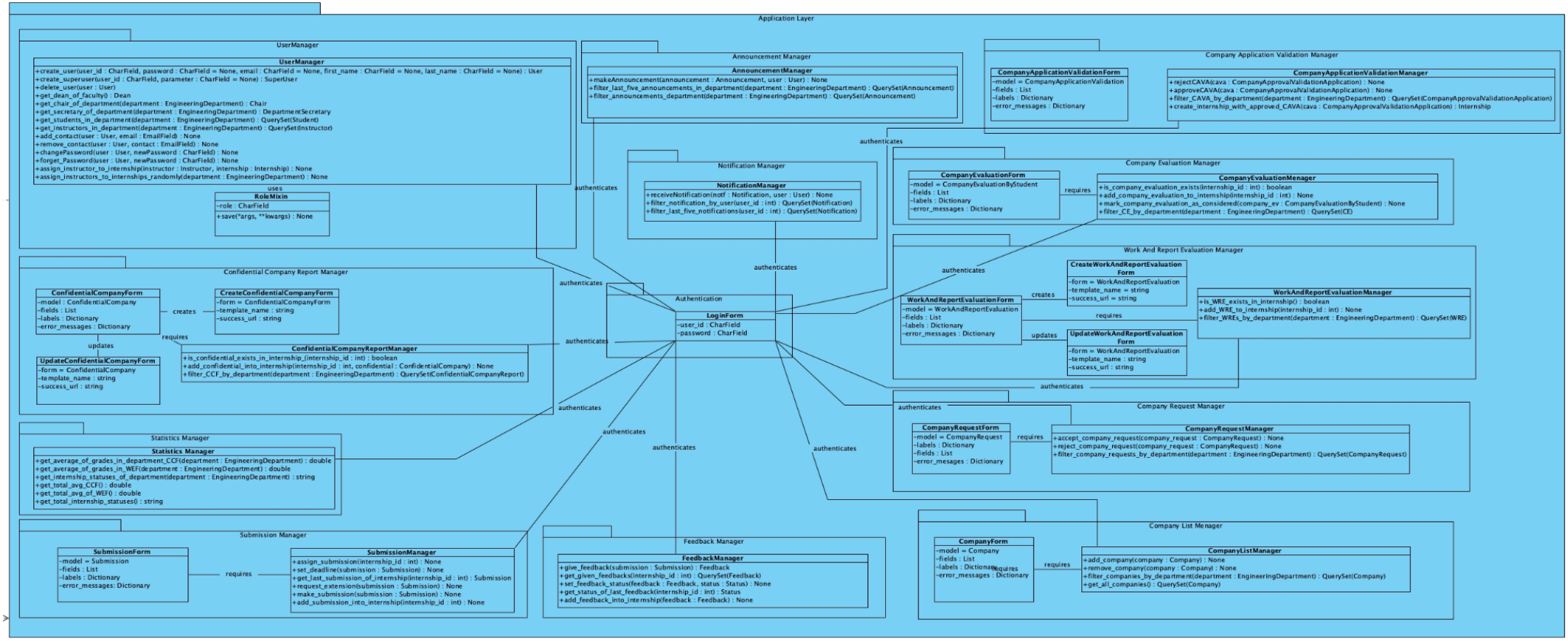

Figure-11: *Application Layer of InternHub*

Recall that in the subsystem decomposition diagram, the application layer was divided into the managers (2.1.3). The above diagram illustrates the classes involved in these managers with the functions provided. Authentication serves as the base, that is because, without valid authorization, none of the above managers will be accessible. For the higher resolution please refer to the same link in 3.2.

3.4.2 Data Layer

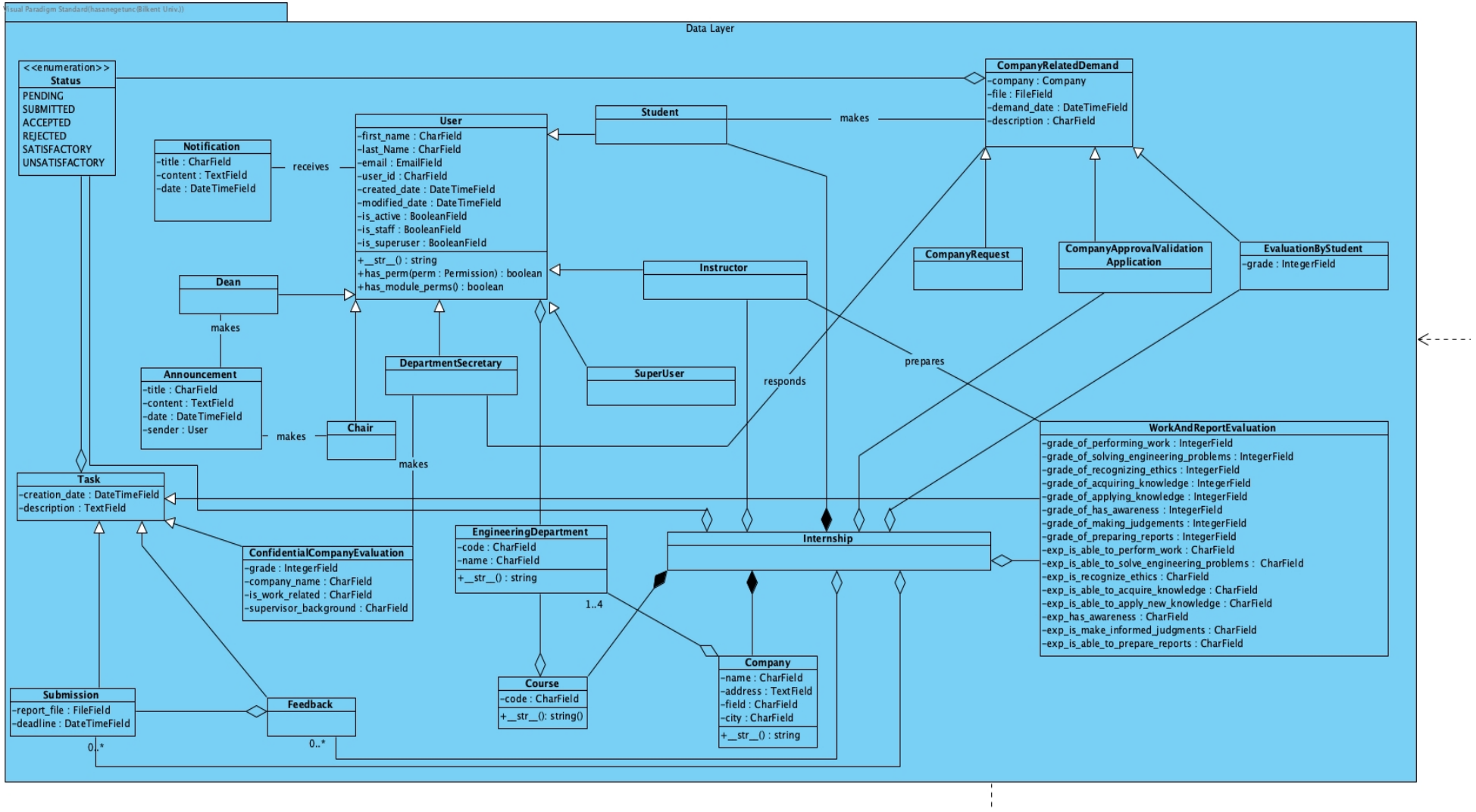

Figure-12: *Data Layer of InternHub*

The above diagram illustrates the data layer of InternHub, this layer's class diagram consists of entity classes. Django's model class data types are used to denote the variable types. For the higher resolution please refer to the same link in 3.2.

3.4.3 Presentation UI Layer

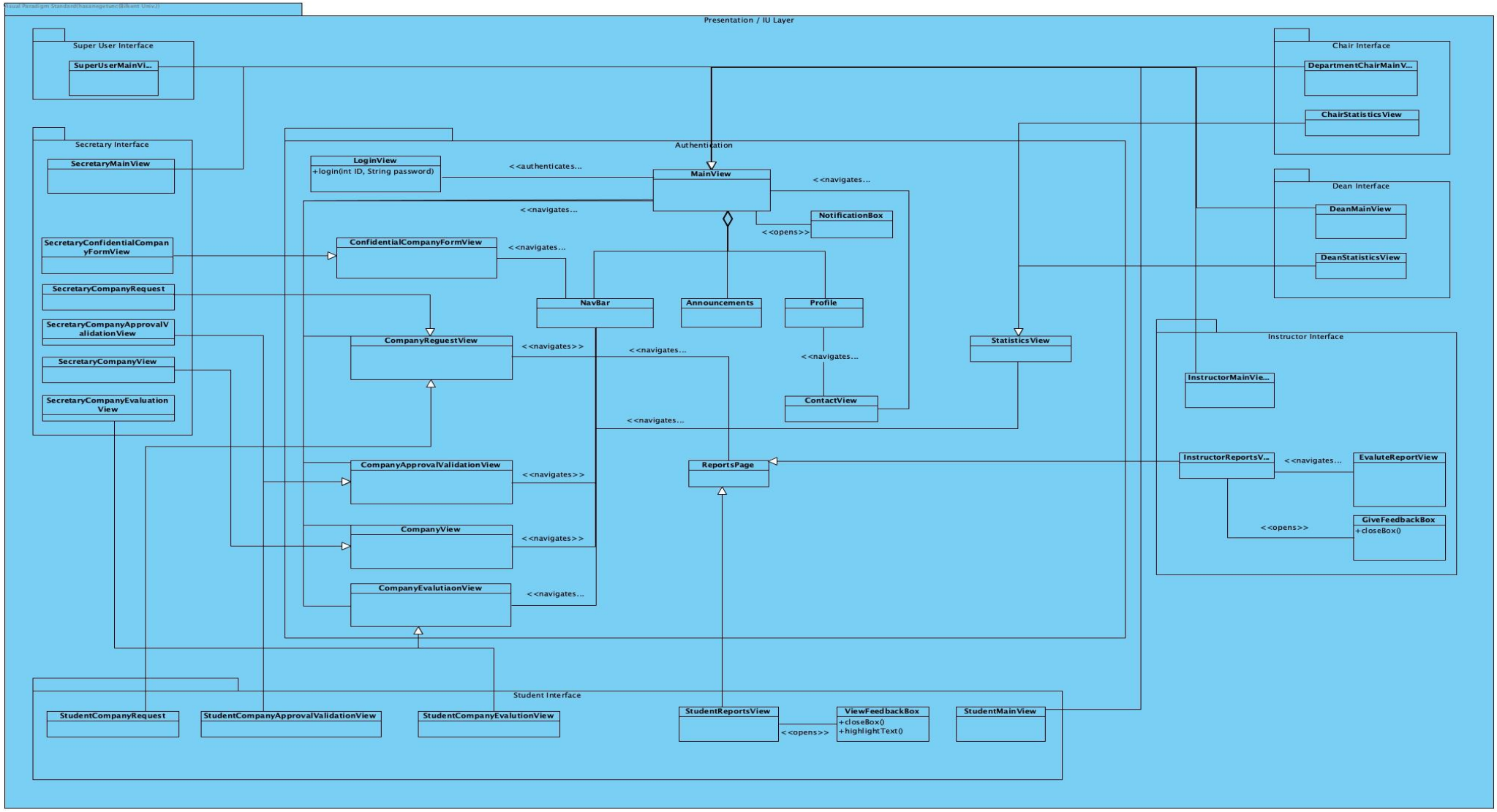

Figure-13: *Presentation UI Layer of InternHub*

The above diagram illustrates the presentation layer of InternHub, note that these are only view operations and implemented through Django class based views. Backend operations are carried out by application layer, so those operations are not be shown in this diagram. Recall that UI Layer has interfaces for each user and see how they are designed. For the higher resolution please refer to the link in 3.2.

3.5 Design Patterns

MVC Pattern: In InternHub developed by Python Django, the architectural pattern used is the Model-View-Controller (MVC) pattern, with Django's own interpretation called the Model-View-Template (MVT) pattern. First of all, the model represents the data structure and handles the interactions with the database. It defines the database schema, encapsulates data access logic, and provides an abstraction layer to interact with the data. It keeps the attributes of classes related to all forms in InternHub. Secondly, the view is responsible for handling requests, processing data, and returning responses. In Django, views are Python functions or classes that receive HTTP requests, interact with models or other data sources, perform business logic, and return HTTP responses. In view, it is decided which features will work and which pages will open.Lastly, the template is responsible for the presentation layer and generating dynamic HTML content. Templates in Django are written in a specific syntax that allows you to include dynamic data, control structures, and template tags. Templates receive data from views and render the final HTML output that is sent back to the user's browser.

Even though the MVT pattern used in Django is comparable to the conventional MVC pattern, there are several implementation-specific differences. The primary goal of both design patterns is to clearly distinguish between data management, business logic, and display in a web application.

Singleton pattern: Singleton pattern is a creational design pattern that guarantees the existence of only one instance of a particular class. An example of this pattern in the InternHub is the BCryptPasswordHasher. It is implemented as a static class with a single instance and is responsible for encrypting passwords during the sign-up process. By using the Singleton pattern, the system ensures that there is only one BCryptPasswordHasher instance, promoting consistency and efficient resource utilization.

Observer Pattern: Observer pattern is indeed a behavioral design pattern that establishes a subscription mechanism to notify multiple objects (observers) about events that occur in the object they are observing (subject). An example of this pattern in the InternHub is the announcements. In this web application, when the users ,who can make announcements such as Department Secretary and Dean, make announcements, all users should be notified about announcements.

Façade Pattern: This is a structural design pattern that provides a simplified interface to a complex system, making it easier to use and understand. The purpose of the Façade pattern is to provide a unified and simplified interface to a set of interfaces and subsystems within a software application. It acts as a front-facing interface that hides the complexities of the underlying subsystems and provides a high-level interface for the clients. This design pattern is benefitted in database manager layer as all of the other layers communicates with database manager, providing them to hide the complexities of database operations.

4.0 Improvement Summary

In the second iteration, regarding the introduction, the purpose of the system was elaborated for each user and the discussion of the other design goals such as performance and functionality was discarded to proceed directly to the top design goals.

For subsystem decomposition, the database manager layer was added, the presentation and the application layer was detailed (i.e. subsystem are included see 2.1) and the entity classes were placed into the data layer. Some additions made into the deployment diagram and some notation fixes were carried out. Some small changes made in persistent data management section to reason our choice. Object access matrix was updated regarding the addition of manager classes and required functionalities. For boundary conditions, each question in the checklist was elaborated by mentioning the boundary condition and how to verify the specified condition.

In low level design section of the second iteration, the final object design is made from zero. To clarify, considering the layered structure of the system, the new object design is decided by decomposing the design into layers. Data layer design is made from scratch, the manager classes were elaborated in terms of methods, their return types and parameters. Finally, for UI layer, Django's class based views were taken essence and the design is made accordingly. By putting all of these layered designs together, the final object design is obtained. Two more external packages were added, and internal packages presented in detail.

5.0 Glossary

AWS: Amazon Web Services

CAVA: Company Approval Validation Application

CCF: Confidential Company Form

CE: Company Evaluation (by Student)

CSS: Cascading Style Sheets

ES5: ECMAScript 5

EC2: Elastic Compute Cloud

GNU: A free and open-source operating system and software ecosystem developed by the GNU Project.

HTML: Hypertext Markup Language

RAM: Random Access Memory

SSD: Solid-State Drive

vCPU: Virtual Central Processing Unit

WRE: Work and Report Evaluation

6.0 References

[1] “Django,” Django Project. [Online]. Available: https://docs.djangoproject.com/en/3.2/. [Accessed: 17-May-2023].

[2] React. [Online]. Available: https://reactjs.org/. [Accessed: 18-May-2023].

[3] SQLite documentation. [Online]. Available: https://www.sqlite.org/docs.html. [Accessed: 18-May-2023].

[4] “Storages,” django. [Online]. Available: https://django-storages.readthedocs.io/en/latest/ . [Accessed: 18-May-2023].

[5] “Promise based HTTP client for the browser and node.js,” Axios. [Online]. Available: https://axios-http.com/. [Accessed: 18-May-2023].

[6] “Home V6.11.1,” Home v6.11.1. [Online]. Available: https://reactrouter.com/. [Accessed: 19-May-2023].

[7] “Cloud computing services - amazon web services (AWS).” [Online]. Available: https://aws.amazon.com/. [Accessed: 20-May-2023].

[8] *Django BCrpytPasswordHasher*. (n.d.-b). Django Project: https://docs.djangoproject.com/en/4.2/topics/auth/passwords/. [Accessed: 21-May-2023].