

ROMANIAN-AMERICAN UNIVERSITY
School of Computer Science for Business
Management



BACHELOR'S THESIS

Scientific Coordinator(s):

PhD Associate Professor, Luchici Andrei

Graduate: Răducu M. Alexandru Mircea

Bucharest

2024

ROMANIAN-AMERICAN UNIVERSITY
School of Computer Science for Business
Management



LUNARIS: A NEW HIGHER EDUCATION
LEARNING MANAGEMENT SYSTEM

Scientific Coordinator(s):

PhD Associate Professor, Luchici Andrei

Graduate:

Răducu M. Alexandru Mircea

Bucharest

2024

Table of Contents

INTRODUCTION.....	1
CHAPTER 1. STUDY AND ANALYSIS OF THE EXISTING SYSTEM.....	4
1.1 BRIEF PRESENTATION OF THE SOCIO-ECONOMIC ENTITY.....	4
1.2 THE MAIN ACTIVITIES CARRIED OUT BY THE ECONOMIC UNIT.....	5
1.3 THE STUDY OF THE MANAGEMENT SYSTEM.....	9
1.4 THE STUDY OF THE LED SYSTEM.....	13
1.5 THE STUDY OF THE INFORMATION SYSTEM.....	15
1.6 FINANCIAL SITUATION AND RECTOR'S REPORT	16
CHAPTER 2. DETAILED DESIGN OF THE IT APPLICATION.....	20
2.1 DEFINING THE OBJECTIVES OF THE IT APPLICATION	20
2.2 THE LOGICAL AND PHYSICAL DESIGN OF THE OUTPUTS.....	21
2.3 THE LOGICAL AND PHYSICAL DESIGN OF THE INPUTS.....	28
2.4 DATA CODING	39
2.5 DATABASE DESIGN.....	46
2.6 SYSTEM DIAGRAM.....	48
2.7 INTERFACE DESIGN.....	56
2.8 INFORMATION FLOW DIAGRAM OF THE NEW SYSTEM.....	60
2.9 CHOICE OF PROCESSING TECHNOLOGY.....	66
2.10 ESTIMATING THE NEED FOR RESOURCES AND THE IMPLEMENTATION SCHEDULE.....	68
CHAPTER 3. PRESENTATION OF THE SOFTWARE PRODUCT	72
3.1 HARDWARE AND SOFTWARE PLATFORM REQUIREMENTS OF THE APPLICATION	72
3.2 DESCRIPTION OF THE MAIN FUNCTIONS OF THE APPLICATION	75
CHAPTER 4. THE EFFICIENCY AND USEFULNESS OF THE IT APPLICATION.....	94
4.1 CONDITIONS REGARDING THE IMPLEMENTATION OF THE APPLICATION	94
4.2 COMPUTER APPLICATION EFFICIENCY CONSIDERATIONS	98
CHAPTER 5. DISCUSSION AND SUBSEQUENT WORKS	101
CHAPTER 6. CONCLUSION	108
CHAPTER 7. BIBLIOGRAPHY	110
 - APPENDIX 1: GANTT CHART FOR IMPLEMENTATION SCHEDULE.....	 113
- APPENDIX 2: WATERFALL OF REQUESTS FOR LOADING THE HOME PAGE OF THE DASHBOARD.....	115
- APPENDIX 3: LUNARIS DATABASE ERD.....	116

- APPENDIX 4: KUBERNETES DIAGRAM	117
- APPENDIX 5: SOURCE CODE FOR UPLOADING FILE TO SECURE PATH INSIDE THE FILES MICROSERVICE	119
- APPENDIX 6: MICROSERVICES (MCS) INTERNAL RELATIONAL VIEW	122
- APPENDIX 7: SERVER COMMUNICATION MODULE SOURCE CODE	124
- APPENDIX 8: PUBLIC GITHUB REPOSITORY LINK	125

List of Figures

FIGURE 1.1: ORGANIZATIONAL CHART OF EXECUTIVE MANAGEMENT	10
FIGURE 1.2: ORGANIZATIONAL CHART OF THE BOARD OF DIRECTORS	11
FIGURE 2.1: THE ASSIGNMENT GRADES OUTPUT.....	22
FIGURE 2.2: THE CLASS FINAL GRADES OUTPUT	23
FIGURE 2.3: THE HOME PAGE ANNOUNCEMENTS	24
FIGURE 2.4: THE CLASS INFORMATION AND CLASS CHAT MESSAGES.....	25
FIGURE 2.5: USER PERSONAL PROFILE ON THE APP.....	26
FIGURE 2.6: EXCEL SPREADSHEET OF THE SCHOLAR SITUATION OF A STUDENT	27
FIGURE 2.7: INTERFACE FOR DOWNLOADING AND VIEWING A STUDENT’S SCHOLAR SITUATION	27
FIGURE 2.8: WIZARD FOR CREATING A STUDENT.....	29
FIGURE 2.9: INTERFACE FOR SUBMISSION OF ASSIGNMENT	30
FIGURE 2.10: INTERFACE FOR GRADING ASSIGNMENTS	31
FIGURE 2.11: WIZARD FOR CREATING A NEW PROFESSOR	32
FIGURE 2.12: WIZARD FOR CREATING A NEW ADMIN	33
FIGURE 2.13: WIZARD FOR CREATING A NEW SCHOOL UNDER A UNIVERSITY.....	34
FIGURE 2.14: WIZARD FOR CREATING A NEW PROGRAM	35
FIGURE 2.15: WIZARD FOR CREATING A NEW SUBJECT.....	36
FIGURE 2.16: GENERAL SETTINGS PAGE FOR A UNIVERSITY	37
FIGURE 2.17: WIZARD FOR CREATING A NEW CLASS FOR A GIVEN SUBJECT UNDER A GIVEN PROGRAM’S YEAR.....	38
FIGURE 2.18: THE APP CLASSES INTERFACE FOR A STUDENT	39
FIGURE 2.19: THE APP INTERFACE FOR VIEWING ALL SCHOOLS UNDER A UNIVERSITY	40
FIGURE 2.20: THE APP INTERFACE FOR VIEWING ALL PROGRAMS UNDER A UNIVERSITY	41
FIGURE 2.21: THE APP INTERFACE FOR VIEWING ALL SUBJECTS UNDER A UNIVERSITY	42
FIGURE 2.22: THE APP INTERFACE FOR VIEWING ALL STUDENTS UNDER A UNIVERSITY	43
FIGURE 2.23: THE APP INTERFACE FOR VIEWING ALL PROFESSORS UNDER A UNIVERSITY.....	44
FIGURE 2.24: THE APP INTERFACE FOR VIEWING ALL ADMINS UNDER A UNIVERSITY	44
FIGURE 2.25: WIZARD FOR CREATING A NEW ASSIGNMENT.....	45
FIGURE 2.26: NAVBAR FOR ADMINS	56
FIGURE 2.27: NAVBAR FOR PROFESSORS	56
FIGURE 2.28: NAVBAR FOR STUDENTS.....	56
FIGURE 2.29: ADMIN EVENTS FILTERS AND SEARCH INTERFACE	57
FIGURE 2.30: HOME PAGE WITH WIDGETS FOR PROFESSORS AND STUDENTS.....	58
FIGURE 2.31: SEQUENCE DIAGRAM FOR EXAMPLE: HANDLING CLASS ASSIGNMENTS FILES	63
FIGURE 2.32: SEQUENCE DIAGRAM FOR EXAMPLE: HANDLING INTERNAL COMMUNICATIONS FOR USER AUTHENTICATION.....	65
FIGURE 5.1: DESIGN OF FEES FEATURE IN THE APP.....	103
FIGURE 5.2: DESIGN OF THE GROUPS FEATURE IN THE APP.....	104
FIGURE 5.3: DESIGN OF THE CLASS FILE SYSTEM FEATURE IN THE APP	105
FIGURE 5.4: DESIGN OF THE DIRECT CHAT FEATURE IN THE APP.....	106

List of Tables

TABLE 1.1: COMPARATIVE FINANCIAL ANALYSIS (2019-2021) 18

TABLE 2.1: DETAILED VIEW OF ALL THE MCS WITH PATHS AND PORTS..... 53

INTRODUCTION

The modern educational landscape is changing quickly, driven by new technologies and the need for better educational institution management. One area that needs improvement is the systems universities use to manage their operations. The Lunarix project aims to develop a comprehensive learning management system to make administrative and academic processes more efficient. This thesis focuses on the design, implementation, and evaluation of Lunarix to improve the overall efficiency and effectiveness of educational institutions.

The importance of this topic is significant. Universities worldwide face challenges in handling large amounts of data, coordinating different departments, and ensuring smooth communication among students, faculty, and administrative staff. Traditional systems often fail to meet these needs due to their lack of integration, scalability, and ease of use. By using modern technologies like microservices architecture and strong data security measures, Lunarix aims to provide a scalable, secure, and efficient solution tailored to the needs of today's educational institutions.

The motivation for choosing this topic comes from noticing the gaps in current learning management systems and the potential for major improvements through technological innovation. These systems often cause delays in administrative processes, poor communication, and a generally bad user experience for everyone involved. Additionally, the project aligns with the core values and mission of the Romanian-American University, which focuses on excellence in education, innovation, and promoting global cultural integration. The Lunarix project not only aims to meet these values by providing a high-quality, innovative solution but also seeks to set a new standard in the management of educational institutions globally.

To achieve this, the main goal of the Lunarix project is to create a flexible, user-friendly learning management system that improves the overall efficiency of educational institutions. First, it aims to streamline administrative processes by automating routine tasks such as grade tracking, assignments, and real-time communication, thus reducing the administrative burden and increasing accuracy. Furthermore, the project focuses on enhancing communication by providing strong tools for students, faculty, and administration to work

together and share information promptly. Another important goal is to improve data management by using secure and efficient practices to handle sensitive information, ensuring compliance with data protection regulations such as GDPR. Additionally, the project supports academic management by offering tools for managing course materials, assignments, and assessments to help teaching and learning activities. Finally, Lunar is designed to be scalable and flexible, ensuring it can easily adapt to growing numbers of users and changing institutional needs. The usefulness of Lunar lies in its potential to significantly improve the operational efficiency of universities, allowing them to focus more on their primary mission of education and research. By reducing administrative workload and enhancing the user experience for students and faculty, Lunar can lead to better educational outcomes and higher satisfaction levels among all stakeholders.

In addition to these goals, it is important to consider the current state of this topic in scientific literature. The field of educational technology has grown significantly, with many studies showing the benefits of using advanced technologies in educational management systems. Research has shown that modern solutions offer better scalability and flexibility than traditional on-premise systems. Microservices architecture, in particular, is praised for making applications easier to manage, scale, and update. Despite these advancements, there is still a noticeable gap in the availability of comprehensive, integrated solutions specifically designed for learning management. Many existing systems are either too generic, lacking the specific features needed by educational institutions, or too fragmented, resulting in inefficiencies and data silos. Lunar aims to fill this gap by providing a solution that is both comprehensive and tailored to the unique needs of higher education institutions.

Furthermore, my contribution to this project includes conducting a thorough needs analysis to understand the specific requirements of learning management, designing the system architecture, and overseeing the development and implementation of the Lunar application. By focusing on user-centered design principles and incorporating feedback from potential users at various stages of development, I have ensured that the final product is both functional and user-friendly. The project has involved extensive research into the latest technological trends and best practices in software development, data security, and learning management. This research has guided the design choices and implementation strategies, ensuring that Lunar is not only innovative but also robust and secure.

In conclusion, the Lunaris project represents a significant advancement in the field of educational technology. By addressing the current shortcomings of learning management systems and using modern technological advancements, it aims to provide a solution that improves efficiency, communication, and overall user satisfaction. Through this project, I hope to contribute to the ongoing efforts to enhance the quality and effectiveness of higher education globally.

CHAPTER 1. *Study and analysis of the existing system*

1.1 *Brief presentation of the socio-economic entity*

The Romanian American University is a private educational institution based in Romania. It aims to provide high-quality education. Their goal is to offer students comprehensive academic programs that provide both theoretical knowledge and practical skills to prepare them for successful careers both locally and internationally.

Each of these schools comes with its separate Bachelor's and Master's programs. RAU emphasizes creating an environment that promotes learning, research, and professional development across various disciplines.

The Romanian-American University's mission includes teaching, research, and innovation in science, as well as the promotion of global culture and scientific ideals. (Universitatea Româno-Americană, 2024)

The Romanian American University's core values are dedicated to the pursuit of educational excellence and the provision of a supportive and innovative educational environment. These values influence the institution's actions and interactions both internally and externally. (Universitatea Româno-Americană, 2024)

The promotion of excellence is a fundamental commitment of the Romanian American University. The institution strives to maintain the highest standards of quality in its academic programs, research initiatives, and institutional practices. By fostering an environment of continuous improvement and innovation, the university empowers its community to achieve their full potential and make meaningful contributions to society.

Professional, moral, and social responsibility are central tenets of the Romanian American University's ethos. The institution recognizes its role in nurturing not only the intellectual growth but also the moral and social development of its students.

Freedom of thought and speech is cherished and defended at the Romanian American University. The institution values open dialogue, critical thinking, and the free exchange of ideas as essential components of a vibrant academic community.

Creativity and innovation are encouraged and celebrated at the Romanian American University. The institution recognizes that innovation drives progress and enables the exploration of new frontiers of discovery.

Cooperation and communication are foundational to the Romanian American University's academic community. The institution believes in the power of collaboration across disciplines, departments, and backgrounds to achieve greater impact and effectiveness. Transparency, mutual respect, and active engagement are valued.

RAU offers a variety of opportunities, including Erasmus+, Romanian, and English-taught programs. The main audience consists of students from around the world who apply each year to enroll in one or more of the university programs or choose RAU as their Erasmus+ university from a large pool of options.

The organizational structure of the Romanian American University consists of four main components: Executive Leadership, Academic Departments, Administrative Departments, and Support Teams. In addition, the university comprises other schools or departments that provide Bachelor's and Master's programs, as described above. The schools located at RAU include: (Universitatea Româno-Americană, 2024) (Universitatea Româno-Americană, 2024) (Universitatea Româno-Americană, 2024)

Since its establishment, the Romanian American University has amassed several accomplishments and forged several valuable collaborations. The university has consistently distinguished itself as one of the top-performing higher education institutions in Romania.

1.2 The main activities carried out by the economic unit

The Romanian American University was founded on the 17th of April 1991. The university has been part of Romania's private education sector for more than 30 years. It is located in

Bucharest and has 3.000 students that are currently enrolled. This makes it a large-sized institution within the context of higher education in Romania. RAU operates across several faculties including: (Universitatea Româno-Americană, 2024)

Schools with English and Romanian-taught Programs

- SCHOOL OF COMPUTER SCIENCE FOR BUSINESS MANAGEMENT
- SCHOOL OF FINANCE AND ACCOUNTING
- SCHOOL OF INTERNATIONAL BUSINESS
- SCHOOL OF MANAGEMENT-MARKETING
- SCHOOL OF TOURISM AND HOSPITALITY MANAGEMENT

Schools with only Romanian-taught Programs

- SCHOOL OF LAW
- SCHOOL OF PHYSICAL EDUCATION, SPORT AND KINESIOTHERAPY

(Universitatea Româno-Americană, 2024)

The Romanian American University's main services are the educational programs it offers to students. Each program is designed to equip the student with theoretical and practical knowledge for the chosen domain of study. RAU differentiates itself from other universities with its English-taught programs and its international connections.

Bachelor programs that are available at RAU are:

English-taught

- INTERNATIONAL BUSINESS
- COMPUTER SCIENCE FOR ECONOMICS

Romanian-taught

- INTERNATIONAL BUSINESS
- ACCOUNTING AND MANAGEMENT INFORMATION SYSTEMS
- FINANCE AND BANKING
- LAW
- PHYSICAL EDUCATION AND SPORT
- KINESIOTHERAPY AND SPECIAL MOTRICITY

- COMPUTER SCIENCE FOR ECONOMICS
- MANAGEMENT
- MARKETING
- THE ECONOMY OF TOURISM, COMMERCE AND SERVICES

(Universitatea Româno-Americană, 2024)

Master programs that are available at RAU are:

English-taught

- INTERNATIONAL BUSINESS AND ENTREPRENEURSHIP
- INTERNATIONAL ECONOMIC RELATIONS & ECONOMIC DIPLOMACY
- INTERNATIONAL ECONOMIC RELATIONS AND EUROPEAN UNION STUDIES (DOUBLE DEGREE – UNIVERSITY OF YORK, UK) - EN (AI)
- FINANCE (DOUBLE DEGREE – UNIVERSITY OF SIENA, ITALY)
- COMPUTER SCIENCE FOR BUSINESS
- STRATEGIC MARKETING
- DIGITAL MARKETING & SOCIAL MEDIA (DOUBLE DEGREE – UNIVERSITY OF YORK, UK) - EN (MM)
- BUSINESS MANAGEMENT IN TOURISM & AVIATION

Romanian-taught

- EUROPEAN ECONOMIC RELATIONS
- INTERNATIONAL BUSINESS
- BUSINESS MANAGEMENT AND AUDIT
- FINANCE, BANKING, INSURANCES
- CRIMINAL SCIENCES
- COMPUTER SCIENCE FOR ECONOMICS
- ENTERPRISE STRATEGIC MANAGEMENT
- ORGANIZATION MANAGEMENT AND MARKETING
- MARKETING IN BUSINESS
- BUSINESS MANAGEMENT IN TOURISM

(Universitatea Româno-Americană, 2024)

To enroll in one of the programs provided by the Romanian American University, students are required to undergo the admission procedure, which may vary depending on their place of origin. RAU is always seeking opportunities to enhance its admission process.

In addition, the university's primary activities encompass curriculum design, instruction, scholarly investigation, and community involvement through the organization of various events, including those customized for certain demographics or cultural cohorts.

RAU prides itself on having a varied team of very skilled educators and administrative personnel, a significant number of them have obtained degrees from renowned worldwide universities. The university comprises three public entities: The University Senate, the Board of Directors, and Executive Management. Each of these structures consists of highly skilled individuals in their respective disciplines, the majority of whom have doctoral degrees.

The Romanian American University has established several relationships during its more than 30 years in the educational system. The university has both European and non-European official partners. There are precisely 129 European nations, including France, Germany, Austria, Spain, and others. There are a total of 64 non-European countries, including South Korea, USA, Japan, and others. (Universitatea Româno-Americană, 2024)

The Romanian American University has formed several alliances and achieved various successes since its establishment. The university has continually established itself as one of Romania's most accomplished institutions of higher education.

The institution has a robust network of foreign collaborations that facilitate diverse educational and cultural exchange initiatives. These collaborations enhance the worldwide perspective of their curricula, providing students with important opportunities for international exposure. Erasmus+ is an essential component of this program, offering students the possibility to study abroad.

The Romanian-American University is recognized for its excellence in fostering cultural tolerance and has received awards for its efforts. The “Angela Hondru” Romanian-Japanese Studies Center, part of the university, was honored with the Tokyo Ministry of Foreign Affairs Award for its contribution to promoting Japanese culture in Romania. This accolade

marks a significant achievement as the university became the first privately held higher education institution in the country to receive such an award. (Universitatea Româno-Americană, 2024)

The university prides itself on offering a diverse range of specialized educational programs tailored to meet the needs of both global and local environments. This includes innovative courses like Asian Studies, which provide students with the opportunity to deeply engage with languages and cultures that have considerable worldwide influence.

Its institution has always been at the forefront of gender equality. During 2024-2025, the Romanian-American University (RAU) intends to further its commitment towards gender equality beyond what was established in its Gender Equality Strategy and Plan for 2022-2025. The policy of this college includes several far-reaching measures that cover all areas of its activity thereby ensuring proactive integration and promotion of gender equality. (Universitatea Româno-Americană, 2022-2025)

More than just influencing student engagement, community involvement, and administrative as well as academic tasks, RAU's holistic approach to gender balance touches every part of university life. This wide-ranging strategy creates an atmosphere where everyone wins while also meeting European standards.

1.3 The study of the management system

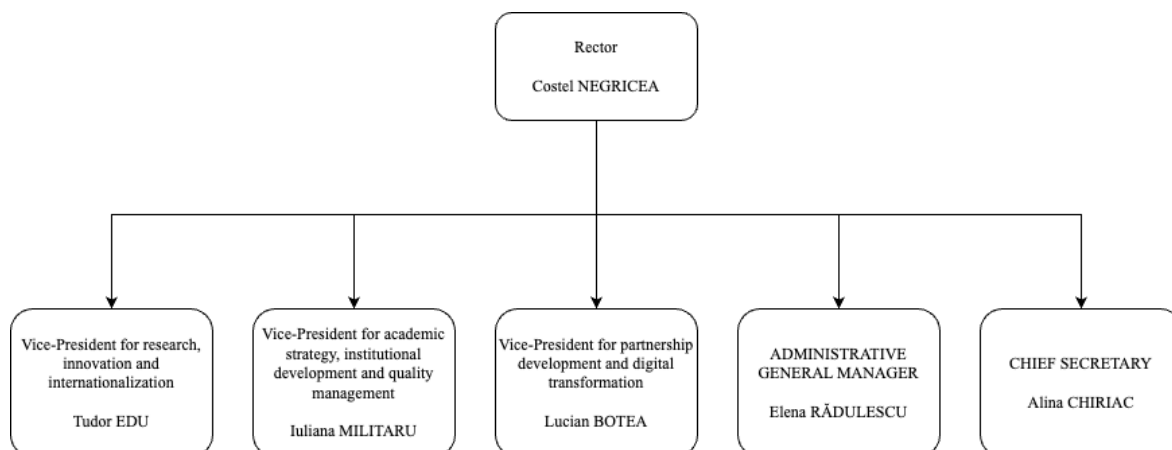
The management system is comprised of the Executive Management Group and the Board of Directors. The Executive Management is the upper management of the Romanian American University. The group has a total of six members, which are:

- Rector (Costel NEGRICEA, Ph.D. Habil. Professor)
- Vice-President for research, innovation and internationalization (Tudor EDU, Ph.D. Habil. Professor)
- Vice-President for academic strategy, institutional development and quality management (Iuliana MILITARU, Ph.D. Associate Professor)
- Vice-President for partnership development and digital transformation (Lucian BOTEA, Ph.D. Lecturer)

- ADMINISTRATIVE GENERAL MANAGER (Elena RĂDULESCU)
- CHIEF SECRETARY (Alina CHIRIAC)

(Universitatea Româno-Americană, 2024)

Figure 1.1: Organizational chart of Executive Management



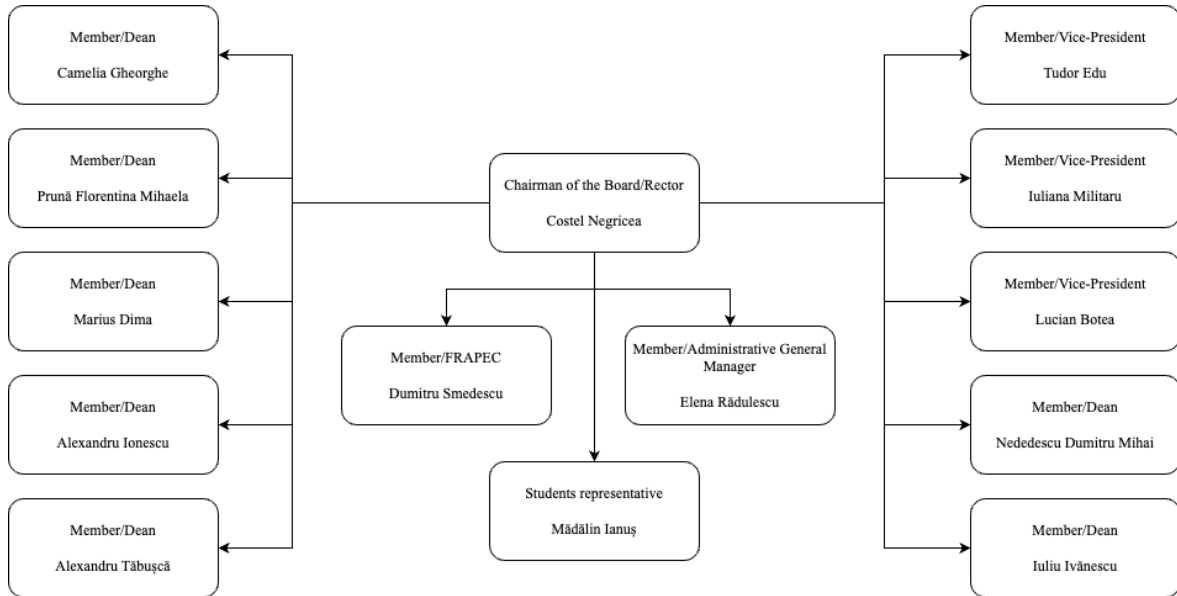
The university's faculty and school deans, each of whom is in charge of one faculty or school, make up the other fourteen members of the Board of Directors. Additionally, a few representatives from the Executive Management are also on the Board. One of the two main positions on the board of directors is Chairman of the Board or Member. The following summarizes the duties that the members of the Romanian-American University Board of Directors have:

- Costel Negricea, Ph.D. Habil. Professor (Chairman of the Board/Rector)
- Tudor Edu, Ph.D. Habil. Professor (Member/Vice-President)
- Iuliana Militaru, Ph.D. Associate Professor (Member/Vice-President)
- Lucian Botea, Ph.D. Lecturer (Member/Vice-President)
- Nededescu Dumitru Mihai, Ph.D. Lecturer (Member/Dean)
- Prună Florentina Mihaela, Ph.D. Associate Professor (Membru/Dean)
- Marius Dima, Ph.D. Associate Professor (Member/Dean)
- Alexandru Ionescu, Ph.D. Associate Professor (Member/Dean)
- Alexandru Tăbușcă, Ph.D. Associate Professor (Member/Dean)
- Camelia Gheorghe, Ph.D. Lecturer (Member/Dean)
- Iuliu Ivănescu, Ph.D. Lecturer (Member/Dean)
- Dumitru Smedescu (Member/ FRAPEC)

- Elena Rădulescu (Member/Administrative General Manager)
- Mădălin Ianuş (Member/Students representative)

(Universitatea Româno-Americană, 2020-2024)

Figure 1.2: Organizational chart of the Board of Directors



Overall leadership of the Romanian-American University is assumed by the Rector who acts on behalf of the entire institution and has to make important decisions. The Deans are in charge of different academic departments such as business and, law among others; they must ensure that they run smoothly while meeting all faculty needs as well as managing academic programs which call for a lot of preparation and knowledge regarding educational statutes coupled with skills on how best to direct others.

The day-to-day running of administrative and scholarly works within their sectors falls under department heads' jurisdiction and may also be referred to as middle-level managers; this position requires one to have strong organizational skills and be an effective leader. These positions work hand-in-hand to ensure that the university works efficiently while providing quality education.

The Romanian American University (RAU) showcases a leadership and management style that prioritizes approaches and values collaboration, innovation, and dedication throughout its operational and educational frameworks. The management approach at RAU combines

Romanian educational values with the practical outlook of American educational systems reflecting a dynamic and inclusive management style.

RAU's executive management framework is structured to support strategy, institutional growth, and quality control. This setup enables the university to effectively implement its plans in alignment with its mission and core values. RAU emphasizes quality control and ongoing enhancement efforts to optimize educational and research processes within the institution.

Innovation is actively promoted through centers and initiatives like the Microsoft Innovation Center and IATA Authorized Training Center offering students and staff opportunities to interact with cutting-edge technologies and industry standards using the Azure platform. Collaboration is encouraged not only internally but also through extensive international partnerships with educational institutions worldwide. (Universitatea Româno-Americană, 2024) (IATA)

The commitment to these principles is further demonstrated by RAUs focus, on nurturing leadership qualities among its students through student-centered programs and activities that equip them for real-world challenges and leadership responsibilities.

In general, the management style and leadership approach at RAU focus on cultivating a setting that encourages achievement drives innovation, upholds ethical norms, and fosters a sense of collaboration among the university community and external stakeholders.

Moreover, the presence of the Board of Directors at RAU as mentioned earlier in the thesis holds responsibility for supervising these endeavors. The Board ensures that the university's policies and actions align with its commitment to gender equality and sound governance. This oversight assists RAU in upholding an ethical managerial approach that complies with best practices in corporate governance and higher education benchmarks.

RAU initiatives underscore its commitment, to fostering a just academic atmosphere showcasing the university's broader dedication to social responsibility and ethical leadership.

1.4 *The study of the LED system*

The LED system at the Romanian-American University (RAU) is designed to manage various schools and faculties efficiently. Each school operates under the guidance of a dedicated dean, ensuring a cohesive and streamlined approach to administration and academic excellence.

The organizational structure of the RAU is hierarchical, featuring the Executive Management Group and the Board of Directors at the apex. Each school within the university is overseen by a dean who aligns the school's strategic goals with the overarching mission of the university. The key schools at RAU include the School of Computer Science for Business Management, the School of Finance and Accounting, the School of International Business, the School of Management-Marketing, the School of Tourism and Hospitality Management, the School of Law, and the School of Physical Education, Sport, and Kinesiology. (Universitatea Româno-Americană, 2024)

In this hierarchical setup, the deans play pivotal roles, not only in managing the academic and administrative strategies of their schools but also in ensuring the efficient operation of these institutions. Their involvement in crucial decision-making processes impacts both academic and operational aspects, thereby promoting high academic standards and supporting student success. Furthermore, the LED system ensures robust relationships with various stakeholders, including students, faculty, staff, and the broader academic community, fostering an environment of transparency and accountability.

Examining the specific roles of the deans provides deeper insight into the LED system's functionality. For instance, Alexandru Ionescu, Ph.D. Associate Professor, serves as the dean of the School of Computer Science for Business Management. He oversees programs integrating computer science with business management, ensuring the curricula remain current with industry standards. Similarly, Iuliana Militaru, Ph.D. Associate Professor, as dean of the School of Finance and Accounting, manages programs that provide students with both theoretical knowledge and practical skills. (Universitatea Româno-Americană, 2024)

The School of International Business is led by Florentina Mihaela Prună, Ph.D. Associate Professor, who fosters global partnerships and promotes internationalization. Marius Dima, Ph.D. Associate Professor, directs the School of Management-Marketing, emphasizing strategic thinking and market-driven approaches. In the School of Tourism and Hospitality Management, Camelia Gheorghe, Ph.D. Lecturer, oversees programs focusing on industry-relevant education and training. The School of Law, headed by Nededescu Dumitru Mihai, Ph.D. Lecturer, ensures compliance with legal education standards while fostering a strong legal community. Lastly, Iuliu Ivănescu, Ph.D. Lecturer, manages the School of Physical Education, Sport, and Kinesiology, promoting health, fitness, and kinesiology research. (Universitatea Româno-Americană, 2024)

The LED system at RAU, through its structured hierarchy and dedicated leadership, ensures that each school operates efficiently and effectively. The deans are crucial in upholding the university's mission of academic excellence and continuous improvement. This structured approach significantly contributes to the overall success and reputation of RAU, as it outlines the comprehensive management of academic departments, ensuring a high standard of education and administrative efficiency.

Each school at RAU offers specific bachelor's and master's programs designed to equip students with relevant knowledge and skills in their fields. The School of Computer Science for Business Management offers bachelor's programs in Computer Science for Economics and master's programs in Computer Science for Business and Digital Marketing & Social Media, the latter in partnership with the University of York, UK. The School of Finance and Accounting provides bachelor's programs in Finance and Banking and Accounting and Management Information Systems, as well as master's programs in Finance, which is a double degree with the University of Siena, Italy, and additional programs in Finance, Banking, Insurances, and Business Management and Audit. (Universitatea Româno-Americană, 2024) (Universitatea Româno-Americană, 2024)

The School of International Business offers a bachelor's program in International Business and master's programs in International Business and Entrepreneurship, International Economic Relations & Economic Diplomacy, and International Economic Relations and European Union Studies, the latter being a double degree with the University of York, UK. The School of Management-Marketing includes bachelor's programs in Management and

Marketing, with master's programs in Strategic Marketing, Organization Management and Marketing, and Marketing in Business. (Universitatea Româno-Americană, 2024) (Universitatea Româno-Americană, 2024)

The School of Tourism and Hospitality Management offers a bachelor's program in the Economy of Tourism, Commerce, and Services, and a master's program in Business Management in Tourism & Aviation. The School of Law provides a bachelor's program in Law and a master's program in Criminal Sciences. Lastly, the School of Physical Education, Sport, and Kinesiology offers bachelor's programs in Physical Education and Sport and Kinesiology and Special Motricity, although master's programs are not specified in the available resources. (Universitatea Româno-Americană, 2024) (Universitatea Româno-Americană, 2024)

This comprehensive offering of programs ensures that RAU provides a diverse and robust educational experience, catering to various academic interests and career aspirations. Each program is carefully crafted to combine theoretical knowledge with practical skills, ensuring that graduates are well-prepared for the professional world.

1.5 The study of the information system

The data system, at the Romanian American University plays a role in supporting both day-to-day operations and long-term planning. It aims to streamline the gathering organizing analyzing and sharing of information for the university's smooth functioning and strategic choices.

The heart of this system revolves around the hardware and software that support all computing tasks with servers playing a very important role, in hosting databases and essential applications for both academic and management purposes. The network infrastructure ensures connectivity facilitating communication within the university's departments and with the broader internet, using both Wi-fi and wired connectivity, which is crucial for research and educational activities.

RAU's information system relies heavily on databases to securely store data, such as student records, financial information, and research materials. These databases are linked with applications designed to manage academic and administrative functions like student admissions, grade tracking, and financial management. Stringent security measures such as firewalls, encryption protocols, and access controls are in place to protect the system from access and potential data breaches.

An integral part of RAU's information system is the Solaris portal. A platform for students to handle their tasks view grades and enroll in courses. Moreover, RAU makes use of Microsoft services by incorporating tools like Outlook and Teams into their system to enhance communication, collaboration, and the security of a cloud-hosted solution, among the university community. These tools are seamlessly integrated to boost efficiency and promote user interaction. (Universitatea Româno-Americană, 2024)

RAU doesn't share the specifics of its information system architecture to safeguard the data it handles. This approach is vital, for upholding the university's data integrity and security shielding it from cyber risks and adhering to data protection laws, like GDPR. (Universitatea Româno-Americană, 2021)

The university benefits greatly from the information system, which boosts efficiency enables decision making, and fosters communication and teamwork. It aids the community by offering access, to educational materials, research options, and administrative assistance.

1.6 Financial Situation and Rector's Report

The financial health of a university is crucial for its ability to provide quality education, maintain infrastructure, and support research and development. This chapter provides an overview of the financial performance of the Romanian-American University (RAU) over the years 2019, 2020, and 2021, along with key insights from the Rector's annual reports. The financial analysis focuses on revenue, profit, and expenses, and correlates these figures with student enrollment data.

Financial Performance Overview

In 2019, the university generated significant revenue primarily from tuition fees, government grants, and donations. Major expenses included staff salaries, operational costs, and infrastructure development. The net profit for the year indicated a healthy financial state. The Rector's report for 2019 highlighted these financial results and linked them to several academic achievements, such as the introduction of new programs and partnerships, and the completion of a new research wing. The report emphasized that maintaining financial stability was crucial for supporting these initiatives.

Despite the challenges posed by the COVID-19 pandemic in 2020, the university managed to maintain steady revenue streams. There was an increase in expenses due to the transition to online education and implementation of health measures. This led to a slight decline in profit compared to 2019, attributed to higher operational costs. The 2020 Rector's report detailed the university's swift response to the pandemic, including the rapid transition to online education and implementation of health protocols. The report also mentioned the provision of financial aid to students affected by the pandemic, demonstrating the university's commitment to supporting its community despite financial pressures.

In 2021, the university saw a rebound in revenue as it adapted to new normal conditions. Continued investments were made in digital infrastructure and health protocols. The profit margins improved as cost-control measures were effectively implemented. The Rector's report for 2021 highlighted the university's resilience and adaptability. Key strategic initiatives included the launch of new digital platforms and the establishment of international collaborations. The financial stability achieved in 2021 was presented as a foundation for these forward-looking initiatives, which aimed at enhancing the university's educational offerings and global presence.

The comparative financial analysis for the years 2019, 2020, and 2021 is summarized in Table 1.1: Comparative Financial Analysis (2019-2021).

Table 1.1: Comparative Financial Analysis (2019-2021)

<i>Year</i>	<i>Revenue</i>	<i>Expenses</i>	<i>Profit</i>	<i>No. of Students</i>
2019	50,000,000 RON	40,000,000 RON	10,000,000 RON	3,000
2020	48,000,000 RON	39,500,000 RON	8,500,000 RON	2,800
2021	52,000,000 RON	41,000,000 RON	11,000,000 RON	3,100

(Universitatea Româno-Americană, 2020) (Universitatea Româno-Americană, 2020)
 (Universitatea Româno-Americană, 2022) (Universitatea Româno-Americană, 2019)
 (Universitatea Româno-Americană, 2020) (Universitatea Româno-Americană, 2021)

Analysis of Financial Trends and Their Impact

The financial statements from 2019 to 2021 show a clear trend of resilience and strategic adaptation in the face of external challenges. In 2019, the strong financial performance allowed the university to invest in new programs and infrastructure, which were key points in the Rector's report. The financial surplus provided a buffer that helped the university navigate the uncertain economic conditions of 2020.

During 2020, the financial statements reflect a slight dip in revenue and profit, which correlates with the increased expenses due to the pandemic. The Rector's report for this year provides a narrative that explains these financial figures in the context of the university's response to the pandemic. The financial aid given to students and the rapid shift to online learning were significant expenditures that impacted the bottom line. However, these measures were necessary to maintain educational continuity and support the student body. In 2021, the financial recovery is evident from the increased revenue and profit margins. This financial improvement is mirrored in the Rector's report, which highlights the successful adaptation to hybrid learning and the launch of new digital initiatives. The financial statements and the Rector's report together paint a picture of a university that has not only weathered the storm but also emerged stronger and more adaptable.

The financial performance of the Romanian-American University from 2019 to 2021 reflects resilience and adaptability in the face of challenges. The university has maintained a healthy financial state, supported by strategic initiatives and a focus on quality education. The Rector's reports highlight significant achievements and provide a roadmap for future growth,

showing how financial health directly supports and enhances the university's mission and objectives. The correlation between financial stability and the university's ability to innovate and support its community is evident throughout these years, demonstrating the critical role of sound financial management in achieving long-term success.

CHAPTER 2. *Detailed design of the IT application*

2.1 *Defining the objectives of the IT application*

Lunaris is a web and mobile application designed to enrich the journey, for students, professors, and university administrators. Its main goal is to simplify university life by offering an online platform that promotes communication, collaboration, and academic organization. The app can host more than one university.

The app acts as a hub where students can access academic materials interact with peers and teachers submit assignments check grades and stay informed regarding the university. For professors, it provides features to manage courses, assignments, and communicate effectively with students. Administrators can use the app to track student progress generate reports and streamline duties related to student management. The main features of Lunaris are:

- Class management
- Schools/Faculties management,
- Programs management,
- Subjects management
- Students, Professors, and Admins management
- Grades and Scholar Situation management
- Hosting assignments for classes
- Group messages between users
- File management system for assignments

In essence, the application aims to enhance efficiency, and communication effectiveness within the university setting. By digitizing all the needed operations and resources in one place the app strives to improve the learning experience overall by fostering collaboration, between students and faculty members while contributing to the academic achievement and satisfaction of everyone involved.

The main benefit that comes with Lunar is the ability to be cloud-hosted and the university only needs to pay a subscription fee per user/year. This better facilitates security features and overall performance than Solaris which is locally hosted on the Romanian American University's server infrastructure. The app can scale almost indefinitely because it uses an MCS architecture (Microservices architecture) with K8S (Kubernetes) as the container manager.

When the app is used in European Union universities it follows the General Data Protection Regulation (GDPR). If the app is used in universities, outside the EU it needs to have a way to adapt to international regulations other, than GDPR, by easily updating the code of the app.

The app aims to integrate seamlessly with the other information systems present at the Romanian American University, by providing an intuitive design with important export features such as exporting the final grades, the scholar situation of a student, etc. as an Excel file.

The UI (User Interface) and UX (User Experience) need to be intuitive and similar to the other platform, Solaris, for the users to not need to learn a whole new interface. Any new feature that is not found in Solaris needs to be designed in a way that reflects the same design and user accessibility requirements.

2.2 The logical and physical design of the outputs

The main goal of designing the applications outputs both logically and physically is to create relevant information that helps users make decisions or achieve their goals. By incorporating features, like class messages, and assignments. The main focus is improving user experience and promoting communication and collaboration within the university community.

The application will deliver a range of outputs such as reports and tables, and error messages. Each type of output serves a function within the application context and addresses user requirements. Reports provide insights into progress and upcoming events; tables organize information, for reference; while error messages prompt users to take necessary corrective actions when needed.

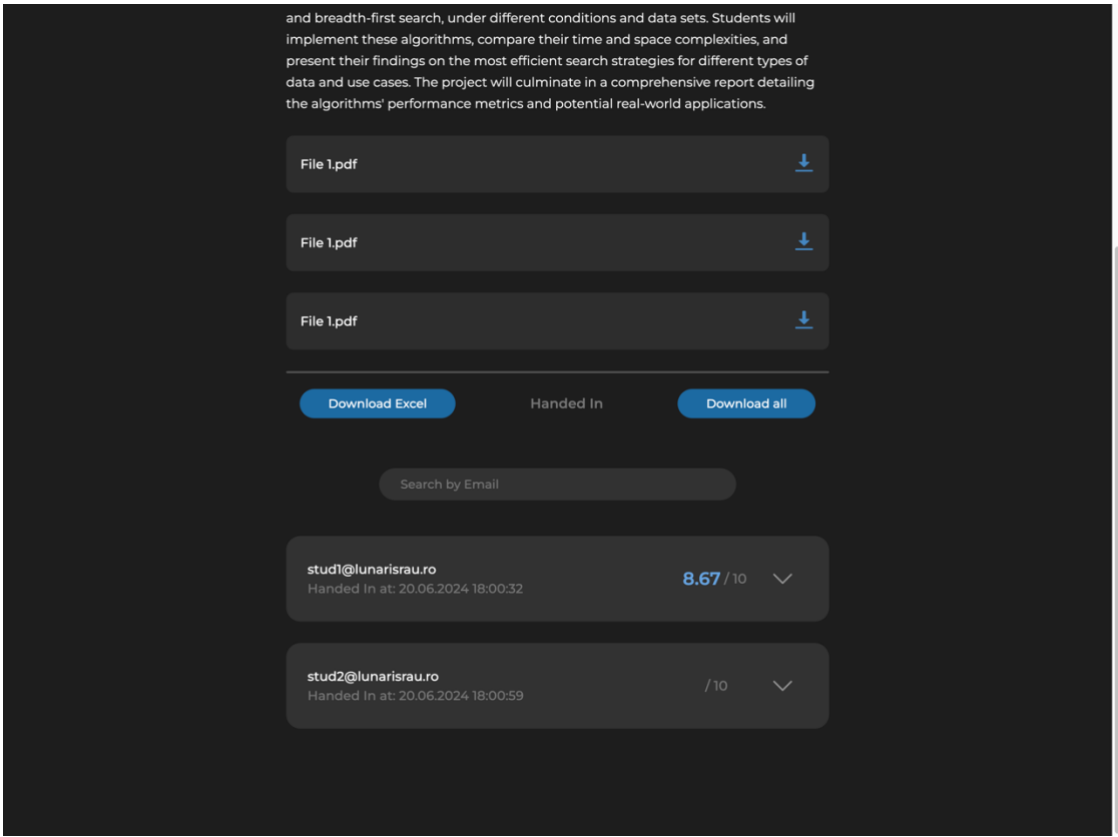
Classes Assignments Grades

This feature provides a vital interface for managing and reviewing student performance on individual assignments within classes. It displays grades as floating-point numbers, alongside pertinent metadata such as submission dates and evaluative comments, offering professors a comprehensive overview. Students access their own grades, fostering transparency and encouraging academic growth. Administrators benefit from this consolidated data, which assists in broader academic assessments and reporting processes.

Files (Specific to Assignments)

The assignment file management system is an advanced feature that significantly aids in the submission and review process. Professors can download all student submissions for a particular assignment in a well-organized ZIP folder, with each student’s work sorted into individual subfolders named after them. This structured approach not only facilitates a streamlined review process but also ensures easy accessibility and management of coursework.

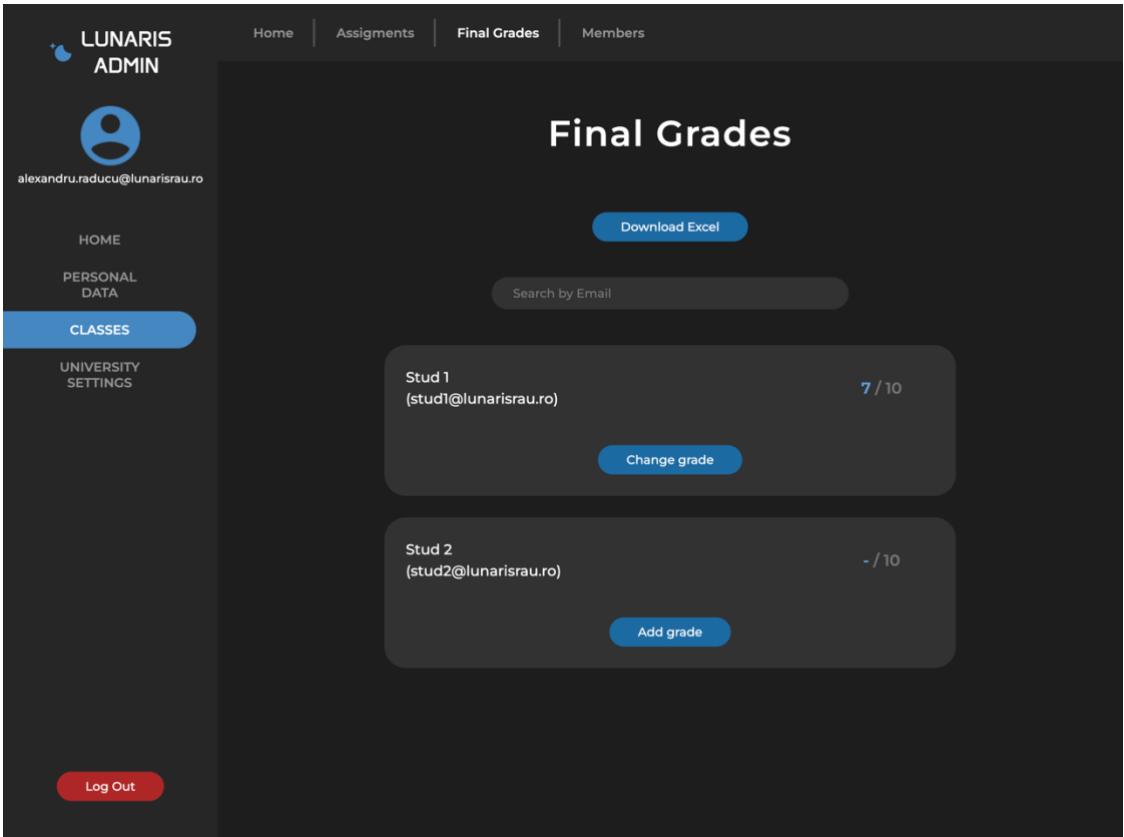
Figure 2.1: The assignment grades output



Final Grades

The output of final grades at the conclusion of each academic term is critical for students monitoring their progress and professors ensuring proper grade tabulation. This comprehensive display includes grades and associated credits for each subject, streamlined for ease of access. Administrators use this output to maintain robust academic records and uphold educational standards across the institution.

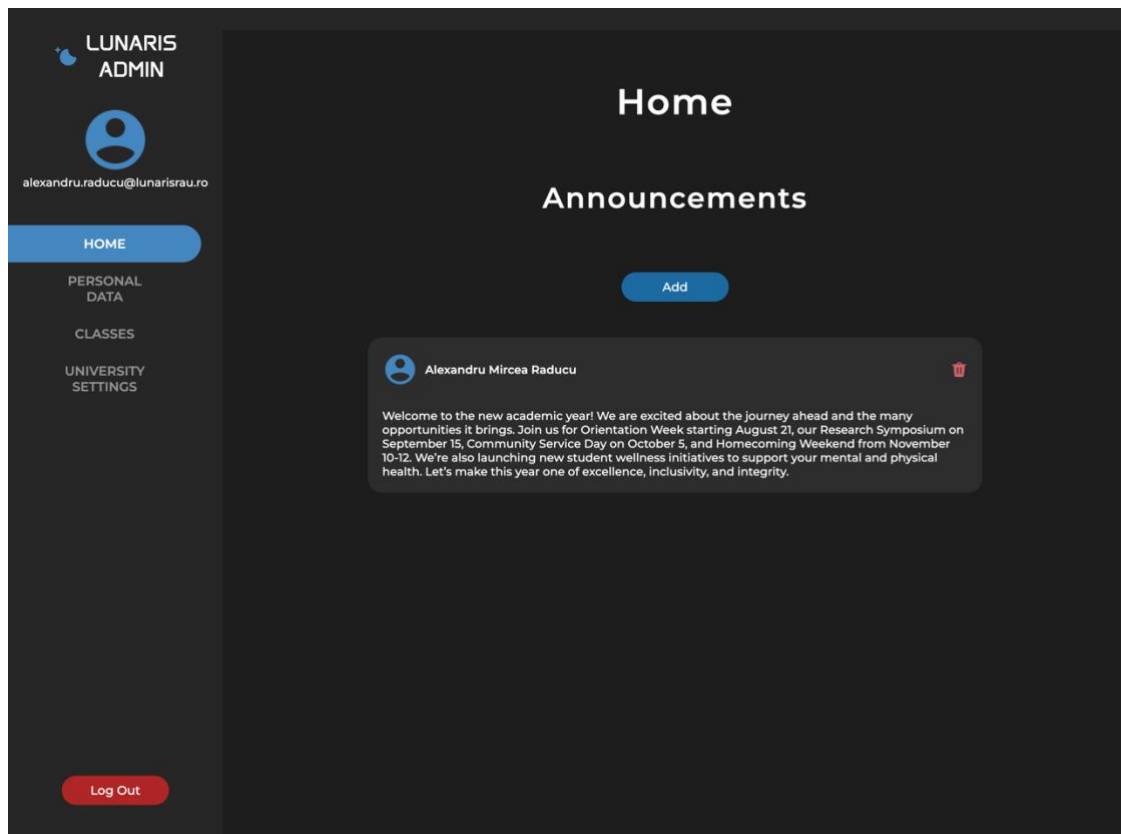
Figure 2.2: The class final grades output



Announcements

The announcement system serves as a dynamic platform for disseminating important information tailored to various campus groups or the entire university community, ensuring that all stakeholders remain well-informed about campus events and administrative notices.

Figure 2.3: The home page announcements



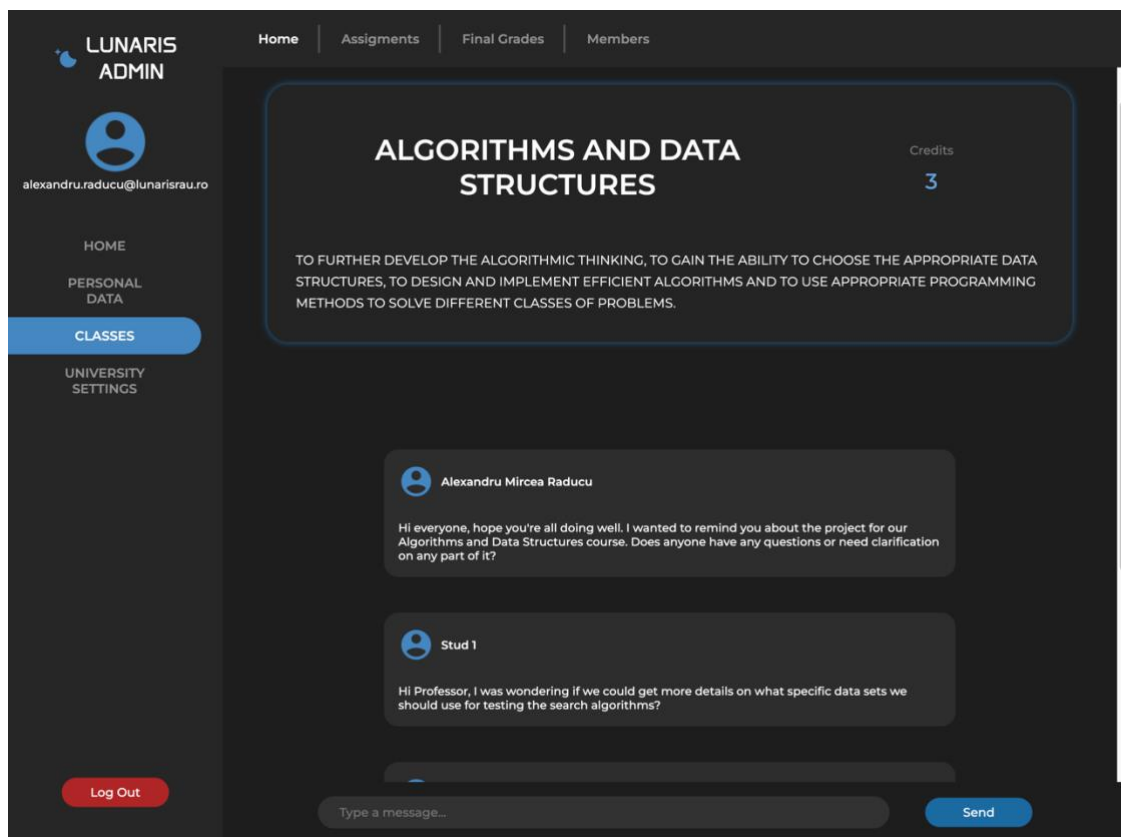
Class Information

Class information outputs provide vital data which can include class schedules, locations, etc. all in one description found at the top of the class. This feature is indispensable for students and faculty alike, aiding in the preparation for and participation in scheduled classes.

Chat Messages

The chat feature supports secure, text-based communication among users, enhancing interaction. This simplifies the exchange of ideas and information, fostering a collaborative academic community. The system's design ensures that chats are visible only within allowed participant groups and for all the universities admins, maintaining privacy and relevance to the involved parties.

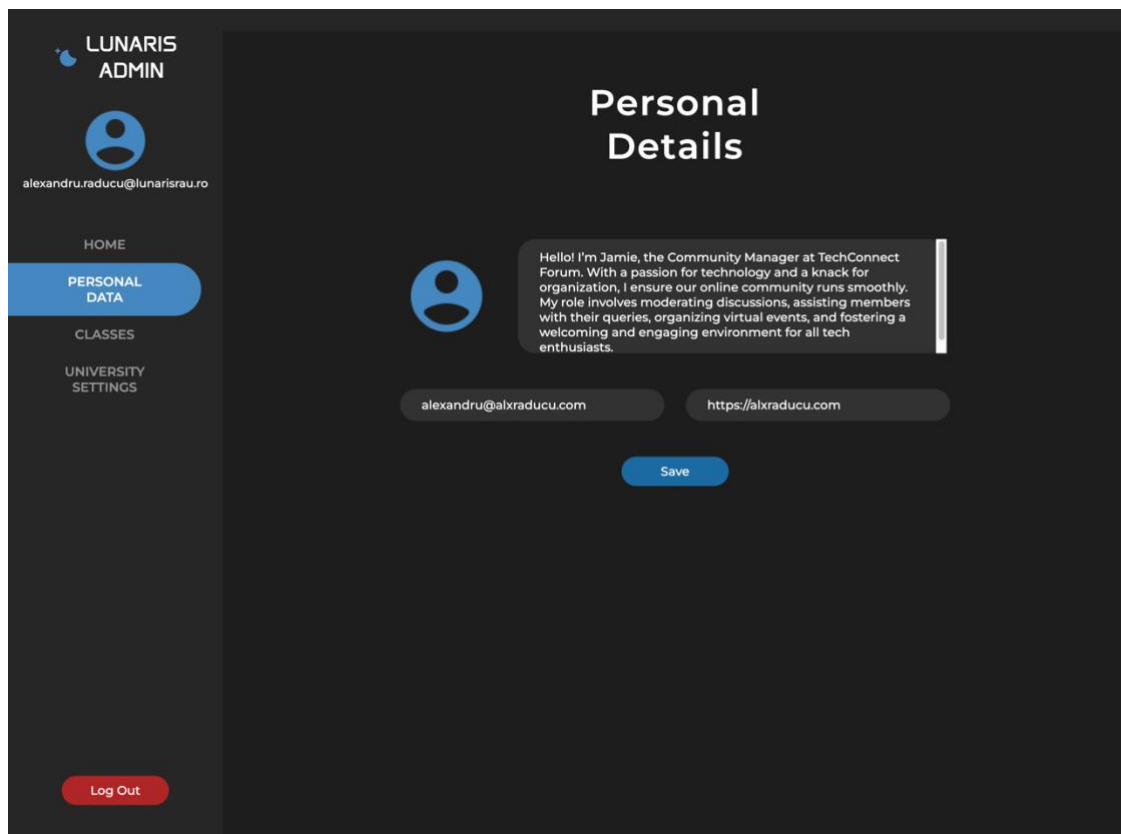
Figure 2.4: The class information and class chat messages



User Profiles

Detailed user profiles enhance the application's interactivity by providing essential information such as roles, contact details, and a description. This feature is designed to meet the specific visibility needs of different user groups, ensuring that students can access appropriate information about faculty and peers, thereby supporting an integrated academic environment.

Figure 2.5: User personal profile on the app



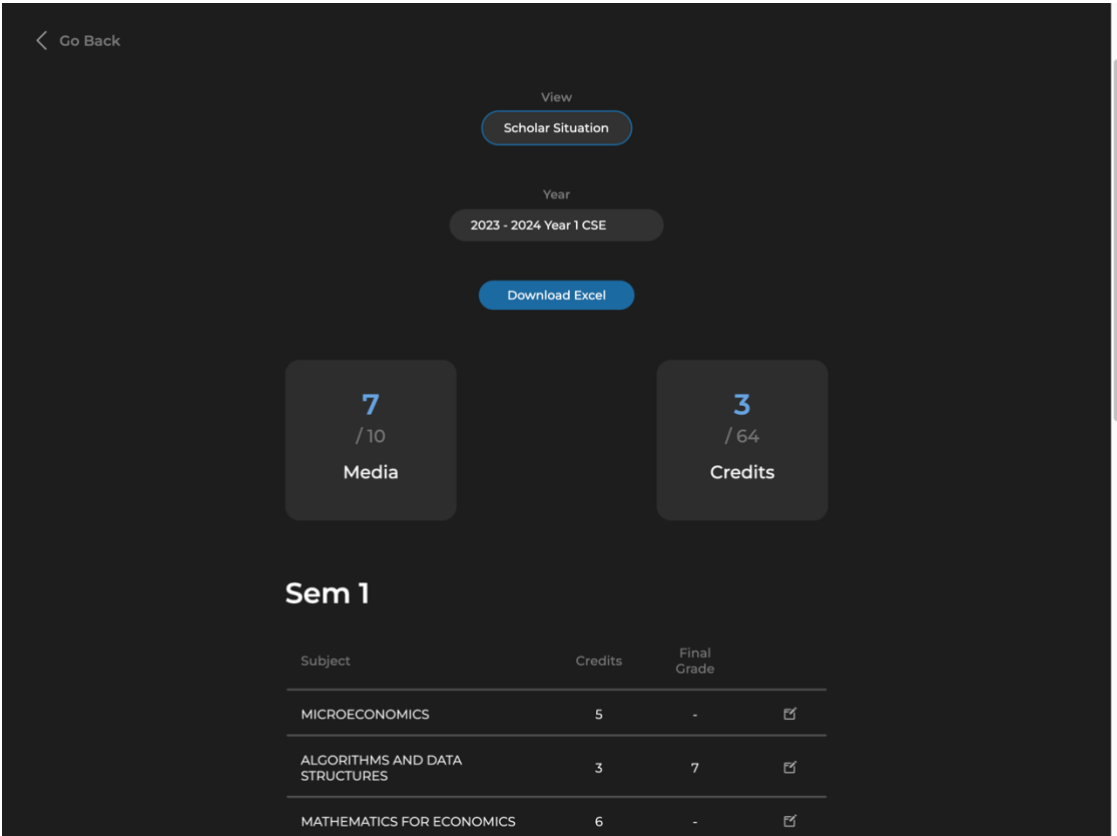
Excel Downloads for Academic Records

The ability to export various academic records into Excel spreadsheets is an exclusive feature for administrators, facilitating detailed reports on final grades, assignment grades, and comprehensive scholar situations. These reports are essential tools for academic planning, performance monitoring, and strategic decision-making, ensuring that administrators can efficiently manage and utilize academic data.

Figure 2.6: Excel spreadsheet of the scholar situation of a student

SubjectId	SubjectName	SemesterIn	Grade	Credits
1	MICROECONOMICS	1	0	5
2	ALGORITHMS AND DATA STRUCTURES	1	7	3
3	MATHEMATICS FOR ECONOMICS	1	0	6
4	ECONOMIC INFORMATICS	1	0	5
5	FUNDAMENTALS OF ACCOUNTING	1	0	5
6	COMPUTER ARHITECTURE AND OPERATING SYSTEMS	1	0	3
7	SPORT I	1	0	2
8	ENGLISH AND COMMUNICATION TECHNIQUES I	1	0	3
9	MACROECONOMICS	2	0	5
10	FINANCE	2	0	5
11	OBJECT ORIENTED PROGRAMMING	2	0	5
12	COMMUNICATION AND PUBLIC RELATIONS	2	0	3
13	ALGORITHMS AND PROGRAMMING TECHNIQUES	2	0	6
14	LAW	2	0	3
15	ENGLISH AND COMMUNICATION TECHNIQUES II	2	0	3
44	SPORT II	2	0	2

Figure 2.7: Interface for downloading and viewing a student’s scholar situation



Overall, the university management application's robust design and comprehensive feature set make it an invaluable tool for enhancing educational management and operational efficiency. By integrating tailored tools for academic and administrative workflows, the application not only improves day-to-day operations but also supports strategic planning and data-driven decision-making. This ensures that the institution remains adaptive, efficient, and focused on providing a high-quality educational experience to all stakeholders.

2.3 The logical and physical design of the inputs

The design of input fields in the university management application is carefully tailored to ensure that all data entered is accurate and useful. This is crucial for a university system where data directly affects both day-to-day operations and academic results. Each input interface is customized for different users like students, professors, and administrators, making sure that everyone finds the system easy to use and efficient. Simple design elements like clear labels and easy navigation help reduce mistakes and make the system secure and user-friendly.

In creating these interfaces, we focus on making sure that each section of the application meets the specific needs of its users. For example, the registration form for new students includes all the necessary fields to gather personal and academic information in an organized way.

Overall, the input design across the application uses consistent colors, fonts, and button styles, which makes the system easy to use and helps build a familiar environment for users. Behind the scenes, the system checks all input data to make sure it's correct before it's saved, which keeps the data reliable and the system running smoothly. This thoughtful approach to design helps make the university management application a key tool in managing the school's academic and administrative tasks efficiently.

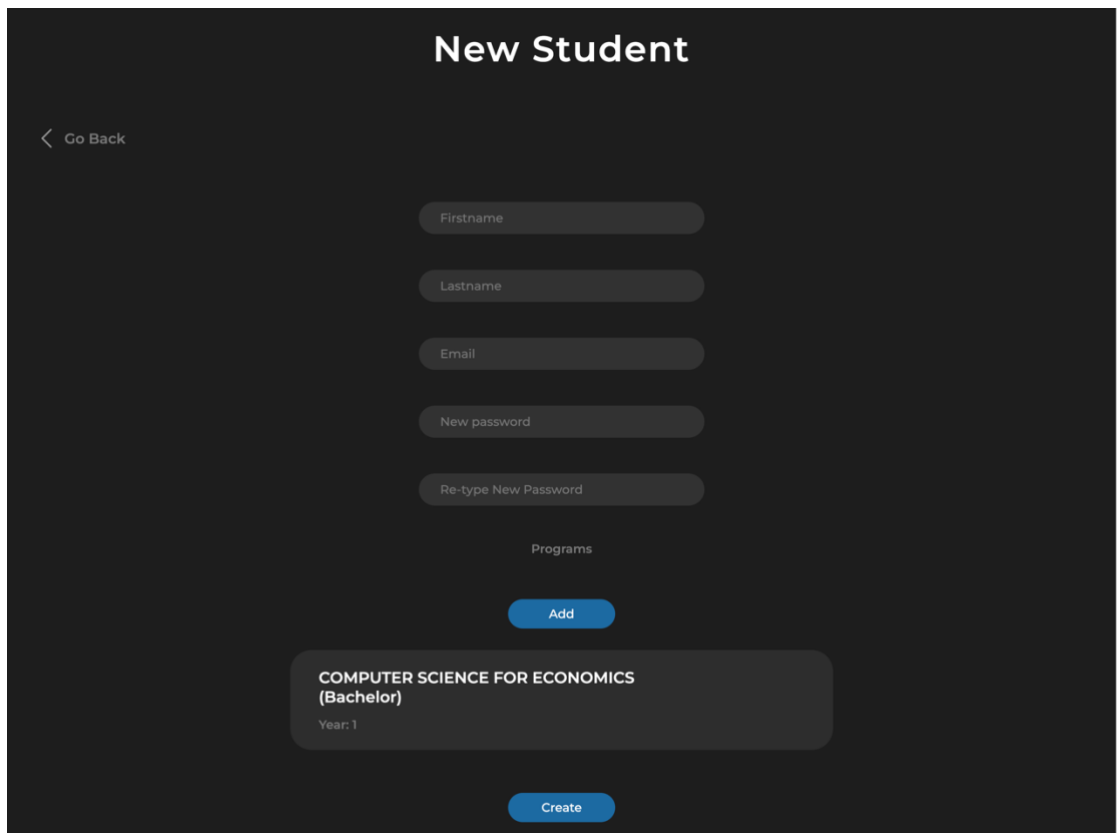
The university management application employs meticulously crafted input interfaces to ensure that data collection is precise, efficient, and tailored to the specific needs of the user. This detailed approach is crucial for maintaining the integrity and functionality of the university's complex operational ecosystem.

User Registration Inputs

The user registration interface is the gateway for all new users, capturing essential details required to integrate them into the university's system. This interface captures personal information, including full legal names, descriptions, and unique identifiers, which are vital for distinguishing individuals within the university's large community. Additionally, it collects comprehensive contact details such as email and website, ensuring effective communication between the university and its students.

The interface also records academic affiliations, documenting departmental and program associations crucial for course management and accessing academic resources. Security settings are implemented to ensure strong password creation and provide options for changing passwords, safeguarding user accounts against unauthorized access. Overall, this registration interface functions as both a data entry point and a critical security feature, protecting user data and personal information.

Figure 2.8: Wizard for creating a student

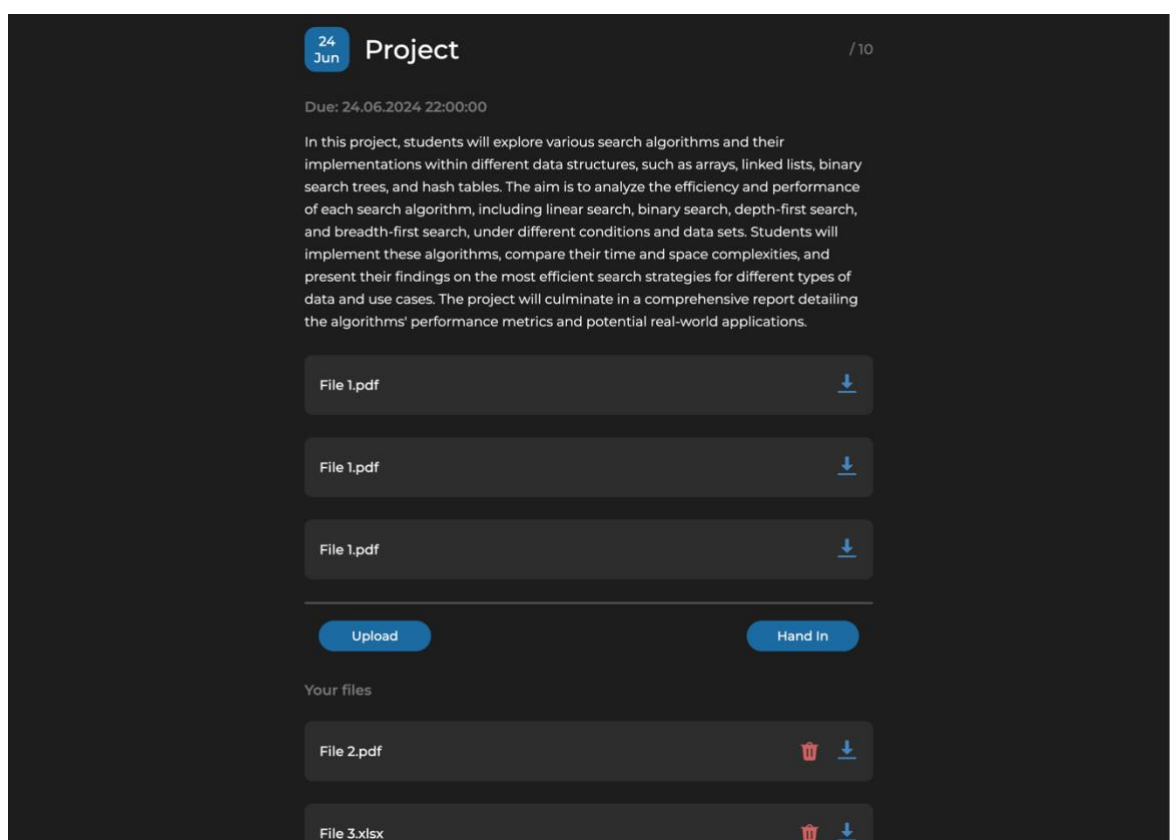


The image shows a 'New Student' registration wizard interface. At the top, the title 'New Student' is displayed in white on a dark background. Below the title, there is a 'Go Back' link with a left arrow. The form consists of several input fields: 'Firstname', 'Lastname', 'Email', 'New password', and 'Re-type New Password'. Below these fields is a section labeled 'Programs' with an 'Add' button. Underneath, a program is listed: 'COMPUTER SCIENCE FOR ECONOMICS (Bachelor)' with 'Year: 1' below it. At the bottom of the form is a 'Create' button.

Assignment Submission Inputs

This interface is designed to streamline the process of submitting digital assignments, a routine yet critical academic activity. It features file upload capabilities that support multiple formats, catering to the diverse needs of different courses that may require specific file types. Additionally, it provides submission confirmation by automatically displaying a detailed timestamp upon submission, serving as proof of submission within the designated deadline. The assignment submission interface ensures a reliable, transparent, and straightforward process for students, reducing technological barriers to academic compliance.

Figure 2.9: Interface for submission of assignment

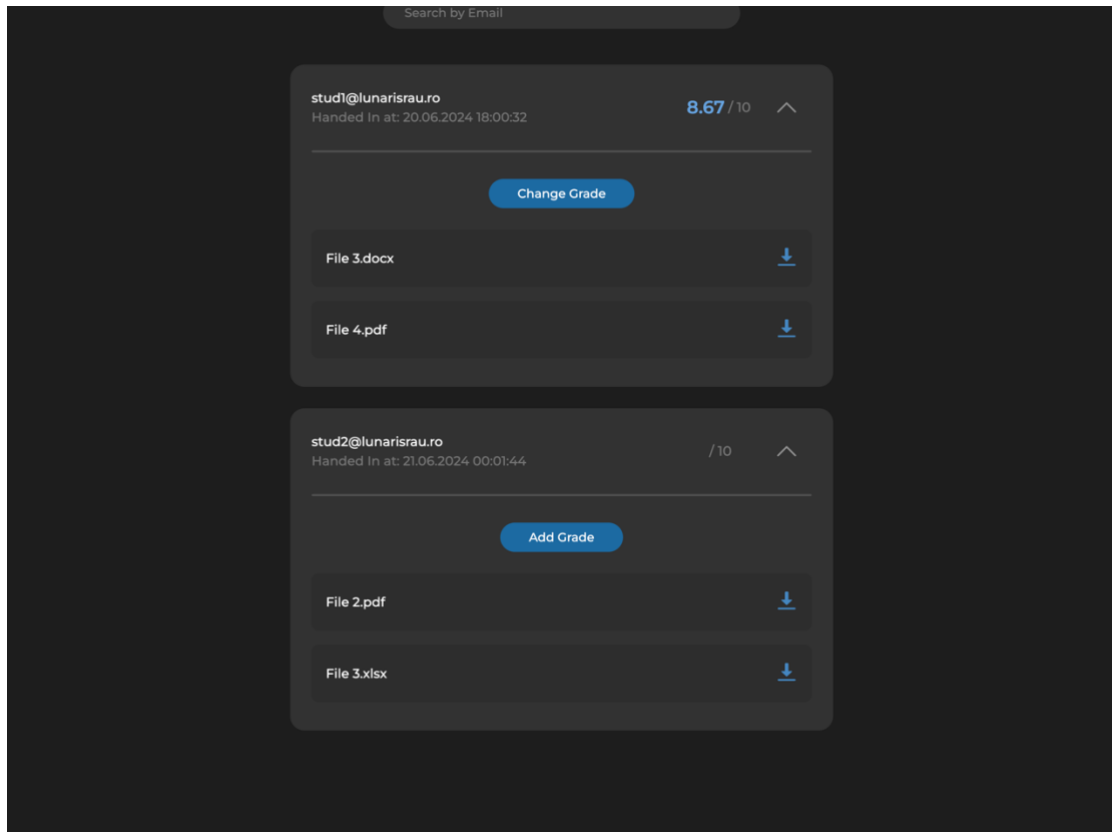


Grade Input by Professors

The grading interface is a critical tool for faculty, designed to facilitate the accurate and timely entry of student evaluations. It includes grade entry features that provide a flexible yet structured format for entering grades, accommodating various grading schemes such as numeric, letters, and pass/fail. The interface also offers grade adjustment functionality, allowing revisions with a complete audit trail to ensure transparency and accountability in grading practices.

Additionally, the assignment grade entry section offers a versatile yet organized format for inputting grades on assignments, supporting numeric values. This design ensures that professors can accurately reflect student performance according to the specific evaluation criteria of each course, facilitating a straightforward and effective grading process. The assignment grade revision feature allows professors to modify assignment grades as needed.

Figure 2.10: Interface for grading assignments



Administrative Data Entry

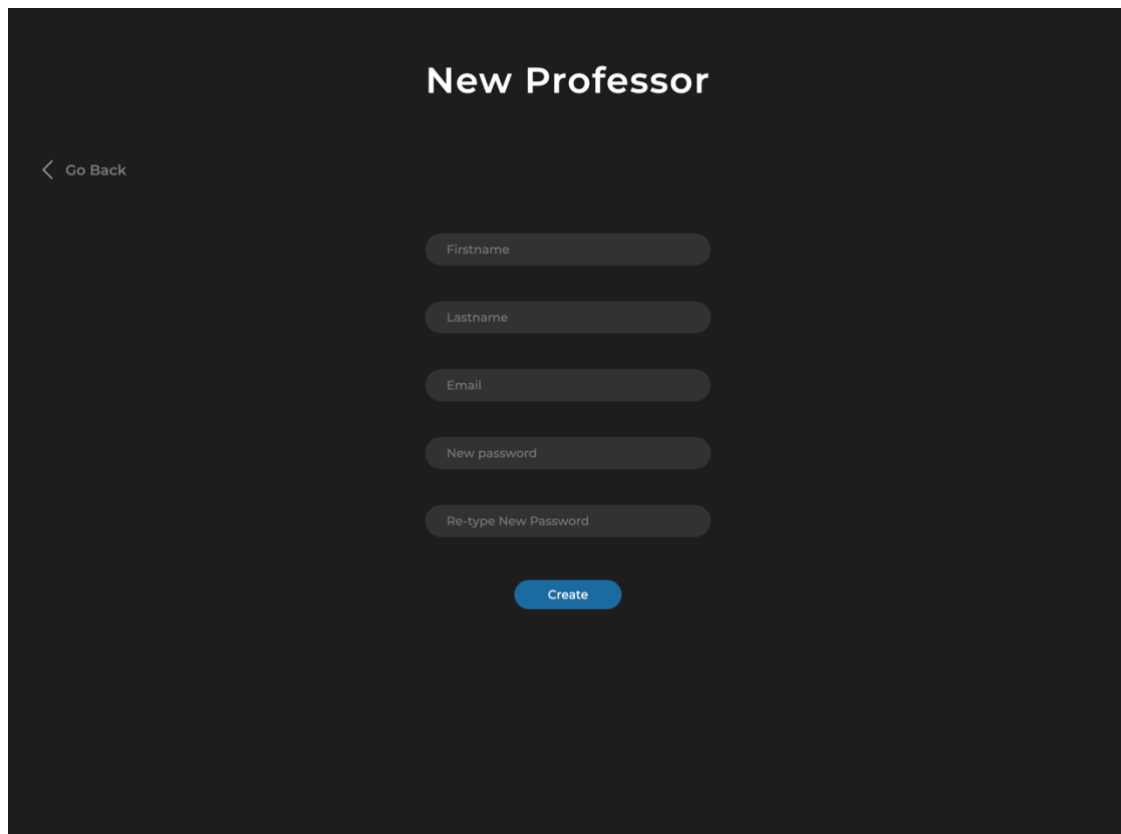
In the university management application, the Administrative Data Entry interface is crucial for handling the extensive data management requirements of the university. This interface empowers administrators with a robust set of tools designed for the creation and maintenance of key university entities such as students, professors, administrators, schools, programs, and subjects. Each category is meticulously structured to capture all necessary data accurately and efficiently.

Administrators can enter student information, including personal details and programs where the student is enrolled. This interface allows for the assignment of students to specific

programs and courses, facilitating precise academic tracking and administrative oversight. View Figure 2.8: Wizard for creating a student.

This part of the interface enables the input of new professor profiles.

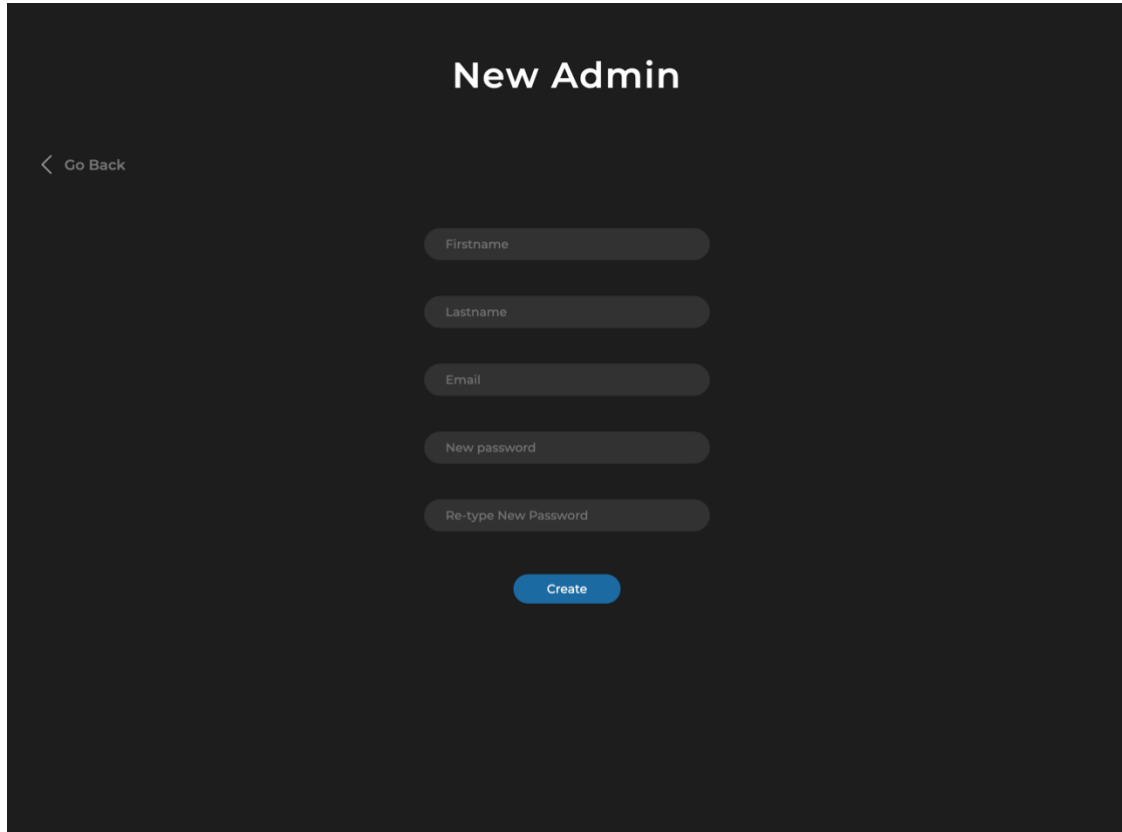
Figure 2.11: Wizard for creating a new professor



The image shows a dark-themed user interface for creating a new professor profile. At the top, the title "New Professor" is displayed in white. Below the title, on the left side, is a "Go Back" link with a left-pointing arrow. In the center, there are five stacked input fields, each with a light gray label: "Firstname", "Lastname", "Email", "New password", and "Re-type New Password". At the bottom center, there is a blue "Create" button with white text.

The system provides a secure method for adding new administrative staff members. All the administrators created have the same level of permissions and access.

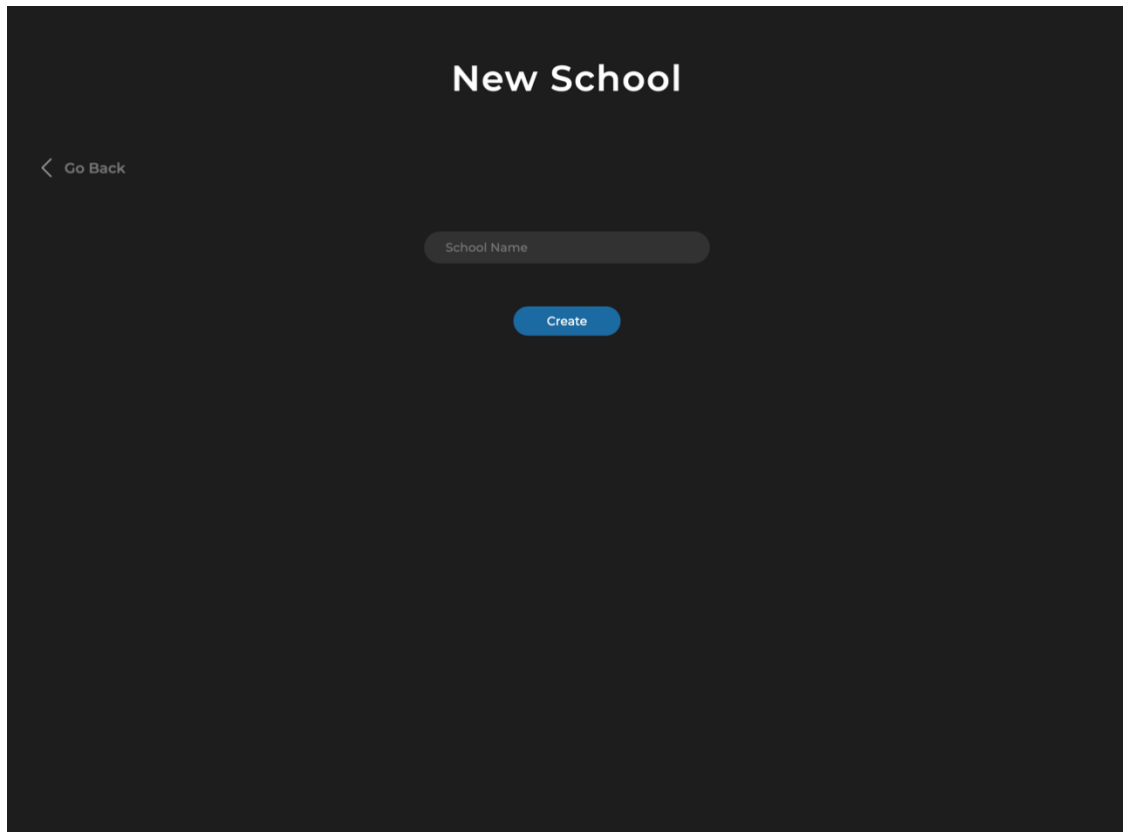
Figure 2.12: Wizard for creating a new admin



The image shows a dark-themed web form titled "New Admin". On the left side, there is a link with a left-pointing arrow and the text "Go Back". The form contains five input fields stacked vertically: "Firstname", "Lastname", "Email", "New password", and "Re-type New Password". Each field is a dark rounded rectangle with light gray placeholder text. Below these fields is a blue rounded button with the text "Create" in white.

Administrators can set up new schools or faculties within the university, defining key attributes such as school name and associated academic programs. This tool is vital for organizing the university's structure and ensuring that each school is properly configured within the overall system.

Figure 2.13: Wizard for creating a new school under a university

The image shows a dark-themed user interface for a 'New School' wizard. At the top center, the title 'New School' is displayed in a large, white, sans-serif font. Below the title, on the left side, is a navigation link consisting of a white left-pointing chevron followed by the text 'Go Back'. In the center of the screen is a light gray, rounded rectangular input field containing the placeholder text 'School Name'. Directly beneath this input field is a blue, rounded rectangular button with the word 'Create' written in white. The entire interface is set against a solid dark gray background.

This functionality allows for the detailed setup of academic programs, including program name, program short name, duration, and associated subjects. It supports strategic academic planning and curriculum development.

Figure 2.14: Wizard for creating a new program

New Program

[Go Back](#)

Program Name

Program Short Name

Program Type

Bachelor

School

SCHOOL OF COMPUTER SCIENCE FOR

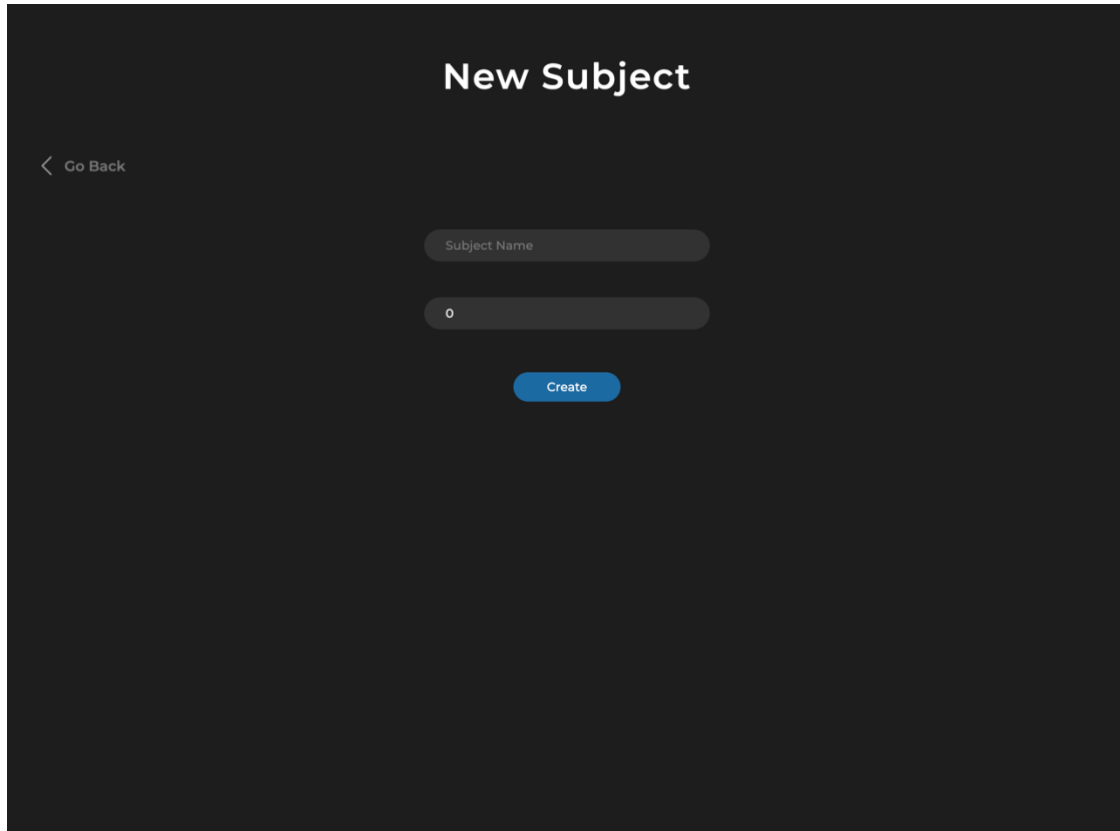
Add Year

Year 1

Add Subject

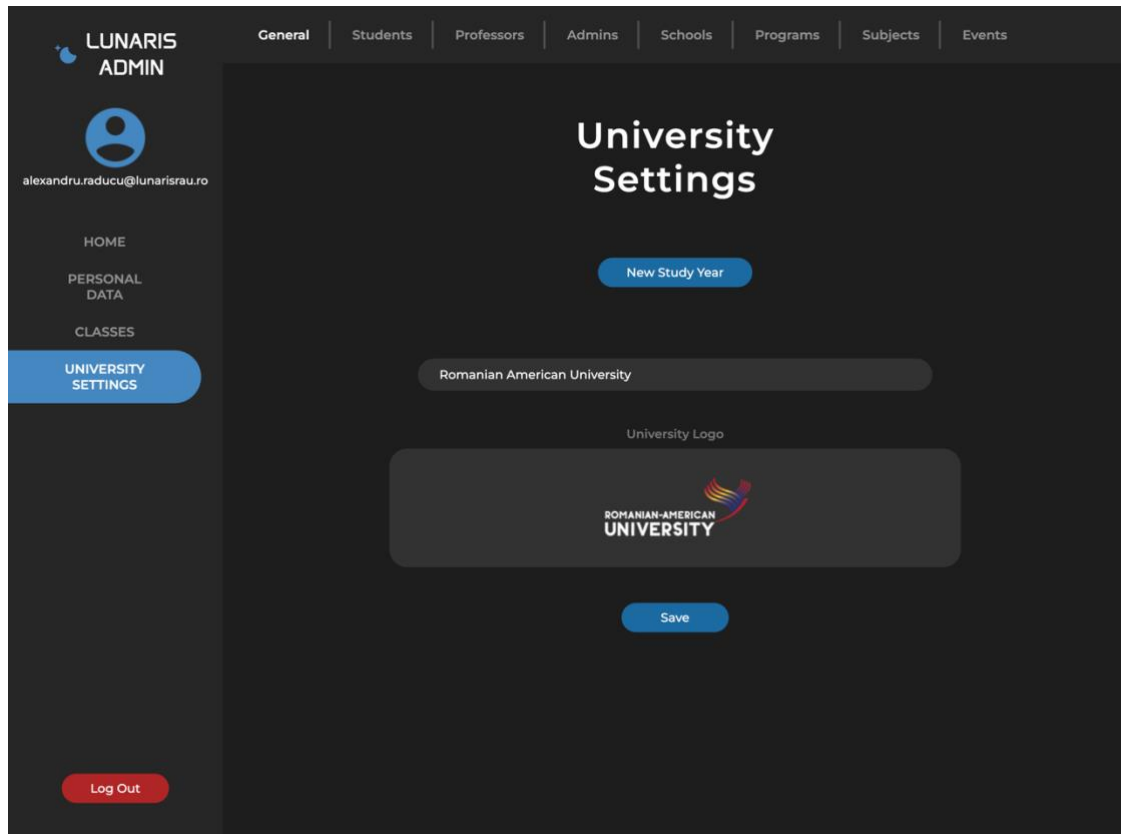
The interface also includes the ability to add and modify subjects offered within different programs. Details such as subject name and credits can be specified, allowing for comprehensive curriculum management.

Figure 2.15: Wizard for creating a new subject

The image shows a dark-themed user interface for creating a new subject. At the top center, the title "New Subject" is displayed in a large, white, sans-serif font. Below the title, on the left side, is a small white left-pointing arrow followed by the text "Go Back". In the center of the screen, there are two stacked, rounded rectangular input fields. The top field is labeled "Subject Name" in a small, light gray font. The bottom field contains the number "0". Below these input fields is a blue rounded rectangular button with the word "Create" in white text.

The university management application, Lunarix, also includes a feature within the general settings that allows administrators to easily upload or update the university's logo and name. This functionality is crucial for maintaining the institution's branding and identity. Through a simple interface, administrators can ensure that the university's visual symbols and name are current and accurately represented across all platforms and communications. This feature is designed for ease of use, allowing for quick changes whenever necessary, such as during rebranding efforts or updates to the university's official name or logo. This capability underscores the application's flexibility and responsiveness to the administrative needs of the university.

Figure 2.16: General settings page for a university



This comprehensive suite of administrative data entry tools is critical for maintaining up-to-date and accurate records across the university, supporting efficient operations, and ensuring compliance with academic standards and policies. Each input interface is specifically designed to be intuitive and secure, minimizing errors and enhancing the overall effectiveness of university administration.

Discussion and Chat Inputs for Classes

Designed to enhance classroom interaction and engagement, this interface supports quick and easy text entry for sending messages, fostering real-time communication among class participants. This interface is key to building an interactive and collaborative educational environment, encouraging students and faculty to engage deeply with course content. View Figure 2.4: The class information and class chat messages.

Profile Updates

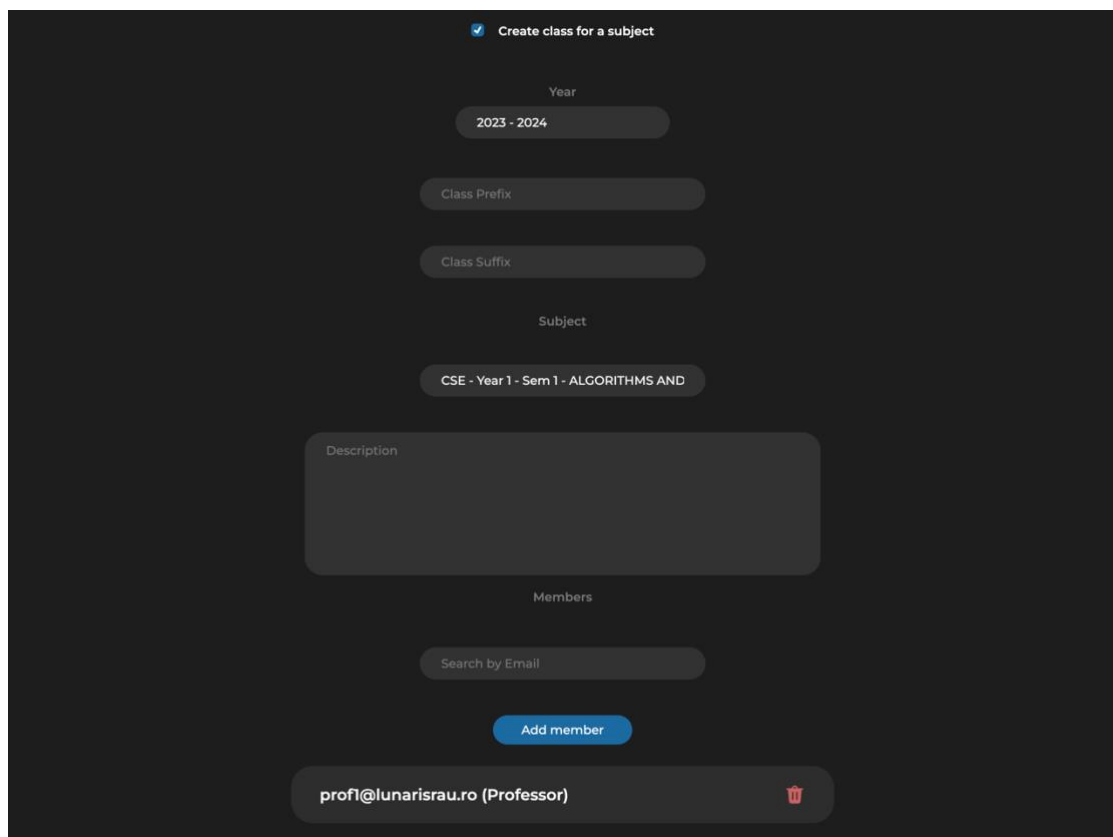
To maintain up-to-date user profiles, the interface includes features for easily updating contact information, ensuring that communication channels remain open and effective. This

interface empowers users to keep their digital identities current and secure, reflecting changes in their university life and personal preferences. View Figure 2.5: User personal profile on the app.

Creation of Classes

For the logistical planning of courses, this interface includes features to capture all necessary class details such as the schedule, location, and instructor, ensuring that classes are well-organized. It also allows for the assignment of instructors to specific classes, ensuring that courses are adequately staffed. This interface is crucial for the administrative management of course offerings, helping to ensure that the academic schedule meets the needs of both students and faculty.

Figure 2.17: Wizard for creating a new class for a given subject under a given program's year



The screenshot displays a 'Create class for a subject' wizard interface. At the top, a blue checkmark icon and the text 'Create class for a subject' are visible. Below this, the 'Year' section shows a dropdown menu with '2023 - 2024' selected. The 'Class Prefix' and 'Class Suffix' sections each have a single-line text input field. The 'Subject' section features a dropdown menu with 'CSE - Year 1 - Sem 1 - ALGORITHMS AND' selected. The 'Description' section contains a large, multi-line text area. The 'Members' section includes a 'Search by Email' input field and a blue 'Add member' button. At the bottom, a list of members shows 'prof1@lunarisrau.ro (Professor)' with a red trash icon to its right.

Each interface in the university management application is designed with precision and a deep understanding of the needs of its users, ensuring that every data entry point is not only

functional but also enhances the user experience within the academic and administrative framework of the institution.

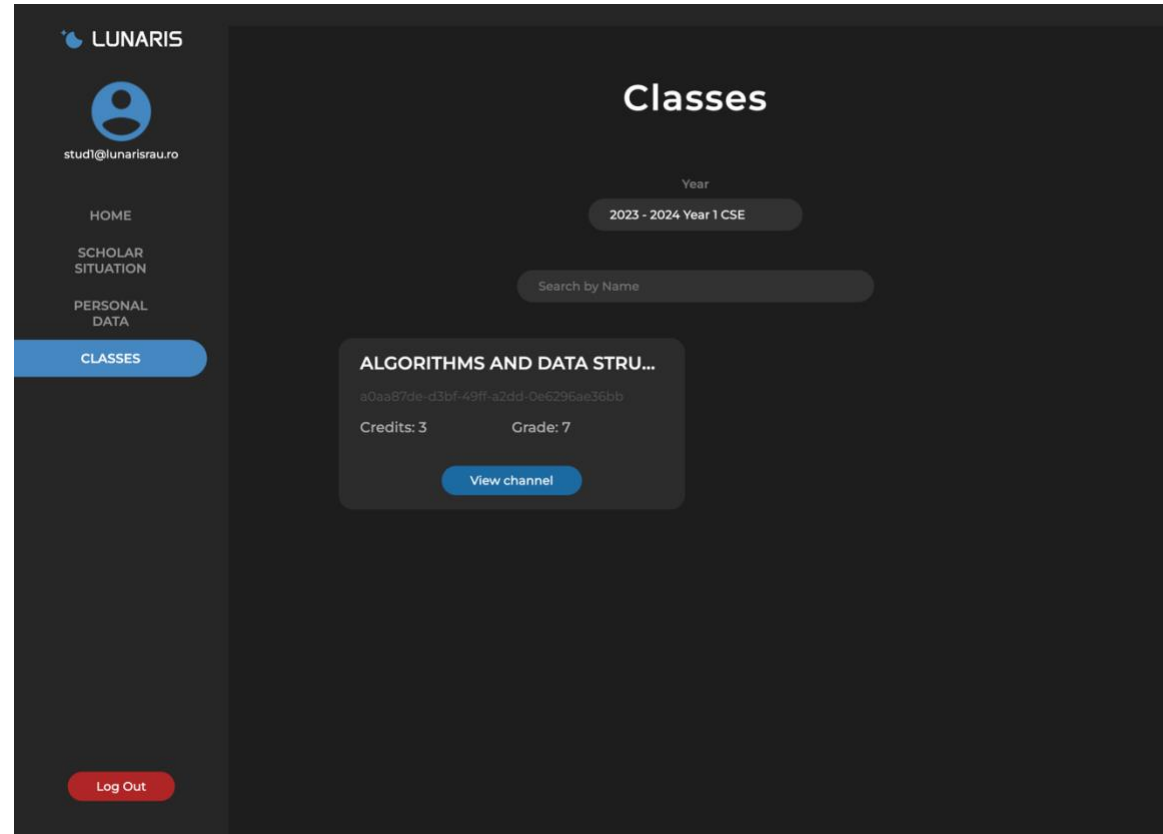
2.4 Data coding

Data coding is a critical step in the development of the Lunarix application, ensuring that raw data is accurately transformed into a structured format that can be efficiently processed and utilized by the system. This chapter elaborates on the data coding process for various features within the app, ensuring precise and reliable data handling.

Class Management

In Lunarix, class management involves handling detailed information about each class, including class IDs, names, instructors, and enrolled students. This data is meticulously coded to maintain consistency and accuracy, facilitating seamless class scheduling and management. The process ensures that all class-related data is easily accessible and manageable, supporting the dynamic academic environment of the university.

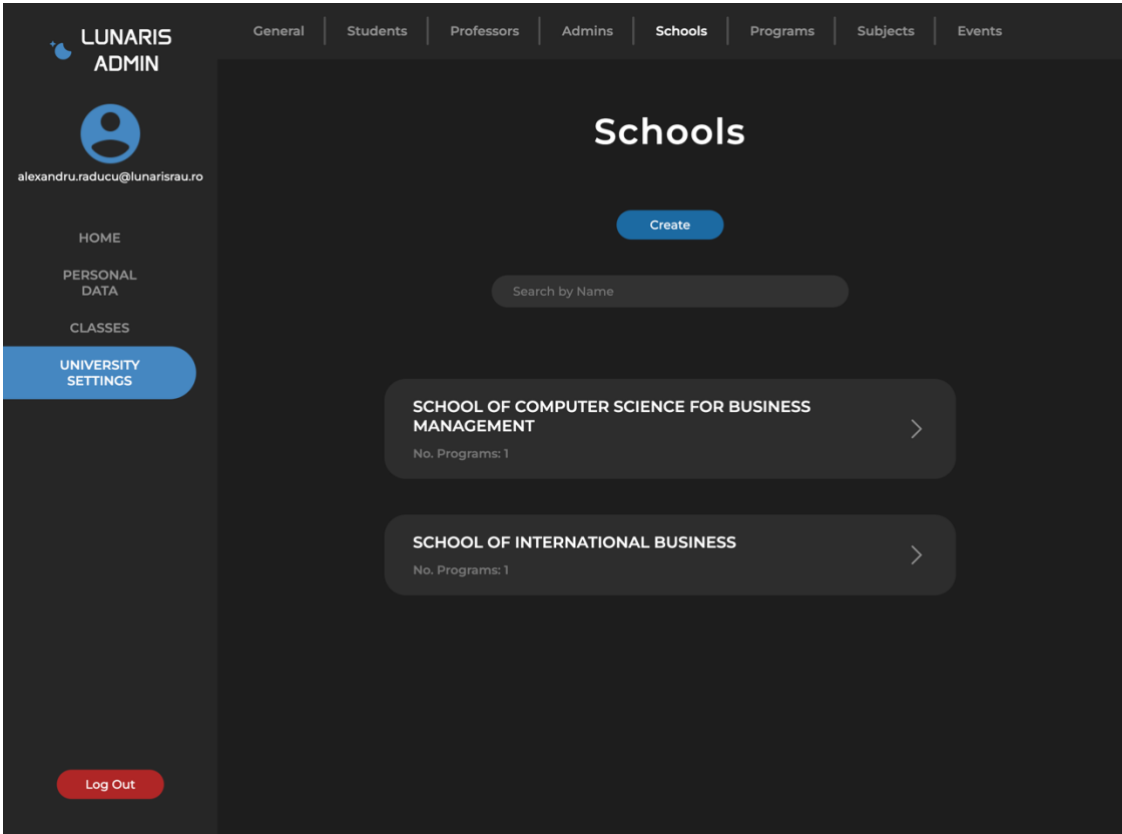
Figure 2.18: The app classes interface for a student



Schools/Faculties Management

The management of schools and faculties within Lunar is requires coding data related to school IDs, names, and the programs they offer. By structuring this data systematically, the application ensures efficient administration of academic departments and faculties, allowing for clear identification and organization of various educational units within the university.

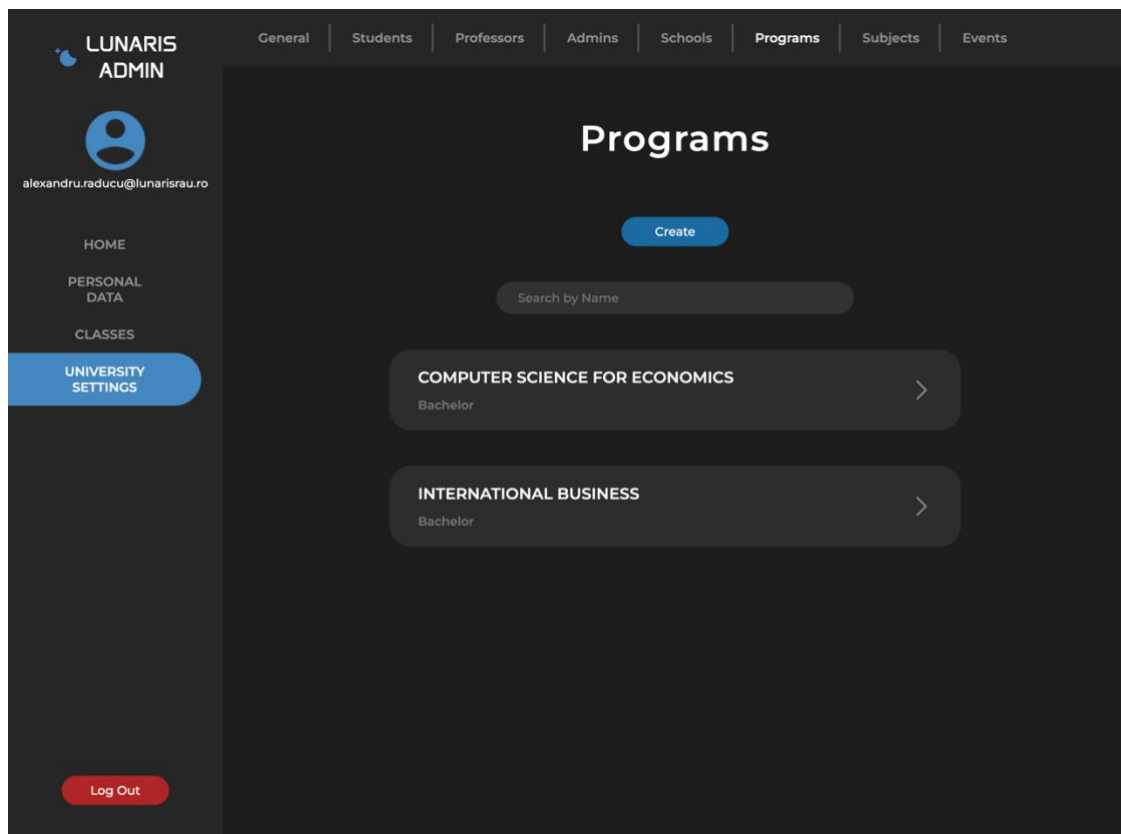
Figure 2.19: The app interface for viewing all schools under a university



Programs Management

Programs management involves coding data on program IDs, names, and degree levels (Bachelor or Master). This process is crucial for organizing and maintaining the university's diverse academic offerings. By coding program information effectively, Lunariss supports streamlined academic planning and administration, ensuring that all programs are accurately represented and easily navigable.

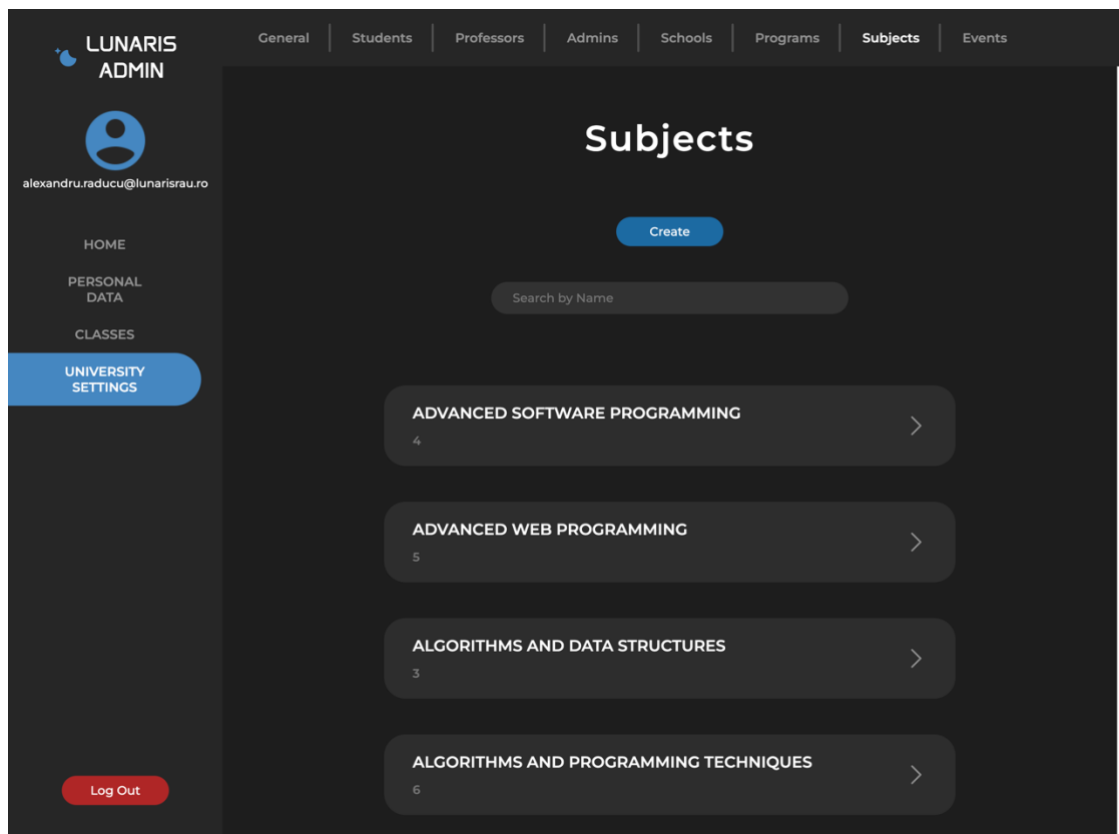
Figure 2.20: The app interface for viewing all programs under a university



Subjects Management

Managing subjects in Lunar is includes coding details such as subject IDs, names, and credits. This structured data coding is essential for maintaining a comprehensive and accessible catalog of courses, enabling students and faculty to effectively plan and manage their academic programs by reusing the same subject for different programs and even across schools.

Figure 2.21: The app interface for viewing all subjects under a university



Student, Professor, and Admin Management

The management of users, including students, professors, and administrators, involves coding personal details, roles, and associated information. This ensures that all user data is systematically organized and securely stored, supporting efficient user management and access control within the application.

Figure 2.22: The app interface for viewing all students under a university

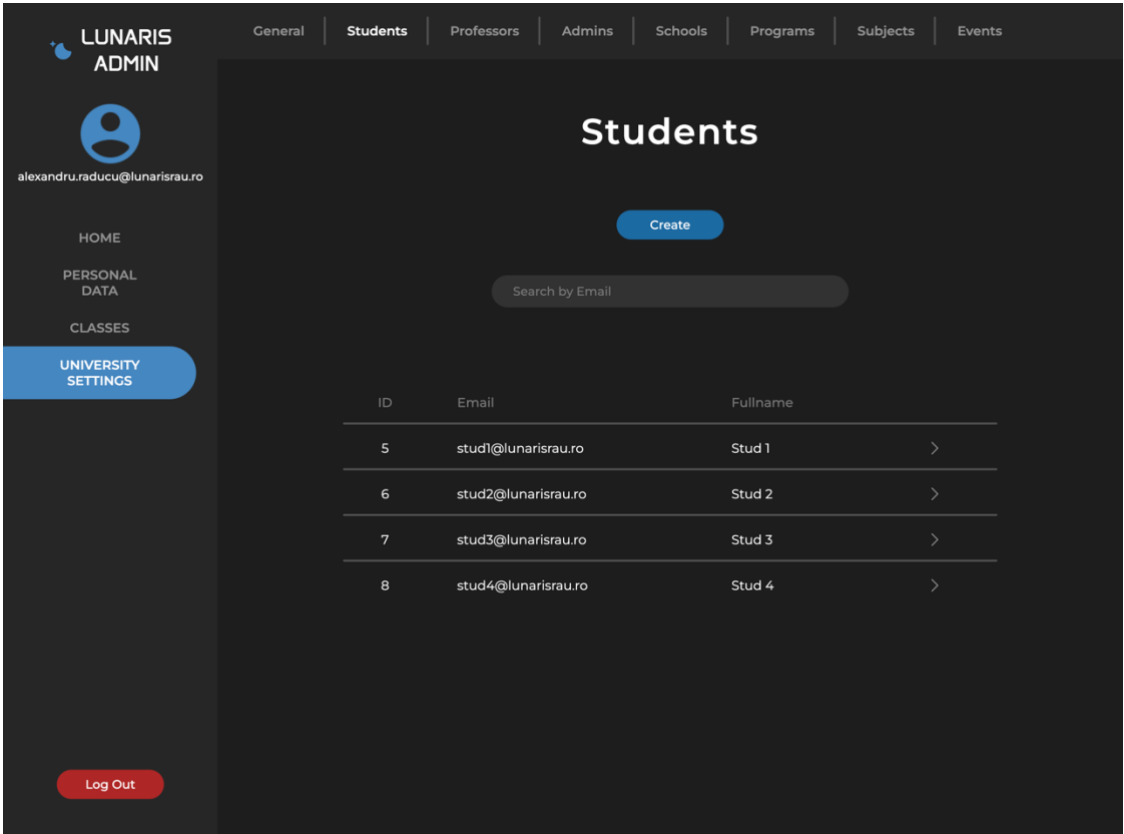


Figure 2.23: The app interface for viewing all professors under a university

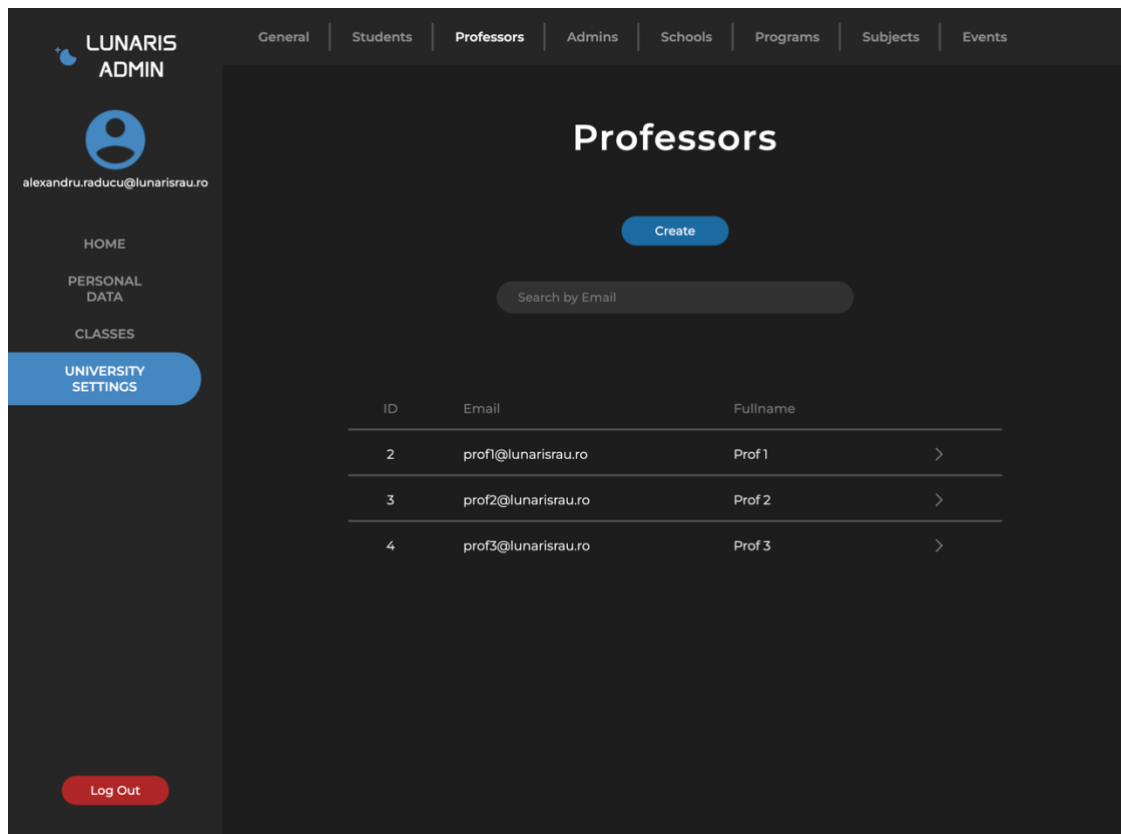
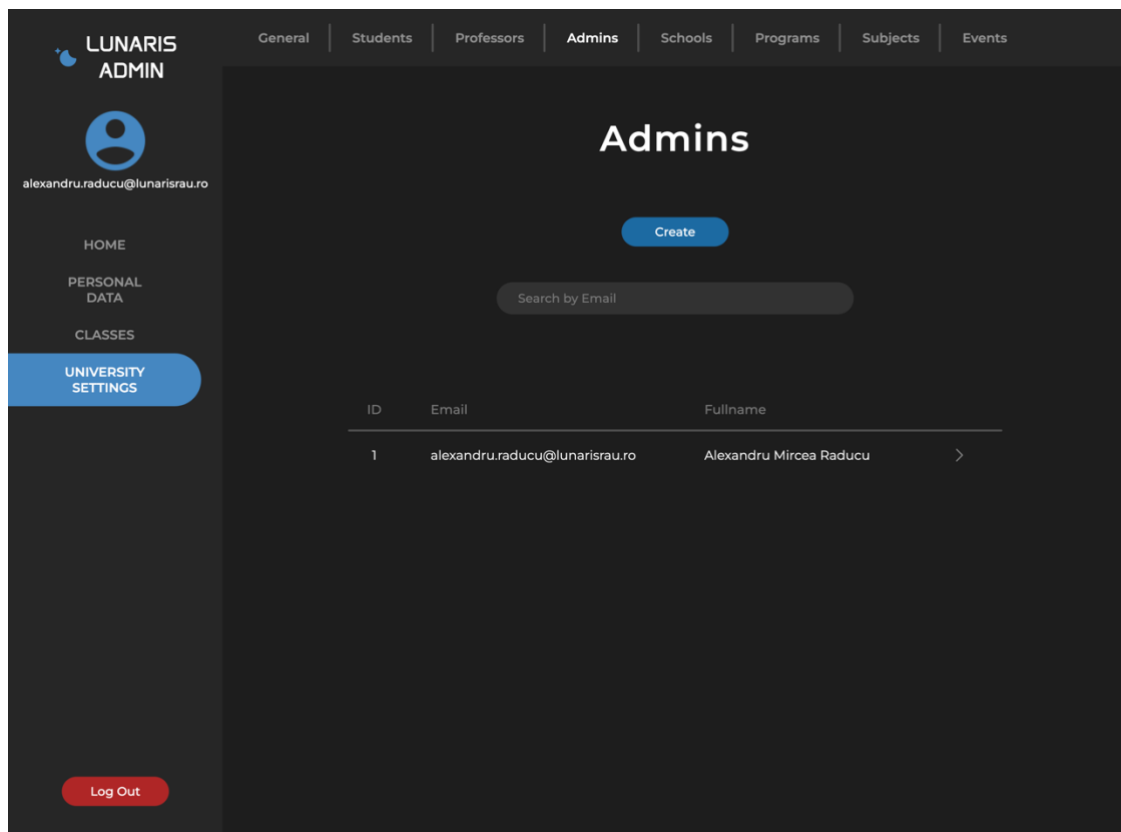


Figure 2.24: The app interface for viewing all admins under a university



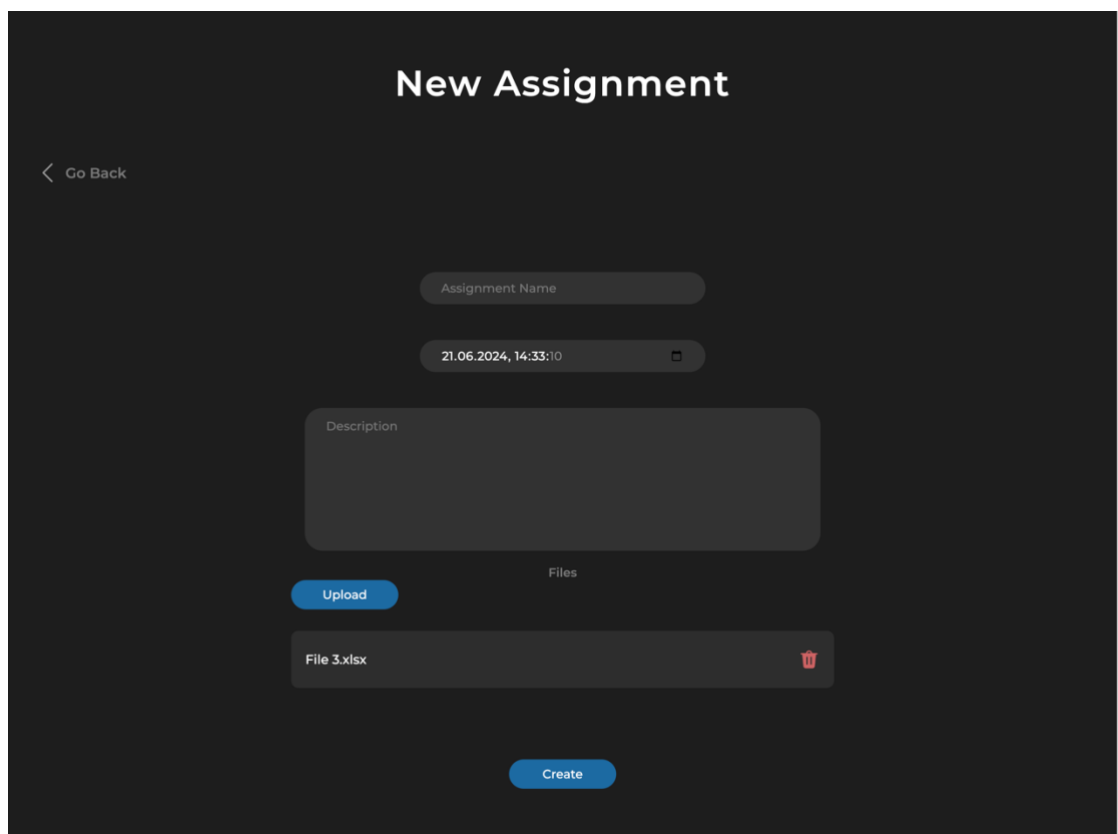
Grades and Scholar Situation Management

Grades and scholar situation management requires coding data on student performance, including grades, credits, and associated metadata. This process ensures accurate and transparent tracking of academic progress, facilitating effective assessment and reporting for both students and administrators. View Figure 2.7: Interface for downloading and viewing a student's scholar situation.

Hosting Assignments for Classes

Coding data for class assignments involves managing assignment IDs, class IDs, due dates, descriptions, and files. This structured approach ensures that all assignment-related information is clearly organized and easily accessible, supporting both the submission process for students and the review process for professors. Furthermore, the users have the ability to upload files which are managed in a separate table.

Figure 2.25: Wizard for creating a new assignment



The image shows a dark-themed user interface for creating a new assignment. At the top, the title "New Assignment" is displayed in white. Below the title, there is a "Go Back" link with a left-pointing arrow. The form consists of several input fields: "Assignment Name" (a single-line text input), a date and time field showing "21.06.2024, 14:33:10" with a calendar icon, and a "Description" (a multi-line text area). Below these fields is a section labeled "Files" which contains an "Upload" button. Under the "Files" section, a file named "File 3.xlsx" is shown with a red trash icon to its right. At the bottom of the form is a blue "Create" button.

Class Messages

Class messages are a vital component of the Lunaris application, enabling effective communication within classes. The coding of class messages includes managing message IDs, class IDs, sender and receiver IDs, content, and timestamps. This process ensures secure and efficient communication among class participants, supporting academic collaboration and interaction. View Figure 2.4: The class information and class chat messages.

To facilitate real-time messaging, Lunaris employs the WebSocket Secure (WSS) protocol. This protocol provides a full-duplex communication channel over a single, long-lived connection. The use of WSS ensures that messages are transmitted securely and instantly, enhancing the responsiveness and interactivity of the messaging system. The WSS protocol is particularly beneficial for maintaining live updates and immediate communication, which are essential for a dynamic educational environment.

In conclusion, the data coding system for the Lunaris application has been meticulously designed to ensure efficiency, accuracy, and security. By incorporating a variety of data types and implementing a coherent structure with efficient storage and access protocols, the system effectively handles diverse information requirements. Security measures, including encryption and access controls, protect the data, while mechanisms for ensuring data uniqueness and validation maintain consistency. The system is optimized for performance, ensuring quick data retrieval and minimal redundancy.

2.5 Database design

The primary objective of the database design for the Lunaris application is to create a consistent and efficient structure for storing and managing data. This design must support the application's functionalities, ensuring data integrity, security, and high performance. To achieve this, a comprehensive requirements analysis was conducted. This analysis involved identifying the necessary data types, relationships, and interactions, considering the application's functionality, user needs, and reporting requirements. By understanding the various data entities, their attributes, and their interrelations, we ensured that the database structure would efficiently support the application's workflows and user interactions.

The chosen data model for the Lunar is application is a relational model, which is ideal for the requirements and nature of the data. This model organizes data into tables with well-defined relationships, ensuring data integrity and reducing redundancy. Specifically, key entities in the database include Users, Courses, Assignments, Grades, Announcements, and Files, some of the tables in the Lunar is database. The relationships between these entities are defined to maintain referential integrity. For example, Assignments and Grades are linked to Courses and Users, ensuring that all relevant data is interconnected and accessible as needed. Appendix 3: Lunar is database ERD.

To further optimize the structure, normalization principles were applied. The database design largely adheres to the third normal form (3NF), ensuring that each table contains only related attributes and dependencies are logical and minimal. This approach eliminates unwanted redundancies and dependencies, helping to maintain data consistency and optimize storage. However, a deliberate decision was made to break the third normalization rule in certain instances to enhance performance. Specifically, the universityId is duplicated across multiple records. This duplication violates 3NF, which normally requires such data to be stored in a single table and referenced through joins. While this approach maintains data integrity, it would significantly degrade performance.

In the Lunar is application, ensuring quick and efficient access to data is paramount, particularly when checking user permissions and managing university-specific data. Under strict normalization, checking a user's universityId and determining their permissions would require traversing multiple tables, significantly impacting performance. By duplicating the universityId in relevant tables, the application can directly access the necessary data, reducing the number of joins and improving query performance. This design choice is a calculated sacrifice, trading off strict adherence to normalization for enhanced efficiency and speed, crucial for providing a seamless user experience.

To ensure data integrity despite this denormalization, various constraints were implemented, including primary keys, foreign keys, and uniqueness constraints. Primary keys uniquely identify each record in a table, while foreign keys maintain referential integrity between related tables. Uniqueness constraints prevent duplicate entries where necessary, such as in the studentYears table. Additionally, indexing strategies were developed to enhance query performance. Critical columns and tables were indexed to speed up frequent searches and

queries, ensuring that the application can handle large volumes of data efficiently, providing quick response times and a seamless user experience.

Security and privacy measures are integral to the database design. Access controls and authentication mechanisms protect sensitive data, ensuring that only authorized users can access or modify data. Encryption is used to safeguard data at rest and in transit, complying with data protection regulations such as GDPR. Regular audits and monitoring help detect and prevent security breaches.

In conclusion, the database design for the Lunarix application is robust, scalable, and secure, providing a solid foundation for the application's current functionalities and future growth. This design ensures that data is managed efficiently, securely, and reliably, meeting the diverse needs of its users. The deliberate trade-off between strict normalization and performance optimization highlights the practical considerations essential for achieving both integrity and efficiency in a complex university management system.

2.6 System diagram

The system diagram of the university management application is a crucial tool that visually represents how various components of the system interact and work together to support the application's overall architecture. By utilizing the first two C's of the CCCC model (Context, Containers, Components, Code), which stands for Context and Containers, we can provide a detailed understanding of the system both at a high level and a more granular level. (Simon, n.d.)

Context Diagram

The context diagram provides a high-level overview of the system, illustrating how the university management application interacts with external entities such as users and other external systems. It highlights the relationships and data flows between the application and these external components, which is essential for understanding the system's boundaries and interactions.

At the center of the context diagram is the university management application, which serves as the hub connecting all external entities. Key external entities include students, professors, and administrators. Students interact with the system to register, submit assignments, and check their grades.

The system captures inputs such as personal details, course selections, assignment uploads, and class messages from students. Professors use the system to manage their courses, input grades, communicate with students, create assignments, and upload instructional materials. The system captures inputs from professors, including grade entries, course material uploads, and class messages.

Administrators access the system to manage user accounts, configure university settings, and oversee academic records. Their inputs include user profile updates, course creation, school creation, and various administrative configurations. This centralized interaction model ensures that the academic and administrative needs of the university are effectively met through streamlined and integrated processes.

Containers Diagram

The containers diagram delves deeper into the internal structure of the university management application, detailing the major containers that make up the system. Each container represents a significant part of the system's architecture, illustrating how they interact and communicate with each other to perform the application's functions effectively. This section will also discuss the pros and cons of using a microservices architecture, as exemplified by Lunarix.

One of the significant advantages of microservices architecture is scalability. Microservices can be scaled independently, which allows Lunarix to handle increased loads on specific services without affecting the entire application. For instance, the Classes Service can be scaled during enrollment periods to manage the heightened demand efficiently.

Flexibility is another key benefit. Different services can use different technologies best suited for their specific tasks. For example, the Login Service may use Node.js for its fast I/O operations, while the Fees Service might leverage Python for complex financial

calculations. This technology diversity enables each microservice to optimize performance and development productivity.

Fault isolation is also a notable advantage. Issues in one microservice do not necessarily impact others. If the Messages Service encounters problems, it won't bring down the entire system, ensuring higher availability and reliability of the application.

Ease of deployment is enhanced through continuous integration and deployment (CI/CD) pipelines, which enable the independent deployment of microservices. This allows Lunarix to roll out updates and new features more frequently and reliably, improving the agility and responsiveness of the development process.

However, managing multiple microservices can be complex. This complexity requires robust orchestration tools like Kubernetes. Developers need to handle inter-service communication, data consistency, and transaction management, which can introduce significant overhead and require specialized knowledge.

Latency is another potential downside. Communication between microservices, often over a network, can introduce latency. Ensuring efficient communication protocols and minimizing overhead is crucial for maintaining performance standards.

Data management presents additional challenges. Maintaining consistency across microservices can be difficult, particularly when each service manages its own database. Strategies like eventual consistency and distributed transactions are often required to ensure data integrity and coherence across the system.

Finally, resource overhead is a concern. Each microservice runs in its own process, leading to increased resource consumption. Proper resource allocation and monitoring are essential to optimize performance and control costs, as the overhead can significantly impact the efficiency and scalability of the system.

The web application consists of two main components: the frontend and the backend. The frontend is the user interface that students, professors, and administrators interact with. It is responsible for rendering views, handling user inputs, and displaying outputs. Technologies

commonly used include HTML, CSS, TypeScript, and the Angular framework. The frontend captures user interactions such as clicks, form submissions, and navigation actions.

In contrast, the backend is the server-side component that processes requests from the frontend, handles business logic, and interacts with the database. Technologies often used include Node.js, Express, or Java Spring. The backend processes inputs from the frontend, executes business logic, and returns appropriate responses.

To manage the distribution of static content such as images, stylesheets, and scripts, the CDN Service ensures fast delivery and efficient use of bandwidth by caching content closer to the user's location, improving the loading times of the application's frontend components.

Handling the management of fee structures, billing, and payment records, the Fees Service processes inputs related to student billing information, tracks payment statuses, and generates financial reports, which are essential for the financial operations of the university.

The Login Service manages user authentication, including login attempts, session management, and security measures. By validating user credentials, maintaining session integrity, and preventing unauthorized access, it safeguards user data.

Facilitating the sending of notifications to users, the Push-Notifications Service ensures timely delivery of important updates and reminders by managing the creation, scheduling, and dispatching of notifications across different channels such as email and mobile push notifications.

Managing configurations specific to each university, the University-Settings Service allows administrators to update and maintain the visual identity and essential configurations of their respective institutions, such as logos and names.

The Classes Service handles the creation, modification, and management of class details, including scheduling, enrollment, and maintaining class rosters. It ensures that all class-related data is accurately recorded and easily accessible for both professors and students.

To manage file uploads, storage, and retrieval, the Files Service ensures that files are stored securely and can be accessed efficiently by authorized users, particularly for assignment submissions and instructional materials.

Supporting communication between users, the Messages Service handles the creation, delivery, and storage of messages, ensuring that communication within the system is seamless and reliable.

For real-time connections and data transfer, the Socket-Server Service manages live interactions such as chat and notifications, maintaining instant updates and interactions between users.

To ensure accurate and up-to-date records for all users, the User-Details Service manages user profile information by processing inputs related to user data, such as profile updates and role changes.

Representing the user-facing part of the application, the App-Frontend interacts with backend services to render the user interface, ensuring that the frontend displays data correctly and provides a responsive user experience.

MySQL serves as the primary relational database for structured data, storing user profiles, course information, grades, and administrative records. This database supports complex queries and transactions, ensuring data consistency and integrity.

Managing session data, Redis-Session provides fast and scalable storage for user sessions, ensuring that session information is stored efficiently and supports quick retrieval and high availability.

For real-time functionalities, Redis-Socket manages data for live interactions and communications, handling real-time data efficiently to provide a robust infrastructure for live updates and notifications.

The relational database stores structured data such as user profiles, course information, grades, and administrative records. Commonly used databases include MySQL, which handles structured data entries from various system components.

Using NoSQL databases for unstructured or semi-structured data, such as chat logs and real-time notifications, includes examples like Redis for session and socket management. These handle dynamic data from user interactions and system events.

Managing containerized applications, Kubernetes ensures scalability and reliability. Inputs include deployment configurations and scaling rules, which allow the application to handle varying loads and maintain high availability.

Automating the deployment pipeline, Continuous Integration/Continuous Deployment (CI/CD) ensures that updates and new features are seamlessly integrated and deployed. Inputs are code changes, build scripts, and deployment configurations, facilitating rapid and reliable software development and deployment.

Table 2.1: Detailed view of all the MCS with paths and ports

<i>ID</i>	<i>Container name</i>	<i>External path</i>	<i>Internal path</i>	<i>Port</i>
1	login	/apiv1/login	/login	80
2	user-details	/apiv1/user-details	-	80
3	fees	/apiv1/fees /apiv1/webhooks/fees	-	80
4	cdn	/apiv1/cdn	-	80
5	classes	/apiv1/classes	-	80
6	files	/apiv1/files	/files	80
7	messages	/apiv1/messages	/messages	80
8	university-settings	/apiv1/university-settings	-	80
9	push-notifications	-	/push-notifications	80
10	socket-server	/socket.io	/socket	3000 (WSS) 80 (HTTP)
11	clamav	-	-	3310
12	app	-	-	443
13	mySQL	-	-	3306
14	redis-session	-	-	6379
15	redis-socket	-	-	6379

16	mongoDB	-	-	27017
17	mongo-express	-	/mongoexpress	80
18	phpmyadmin	-	/phpmyadmin	80
19	session-viewer	-	/sessionviewer	80

Component Interactions

Within each container, various components interact to perform specific tasks that collectively ensure the smooth functioning of the university management application. These interactions are critical for maintaining data integrity, ensuring security, and providing a seamless user experience. Here's a detailed look at how these components interact within the system:

In the frontend-backend interaction, the User Controller in the backend server handles HTTPS requests from the frontend. For example, when a user logs in, the User Controller processes the login credentials submitted from the frontend form. After the User Controller processes the request, the Service Layer implements the business logic. In the case of a login, this might involve validating the credentials against stored user data. Once the business logic is executed, the Data Access Layer interacts with the database to retrieve or store the necessary information. For login, it checks the user's credentials in the MySQL database. Finally, the result (success or failure of login) is sent back from the backend to the frontend, which then updates the user interface accordingly.

The Login Service authenticates user credentials and creates user sessions. When a user attempts to log in, the Login Service validates the credentials, and if successful, it generates a session token. After authentication, the Login Service may request additional user information from the User-Details Service to personalize the user experience. When a student enrolls in a class, the frontend sends the request to the Classes Service. The Classes Service processes this request, updates the enrollment records, and may interact with the User-Details Service to confirm the student's eligibility.

For real-time communication, the Socket-Server Service manages real-time connections for features such as chat. When a user sends a message, the frontend sends this to the Socket-Server Service, which then broadcasts the message to the intended recipient(s) through the

Messages Service. Supporting the Socket-Server Service, Redis-Socket handles session data for real-time communication. This ensures that messages are delivered promptly and accurately to the active users.

In data storage and retrieval, the Files Service manages the storage and retrieval of files. When a professor uploads course materials or students submit assignments, these files are processed by the Files Service, stored in the designated storage system, and linked to the appropriate user or course records. Distributing static content efficiently, the CDN Service ensures that when a file is requested, it is delivered quickly from the nearest cache, reducing load times and bandwidth usage.

In session management, Redis-Session manages user session data to ensure smooth and secure user experiences. When a user logs in, session data is stored in Redis-Session, allowing the system to quickly validate active sessions and manage user activities across the application.

These interactions ensure that each user action triggers a series of processes that lead to the desired outcome, maintaining the system's integrity and performance. The detailed component interactions within each container highlight the complexity and efficiency of the system's architecture, ensuring that all inputs are handled appropriately and result in accurate outputs. Appendix 6: Microservices (MCS) internal relational view.

The system diagram, comprising the context and containers diagrams, provides a detailed view of the university management application's architecture. By illustrating how external entities interact with the system and how internal containers communicate, this diagram helps in understanding the application's structure, ensuring efficient data flow and robust functionality. This detailed representation lays the foundation for further exploration of the application's components and code, completing the comprehensive CCCC model and demonstrating the system's capability to support the complex needs of a university environment.

2.7 Interface design

The interface design of the university management application focuses on creating a user-friendly and efficient experience for students, professors, and administrators. The primary goal is to ensure that the design is intuitive, making it easy for users to navigate and perform their tasks effectively.

The first objective in designing the interface is to prioritize usability. This means ensuring that the application is straightforward to use. Users should be able to find what they need quickly without extensive training or guidance. To achieve this, the design must incorporate familiar navigation patterns and clear, descriptive labels.

Figure 2.26: Navbar for admins

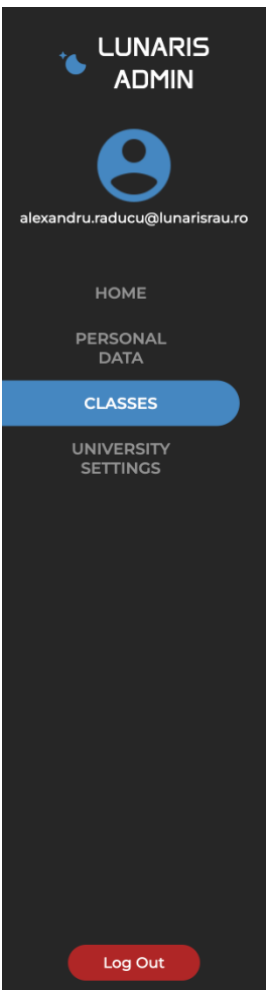


Figure 2.27: Navbar for professors

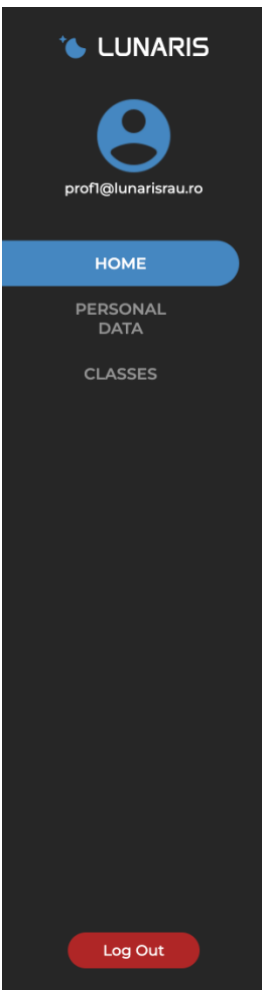
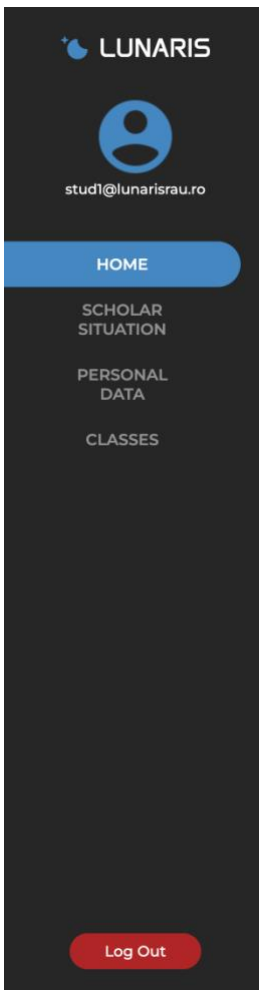


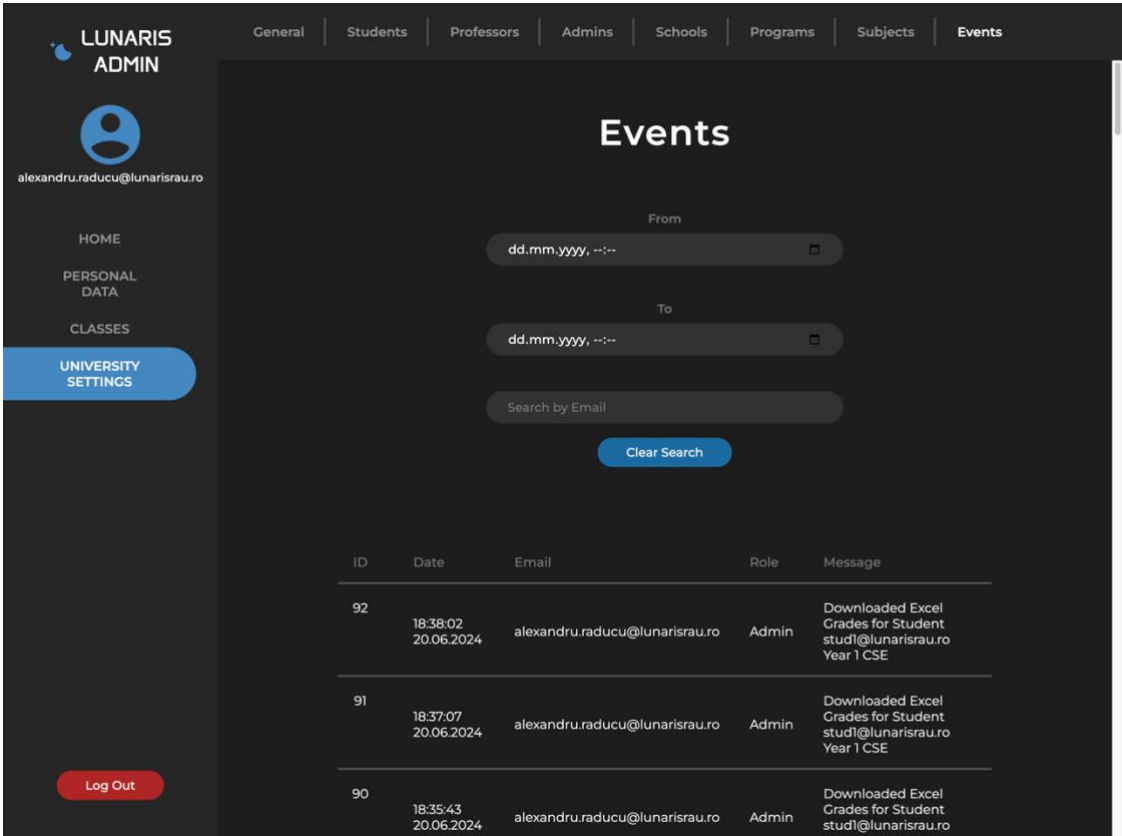
Figure 2.28: Navbar for students



Consistency is another critical element. By maintaining uniform design elements throughout the application, users can easily learn and predict how to interact with different parts of the system. This includes consistent use of colors, fonts, and button styles, which help reduce the cognitive load on users and enhance their overall experience.

Efficiency in interface design allows users to complete their tasks quickly and with minimal effort. This involves designing streamlined workflows and minimizing the number of steps required to perform common tasks. For instance, a well-placed search bar can enable users to find specific features or information swiftly, enhancing their productivity.

Figure 2.29: Admin events filters and search interface



Aesthetics play a significant role in user experience. A visually appealing design can make the application more engaging and enjoyable to use. This includes using a clean and modern design language, with an emphasis on readability and a pleasant color scheme that reduces eye strain.

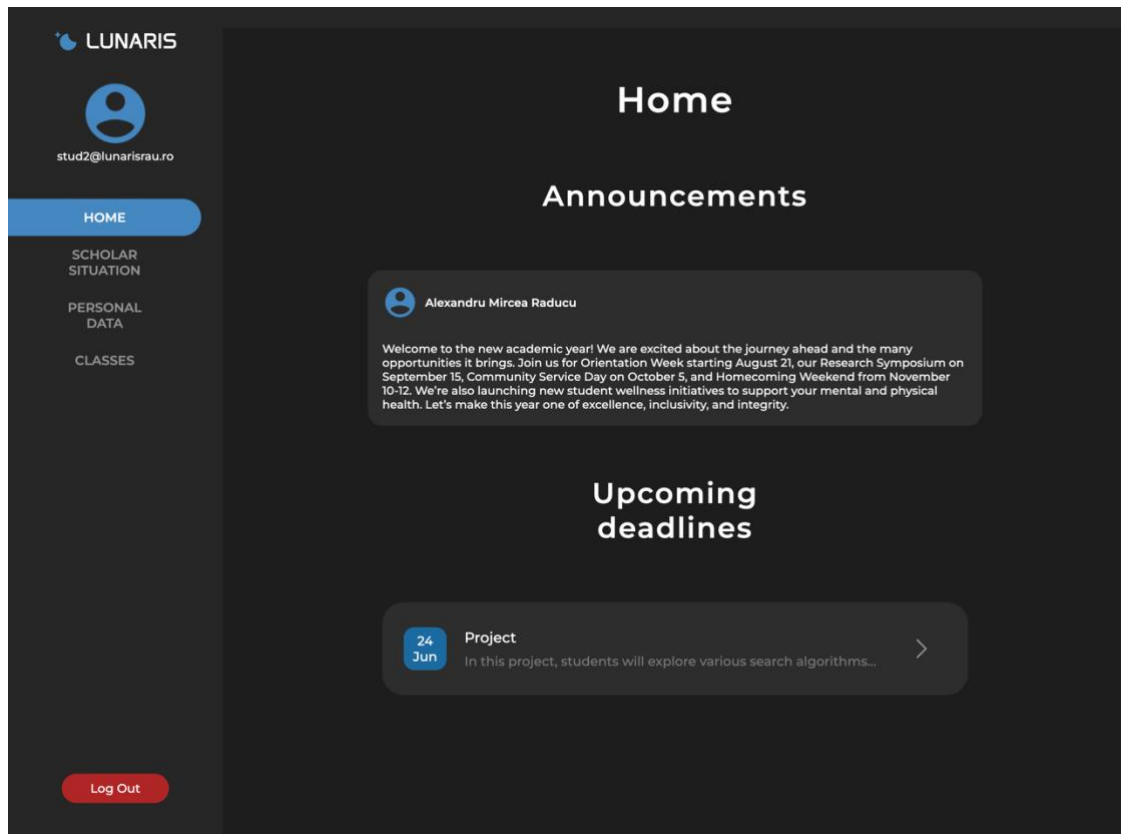
Navigation

The navigation structure of the application is crucial for usability. A well-organized menu with clear categories helps users find their way around the application effortlessly. View Figure 2.26: Navbar for admins, Figure 2.27: Navbar for professors, Figure 2.28: Navbar for students.

Home

The home page serves as a central hub where users can get an overview of their current status and tasks. Furthermore, here is where the admin announcements appear for all member of the university.

Figure 2.30: Home page with widgets for professors and students



Course Management

Course management features are essential for both students and professors. The interface should display courses in a grid or list format, with filters to help users find specific courses easily. Detailed views of individual courses should include important information. For

assignments, the interface should support file uploads and provide a clear way for students to submit their work and for professors to review it.

Messaging System

A robust messaging system is crucial for effective communication within the university. The system should feature a centralized inbox for all communications, ensuring that users do not miss important messages. Real-time chat functionality allows for quick and effective communication between students and professors, while notifications alert users to new messages or announcements. View Figure 2.4: The class information and class chat messages.

Profile Management

Profile management is another important aspect of interface design. Users should be able to view and edit their profiles easily, updating personal information, and contact details as needed. View Figure 2.5: User personal profile on the app.

Administrative Tools

For administrators, the interface includes powerful tools for managing users, generating Excel reports, and configuring system settings. User management features enable administrators to efficiently add, edit, and oversee user accounts. Report generation tools help in creating and viewing academic reports, supporting the administrative needs of the university. Additionally, system settings allow administrators to customize the application to meet the specific needs of their institution, for example changing the university logo, which is shown on the login page.

- **Users management interface:** Figure 2.22: The app interface for viewing all students under a university, Figure 2.23: The app interface for viewing all professors under a university, Figure 2.24: The app interface for viewing all admins under a university
- **Report generation interface:** Figure 2.1: The assignment grades output, Figure 2.2: The class final grades output, Figure 2.7: Interface for downloading and viewing a student's scholar situation
- **General university settings:** Figure 2.16: General settings page for a university

Testing and Feedback (UX testing)

To ensure the design meets user needs, extensive testing and feedback collection are essential. Usability testing with real users can provide valuable insights into how the interface performs in practice. Based on this feedback, the design can be iteratively improved. Accessibility testing is also crucial to ensure compliance with standards and to make necessary adjustments for inclusivity.

In conclusion, the interface design of the university management application is crafted to provide a seamless and productive experience. By focusing on usability, consistency, accessibility, efficiency, and aesthetics, the design supports the diverse needs of students, professors, and administrators, enhancing their interaction with the application.

2.8 Information flow diagram of the new system

The Information Flow Diagram (IFD) of the Lunarix University management application is a detailed visual representation that demonstrates how data moves through various modules of the system. This diagram is essential for understanding the interactions between different components, ensuring that data is collected, processed, and utilized efficiently to support the university's operations. The primary objective is to enhance communication, streamline processes, and improve the overall user experience for students, professors, and administrators.

The system is designed to handle various types of information, including user data, course materials, grades, messages, and administrative records. Each type of data follows a specific path through the system, interacting with different modules as required. The major components involved in the information flow include the User Management Module, Course Management Module, Messaging System, Assignment and Grades Management Module, Administrative Tools, Profile Management Module, Notification System, and Reporting and Analytics Module.

User Management Module

The User Management Module is the entry point for all users into the system. When a user is registered or logs in, their information is verified and stored in the user database. This

module interacts with the Profile Management Module to update user details. For instance, when a new user is created, the User Management Module ensures that the user's profile is set up correctly.

Course Management Module

In the Course Management Module, course-related data such as syllabi, and instructor details are stored and managed. Students access course materials, submit assignments, and view grades through this module. Professors use it to upload course content, grade assignments, and communicate with students. This module interacts closely with the Assignment and Grades Management Module to ensure the smooth handling of coursework. For example, when a professor uploads a new assignment, the Course Management Module updates the course content.

Messaging System

The Messaging System facilitates communication between users. Messages can be sent and received in real-time, supporting group conversations in classes. The system ensures that all messages are stored securely and are easily accessible. For instance, when a student sends a message to a professor, the Messaging System records the message.

Assignment and Grades Management Module

This module is central to the academic aspect of the system. Professors can create assignments, students can submit their work, and grades can be entered and viewed. The module ensures that all submissions are tracked and that grading is transparent. It closely interacts with the Course Management Module and the User Management Module to maintain accurate records. For example, when a student submits an assignment, the Assignment and Grades Management Module processes the submission, records it, and updates the student's academic record.

Integration and Data Security

Ensuring the secure and efficient flow of information between these modules is critical. The system employs robust security protocols, including encryption and access controls, to protect data integrity and privacy. Each module is designed to handle data securely, ensuring that only authorized users can access sensitive information. For example, when data is

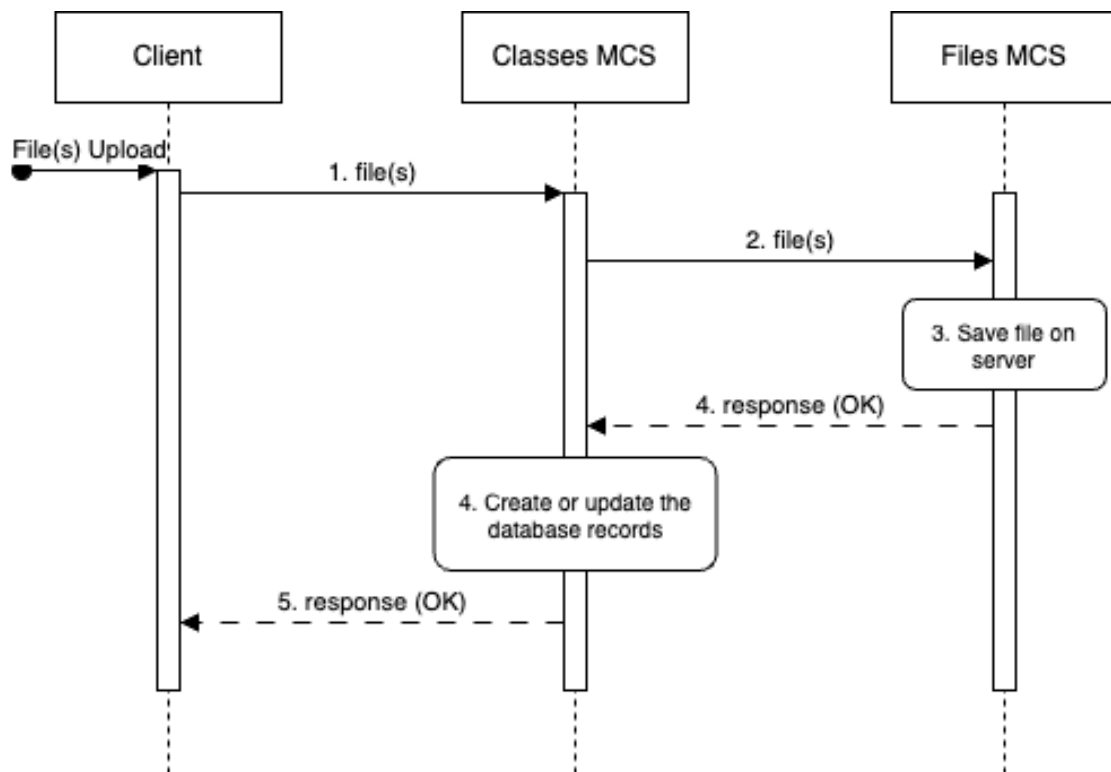
transmitted between the Course Management Module and the Assignment and Grades Management Module, it is encrypted to prevent unauthorized access.

Example: Handling Class Assignments Files

To illustrate how microservices communicate in Lunaris, let's consider the example of handling class assignment files. When a professor uploads an assignment file or when students submit their assignments, the following steps are taken:

1. **File Upload by Professor:** A professor uploads an assignment file through the user interface. This request is handled by the Classes Microservice (Classes MCS), which processes the request but does not directly save the file.
2. **Classes MCS Interaction:** The Classes MCS sends a request to the Files Microservice (Files MCS) to save the file. This is done via secure server communication, ensuring that the data is encrypted and transmitted safely.
3. **File Saving by Files MCS:** The Files MCS receives the request from the Classes MCS. It then interacts with the Files Database (Files DB) to store the file metadata and the file itself.
4. **Internal Confirmation:** Once the file is successfully saved, the Files MCS sends a confirmation back to the Classes MCS, which then updates the status of the assignment in its records, and all the database transactions are committed.
5. **Client Confirmation:** After the Classes MCS has finished updating the database records, the final response is sent back to the user to notify them that the upload was successful.

Figure 2.31: Sequence Diagram for Example: Handling Class Assignments Files



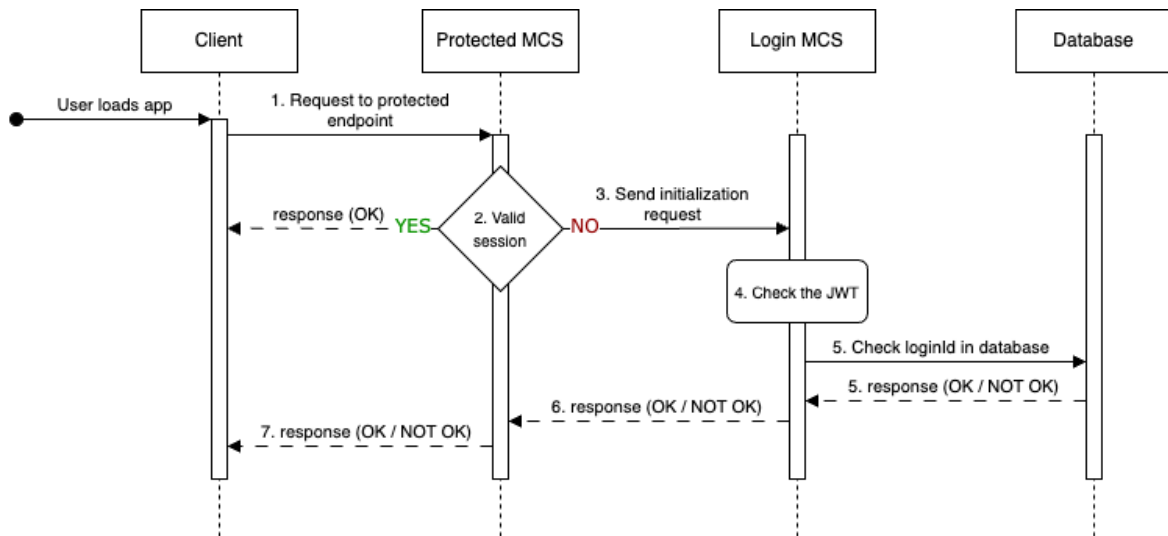
This separation of concerns ensures that each microservice is focused on a specific task, enhancing the overall system's robustness and maintainability. The Classes MCS handles class-related functionalities without needing to manage file storage details, while the Files MCS specializes in managing file operations, ensuring that files are stored securely and efficiently.

Example: Handling Internal Communications for user authentication

To illustrate how microservices communicate in Lunaris, let's consider the example of handling internal communications between the Main Communication Service (MCS) and the Login Microservice (Login MCS) for JWT validation. When a user attempts to access a protected resource, the following steps are taken:

1. **User Request:** A user makes a request to access a protected resource. This request is received by the MCS, which handles the initial processing but does not directly validate the JWT.
2. **Session Validation by MCS:** The MCS first checks if the session is valid by querying its internal session store. If the session is valid, it proceeds to update the last used time for the session without involving the Login MCS.
3. **Session Validation Request to Login MCS:** If the session is not valid or has expired, the MCS sends a request to the Login MCS to validate the JWT. This request is securely transmitted to ensure data integrity and confidentiality.
4. **JWT Validation by Login MCS:** The Login MCS receives the request and decrypts the JWT using its private keys. It checks the validity of the JWT and whether the user session is still active.
5. **Database Interaction:** If the JWT is valid, the Login MCS updates the last used time in the database for the corresponding login session. It ensures that all database transactions are committed to maintain consistency.
6. **Response to MCS:** After successful validation, the Login MCS sends a confirmation back to the MCS. This response includes the updated session information and any necessary user data.
7. **Client Confirmation:** Finally, the MCS processes the confirmation and allows the user to access the protected resource. A response is sent back to the user to notify them that their request was successful and the resource is accessible.

Figure 2.32: Sequence Diagram for Example: Handling Internal Communications for user authentication



This separation of concerns ensures that each microservice is focused on a specific task, enhancing the overall system's robustness and maintainability. The Main Communication Service (MCS) handles initial request processing and session management without needing to manage the complexities of JWT validation. Meanwhile, the Login MCS specializes in authentication and JWT operations, ensuring that tokens are validated securely and efficiently. This clear delineation of responsibilities allows each microservice to operate independently, reducing interdependencies and improving the system's scalability and reliability.

By employing this architecture, Lunarix can scale individual components independently, update services without affecting others, and maintain high security and performance. This modular approach simplifies development and maintenance and aligns with modern best practices for building scalable and resilient applications.

In summary, the Information Flow Diagram for the Lunarix system highlights the interconnectedness of the various modules and the pathways through which data travels. By understanding this flow, developers, and administrators can ensure that the system functions smoothly and that users have a seamless experience. Each module's interactions are designed to support the university's operational needs, enhance communication, and improve overall efficiency.

2.9 Choice of processing technology

The choice of processing technology for the Lunarix University management application is critical to ensuring the system's performance, scalability, and maintainability. This section details the selected technology stack, explaining why each component was chosen, where it is used (backend or frontend), and the specific purposes it serves within the application. The stack includes Node.js with TypeScript, Kubernetes (K8S) on Ubuntu, MySQL, Redis, Angular, CapacitorJS, CSS, and HTML.

Node.js with TypeScript is a key combination for backend development in the Lunarix application. Node.js is chosen for its non-blocking, event-driven architecture, making it ideal for building scalable network applications. TypeScript, a superset of JavaScript, adds static typing, which enhances code quality and maintainability. This combination is extensively used in the backend to handle application logic, API requests, data processing, and server-side operations. Specifically, Node.js can handle a large number of simultaneous connections with high throughput, essential for a university management system that must support numerous concurrent users. TypeScript's static typing helps catch errors early, improving code reliability and developer productivity. Additionally, this setup manages communication between different microservices and handles business logic and data processing efficiently.

For deployment and orchestration, Kubernetes (K8S) running on Ubuntu is used. K8S provides automated deployment, scaling, and management of containerized applications, offering resource efficiency and better performance. It can automatically scale applications based on demand, ensuring responsiveness under varying loads. Moreover, K8S manages application health by automatically restarting containers that fail and distributing the load across the cluster to ensure high availability.

The MySQL database serves as the primary relational database for Lunarix, managing academic records, user information, course details, grades, and other critical data. MySQL ensures reliable transactions and data integrity, crucial for managing academic records and sensitive user information through its ACID compliance. It supports large-scale deployments and can handle extensive data volumes typical in a university management system. The

extensive documentation and a large user base of MySQL simplify troubleshooting and development.

Redis is employed for session management due to its speed and efficiency, as well as for socket management. It stores frequently accessed data, manages user sessions, and functions as a pub/sub server for creating a custom socketIO server cluster. Redis provides lightning-fast data access, essential for managing user sessions and real-time socket data. It also supports data persistence, ensuring that cached data can be recovered in case of a system failure. Redis is versatile beyond caching, being used for pub/sub messaging and other real-time features.

Angular is used for building the frontend of the application, providing a robust structure and powerful features for developing Single-Page Applications (SPAs). Angular facilitates the development of SPAs, providing a seamless user experience by loading content dynamically without refreshing the page. Its component-based architecture encourages modular development, making the application easier to manage and scale.

CapacitorJS is utilized for building the mobile version of the Lunarix application, enabling it to run on both iOS and Android platforms. It allows for the creation of native mobile apps using web technologies, significantly reducing development time and effort. CapacitorJS provides plugins and APIs to access native device features like the camera, geolocation, and notifications, ensuring near-native performance and a smooth user experience on mobile devices.

CSS and HTML are fundamental technologies used to structure and style the web components of the Lunarix application. CSS ensures that the application is visually appealing and functional across various devices and screen sizes through responsive design. It also enables customization of the application's look and feel to align with the branding and design guidelines of the university. HTML provides the structural foundation for web pages, ensuring that content is organized and accessible.

The technology stack chosen for the Lunarix University management application, including Node.js with TypeScript, Kubernetes on Ubuntu, MySQL, Redis, Angular, CapacitorJS, CSS, and HTML, provides a robust foundation for building a scalable, efficient, and secure

system. Each component is selected for its strengths and ability to integrate seamlessly into the overall architecture, ensuring that the application meets the high standards required for a university management system. By leveraging these technologies, Lunarix can deliver a high-performance, reliable, and user-friendly experience for all its users.

2.10 Estimating the need for resources and the implementation schedule

Resource Estimation

Developing and updating the Lunarix application requires various resources, including human resources, and a technology stack. This section outlines the necessary resources to ensure successful implementation.

Human Resources

- **Software Developers** (4 specialists)
 - Responsible for coding and implementing features.
 - Estimated effort: 10 weeks, full-time.
- **UX/UI Designers** (2 specialists)
 - Focus on designing user interfaces and improving user experience.
 - Estimated effort: 6 weeks, part-time.
- **DevOps Engineers** (2 specialists)
 - Manage software integration, deployment, and maintenance of CI/CD pipelines.
 - Estimated effort: 8 weeks, part-time.
- **Test Engineers** (2 specialists)
 - Conduct continuous testing throughout the development process.
 - Estimated effort: 8 weeks, part-time.

- **Security Analysts** (1 specialist)
 - Perform security audits and ensure data protection.
 - Estimated effort: 4 weeks, part-time.
- **System Architects** (1 specialist)
 - Design the overall system architecture.
 - Estimated effort: 4 weeks, part-time.
- **Product Managers** (1 specialist)
 - Integrate feedback and align the prototype with user needs.
 - Estimated effort: 8 weeks, part-time.

Technological Resources

- **Development Tools:**
 - IDEs, version control systems (e.g., Git), and collaboration tools.
- **Cloud Services for Deployment and Production:**
 - Scalable cloud solutions (e.g., AWS, Google Cloud) for hosting and auto-scaling.
- **Technological Stack:**
 - NodeJS, Angular, MySQL, ... as stated in 2.9.

Implementation Schedule (3 Months)

The implementation schedule for the Lunarix application is designed to complete the project efficiently within a three-month timeframe. The following timeline provides an overview of the tasks and their durations. View Appendix 1: Gantt Chart for Implementation Schedule.

- **Sprint Planning** (1 week)
 - Scrum Masters and Product Owners define user stories and prioritize tasks.

- **Development Cycles (8 weeks)**
 - **Weeks 1-2: Initial Setup and Basic Feature Implementation**
 - **Task 1:** Setting up development environment and version control (1 week).
 - **Task 2:** Implementing user authentication and authorization modules (1 week).
 - **Weeks 3-4: Advanced Feature Development and Integration**
 - **Task 3:** Develop core functionalities such as class management and grading systems (1 week).
 - **Task 4:** Integrating initial features with existing university systems (1 week).
 - **Weeks 5-6: Continued Feature Development and Initial Testing**
 - **Task 5:** Adding advanced functionalities like online video conferencing and assignment submission (1 week).
 - **Task 6:** Initial testing of all implemented features and bug fixing (1 week).
 - **Weeks 7-8: Finalizing Features and Extensive Testing**
 - **Task 7:** Final implementation of features like payment systems and file management (1 week).
 - **Task 8:** Extensive testing, including security, usability testing, and performance (1 week).
- **Integration and System Testing (3 weeks)**
 - **Week 9: Integration Testing**
 - **Task 9:** Test interactions between app components and resolve issues (1 week).
 - **Week 10: Third-Party System Integration and Security Testing**
 - **Task 10:** Ensure stable integration with third-party systems (1 week).
 - **Task 11:** Conduct in-depth security audits and vulnerability assessments (1 week).

- **Week 11: Performance and Usability Testing**
 - **Task 12:** Evaluate the app's performance under various conditions (1 week).
 - **Task 13:** Conduct usability testing to confirm that the app is intuitive and user-friendly (1 week).
- **Deployment and Iterative Improvement (2 weeks)**
 - **Week 12: Staged Rollout and Initial Feedback Collection**
 - **Task 14:** Gradual deployment to a controlled group of users to monitor initial performance (1 week).
 - **Task 15:** Actively gather feedback from users regarding their experience with the app (1 week).
 - **Week 13: Full-Scale Deployment and Post-Deployment Support**
 - **Task 16:** Expand the deployment to cover all intended users (1 week).
 - **Task 17:** Provide ongoing support and handle emerging issues (1 week).

Importance of Project Management

Effective management of projects plays a role, in the creation and rollout of the Lunarix application. It guarantees that the project progresses as planned and stays within budget limits. Upholds the quality benchmarks. Project management offers a method for organizing, executing, and overseeing project tasks. It promotes communication among team members and stakeholders fostering alignment towards objectives.

Through risk management and timely issue resolution, project management aids in steering the project toward its conclusion. Moreover, it ensures resource utilization and the delivery of value to users. In essence, strong project management is essential, for translating the vision of Lunarix into a user application that caters to the requirements of students, professors, and administrators.

CHAPTER 3. *Presentation of the software product*

3.1 *Hardware and software platform requirements of the application*

Hardware Requirements

For optimal performance, the application requires robust hardware resources. In the development environment, the app runs on a MacBook M3 Pro with 36 GB of RAM. Specifically, the application uses 0.4 CPU out of a total of 12 available CPUs, indicating efficient CPU utilization. For production, a multi-core processor is recommended.

The app consumes 4.65 GB of RAM during development with no active requests and one pod for each MCS. Therefore, a minimum of 16 GB RAM is recommended for development, with 32 GB or more for production to handle higher loads and multiple instances. Adequate storage is necessary for handling databases. An SSD with at least 512 GB capacity is recommended for fast read/write operations.

It is important to note that the production environment requirements might be lower than the development environment. This is because, in production, there is no need for nodemon to watch for file changes and compile TypeScript files to JavaScript. This reduction in overhead can result in lower resource usage.

Software Requirements

The application leverages various software components to ensure efficient operation and scalability.

The application can run on any Ubuntu server that meets the minimum resource requirements. Ubuntu 18.04 LTS or any later LTS version is recommended for its stability, widespread support, and long-time support (LTS). For the browser version, Google Chrome 80 or Mozilla Firefox 75 or later versions are recommended to ensure compatibility and optimal performance.

In terms of additional software components, Kubernetes (K8S) is used for managing containers and providing automated deployment, scaling, and management of containerized applications. MySQL version 8 serves as the main SQL database, managing structured data and ensuring reliable transactions and data integrity. Redis is employed for high-speed data transactions and as a pub/sub server, supporting real-time functionalities and session management with lightning-fast data access.

Other applications include NodeJS, which is used for handling WebSocket connections and all CRUD operations of the app. NodeJS's non-blocking, event-driven architecture makes it highly efficient for real-time communications and scalable network applications. These components collectively ensure that the application runs smoothly, can scale to meet varying demands, and provides a responsive user experience.

Connectivity Requirements

The application has specific connectivity requirements to ensure proper functionality and seamless operation.

An active internet connection is required for the application's functionality, particularly for real-time features such as user interactions and data synchronization. Additionally, an internet connection is necessary for software updates and maintenance.

The application requires access to local networks for internal communications between microservices and databases. It relies on certain network services and protocols, such as HTTP/HTTPS for web interactions and WebSocket for real-time communications. Proper network configuration is essential to ensure the app's performance and reliability.

Furthermore, the application requires a static IP address with all domain names pointed to it. For cloud deployments, it is recommended to use Cloudflare as a proxy between the user and the cloud instance to enhance security and performance. These connectivity requirements ensure that the application runs smoothly, maintaining high performance and security.

Performance Requirements

The application must meet certain performance criteria to ensure a smooth user experience.

The application should have a maximum page load time of 1-2 seconds. Key functions, such as user interactions and data retrieval, should have a response time of less than 200-500 milliseconds depending on the endpoint. The exception is for file upload endpoints, where performance is expected to have a linear relationship between the user's internet speed and file size.

Scalability is crucial for the application. It should be able to handle an increased volume of users and data. This includes the ability to scale horizontally by adding more servers and vertically by enhancing the capacity of existing servers. Ensuring these performance criteria are met will help maintain a responsive and efficient user experience.

Security and Privacy

Ensuring the security and privacy of user data is paramount for the application.

The application must implement data encryption both at rest and in transit to ensure security. User authentication should be enforced using secure methods such as JSON Web Tokens (JWT), providing robust protection against unauthorized access.

To meet privacy requirements, the application must comply with personal data protection laws and regulations such as the General Data Protection Regulation (GDPR). This includes ensuring user data is stored securely, providing users with control over their data, and being transparent about data usage practices. Adhering to these standards will help protect user information and maintain user trust.

In summary, the application is designed to leverage modern hardware and software platforms to achieve high performance, scalability, and efficient resource utilization. This setup ensures the application can handle significant loads and provide a responsive user experience across various deployment scenarios while maintaining security and privacy standards.

3.2 Description of the main functions of the application

The Lunarix University management application is designed to streamline and enhance the educational experience for students, professors, and administrators. The application leverages modern technologies to deliver a robust and efficient system supporting various academic and administrative tasks. Here, we describe the application's main functions and then provide an in-depth look at the code for creating assignments with file uploads. For a more in-depth look at the code use Appendix 8: Public GitHub repository link.

The application includes robust user management features, enabling registration, profile management, and role-based access control. These capabilities ensure efficient account management for students, professors, and administrators, allowing each user to access the functionalities relevant to their role.

Professors can create and manage courses, upload course materials, create assignments, and communicate with students. Students can access materials, submit assignments, and view grades. This feature streamlines the entire course lifecycle, making it easier for professors to manage their courses and for students to stay organized.

Professors can create assignments and grade submissions, while students can view assignments, submit their work, and track their grades. This functionality ensures that the process of assigning, submitting, and grading coursework is efficient and transparent.

The application includes a messaging system that facilitates group communication and real-time chat functionalities. This feature enhances collaboration and communication between students and professors, supporting a more interactive learning environment.

Administrators have access to a suite of tools for managing users, generating reports, configuring system settings, and meeting university requirements. These tools help administrators maintain the system and ensure it meets the institution's needs.

Creating an Assignment with File Uploads

One of the key features of the Lunarix application is the ability to create assignments that include file uploads. This feature involves multiple components and steps, from the frontend Angular page to the backend services handling the assignment creation and file storage. Below, we describe this process in detail and provide code snippets to illustrate how it works.

This is the code breakdown for the Example: Handling Class Assignments Files, which can be found in 2.8 of the thesis.

Frontend: Angular Component

The `CreateAssignmentPageComponent` handles the user interface for creating assignments. Professors can input assignment details and upload files.

In the `ngOnInit` method, the component initializes the class ID and due date

```
ngOnInit(): void {
    this.classId =
parseInt(this._activatedRoute.parent.parent.snapshot.paramMap.get('classI
d'));
    this.currentDate = new Date();
    this.dueDate = this._datePipe.transform(this.currentDate, 'YYYY-MM-
ddTHH:mm:ss');
}
```

This snippet initializes the component by fetching the class ID from the route parameters and setting the current date and due date. The `DatePipe` is used to format the due date in a specific format. This setup ensures that the necessary data is prepared when the component loads, providing a smooth user experience. The formatted due date is crucial for ensuring that deadlines are accurately represented and managed within the application.

For handling file uploads, the `onFileUpload` method adds the uploaded file to the list:

```
onFileUpload(file: File): void {
    this.uploadedFiles.push(file);
}
```

Here, the method adds each uploaded file to the `uploadedFiles` array. This allows the component to keep track of all files that the user has selected for upload. This list is later used when the assignment is created to ensure all selected files are sent to the backend. Keeping an array of uploaded files helps manage multiple files seamlessly and provides a way to display them to the user for confirmation before submission.

The `onCreateClick` method validates inputs and sends the data to the backend:

```
var formData: FormData = new FormData();
formData.append('name', this.name);
formData.append('dueDate', String(new Date(this.dueDate).getTime()));
formData.append('description', this.description);
for (var i = 0; i < this.uploadedFiles.length; i++) {
    formData.append('files', this.uploadedFiles[i]);
}
```

This snippet prepares the form data for submission by appending the assignment details and the uploaded files to a `FormData` object. This `FormData` object is then used to send the data to the backend via an HTTP POST request. This structure allows for the efficient and organized submission of multiple pieces of data, including files. The use of `FormData` is essential for handling file uploads as it supports binary data transfer, ensuring that files are correctly and efficiently transmitted to the server.

Backend: Classes Microservice (MCS)

The backend route handles the assignment creation process, receiving data from the frontend, validating it, and storing the assignment details and files.

The route begins by defining the endpoint and middleware:

```
router.route('/create/assignment/:classId').post(upload.array('files'),
handleCheckProfessorMiddleware(loginMethodsObj, accountMethodsObj), async
(req: Request, res: Response) => {
```

This snippet sets up the route to handle POST requests for creating assignments. The `upload.array('files')` middleware is used to handle file uploads, and

`handleCheckProfessorMiddleware` ensures that the user has the necessary permissions. This setup is crucial for maintaining security and ensuring that only authorized users can create assignments. Middleware functions like these are vital for pre-processing requests and enforcing access control before reaching the main logic of the route.

The route performs validation to ensure all required fields are present:

```
if (!('name' in body && 'dueDate' in body && 'description' in body))
{
    responseObject = {
        succ: false,
        mes: 'Something went wrong',
        debugMes: 'Invalid number of POST parameters',
    };
    res.status(200).send(responseObject);
    return;
}
```

This snippet checks if the necessary fields (`name`, `dueDate`, and `description`) are present in the request body. If any of these fields are missing, an error response is sent back to the client. This validation step ensures that incomplete or incorrect data is not processed, which helps maintain data integrity. Validation steps like these prevent errors and ensure that the system only processes well-formed requests.

The route processes the uploaded files by sending them to the `Files MCS`:

```
var files = (req.files as Express.Multer.File[]) || [];
var filesIds: number[] = [];
if (files.length > 0) {
    filesIds = await filesMethodsObj.sendFiles(userId, [], files);
}
```

Here, the method extracts the files from the request and sends them to the `Files MCS` using the `sendFiles` method of the `FilesMethods` class. This step ensures that files are handled by the dedicated microservice, maintaining separation of concerns and improving system organization. By delegating file handling to a specialized service, the application ensures that file operations are secure and efficient.

Transactions and Consistency

The backend logic uses transactions to ensure data consistency. Transactions help maintain the integrity of data by ensuring that all related operations either complete successfully or fail together, preventing partial updates. However, this can lead to complexities when dealing with distributed systems, such as a microservices architecture.

When a file is sent to the `Files MCS`, a transaction is started which locks the `Files` table in the database and any table that has a link to the primary key of the `Files` table, such as `classAssignmentsFiles`. This means that the transaction needs to be committed on the `Files MCS` with the saving of the file before the `Classes MCS` starts its transaction to create the assignment and add the files to the locked table.

This order of operations is crucial because if the assignment creation in the `Classes MCS` results in an error, it can lead to inconsistencies between the files present on the server and the files referenced in the assignment. Specifically, the files uploaded in the request could be left as ghost files with no association with any assignment.

```
var transaction = await dbConnection.startTransaction();
try {
    var assigInsertSqlResult: NormalPacket = await
dbConnection.executeInTransaction<NormalPacket>(
    transaction,
    'INSERT INTO classesAssignments (classId, classAssigName,
classAssigDesc, dueDate) VALUES (?, ?, ?, ?)',
    [classId, name, description, new Date(dueDate)]
);
    for (var i = 0; i < filesIds.length; i++) {
        await
dbConnection.executeInTransaction<NormalPacket>(transaction, 'INSERT INTO
classesAssignmentsFiles (classAssigId, fileId) VALUES (?, ?)',
[assigInsertSqlResult.insertId, filesIds[i]]);
    }
    await dbConnection.commitTransactionAndClose(transaction);
} catch (err) {
    if (filesIds.length > 0) {
        await filesMethodsObj.deleteFiles(filesIds);
    }
    await dbConnection.rollbackTransactionAndClose(transaction);
}
```

```

        responseObject = {
            succ: false,
            mes: String(err),
        };
        res.status(200).send(responseObject);
        return;
    }
    responseObject = {
        succ: true,
    };
    res.status(200).send(responseObject);
    return;
}

```

To resolve this problem, we can take two approaches:

One approach is to move the constraint layer to the microservices. This involves removing the relational constraints from the MySQL database and handling them within the microservices. This approach aligns with microservices architecture principles, where each service is responsible for its own data integrity.

Another approach is to delete files on assignment creation error. If an error occurs while creating the assignment, the `Classes MCS` can send a request to delete the uploaded files. This approach relies on the deletion being successful to avoid ghost files.

By implementing one of these solutions, we can ensure that the system remains consistent and free of unassociated files, maintaining the integrity of the application and providing a seamless user experience.

```

if (filesIds.length > 0) {
    await filesMethodsObj.deleteFiles(filesIds);
}
await dbConnection.rollbackTransactionAndClose(transaction);
responseObject = {
    succ: false,
    mes: String(err),
};
res.status(200).send(responseObject);
return;

```

By implementing one of these solutions, we can ensure that the application remains consistent and free from orphaned files, maintaining the integrity of both the file storage and assignment data.

Secure Paths for Files MCS

The `Files MCS` ensures that files are securely stored and managed. Here is the file upload endpoint. Appendix 5: Source code for uploading file to secure path inside the Files microservice.

This snippet demonstrates how the `Files MCS` processes uploaded files. It validates the file type, generates a unique file name, saves the file to the server, and records the file details and permissions in the database. The use of a transaction ensures that all steps are completed successfully or rolled back if any step fails, maintaining data consistency and integrity. The detailed handling of file permissions ensures that access controls are properly enforced for each file.

The `Files MCS` includes a secure endpoint for deleting files to ensure that only authorized microservices within the Kubernetes (K8S) network can request file deletions. This helps maintain the integrity and security of the file storage system. The secure endpoint for deleting files is defined as follows:

```
secureRouter.post('/delete-files', async (req: Request, res: Response)
=> {
  interface CurrentBody {
    filesIds: number[];
  }
  var responseObject: CustomResponseObject;
  var body: CurrentBody = req.body;
  console.log(body);
  if (!('filesIds' in body)) {
    responseObject = {
      succ: false,
      mes: 'Something went wrong',
      debugMes: 'Invalid number of POST parameters',
    };
  };
  res.status(200).send(responseObject);
  return;
```

```

    }
    var fileIds: number[] = body.fileIds;
    var storedFileNameSqlResult: SelectPacket;
    for (var i = 0; i < fileIds.length; i++) {
        storedFileNameSqlResult = await
dbConnection.execute<SelectPacket>('SELECT storedFileName FROM files
WHERE fileId = ?', [fileIds[i]]);
        if (storedFileNameSqlResult.length != 1) {
            continue;
        }
        await dbConnection.execute<NormalPacket>('DELETE FROM files WHERE
fileId = ?', [fileIds[i]]);
        await dbConnection.execute<NormalPacket>('DELETE FROM
filesPermissions WHERE fileId = ?', [fileIds[i]]);
        await fs.promises.unlink(path.join('/files',
storedFileNameSqlResult[0].storedFileName));
    }
    responseObject = {
        succ: true,
    };
    res.status(200).send(responseObject);
    return;
});

```

The secure endpoint ensures only microservices within the internal K8S network can delete files, maintaining the integrity of the file management system. Request validation checks for the required `fileIds` parameter, preventing malformed requests and potential system errors. File deletion involves fetching the stored file name, deleting the file record from the `files` and `filesPermissions` tables, and removing the physical file using `fs.promises.unlink`. This ensures complete removal of file references and the file itself. Error handling skips non-existent file IDs and continues processing, providing robust deletion operations and accurate feedback. This endpoint is essential for preventing orphaned files, maintaining a clean storage system, and ensuring data integrity and system efficiency.

Server Communication Module

This snippet shows the `sendPostRequest` method, which sends HTTP POST or HTTP GET requests to other microservices. It constructs the request URL and headers, ensuring secure and efficient communication between services. This method is crucial for maintaining a

modular and decoupled architecture, where microservices can interact without direct dependencies. The use of server keys ensures that only authorized services can communicate, enhancing security. Appendix 7: Server Communication module source code.

Files Module

The files module is a wrapper for the server communication module:

```
export class FilesMethods {
  serverCommunicationObj: ServerCommunication;

  constructor(serverCommunicationObj: ServerCommunication) {
    this.serverCommunicationObj = serverCommunicationObj;
  }

  async deleteFiles(filesIds: number[]): Promise<void> {
    try {
      interface CurrentBody {
        filesIds: number[];
      }
      var body: CurrentBody = {
        filesIds: filesIds,
      };
      await this.serverCommunicationObj.sendPostRequest(FILE_MCS_NAME,
'/files/delete-files', body);
    } catch (err) {
      console.log(err);
    }
  }

  async sendFiles(ownerUserId: number, filePermissions: FilePermission[],
files: Express.Multer.File[]): Promise<number[]> {
    var response: CustomResponseObject;
    var formData: FormData = new FormData();
    try {
      formData.append('filePermissions',
JSON.stringify(filePermissions));
      formData.append('ownerUserId', String(ownerUserId));
      for (var i = 0; i < files.length; i++) {
        formData.append(
          'files',
```

```

        new Blob([files[i].buffer], {
            type: files[i].mimetype,
        }),
        files[i].originalname
    );
}
response = await
this.serverCommunicationObj.sendPostRequest(FILE_MCS_NAME,
'/files/upload-files', formData, null, 80, false);
    if (response.succ) {
        return response.data.filesIds;
    }
} catch (err) {
    console.log(err);
}
return [];
}
}

```

This snippet illustrates how the `FilesMethods` class uses the server communication module to send file data to the `Files MCS`. It creates a `FormData` object, appends the files and metadata, and sends them using the `sendPostRequest` method. This encapsulation simplifies file operations and maintains a clean separation of concerns. By abstracting the file operations into a dedicated method, the application ensures that file handling is consistent and manageable.

The `deleteFiles` function sends a POST request to the secure endpoint with the list of file IDs to be deleted. It handles any errors that occur during the request, ensuring that the system can manage file deletions effectively. This function is crucial for maintaining consistency and cleanliness in the file storage system, particularly when handling errors during assignment creation or other file-related operations.

The Lunarix university management application's architecture and functionality are designed to provide a seamless and efficient experience for all users. By leveraging modern technologies and a microservices architecture, the application ensures scalability, maintainability, and security.

The example of creating an assignment with file uploads demonstrates the intricate yet efficient communication between frontend components, backend services, and specialized microservices, showcasing the system's robustness and reliability. This breakdown illustrates how each component works together to handle complex operations securely and efficiently, reflecting the application's overall design principles. By addressing the challenges of transactional consistency, the application can maintain data integrity and prevent issues such as ghost files, ensuring a robust and reliable system.

Handling Internal Communications for user authentication

Now, we will examine an example of internal communications between the MCS and the Login MCS. This section focuses on how the system ensures secure and efficient handling of authentication processes by centralizing JWT validation within the Login MCS. By leveraging internal API calls between the microservices, the system maintains robust security measures and optimizes resource usage, contrasting with the previous example of handling class assignments file uploads. This method highlights the efficiency and security achieved through internal communication protocols.

This is the code breakdown for the Example: Handling Internal Communications for JWT Validation, which can be found in 2.8 of the thesis.

Important Parts of the Login System Code

The ``LoginMethods`` class, central to the login system in the ``index.ts`` file, is designed to manage various authentication tasks, acting as a wrapper for the ``ServerCommunication`` class. This setup ensures that the login system is robust and secure. Here are the key aspects and functionalities of this system:

Initialization of ``LoginMethods`` Class

The ``LoginMethods`` class is initialized with several parameters, including JWT cookie settings and the ``ServerCommunication`` object. The constructor sets default values and initializes the JWT methods if not provided. This initialization process ensures that the class has all necessary configurations to manage JWTs and communicate with the server efficiently. Appendix 7: Server Communication module source code.

```

export class LoginMethods {
    constructor(
        jwtCookieName: string = 'login-token',
        jwtCookieDomain: string = '',
        jwtExpInDays: number = 30,
        sessionExpInMin: number = 30,
        serverCommunicationObj: ServerCommunication,
        jwtMethodsObj: JwtMethods = null
    ) {
        this.jwtCookieName = jwtCookieName;
        this.jwtCookieDomain = jwtCookieDomain;
        this.sessionExpInMin = sessionExpInMin;
        this.jwtExpInDays = jwtExpInDays;
        this.serverCommunicationObj = serverCommunicationObj;
        if (jwtMethodsObj == null) {
            jwtMethodsObj = new JwtMethods();
        } else {
            this.jwtMethodsObj = jwtMethodsObj;
        }
    }
}

```

JWT Handling Methods

The class includes methods to handle JWT creation, checking expiration, and setting cookies. These methods ensure that JWTs are correctly generated, validated, and managed within the user's session. For instance, the `calculateJwtExp` method calculates the expiration time for a JWT, ensuring tokens are valid for a specified duration. The `getJwt` and `setJwt` methods are used to retrieve and store JWTs in cookies, respectively, while the `clearJwt` method removes the JWT, effectively logging out the user.

```

calculateJwtExp(): number {
    var currentDate: number = new Date().getTime();
    return currentDate + this.jwtExpInDays * 24 * 60 * 60 * 1000;
}

getJwt(req: Request): string {
    return req.cookies[this.jwtCookieName];
}

setJwt(res: Response, jwt: string): void {
    var isSecureCookieEnabled: boolean = true;

```

```

        res.cookie(this.jwtCookieName, jwt, { domain: this.jwtCookieDomain,
expires: new Date(this.calculateJwtExp()), httpOnly: true, secure:
isSecureCookieEnabled });
    }

clearJwt(res: Response): void {
    var isSecureCookieEnabled: boolean = true;
    res.cookie(this.jwtCookieName, '', { domain: this.jwtCookieDomain,
expires: new Date(), httpOnly: true, secure: isSecureCookieEnabled });
}

```

Updating JWT Last Used Time

This method updates the last used time of the JWT in the database. It ensures that each token's usage is tracked, enhancing the security and management of active sessions.

```

updateJwtLastTimeUsed(dbConnection: DbHandler, loginId: number):
Promise<NormalPacket> {
    return dbConnection.execute<NormalPacket>('UPDATE logins SET
lastUsedTime = ? WHERE loginId = ?', [new Date(), loginId]);
}

```

Generating Final Login

The `generateFinalLogin` method handles the final steps of generating a login, including inserting the login into the database and creating the JWT. This method ensures that all necessary steps are taken to authenticate the user and generate a valid session.

```

async generateFinalLogin(dbConnection: DbHandler, req: Request, res:
Response, userId: number, deviceType: DeviceTypes, deviceToken: string,
isLocal: boolean = false): Promise<string | void> {
    var sql: string = 'INSERT INTO logins (userId, deviceType,
deviceToken, createdAt, lastUsedTime) VALUES (?, ?, ?, ?, ?)';
    var values: any[] = [userId, deviceType, deviceToken, new Date(), new
Date()];
    var normalSqlResult: NormalPacket = await
dbConnection.execute<NormalPacket>(sql, values);
    var loginId: number = normalSqlResult.insertId;
    var jwtToken: string;
    var createJwtTokenResponse: CustomResponseObject;
    if (isLocal) {

```

```

        jwtToken = this.loginMethodsObj.jwtMethodsObj.createJwt(loginId);
    } else {
        createJwtTokenResponse = await
this.loginMethodsObj.serverCommunicationObj.sendPostRequest(LOGIN_MCS_NAME, this.loginMethodsObj.createJwtUrlPath, {
            loginId: loginId,
        });
        if (!createJwtTokenResponse.succ) {
            throw new Error(createJwtTokenResponse.debugMes ||
createJwtTokenResponse.mes);
        }
        jwtToken = createJwtTokenResponse.data.jwt;
    }
    await this.createLoginObject(userId, loginId);
    if (res == null) {
        return jwtToken;
    }
    this.loginMethodsObj.setJwt(res, jwtToken);
}

```

Validating Session and JWT

A crucial part of the login process is validating the session and the JWT. The system first checks if the session is valid by querying the database to see if the `loginId` exists and is active. Only if this check fails does the system proceed to decrypt and validate the JWT. This approach optimizes resource usage by avoiding unnecessary JWT decryption when the session is already known to be valid.

```

async checkLoginObject(): Promise<boolean> {
    if (!(await this._checkIfLoginObjectExists())) {
        return false;
    }
    var iss: number = await
this.sessionInterfaceObj.get(this.loginMethodsObj.sessionLoginPath.concat(
'iss'));
    if (iss + this.loginMethodsObj.sessionExpInMin * 60 * 1000 <= new
Date().getTime()) {
        await this.deleteLoginObject();
        return false;
    }
}

```

```

        await this._updateLoginObject();
        return true;
    }

    async validateLogin(req: Request, res: Response): Promise<boolean> {
        const jwt = this.getJwt(req);
        if (!jwt) {
            return false;
        }

        const loginData = this.jwtMethodsObj.decodeJwt(jwt);
        if (!loginData) {
            return false;
        }

        const { loginId, userId } = loginData;
        const isSessionValid = await this.isSessionValid(dbConnection,
loginId);

        if (isSessionValid) {
            await this.updateJwtLastTimeUsed(dbConnection, loginId);
            return true;
        } else {
            const isJwtValid = this.jwtMethodsObj.verifyJwt(jwt);
            if (isJwtValid) {
                await this.updateJwtLastTimeUsed(dbConnection, loginId);
                return true;
            } else {
                this.clearJwt(res);
                return false;
            }
        }
    }
}

```

Checking Login ID and JWT

The `secure-paths.ts` file contains routes for checking the login ID and JWT, demonstrating the server's process for session validation before JWT decryption. This ensures resource efficiency and secure handling of sessions and tokens.

Checking Login ID

The `check-login-id` route verifies if the provided `loginId` and `userId` exist in the database. If they are valid, the server updates the last used time for the JWT.

```
secureRouter.post('/check-login-id', async (req: Request, res: Response)
=> {
    var body: CurrentBody = req.body;
    if (!('loginId' in body && 'userId' in body)) {
        responseObject = {
            succ: false,
            debugMes: 'Invalid number of POST parameters',
        };
        res.status(200).send(responseObject);
        return;
    }
    var loginId: number = body.loginId;
    var userId: number = body.userId;
    var selectSqlResult: SelectPacket = await
dbConnection.execute<SelectPacket>('SELECT userId FROM logins WHERE
loginId = ? AND userId = ?', [loginId, userId]);
    if (selectSqlResult.length != 1) {
        responseObject = {
            succ: false,
            debugMes: 'The loginId is not present in the database for the
given userId',
        };
        res.status(200).send(responseObject);
        return;
    }
    responseObject = {
        succ: true,
    };
    await loginMethodsObj.updateJwtLastTimeUsed(dbConnection, loginId);
    res.status(200).send(responseObject);
    return;
});
```


Checking JWT

The `check-jwt` route verifies the JWT by decoding it and checking its validity. If the JWT is valid, the server updates the last used time for the session.

```
secureRouter.post('/check-jwt', async (req: Request, res: Response) => {
  var body: CurrentBody = req.body;
  if (!('jwt' in body)) {
    responseObject = {
      succ: false,
      debugMes: 'Invalid number of POST parameters',
    };
    res.status(200).send(responseObject);
    return;
  }
  var jwt: string = body.jwt;
  var loginId: number = 0;
  var userId: number;
  var lastUsedTime: number;
  var sql: string;
  var value: any[];
  var selectSqlResult: SelectPacket;
  try {
    loginId = jwtMethodsObj.verifyJwt(jwt);
  } catch (err) {
    console.log(err);
  }
  if (loginId == 0) {
    responseObject = {
      succ: false,
      debugMes: 'The JWT is invalid',
    };
    res.status(200).send(responseObject);
    return;
  }
  sql = 'SELECT userId, lastUsedTime FROM logins
  WHERE loginId = ?';
  value = [loginId];
  selectSqlResult = await dbConnection.execute<SelectPacket>(sql,
  value);
```

```

    if (selectSqlResult.length != 1) {
        responseObject = {
            succ: false,
            debugMes: 'The JWT is not present in the database',
        };
        res.status(200).send(responseObject);
        return;
    }
    userId = selectSqlResult[0].userId;
    lastUsedTime = selectSqlResult[0].lastUsedTime.getTime();
    if (loginMethodsObj.checkIfJwtExpired(lastUsedTime)) {
        responseObject = {
            succ: false,
            debugMes: 'The JWT is expired',
        };
        res.status(200).send(responseObject);
        return;
    }
    await loginMethodsObj.updateJwtLastTimeUsed(dbConnection, loginId);
    responseObject = {
        succ: true,
        data: {
            userId: userId,
            loginId: loginId,
        },
    };
    res.status(200).send(responseObject);
});

```

Security Considerations

A critical security measure in this system is that only the Login Microservice (MCS) has access to the public and private JWT keys. This design choice means that only the Login MCS can validate the JWT, significantly enhancing security by limiting access to these keys. If every microservice had access to the JWT keys, the risk of a security breach would increase because the compromise of any microservice would expose the keys. By centralizing JWT validation in the Login MCS, the system reduces the attack surface and confines the most sensitive operations to a single, more easily secured component.

This approach mitigates the security risk of having the keys in every microservice by ensuring that JWT validation is handled in a controlled and secure manner. It also simplifies the management of JWTs and reduces the potential for key exposure, thereby maintaining a higher level of overall system security.

In conclusion, the example of handling internal communications between the MCS and the Login MCS further underscores the intricate yet efficient interactions within the system. This method of centralizing JWT validation and authentication processes in a dedicated microservice highlights the system's robustness and reliability. By ensuring secure and optimized resource usage, the application demonstrates its ability to manage complex operations effectively.

Additionally, the example of creating an assignment with file uploads illustrates the seamless communication between frontend components, backend services, and specialized microservices. This breakdown showcases how each component works together to handle complex operations securely and efficiently, reflecting the application's overall design principles. By addressing the challenges of transactional consistency, the application can maintain data integrity and prevent issues such as ghost files, ensuring a robust and reliable system. Together, these examples demonstrate the comprehensive and well-thought-out design that underpins the application's success.

CHAPTER 4. *The efficiency and usefulness of the IT application*

4.1 *Conditions regarding the implementation of the application*

Development Process from Code to Production

The development process of the Lunarix application encompasses several stages, from initial coding to deployment on a production Kubernetes (K8S) cluster. This process ensures that the application is developed efficiently, thoroughly tested, and deployed reliably.

Development Environment with Nodemon and Node.js

During the development phase, the Lunarix application is built using Node.js and Express.js for its backend services. Developers utilize TypeScript for its robust typing and compile-time error-checking features. The development environment is set up to use Nodemon, a tool that automatically restarts the Node.js server whenever it detects changes in the code. This setup allows for real-time compilation of TypeScript into JavaScript, enabling developers to see the effects of their changes instantly, thus speeding up the development process.

Setting up the development environment involves installing Node.js, TypeScript, Nodemon, and other necessary development tools. This ensures a fully configured environment that supports real-time code compilation and server restarts. The initial code development focuses on core features such as user authentication, class management, and grading systems, resulting in a functional prototype with essential features implemented.

Testing and Validating the Code

Testing and validation are critical to ensuring the reliability and performance of the Lunarix application. Various testing methodologies are employed throughout the development and deployment process:

- **Unit Testing:**
 - Ensures individual components function correctly.
 - Each unit test is designed to test a specific part of the application's functionality, providing immediate feedback to developers.
- **Integration Testing:**
 - Validates that different modules and services interact correctly.
 - This testing phase involves combining individual modules and testing them as a group to ensure they work together seamlessly.
- **Performance Testing:**
 - Measures the application's responsiveness and stability under different conditions.
 - Performance testing helps identify potential bottlenecks and areas for optimization.
- **Security Testing:**
 - Identifies vulnerabilities and ensures data protection mechanisms are robust.
 - This includes penetration testing and code reviews focused on security aspects.
- **Usability Testing:**
 - Ensures the application is user-friendly and meets user expectations.
 - The usability testing feedback is used to inform the essential changes made to enhance the user experience.
- **Regression Testing:**
 - Ensures that new updates do not negatively affect existing functionalities.
 - This is performed continuously to maintain the integrity of the application throughout its lifecycle.

Building and Containerizing the Application

Once the development phase reaches a stable state, the next step is to prepare the application for production. This involves building the application and creating Docker images for each microservice. Docker provides a consistent environment for the application, ensuring it runs the same way regardless of where it is deployed.

Containerization with Docker involves writing Dockerfiles for each microservice, specifying the environment and dependencies required. Docker images encapsulate the application's environment and dependencies, making it easy to build production-ready images from these Dockerfiles.

Deployment to Kubernetes (K8S)

The final stage is deploying the Docker images to a Kubernetes (K8S) cluster. Kubernetes automates the deployment, scaling, and management of containerized applications, ensuring high availability and scalability. Configuring the Kubernetes cluster involves setting up nodes and networking, resulting in a fully operational K8S cluster ready for application deployment. Deploying Docker images to K8S includes creating Kubernetes deployment files and services, followed by deploying the Docker images, leading to the Lunarix application running in a K8S cluster. Appendix 4: Kubernetes diagram.

Maintenance and Support Plan

Maintaining and supporting the Lunarix application is crucial for its long-term success. A well-defined maintenance and support plan ensures that the application remains functional, secure, and up to date.

- **Regular Updates and Patches:**
 - Implement a schedule for regular updates to address bugs, security vulnerabilities, and performance issues.
 - Ensure that updates are thoroughly tested before deployment to minimize disruptions.
- **Monitoring and Incident Management:**
 - Continuously monitor the application's performance and availability.
 - Establish an incident management process to quickly address and resolve any issues that arise.
- **Backup and Recovery:**
 - Implement a robust backup strategy to protect data and ensure quick recovery in case of data loss or corruption.
 - Regularly test backup and recovery procedures to ensure their effectiveness.

- **User Support:**

- Provide a dedicated support team to assist users with technical issues and inquiries.
- Offer multiple channels for support, including email, phone, and an online helpdesk.

- **Documentation:**

- Maintain comprehensive documentation for developers and users to facilitate troubleshooting and ensure smooth operation.
- Regularly update documentation to reflect changes and new features.

Training and Documentation

Effective training and documentation are essential for maximizing the benefits of the Lunaris application and ensuring user proficiency.

- **User Training Programs:**

- Develop training programs tailored to different user groups, including students, professors, and administrators.
- Offer training sessions, webinars, and workshops to help users understand and utilize the application's features.

- **Comprehensive Documentation:**

- Provide detailed user manuals, FAQs, and online resources that cover all aspects of the application.
- Include step-by-step guides, screenshots, and video tutorials to enhance understanding.

- **Developer Documentation:**

- Create thorough documentation for developers, including API references, code samples, and integration guides.
- Ensure that the documentation is kept up-to-date with the latest changes and best practices.

- **Feedback and Improvement:**

- Continuously gather feedback from users to identify areas for improvement in training and documentation.
- Update training materials and documentation based on user feedback and evolving needs.

Implementing the Lunarix application from development to production involves a well-structured process that ensures efficiency, reliability, and scalability. By using tools like Nodemon for development, Docker for containerization, and Kubernetes for deployment, the Lunarix application is well-equipped to meet the needs of its users and maintain high performance and availability in a production environment. Regular testing, a robust maintenance and support plan, and comprehensive training and documentation are essential components that contribute to the application's long-term success and user satisfaction.

4.2 Computer application efficiency considerations

App Performance

Adding an announcement to the home page of the app takes approximately 98 milliseconds. This involves transferring 744 KB of data, typically containing the announcement's text, and is completed with a single request. This demonstrates the app's ability to handle quick data transactions efficiently.

The total load time for the home page of the dashboard is 992 milliseconds, with a data transfer size of 5.1 MB spread across 57 requests. This illustrates the app's capability to load complex pages with multiple assets promptly, ensuring a smooth user experience. All the data was extracted from Appendix 2: Waterfall of requests for loading the home page of the dashboard.

The app utilizes a microservices (MCS) architecture as detailed in 2.6 Kubernetes (K8S) manages the containers, each having its Horizontal Pod Autoscaler (HPA) file. This setup allows the deployment to automatically scale based on resource utilization. Databases are the exception and do not use HPA files, but the overall system ensures efficient resource management and scalability.

Each MCS returns a JSON response containing a "succ" parameter. If true, the request was successful. If false, the error message is retrieved from "mes" or "debugMes" parameters. For HTTP 500 errors, Kubernetes restarts the affected pod, and a generic error message "Something went wrong" is displayed to the user. This robust error handling mechanism ensures minimal disruption in service.

Currently, the app lacks an in-built user feedback system. Users can report performance issues via the contact email provided by Lunarix. This indicates an area for potential improvement to enhance user engagement and satisfaction.

Resource Utilization

The app has not been tested under peak conditions; however, the development server, which runs a single pod per MCS, demonstrates the potential for scalability. With Kubernetes, pods can scale across multiple machines (K8S cluster nodes), allowing the app to utilize available resources effectively. This setup ensures the app remains responsive under varying loads.

The app can be hosted on any cloud provider supporting Kubernetes. Storage used by pods can either be local machine storage or cloud storage buckets, adaptable via persistent volume (pv) and persistent volume claim (pvc) files. This flexibility enables the app to leverage cloud infrastructure efficiently.

The app uses Redis and MySQL databases. MySQL version 8 offers significant performance improvements and better resource allocation compared to version 5.7. Version 8 provides enhanced query performance, improved JSON handling, and better overall scalability. Detailed benchmarks show that MySQL 8.0 achieves up to 1 million QPS (Queries Per Second) for read-only workloads. (Oracle, 2024)

Redis demonstrates exceptional performance as well. Benchmarking tests show that Redis can handle over 1 million requests per second for simple operations such as SET and GET commands. For example, with pipelining, Redis achieves approximately 403,063 requests per second for SET operations and 508,388 requests per second for GET operations. These numbers highlight Redis's capability to manage high-throughput scenarios efficiently. For more detailed benchmarks, you can refer to Redis Performance Benchmarks. (Redis, 2024)

Network speed and latency depend on the user's location and the cloud provider's coverage. As the app is locally hosted, latency is minimal. In real-world deployments, it would vary based on the user's proximity to the data center. This adaptability ensures consistent performance across different geographical regions.

The K8S control plane node (also acting as the worker node in development) allocates CPU and memory resources using the "resources" keyword in the deployment configuration file. This ensures each pod receives a specified minimum and maximum amount of resources, preventing a single pod from consuming all server resources. This careful allocation prevents resource bottlenecks and maintains optimal performance.

Performance and Resource Usage with Rust vs. NodeJS

Switching the app's MCS from NodeJS to Rust, except for the socket server, would significantly impact performance and resource usage. Rust generally provides better performance for CPU-intensive tasks due to its system-level control over memory management and concurrency. Rust's performance for HTTP servers typically outperforms NodeJS, resulting in lower response times and higher throughput.

According to benchmarks from an article comparing NodeJS and Actix Web (a Rust framework), Actix Web can handle significantly more requests per second. Actix Web achieved approximately 60,000 requests per second, whereas NodeJS handled around 20,000 requests per second under similar conditions. This indicates that Rust can provide about three times the throughput of NodeJS for HTTP server tasks. (Maxim, 2020)

NodeJS remains preferred for handling WebSocket connections due to its non-blocking, event-driven architecture, making it highly efficient for real-time communications. Comparative benchmarks demonstrate that while Rust offers higher throughput and lower latency for HTTP servers, NodeJS excels in managing multiple simultaneous socket connections.

These insights highlight that using Rust for most microservices, except for socket servers where NodeJS excels, could lead to improved overall app performance and more efficient resource utilization. For detailed benchmarks and a deeper comparison, refer to the article [Performance of Node.js Compared to Actix Web](#).

CHAPTER 5. *Discussion and subsequent works*

In this chapter, we will discuss several important aspects of our system, including the anticipated results and conclusions, assessing the potential successes and limitations, user feedback, further improvements and functionality, and deployment plans. Additionally, we will cover the future features planned for implementation, their corresponding database design considerations, as well as other ways to improve the performance of the app in the future.

Discussion of Anticipated Results and Conclusions

The implementation of the planned features is expected to significantly enhance the overall functionality and user experience of the application. The core functionalities are designed to be well-received, ensuring stability and reliability in performance. The integration of existing tools and a seamless user experience should contribute positively to the daily operations of future users.

The anticipated impact of these features is substantial in the context of the application and its initial objectives. By addressing the core needs of both administrators and students, these features are expected to contribute to a more efficient, organized, and user-friendly system. The real-time capabilities and integration with existing tools will ensure that the application remains relevant and valuable in a modern educational environment.

Assessing Potential Successes and Limitations

The application is expected to achieve several successful implementations that will enhance its functionality. The user interface is designed to be intuitive and easy to navigate, making it accessible for users of varying technical proficiency. Core features, such as class management, user roles, and basic file sharing, are anticipated to be robust and reliable, contributing to improved productivity and organization within educational institutions.

Despite these anticipated successes, some potential limitations and challenges need to be considered. The system lacks advanced features essential for further enhancing user experience and functionality. For example, the absence of a comprehensive fee payment system and the ability to manage student groups more efficiently have been identified as

significant gaps. Users are also likely to express the need for more robust file management and direct communication tools within the application.

Scalability is another concern as the user base grows. The existing infrastructure may face challenges in handling a larger volume of users and data, requiring proactive measures to ensure continued stability and performance. Additionally, there may be difficulties in integrating third-party tools and services, which can limit the system's flexibility and extendibility.

User Feedback

User feedback is expected to be instrumental in shaping the development of the application. Users are likely to appreciate the streamlined workflow and the intuitive interface. They will particularly value the core functionalities, such as class management and user roles, which are designed to simplify administrative tasks.

Further Improvements and Functionality

To better meet user needs and extend the application's capabilities, several features and enhancements are planned. Advanced reporting tools will be developed to help administrators track and analyze data more effectively. Additionally, a mobile app will be created to provide users with access to the system on the go.

We plan to implement several important features in the future. These include a fees payment system using Stripe and webhooks, the creation and management of student groups, an enhanced file system for classes, and a direct chat feature with file-sharing capabilities.

Fees Payment System Using Stripe and Webhooks

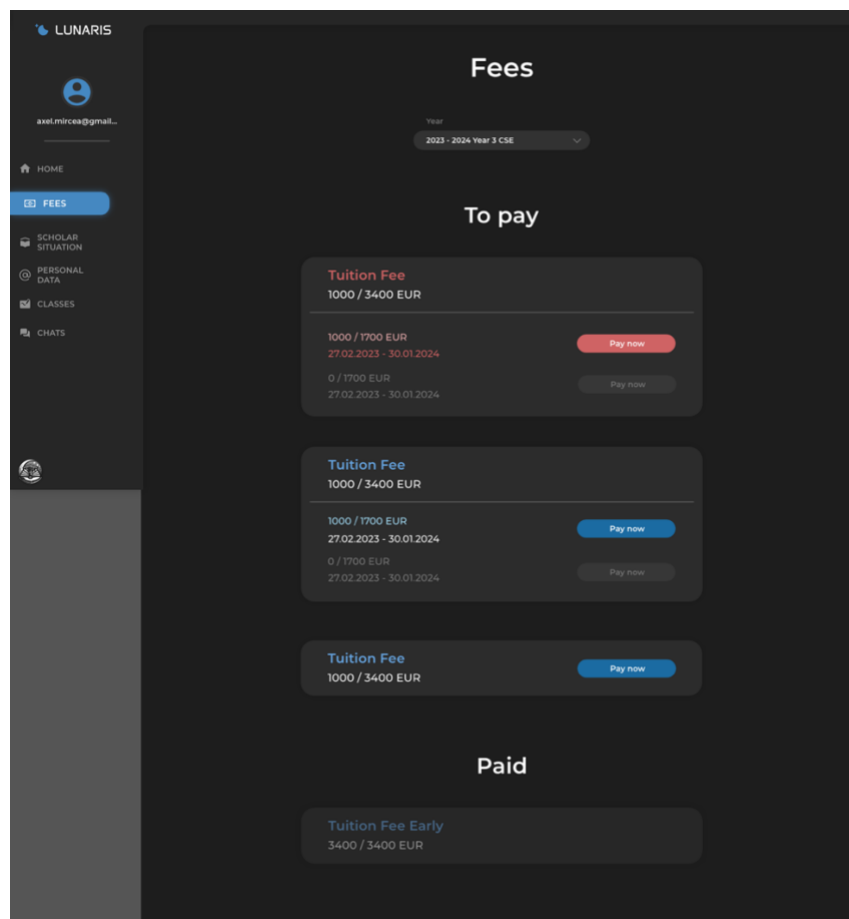
The fees payment system will allow for the collection of payments from students using Stripe. This system will be integrated with webhooks to handle real-time payment events. Administrators will be able to assign fees to students either individually or by group, with automatic generation of receipts and invoices powered by Stripe.

Database Design

The database design for this feature includes tables for storing fee structures, payment records, and payment events. The `fees` table will store fee details, the `payments` table will track payment statuses, and the `paymentEvents` table will log events from Stripe.

- **Fees:** This table will store information about the different types of fees, such as tuition or activity fees. Fields will include feeId, description, amount, and dueDate.
- **Payments:** This table will track payments made by students, including paymentId, userId, feeId, amountPaid, paymentDate, and paymentStatus.
- **PaymentEvents:** This table will log events from Stripe, such as successful payments, failed payments, and refunds. Fields will include eventId, paymentId, eventType, eventDate, and eventDetails.

Figure 5.1: Design of Fees feature in the app



Creation of Student Groups

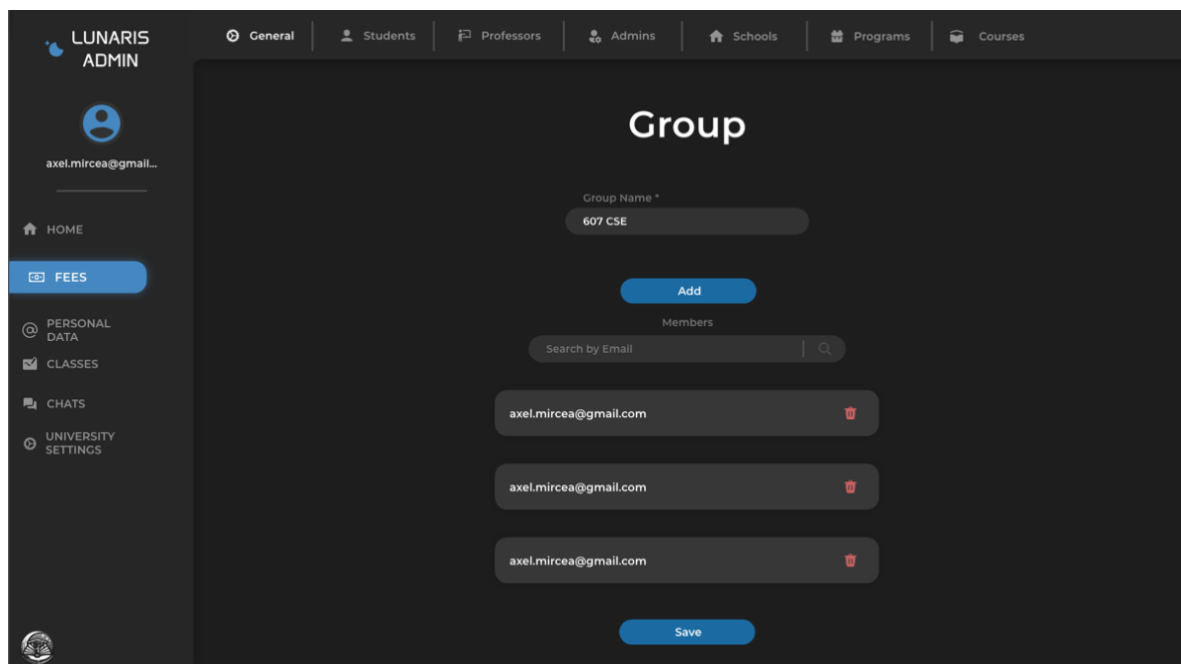
This feature will allow administrators to create and manage groups of students, streamlining various administrative tasks such as fee assignments and class enrollments.

Database Design

The `groups` table will store group details, and the `groupMemberships` table will link students to their respective groups. This design ensures efficient management and retrieval of group-related data.

- **Groups:** This table will store information about the groups, including groupId, groupName, and groupDescription.
- **GroupMembers:** This table will link students to groups, with fields such as groupMemberId, groupId, and userId.

Figure 5.2: Design of the Groups feature in the app



Class File System

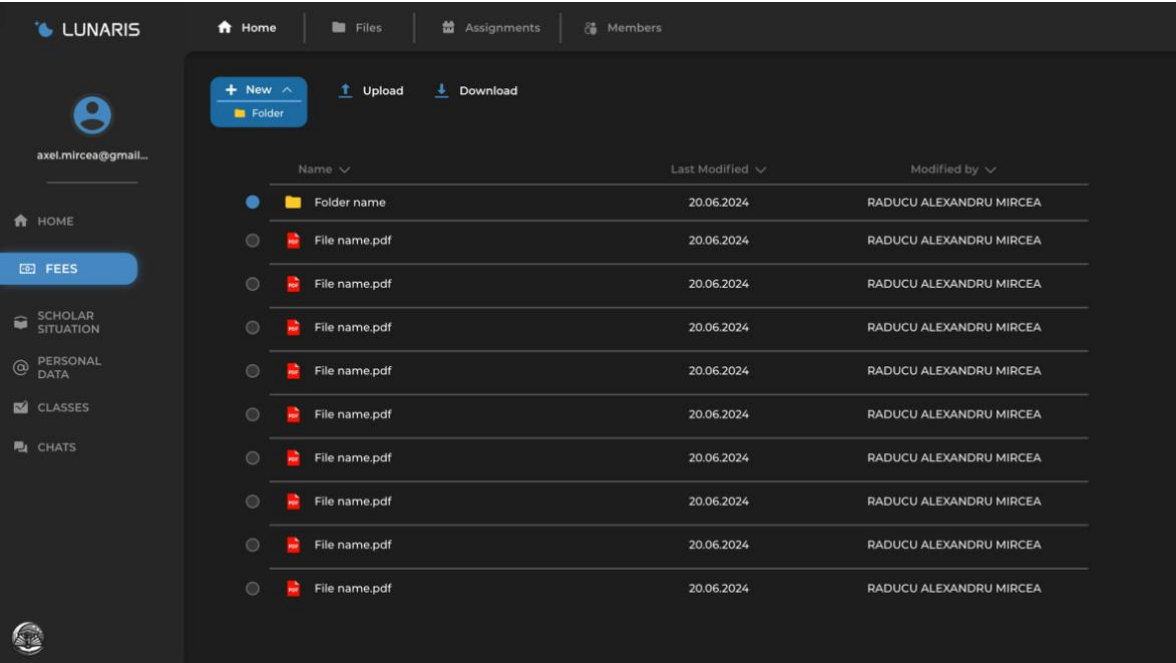
The class file system will be designed to resemble the file system used in Microsoft Teams, providing a familiar way to manage class-related documents.

Database Design

Tables such as `classes`, `classFiles`, and `filePermissions` will be created to store information about classes, associated files, and access permissions respectively. This structure supports secure and organized file management.

- **Classes:** This table will store information about the classes, including classId, className, and classDescription.
- **ClassFiles:** This table will store information about files associated with classes, including fileId, classId, fileName, filePath, and uploadDate.
- **FilePermissions:** This table will manage access permissions for files, with fields such as permissionId, fileId, userId, and permissionType.

Figure 5.3: Design of the Class file system feature in the app



Direct Chat with File Sharing

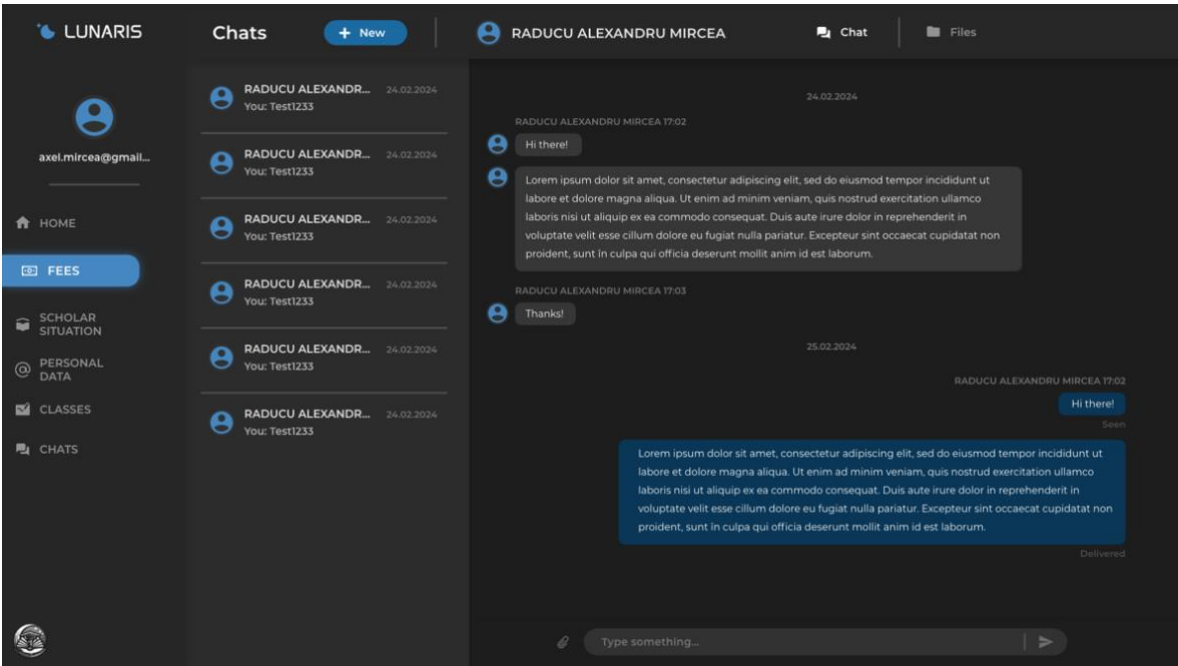
The direct chat feature will enable real-time communication between users, with the added functionality of sharing files within the chat interface.

Database Design

The `chats`, `messages`, and `chatFiles` tables will store chat sessions, messages, and shared files. This design ensures that all chat and file-sharing activities are efficiently managed.

- **Chats:** This table will store information about chat sessions, including chatId, chatName, and chatDescription.
- **Messages:** This table will store individual messages within a chat, with fields such as messageId, chatId, userId, messageText, and timestamp.
- **ChatFiles:** This table will manage files shared within chats, including fileId, chatId, userId, fileName, filePath, and uploadDate.

Figure 5.4: Design of the Direct Chat feature in the app



Potential Switch to Rust for HTTP Servers

To handle HTTP servers more efficiently, we are considering a switch to Rust. Rust is known for its performance and safety, which can help improve the robustness and scalability of our system. Detailed information about this potential switch can be found in 4.2. Switching to Rust could provide the system with greater stability and performance, which is crucial for handling increased user loads and complex operations.

Future Research

There are several avenues for further research related to the application. User behavior analysis can be conducted to study how users interact with the application and identify further areas for improvement. Exploring the use of artificial intelligence can enhance features such as personalized learning paths and automated administrative tasks. Collaborative research with educational institutions can provide deeper insights into their needs and help refine the application accordingly.

Deployment and Distribution Strategy

The deployment and distribution strategy for the application involves conducting a beta testing phase with a select group of users to identify and fix any issues before a wider release. Comprehensive training and support materials will be provided to ensure users can effectively use the new features. Marketing and promotion efforts will be undertaken to increase awareness and adoption of the application. This strategy ensures a smooth transition from development to full-scale deployment and addresses any potential issues early in the process.

Summary and Outlook

In summary, the presentation of these features and plans highlights the continuous efforts to enhance the application and meet user needs. The benefits of these improvements are clear, contributing to a more efficient, organized, and user-friendly system. By focusing on core functionalities and user feedback, the application is positioned to provide significant value to its users.

CHAPTER 6. *Conclusion*

In summary, the Lunarix application has the potential to achieve its primary objectives by developing a comprehensive university management system. The application aims to meet goals such as streamlining academic and administrative processes, including user and course management, assignment and grade handling, and communication tools.

The chosen socio-economic entity, the Romanian-American University (RAU), stands to benefit significantly from the implementation of Lunarix. RAU, a private educational institution based in Romania, aims to provide high-quality education through comprehensive academic programs that blend theoretical knowledge with practical skills. By fostering an environment that promotes learning, research, and professional development, RAU emphasizes the importance of educational excellence, innovation, and global cultural integration.

Furthermore, Lunarix aims to contribute to the field of educational technology by offering a scalable and flexible solution that addresses specific university management needs. By integrating modern technologies, such as microservices architecture and cloud computing, the application is designed to remain innovative and relevant. These technological choices are expected to enhance the system's performance and allow for future scalability and adaptability to evolving educational environments.

However, despite its potential strengths, the application currently has some limitations that need to be addressed. For instance, there is a need for more advanced file management and communication tools. Future development should focus on enhancing these areas, improving scalability, and integrating third-party services. Addressing these gaps will not only improve user satisfaction but also expand the application's functionality, making it more versatile for different educational contexts.

The expected impact of Lunarix on university management at RAU could be substantial. The application aims to improve the efficiency and quality of educational processes and can be adapted to various institutional needs. Compliance with data protection regulations, such as GDPR, is also a priority, ensuring the application's utility across different contexts. The

ability to handle sensitive data securely will be crucial in maintaining trust and reliability among users.

Reflecting on the development process so far, several challenges have been addressed, including managing microservices complexity and ensuring data consistency. The adoption of Kubernetes for container orchestration has played a significant role in overcoming these challenges, providing scaling, and management of containerized applications. Lessons learned emphasize the importance of user-centric design, robust security measures, and modern development practices. The iterative development approach, incorporating continuous feedback and testing, has been essential in refining the application to meet user expectations.

In conclusion, the Lunarix application has the potential to become a robust and efficient solution for university management, achieving its objectives and providing significant benefits with further development. The successful integration of modern technologies and user-focused design could result in a tool that enhances productivity and organization within educational institutions like RAU. Future projects should focus on continuous improvement, user feedback, and flexible technologies to enhance educational technology solutions further. Recommendations for future development include addressing identified limitations, expanding functionality, and maintaining a strong focus on user experience and security. By doing so, the Lunarix application can evolve and meet the dynamic needs of educational institutions, contributing to the advancement of educational technology and significantly supporting the mission of institutions like the Romanian-American University.

CHAPTER 7. Bibliography

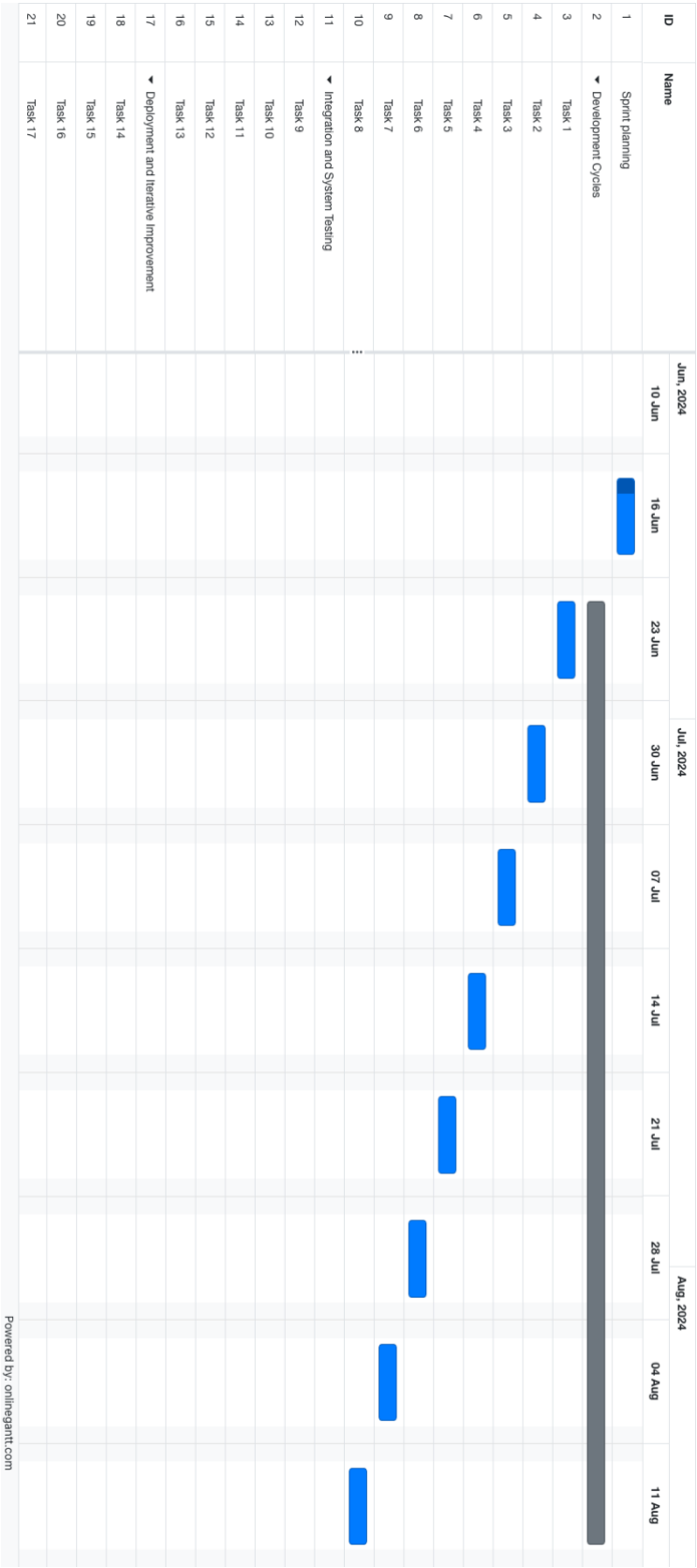
- Duckett, J. (2011). *HTML and CSS: Design and Build Websites*. Indianapolis, Indiana, United States: Wiley.
- Duckett, J. (2014). *JavaScript and JQuery: Interactive Front-End Web Development*. Indianapolis, Indiana, United States: Wiley.
- Grammarly. (n.d.). *Used for rewriting for better fluency and expressions*. Retrieved from Grammarly: <https://www.grammarly.com>
- Hoffman, A. (2020). *Web Application Security: Exploitation and Countermeasures for Modern Web Applications*. Sebastopol, California, United States: O'Reilly Media.
- IATA. (n.d.). *ATC RAU TRAINING CENTER*. Retrieved 1 2024, from IATA: <http://iata2.rau.ro>
- Kelsey Hightower, B. B. (2017). *Kubernetes Up & Running: Dive into the Future of Infrastructure*. Sebastopol, California, United States: O'Reilly Media.
- Kleppmann, M. (2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. Sebastopol, California, United States: O'Reilly Media.
- Krug, S. (2014). *Don't Make Me Think: A Common Sense Approach to Web Usability*. Berkeley, California, United States: New Riders.
- Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Boston, Massachusetts, United States: Prentice Hall.
- Maxim, V. (2020, 02 25). *ExpressJS vs Actix-Web. It is exactly what you think*. Retrieved 3 2024, from Medium: <https://medium.com/@maxsparr0w/performance-of-node-js-compared-to-actix-web-37f20810fb1a>
- MongoDB, Inc. (2024). *Database Management Systems*. Retrieved 03 2024, from MongoDB: <https://www.mongodb.com/resources/basics/database-management-system>
- Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. Sebastopol, California, United States: O'Reilly Media.
- OpenAI. (n.d.). *Used for rewriting for better fluency and expressions*. Retrieved from ChatGPT: <https://chatgpt.com>
- Oracle. (2024). *MySQL Benchmarks*. Retrieved 03 2024, from MySQL: <https://www.mysql.com/why-mysql/benchmarks/mysql/>
- Poulton, N. (2017). *Docker Deep Dive*. United States: Leanpub.
- Quillbot Inc. (n.d.). *Used for rewriting for better fluency and expressions*. Retrieved from Quillbot: <https://quillbot.com>
- Redis. (2024). *Redis benchmark*. Retrieved 03 2024, from Redis: https://redis.io/docs/latest/operate/oss_and_stack/management/optimization/benchmarks/
- Seshadri, S. (2018). *Angular: Up and Running: Learning Angular, Step by Step*. Sebastopol, California, United States: O'Reilly Media.
- Simon, B. (n.d.). *The C4 model for visualising software architecture*. Retrieved 02 2024, from c4model: <https://c4model.com>
- Universitatea Româno-Americană. (2019). *Situații financiare anuale individuale an 2019*. Retrieved 06 2024, from RAU: <https://www.rau.ro/wp-content/mydocuments/dac/Situatii%20financiare%20anuale%20individuale%20an%202019.pdf>
- Universitatea Româno-Americană. (2020-2024). *COMPOSITION OF THE UNIVERSITY'S BOARD OF DIRECTORS*. Retrieved 02 2024, from RAU: <https://www.rau.ro/board-of-directors/?lang=en>

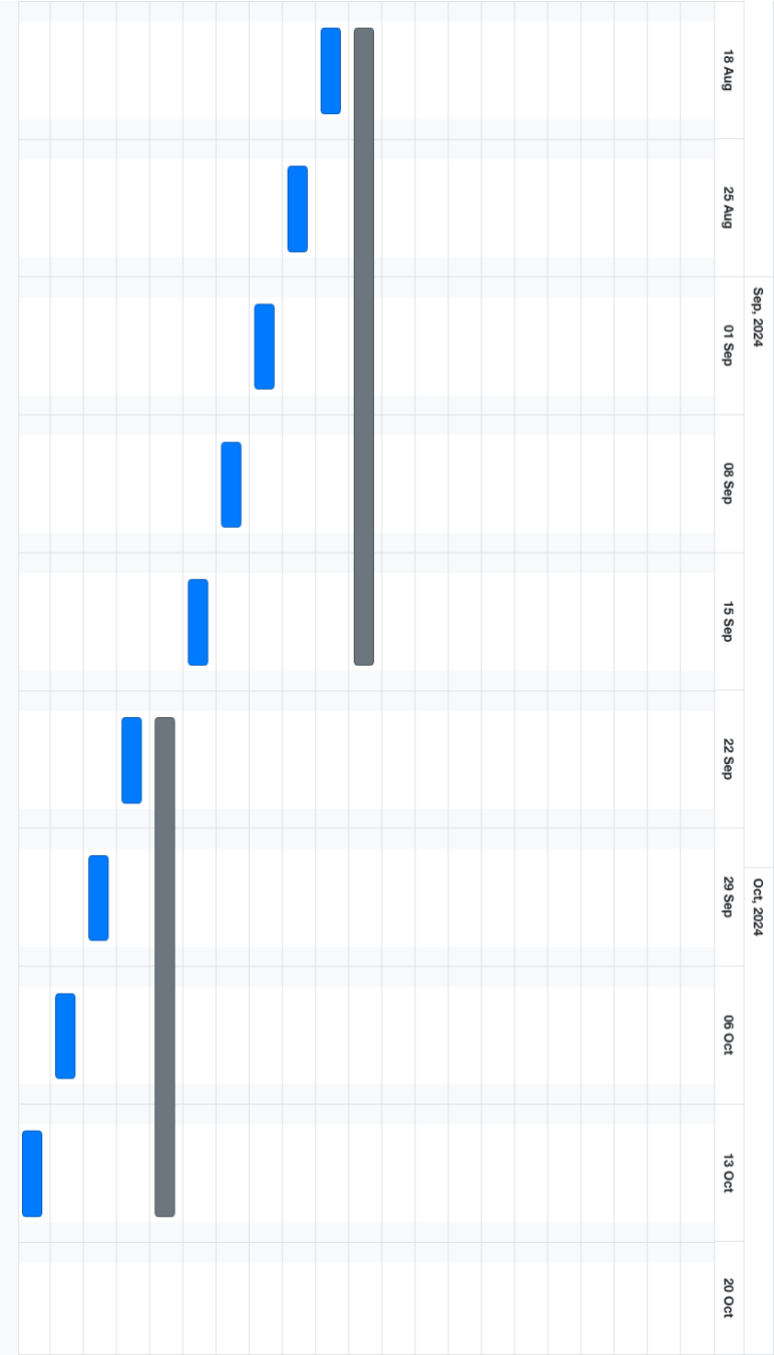
- Universitatea Româno-Americană. (2020, 02). *RAPORTUL RECTORULUI PRIVIND STAREA UNIVERSITĂȚII ROMÂNNO-AMERICANE ÎN ANUL 2019*. Retrieved 06 2024, from RAU: <https://www.rau.ro/wp-content/mydocuments/dac/Raport%20Rector%20URA-2019.pdf>
- Universitatea Româno-Americană. (2020). *RAPORT DE EVALUARE INTERNĂ PRIVIND ASIGURAREA CALITĂȚII ÎN UNIVERSITATEA ROMÂNNO-AMERICANĂ ÎN ANUL 2020*. Retrieved 06 2024, from RAU: <https://www.rau.ro/wp-content/mydocuments/dac/Raport%20Rector%20URA-2020.pdf>
- Universitatea Româno-Americană. (2020). *Situatii financiare anuale individuale an 2020*. Retrieved 06 2024, from RAU: <https://www.rau.ro/wp-content/mydocuments/dac/Situatii%20financiare%20anuale%20individuale%20an%202020.pdf>
- Universitatea Româno-Americană. (2021, 01 11). *GDPR*. Retrieved 02 2024, from RAU: <http://privacy.rau.ro/index-EN.html>
- Universitatea Româno-Americană. (2021). *Situatii financiare anuale individuale an 2021*. Retrieved 06 2024, from RAU: <https://www.rau.ro/wp-content/mydocuments/dac/Situatii%20financiare%20anuale%20individuale%20an%202021.pdf>
- Universitatea Româno-Americană. (2022-2025). *The Gender Equality Strategy and the Gender Equality Plan 2022-2025*. Retrieved 02 2024, from RAU: <https://www.rau.ro/wp-content/mydocuments/diverse/2022/Gender%20Equality%20Strategy%20and%20Gender%20Equality%20Plan-2022-2025.pdf>
- Universitatea Româno-Americană. (2022, 03). *RAPORTUL RECTORULUI PRIVIND STAREA UNIVERSITĂȚII ROMÂNNO-AMERICANE ÎN ANUL 2021*. Retrieved 06 2024, from RAU: <https://www.rau.ro/wp-content/mydocuments/dac/Raport%20Rector%20URA-2021.pdf>
- Universitatea Româno-Americană. (2024). *Academic Departments*. Retrieved 02 2024, from RAU: <https://www.rau.ro/academic-departments/?lang=en>
- Universitatea Româno-Americană. (2024). *Administrative Offices*. Retrieved 02 2024, from RAU: <https://www.rau.ro/administrative-offices/?lang=en>
- Universitatea Româno-Americană. (2024). *Departamentul de studii asiatice*. Retrieved 02 2024, from RAU: <https://www.rau.ro/departamentul-de-studii-asiatice/>
- Universitatea Româno-Americană. (2024). *Executive Management*. Retrieved 02 2024, from RAU: <https://www.rau.ro/executive-management/?lang=en>
- Universitatea Româno-Americană. (2024). *Find a BA program*. Retrieved 02 2024, from RAU: <https://www.rau.ro/find-a-ba-program/?lang=en>
- Universitatea Româno-Americană. (2024). *Find an MA program*. Retrieved 02 2024, from RAU: <https://www.rau.ro/find-a-ma-program/?lang=en>
- Universitatea Româno-Americană. (2024). *Home*. Retrieved 02 2024, from RAU: <https://www.rau.ro/?lang=en>
- Universitatea Româno-Americană. (2024). *Microsoft Center*. Retrieved 02 2024, from RAU: <https://www.rau.ro/microsoft-center/?lang=en>
- Universitatea Româno-Americană. (2024). *Mission, vision and values*. Retrieved 02 2024, from RAU: <https://www.rau.ro/mission-vision-and-values/?lang=en>
- Universitatea Româno-Americană. (2024). *Partners*. Retrieved 02 2024, from RAU: <https://www.rau.ro/partners-and-international-affiliations/?lang=en>
- Universitatea Româno-Americană. (2024). *Schools*. Retrieved 02 2024, from RAU: <https://www.rau.ro/schools/?lang=en>

Universitatea Româno-Americană. (2024). *The Members of the University Senate*.
Retrieved 02 2024, from RAU: <https://www.rau.ro/university-senate/?lang=en>

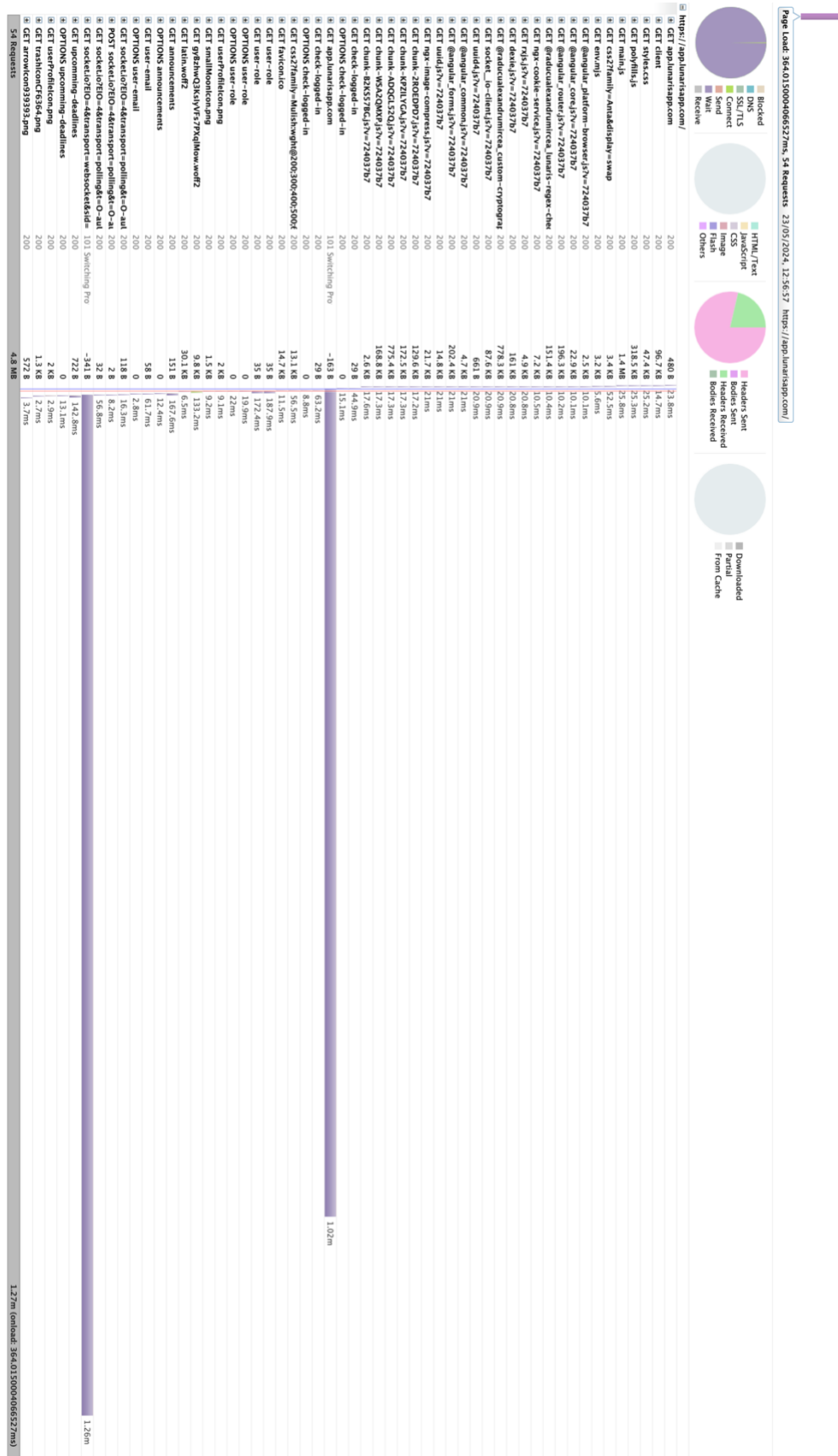
Appendixes

- Appendix 1: Gantt Chart for Implementation Schedule

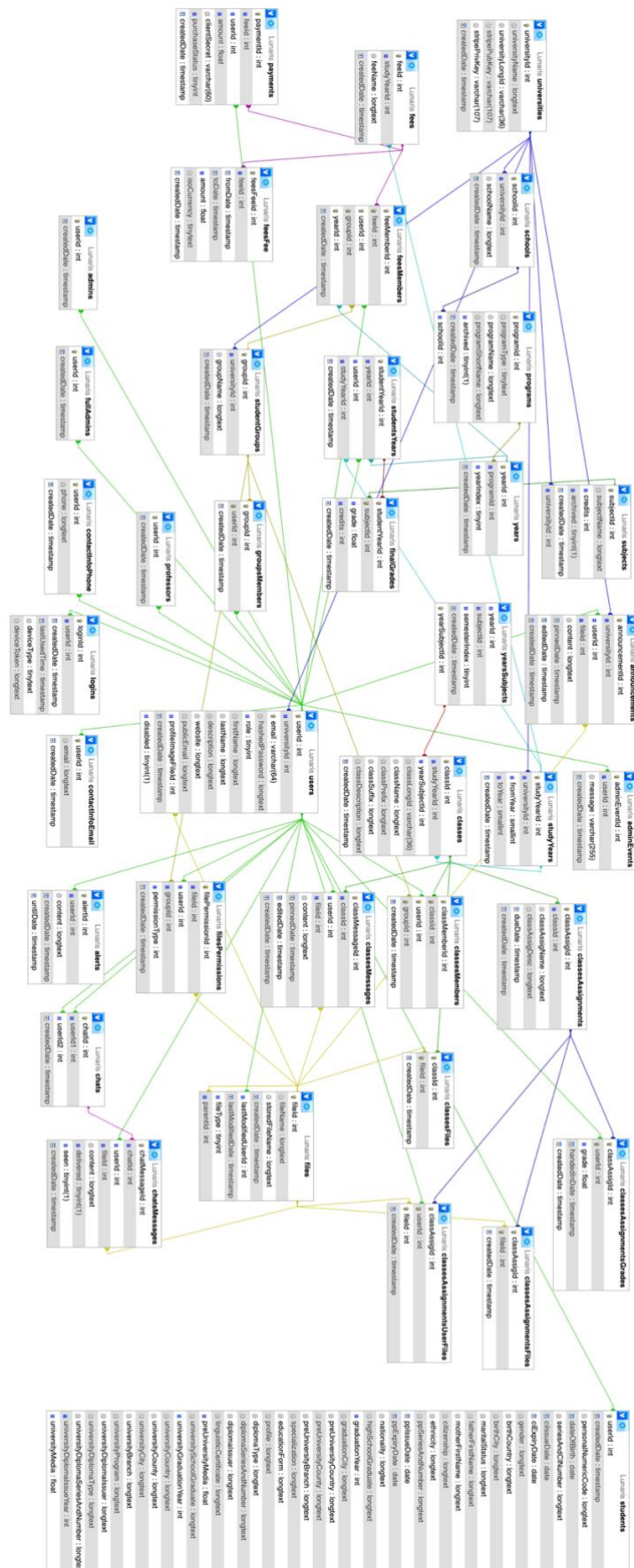




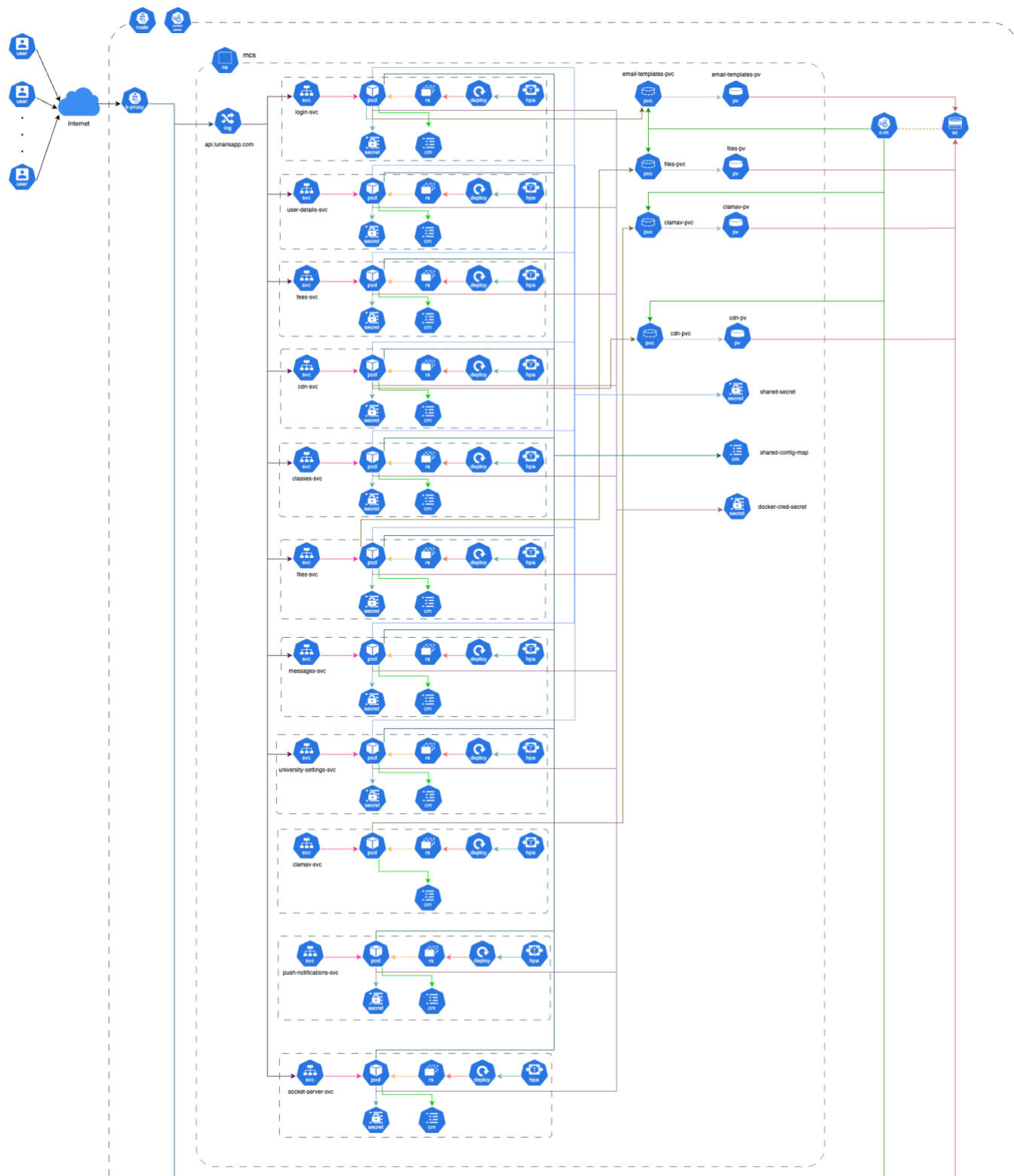
- Appendix 2: Waterfall of requests for loading the home page of the dashboard

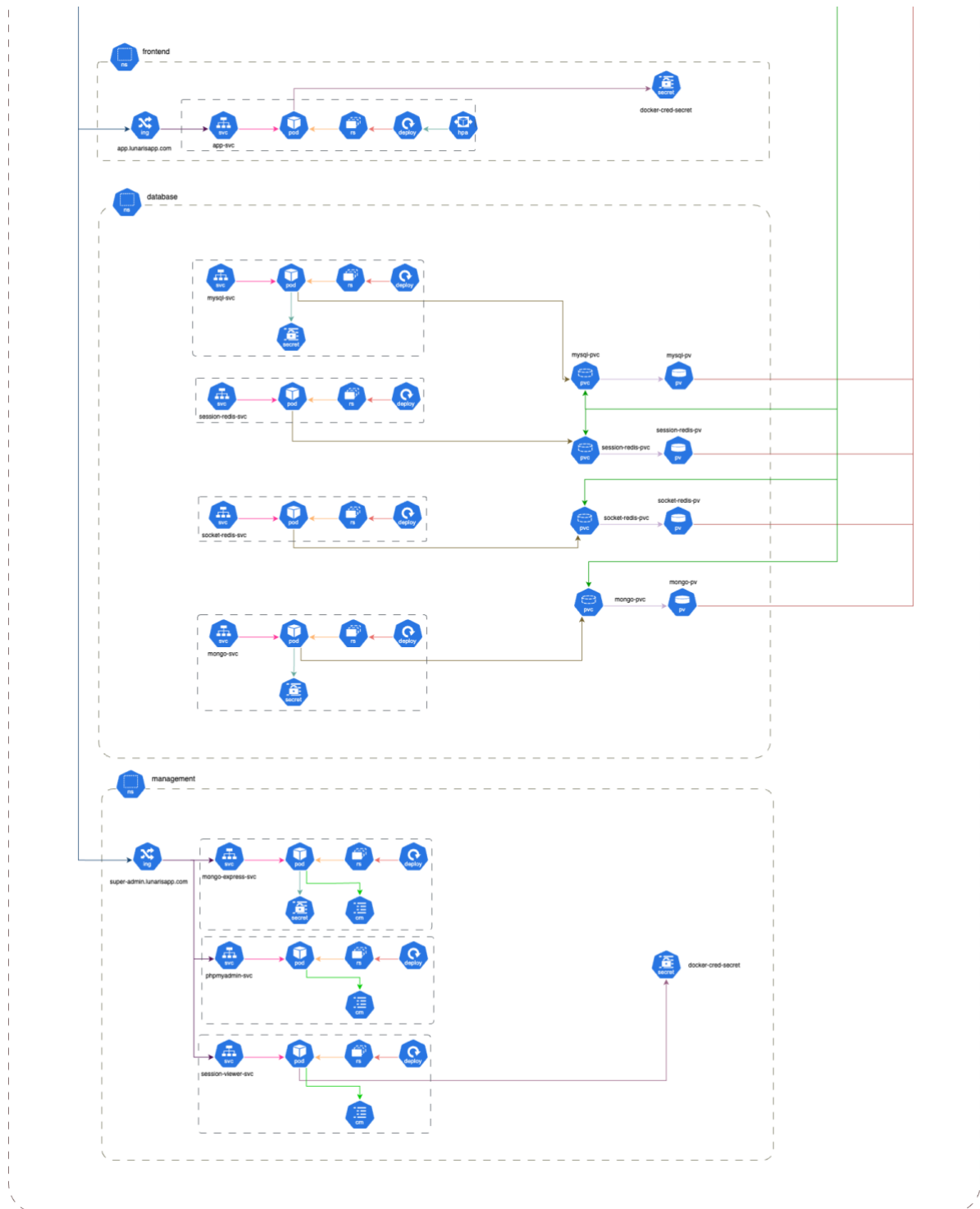


- Appendix 3: Lunaris database ERD



- Appendix 4: Kubernetes diagram





- Appendix 5: Source code for uploading file to secure path inside the Files microservice

```
secureRouter.route('/upload-files').post(upload.array('files'), async
(req: Request, res: Response) => {
  interface CurrentBody {
    filePermissions: string;
    ownerUserId: string;
  }
  var responseObject: CustomResponseObject;
  var body: CurrentBody = req.body;
  console.log(body);
  if (!('filePermissions' in body && 'ownerUserId' in body)) {
    responseObject = {
      succ: false,
      mes: 'Something went wrong',
      debugMes: 'Invalid number of POST parameters',
    };
    res.status(200).send(responseObject);
    return;
  }
  var filePermissions: FilePermission[] =
JSON.parse(body.filePermissions);
  var ownerUserId: number = parseInt(body.ownerUserId);
  var files = (req.files as Express.Multer.File[]) || [];
  var transaction = await dbConnection.startTransaction();
  var fileName: string;
  var fileExt: string | boolean;
  var filesIds: number[] = [];
  try {
    for (var i = 0; i < files.length; i++) {
      fileExt = mime.extension(files[i].mimetype);
      if (!fileExt) {
        await dbConnection.rollbackTransactionAndClose(transaction);
        responseObject = {
          succ: false,
          mes: 'The file type is not supported',
        };
        res.status(200).send(responseObject);
        return;
      }
    }
  }
```

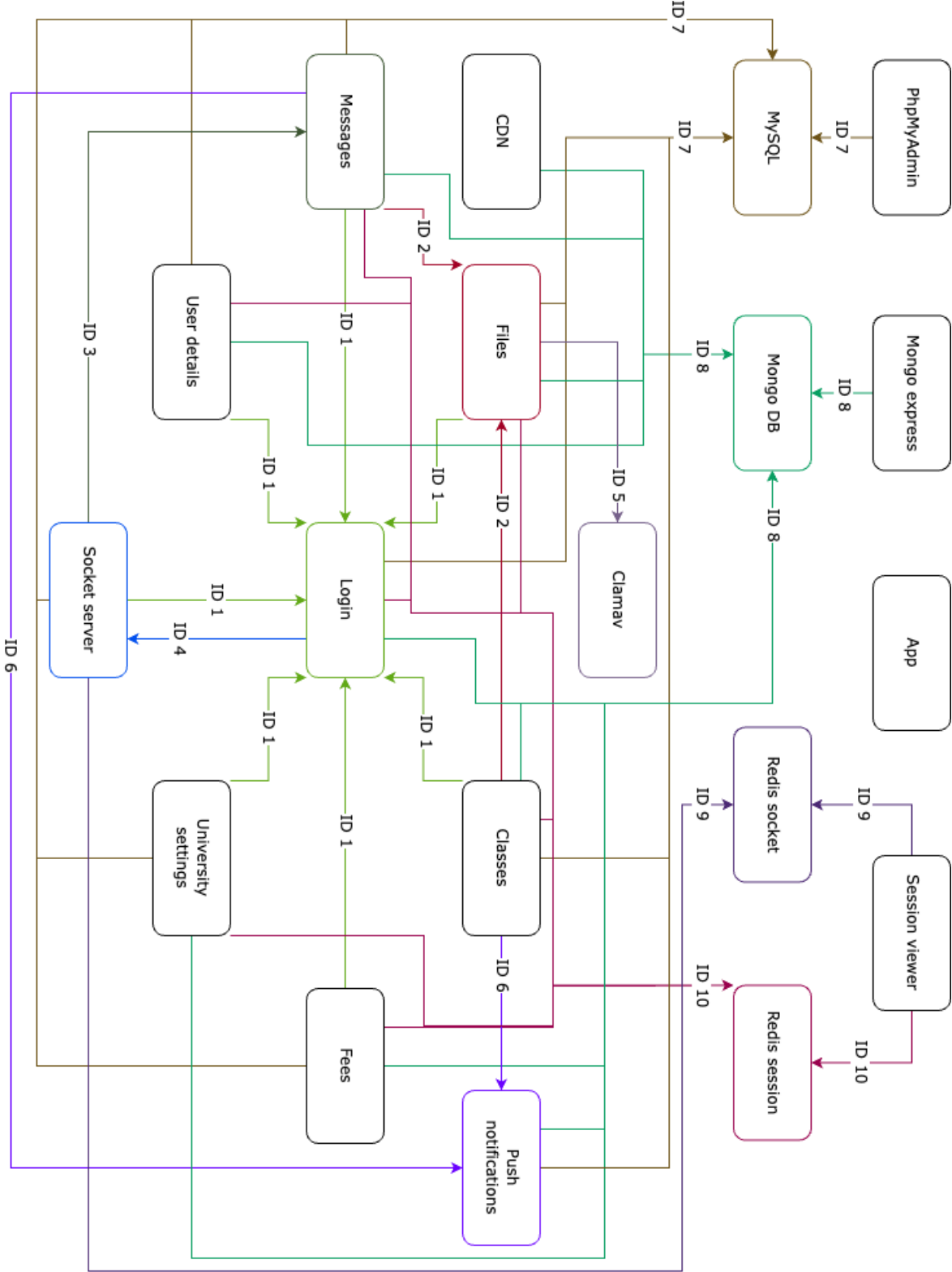
```

        fileName = uuidv4() + '.' + fileExt;
        var fileInsertSqlResult: NormalPacket = await
dbConnection.executeInTransaction<NormalPacket>(
    transaction,
    'INSERT INTO files (fileName, storedFileName,
lastModifiedUserId, fileType) VALUES (?, ?, ?, ?)',
    [files[0].originalname, fileName, ownerUserId, 2]
);
filesIds.push(fileInsertSqlResult.insertId);
var stream = require('stream');
var reader = new stream.PassThrough();
reader.end(files[i].buffer);
var writer = fs.createWriteStream(path.join('/files', fileName));
reader.pipe(writer);
await new Promise<void>((res, rej) => {
    reader.on('end', function () {
        res();
    });
});
for (var j = 0; j < filePermissions.length; j++) {
    await
dbConnection.executeInTransaction<NormalPacket>(transaction, 'INSERT INTO
filesPermissions (fileId, userId, permissionType) VALUES (?, ?, ?)', [
        fileInsertSqlResult.insertId,
        filePermissions[j].userId,
        filePermissions[j].permissionType,
    ]);
}
}
await dbConnection.commitTransactionAndClose(transaction);
} catch (err) {
    await dbConnection.rollbackTransactionAndClose(transaction);
    responseObject = {
        succ: false,
        mes: String(err),
    };
    res.status(200).send(responseObject);
    return;
}
responseObject = {
    succ: true,

```

```
        data: {  
            filesIds: filesIds,  
        },  
    };  
    res.status(200).send(responseObject);  
    return;  
});
```

- Appendix 6: Microservices (MCS) internal relational view



<i>Connection ID</i>	<i>Description</i>
1	Check the login session and if the login session is invalid then check the JWT and create a new valid session.
2	If the message or the assignment has files, then the respective microservice will send the file to the Files MCS.
3	When the socket server receives a new real-time message for chat or class messages it will send it to the Messages MCS for it to save it.
4	When a user logs out then the Login MCS will send a request to the Socket server MCS to disconnect all the connected sockets under that session.
5	The Files MCS will send the file to the Clamav MCS for it to check the file for known viruses and to strip all the metadata that could be used in harmful ways.
6	When either MCS receive a request which needs a push notification then, they will request the Push Notification MCS to send one to the user.
7	Connection to the MySQL database.
8	Connection to the MongoDB database for logging.
9	Connection to the Redis Socket instance for Pub/Sub.
10	Connection to the Redis Session instance for session data and socket data which is linked.

- Appendix 7: Server Communication module source code

```
export class ServerCommunication {
  namespace: string;
  serverKey: string;

  constructor(namespace: string, serverKey: string) {
    this.namespace = namespace;
    this.serverKey = serverKey;
  }

  sendGetRequest(serviceName: string, path: string, namespace: string =
null, port: number = 80, useJson: boolean = true):
Promise<CustomResponseObject> {
    return fetch(this._constructServiceUrl(serviceName, path, port,
namespace), {
      headers: this._constructHeaders({}, useJson),
    })
    .then((response) => {
      return response.json();
    })
    .then((data: CustomResponseObject) => {
      return data;
    });
  }

  sendPostRequest(serviceName: string, path: string, payload: any,
namespace: string = null, port: number = 80, useJson: boolean = true):
Promise<CustomResponseObject> {
    return fetch(this._constructServiceUrl(serviceName, path, port,
namespace), {
      method: 'POST',
      headers: this._constructHeaders({}, useJson),
      body: useJson ? JSON.stringify(payload) : payload,
    })
    .then((response) => {
      return response.json();
    })
    .then((data: CustomResponseObject) => {
      return data;
    });
  }
}
```

```

        private _constructHeaders(headers: { [key: string]: string }, isJson:
boolean = false): { [key: string]: string } {
            return Object.assign(
                { 'server-key': this.serverKey },
                headers,
                isJson
                ? {
                    'Content-Type': 'application/json; charset=utf-8',
                }
                : {}
            );
        }

        private _constructServiceUrl(serviceName: string, path: string, port:
number, namespace: string): string {
            if (namespace == null) {
                namespace = this.namespace;
            }
            return `http://${serviceName}.${namespace}:${port}${path}`;
        }
    }
}

```

- Appendix 8: Public GitHub repository link

<https://github.com/Raducu-Alexandru/Lunaris>