Job Search Engine (JSE) Project Documentation

1. Introduction

**Project:** Job Search Engine (JSE)

**Contractors:** Ospan Ramazan, Saparbek Ulpan, Abdigali Nurasyl

**Purpose:**

JSE is an online platform designed to make it easier for employers and job seekers to connect. It offers a wide set of tools for job searching and recruitment, ensuring a smooth and effective process.

1. Functional Requirements

**User Authentication:**

* Secure authentication mechanism.
* Users can register as job seekers or employers.
* Account verification is required through email confirmation.

**Job Search and Posting:**

* Job seekers can search for job openings using different criteria like job title, location, and salary.
* Employers can post job openings with detailed descriptions and requirements.

**Application Submission and Review:**

* Job seekers can apply for jobs through the platform.
* Employers receive notifications about new applications and can view applicant profiles.

**Communication Tools:**

* The platform provides messaging between employers and job seekers.
* Supports interview scheduling.

**Logging and Monitoring:**

* The system logs all actions (applications, job postings, communications).
* Administrators can remotely access these logs to monitor the system.

**Error Handling:**

* The platform should display clear explanations when errors occur.
* Users should have the option to retry after a failed transaction.

3. About Risk

1. Risk: Lack of clarity of requirements:

* Description: Lack of a clear mechanism for defining requirements for user accounts (jobseekers, employers), which can lead to malfunctions in the registration and authentication process.
* Importance: 4 (important to prevent problems in the initial stage of use)
* Complexity: 3 (requires additional clarity in documentation)
* Mitigation Strategy Techniques:

1. Develop detailed enrollment scripts for each type of user.
2. Testing the registration scripts with users to identify potential problems.

* Considered: yes.

1. Risk: Improper dependency management.

* Description: Lack of integration between different components of the system, such as between the job search system and the application process, which can lead to unexpected errors.
* Importance: 4 (important to ensure the coherent and efficient functioning of the system)
* Complexity: 3 (requires more careful definition of dependencies)
* Mitigation Strategy Techniques:

1. Developing clear interfaces between the various components of the system.
2. Regular testing of processes that depend on each other.

* Considered: yes.

1. Risk: Ineffective technology management.

* Description: Potential problems with payment systems or bugs in the mobile app could cause disruptions that affect the usability of the system..
* Importance: 4 (critical for business continuity)
* Complexity: 3 (more testing and updates are needed
* Mitigation Strategy Techniques:

1. Implement a monitoring system and periodically update technology elements.
2. Provide training to staff on safe and effective methods of using technology.

* Considered: yes.

1. Risk: Poor security.

* Description: Lack of strong security makes the system vulnerable to web-based attacks such as SQL injection, XSS and others, which can lead to leakage of sensitive information and loss of user trust.
* Importance: 5 (critical importance as insecure practices can lead to leakage of sensitive information and loss of user trust)
* Complexity: 4 (requires high technical expertise and implementation of additional security features)
* Mitigation Strategy Techniques:

1. Implement a robust Web Application Firewall (WAF) to filter malicious traffic and prevent attacks.
2. Regularly updating and auditing application code to identify and remediate potential vulnerabilities.
3. Training staff on secure development practices and ensuring compliance with security standards.

* Considered: yes.

1. Risk: Failure to send a confirmation email when registering a user.

* Description: In case of a trivial server glitch during registration, the user may not receive a confirmation email, which creates the risk of missing a potential user and loss of trust.
* Importance: 5 (critical importance, missed users and loss of trust).
* Complexity: 2 (technical failure).
* Mitigation Strategy Techniques:

1. Replacing servers with more productive and stable servers.
2. Ability to resend the request for written confirmation.

* Considered: yes.

1. Risk: Lack of platform administrators and support staff.

* Description: The project will definitely serve a lot of traffic someday, to control it, maintaining a large staff may cost a young company a lot of money.
* Importance: 4 (very important in terms of perceived interest of the parties).
* Complexity: 3 (finding a reliable CRM service provider).
* Mitigation Strategy Techniques:

1. Monitor traffic to forecast staffing needs.
2. Establishing a long-term partnership with a reliable CRM service provider..
3. Automating routine tasks via artificial intelligence bot.

* Considered: yes.

1. Risk: Users misuse their personal profile and publish phishing links.

* Description: Users may abuse their profile by publishing phishing links, which directly affects the security of users' data and devices.
* Importance: 5 (extreme importance, affects data and device security).
* Complexity: 2 (solution from the technical part).
* Mitigation strategy methods:

1. Every time a user authorizes on a service, remind the user to keep their eyes open and remind them of possible malicious users.
2. Ability to check links before publishing via CRM system.

* Considered: yes.

1. Risk: Confusion among messages/requests to the platform administrator.

* Description: Confusion among different requests can slow down processing and responses, affecting the responsiveness of the service.
* Importance: 4 (affects the speed of service responsiveness).
* Complexity: 1 (easily solvable problem).
* Mitigation Strategy Techniques:

1. Adding template topics to intelligently sort messages..
2. Redirecting to an existing administrator when urgently needed.

* Considered: yes.

1. Risk: Error Management.

* Description: Ambiguity when errors occur can slow down the error management process.
* Importance: 4 (affects the speed of bug fixing)
* Complexity: 2 (difficulty in identifying faults in a timely manner)
* Mitigation Strategy Techniques:

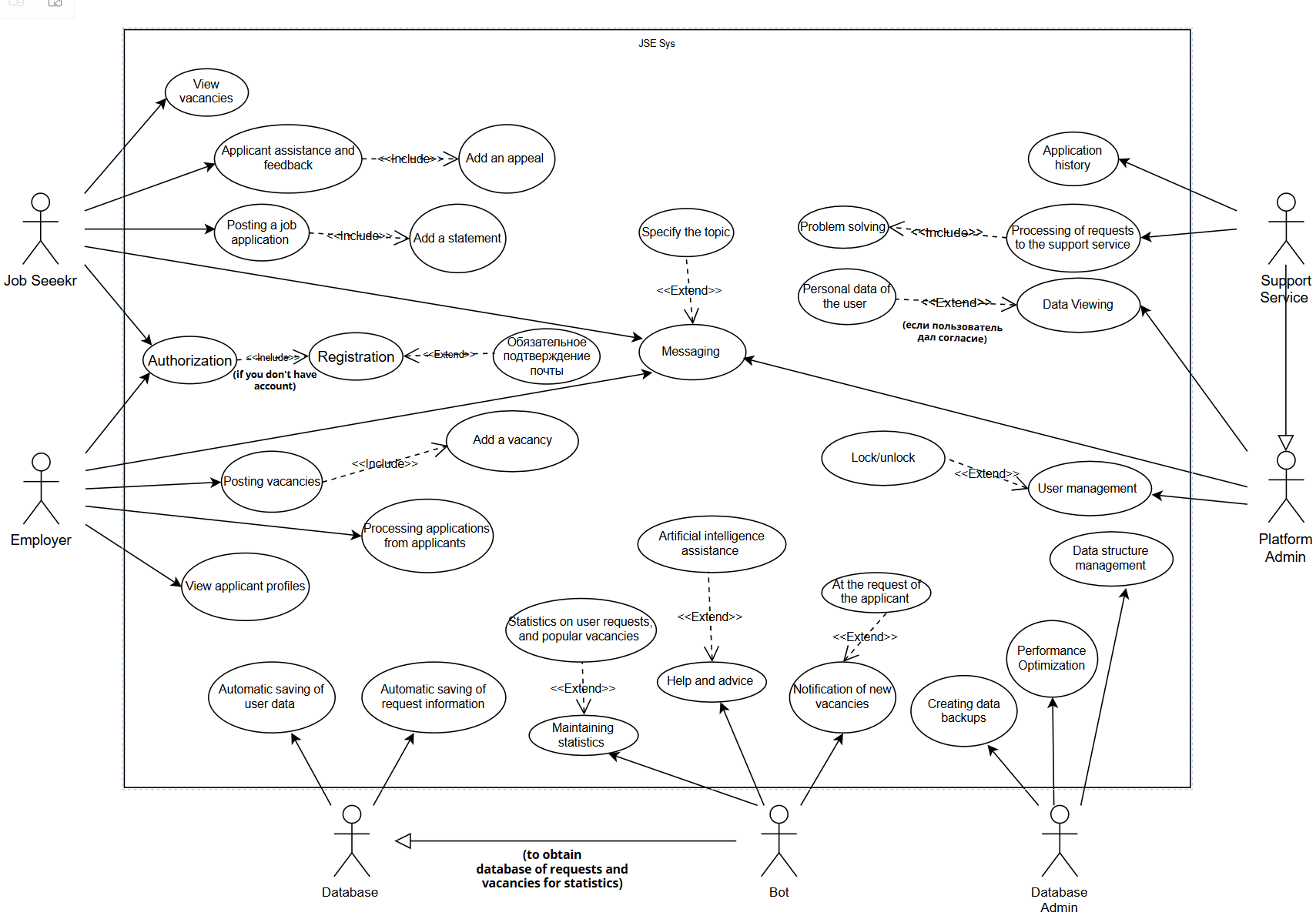
1. It is important to create informative error messages in plain language.
2. mplement an error log that will keep a report on the origin of the error and the cause, for more productive troubleshooting.
3. Designate individuals to periodically test different aspects of the system and collect data.

* Considered: yes.

4. Use Case Diagram

Actors:

* **Job Seeker:** Posts applications, seeks help and feedback, views company profiles and job openings, exchanges messages, and handles login and registration.
* **Employer:** Posts jobs, processes applications, views applicant profiles, exchanges messages, and handles login and registration.
* **Platform Administrator:** Views user data, verifies authenticity, manages users, and exchanges messages.
* **Support Service:** Handles support requests and maintains a history of interactions.
* **Database Administrator:** Manages data structure and performance, creates backups.
* **Bot:** Manages statistics, provides help and advice, notifies about new job openings.
* **Database:** Automatically saves user data and query information.



5. Creating use case template

| **ID** | **Description** | **Actors** | **Assumptions** | **Steps** | **Variations** | **Quality Concerns** | **Issues** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Размещение заявления на работу | Job Seeker | имеет аккаунт на платформе.  успешно авторизован в системе. | входит в свой аккаунт на платформе.  выбирает опцию "Разместить заявление на работу".  Система запрашивает необходимую информацию, такую как опыт работы, образование и навыки.  вводит требуемую информацию.  Система сохраняет заявление и генерирует уведомление об успешном размещении. | Если Job Seeker вводит недостаточную информацию, система отображает сообщение об ошибке и запрашивает дополнительные данные. | размещение заявления должно быть выполнено в течение определенного времени. | Как обрабатывать случай, если система временно недоступна в момент размещения заявления? |
| 2 | Просмотр вакансий | Job Seeker | успешно авторизован в системе. | 1. входит в свой аккаунт на платформе.  2. выбирает опцию "Просмотр вакансий".  3. Система отображает список доступных вакансий с информацией о них, такой как название, местоположение и зарплата.  4. может использовать фильтры для уточнения результатов поиска  5. выбирает интересующую вакансию для получения дополнительной информации. | Возможность сортировки вакансий по различным критериям (например, по дате публикации или релевантности). | Точность поиска: Система должна предоставлять соответствующие списки вакансий | Как обрабатывать ситуацию, когда у вакансии нет дополнительной информации? |
| 3 | Размещение вакансии | Employer | имеет аккаунт на платформе.  успешно авторизован в системе. | 1. входит в свой аккаунт на платформе.  2. выбирает опцию "Разместить вакансию".  3. Система запрашивает необходимую информацию, такую как название вакансии, требования, местоположение и срок подачи заявок.  4. вводит требуемую информацию.  5. Система сохраняет вакансию и генерирует уведомление об успешном размещении. | Возможность добавления дополнительных деталей, таких как тип занятости (полная занятость, частичная занятость) или вид оплаты. | Обеспечение актуальности вакансий: система должна регулярно обновлять список вакансий. | Как обрабатывать ситуацию, когда Employer забыл указать местоположение вакансии? |
| 4 | Обработка заявок от соискателей | Employer | успешно авторизован в системе.  разместил вакансию. | 1. входит в свой аккаунт на платформе.  2. входит в свой аккаунт на платформе.  3. Система отображает список заявок с информацией о соискателях.  4. может рассматривать профили соискателей и принимать решение о приглашении на собеседование или отказе.  5. Система генерирует уведомление для соискателей о решении работодателя. | Возможность фильтрации заявок по определенным критериям (например, по уровню опыта или образования). | страница с вакансиями должна загружаться быстро даже при большом объеме данных. | Как обрабатывать ситуацию, когда Employer хочет связаться с соискателем для дополнительной информации? |
| 5 | Уведомление об отказе | Job Seeker, Employer | 1. Job Seeker отправил заявление на вакансию.  2. Employer рассмотрел заявление и принял решение об отказе. | 1.System получает уведомление от Employer об отказе в заявлении соискателя.  2. System проверяет, что заявление соискателя было рассмотрено.  3. System отправляет уведомление Job Seeker о том, что его заявление было отклонено.  4. Job Seeker получает уведомление и может просмотреть причины отказа. | Возможность Employer прикрепить комментарии или обоснование к отказу. | Быстрота доставки уведомлений: система должна оперативно отправлять уведомления об отказе. | Как обрабатывать случай, когда у Job Seeker есть вопросы по отказу и он хочет связаться с работодателем для получения обратной связи? |

6. SOFTWARE LIFE CYCLE MODELING

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Description** | **Advantages** | **Disadvantages** |
| Waterfall | In the Development Cascade model, the process progresses linearly through sequential phases: requirements definition, design, implementation, testing, deployment, and maintenance. | Simple and straightforward, suitable for projects with stable and well-defined requirements. | Lack of flexibility, difficulty in making changes after development has started, high risk of defects at later stages. |
| Iterative | In the Iteration model, the development process is broken down into smaller cycles, each consisting of planning, implementation, testing, and evaluation. Each iteration creates a step-by-step increase in software functionality. | Allows you to get feedback and develop the product incrementally, reduces risks by ensuring that important requirements are prioritized, easier to make changes compared to the Cascade model. | Requires more customer involvement and feedback, may increase complexity if iterations are not managed well, possible overgrowth of workload if requirements are not sufficiently controlled. |
| Scrum | Scrum is an agile methodology that emphasizes teamwork, accountability, and iterative movement toward a clearly defined goal. It involves breaking a project into short cycles, called sprints, with frequent inspections and adaptations. | Highly flexible and adaptable to changing requirements, promotes collaboration and communication within the team, allows for continuous improvement through regular retrospectives. | Requires experienced and self-organizing teams, can be difficult to implement in organizations unfamiliar with agile practices, depends heavily on effective communication and coordination. |

Summary: In selecting Waterfall, Iterative and Scrum models, we considered the specifics of the "job search engine" project. Waterfall will support the clear definition of requirements, especially in the context of accurate job search. The Iterative approach will provide flexibility to make changes, which is critical in an ever-changing labor market. Using Scrum, we will be able to better coordinate the team, supporting regular progress reviews and maximizing performance in a rapidly evolving employment landscape. This combined approach will optimize the development of the job search engine, making it more adaptive and efficient.

7. Activity diagram

**Action Initiation**

* **Data Entry:** The user starts interacting with the system by entering their personal data.

**Role Selection**

* **The user selects the role under which they will continue to work in the system:** Employer or Job Seeker.

**Employer**

1. **Job Posting:** The employer posts job information in the system.
2. **Application Review:** The employer reviews applications from job seekers for the posted jobs.

**Job Seeker**

1. **Job Search:** The job seeker searches for jobs using various search criteria.
2. **Job Application Submission:** The job seeker submits applications for selected job openings.

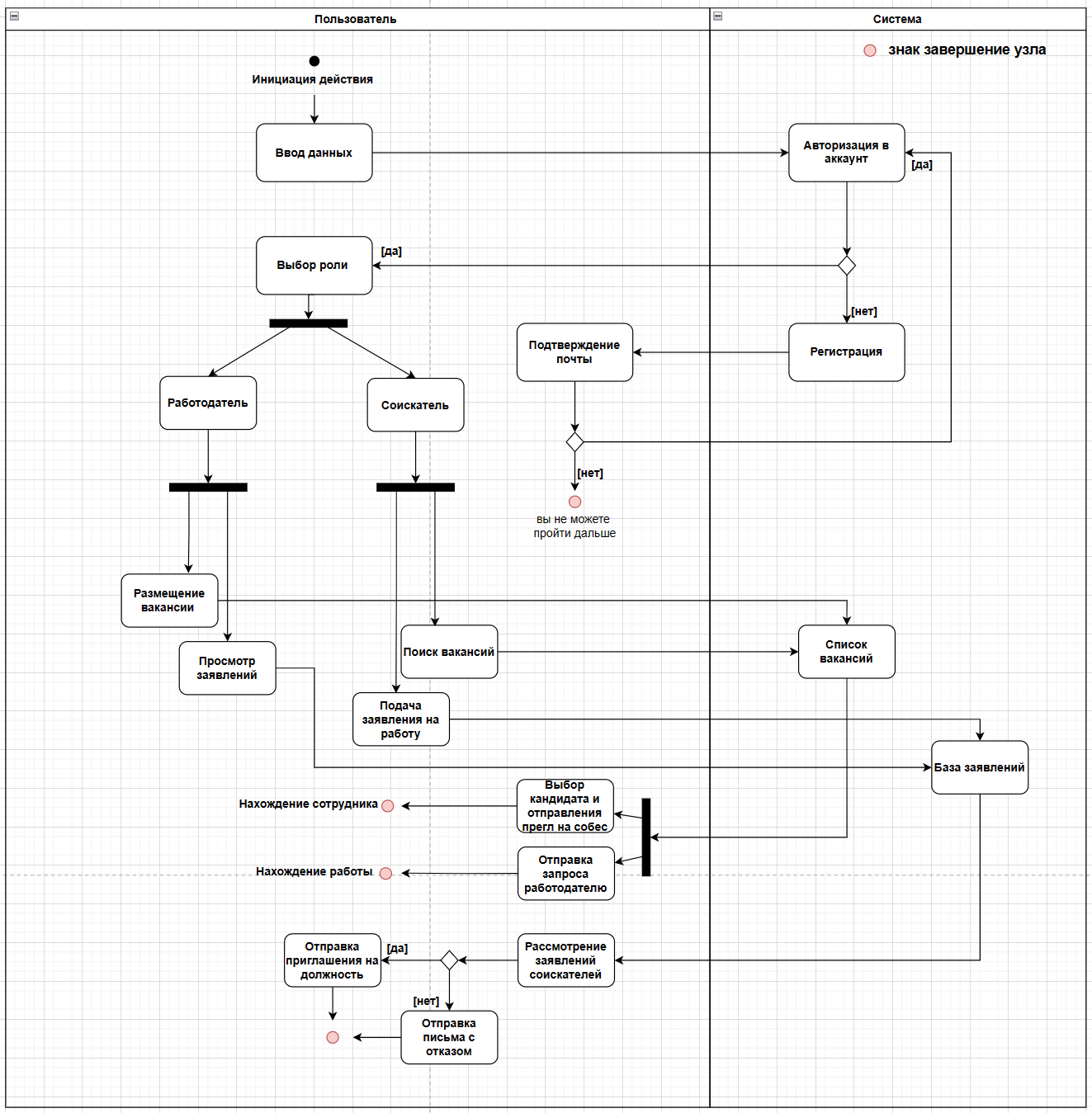
**Common Processes for Employers and Job Seekers**

1. **Invitation/Rejection Sending:** Depending on the employer's decision, the job seeker receives an interview invitation or a rejection notice.
2. **Candidate Selection and Offer Sending:** The employer selects candidates and sends them job offers.
3. **Employer Inquiry Sending:** The job seeker can initiate feedback with the employer.
4. **Job Application Review:** The employer reviews job applications from job seekers.

**System**

1. **Account Authorization/Registration:** The user authorizes their account in the system or goes through the registration process if they do not have an account.
2. **Email Confirmation:** This is a mandatory step to complete the registration and proceed further in the system.
3. **Job Listing:** The system provides a list of jobs to employers, which can be used to manage current job openings.
4. **Application Database:** The system stores all job applications from job seekers in a database accessible to employers.

**Node Completion Symbol:** This symbol indicates the completion of a specific process flow in the diagram.



8. Sequence diagram

**Diagram Elements:**

**1. Users:**

* Job Seeker
* Employer

**2. System Components:**

* platform frontend
* platform backend (Django)
* Database (PostgreSQL)
* email system (MAILGUN)

**Registration Process:**

**1. Users start registration by entering their personal data.**

* Job Seeker or Employer begins the registration.

**2. Frontend checks the entered data.**

* Ensures the data is correct and complete.

**3. Frontend sends the checked data to the backend.**

**4.Backend checks the data again for security and processes it.**

* Includes extra checks for data integrity.

**5. Backend sends a request to the database to add the new user.**

* Stores the data securely.

**6. Database confirms the new user is added.**

**7. Backend sends a success message to the frontend.**

**8. Frontend shows a success message to the user.**

**9. Backend sends a confirmation email via MAILGUN.**

* Email contains a link to confirm the registration.

**Authorization Process:**

**1. Users enter their login credentials.**

**2. Frontend checks the login credentials.**

* Ensures the data is correct and complete.

**3. Frontend sends the checked credentials to the backend.**

**4. Backend processes the credentials and checks them in the database.**

**5. Database returns the user data if credentials are correct.**

**6. Backend confirms the credentials are valid.**

**7. Backend sends a success or error message to the frontend.**

**8. Frontend shows the result to the user.**

**Creating an Application/Job Posting:**

**1. Authorized users enter details for an application or job posting.**

**2. Frontend checks the entered details.**

* Ensures the data is correct and complete.

**3. Frontend sends the checked details to the backend.**

**4. Backend processes the details and sends a request to the database to store them.**

**5. Database confirms the storage of the details.**

**6. Backend sends a success message to the frontend.**

**7. Frontend shows a success message or error to the user.**

**Password Recovery Process:**

**1. Users request password recovery by entering their email.**

**2. Frontend checks the entered email.**

* Ensures the email is in the correct format.

**3. Frontend sends the checked email to the backend.**

**4. Backend checks the email in the database.**

* Sends a request to the database to verify the email address.

**5. Database confirms if the email is found.**

* If not found, an error message is sent to the user.

**6. If the email is found, the backend generates a password recovery token and stores it in the database.**

**7. Backend sends a password recovery email via MAILGUN.**

* Email contains a link to reset the password.

**8. Users receive the email and follow the link to reset their password.**

* The link takes them to a page to enter a new password.

**9. Frontend checks the new password.**

* Ensures the password meets security criteria.

**10. Frontend sends the new password and token to the backend.**

**11. Backend checks the token and updates the user's password in the database.**

**12. Database confirms the update.**

**13. Backend sends a success message to the frontend.**

**14. Frontend shows the result to the user.**

**Error Handling:**

**1. If the email is not found:**

* An error message is sent to the user.

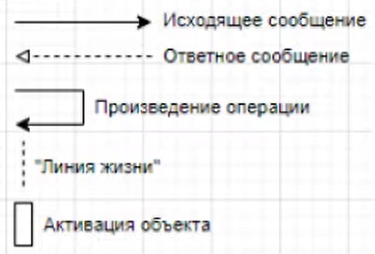
**2. If sending the email fails:**

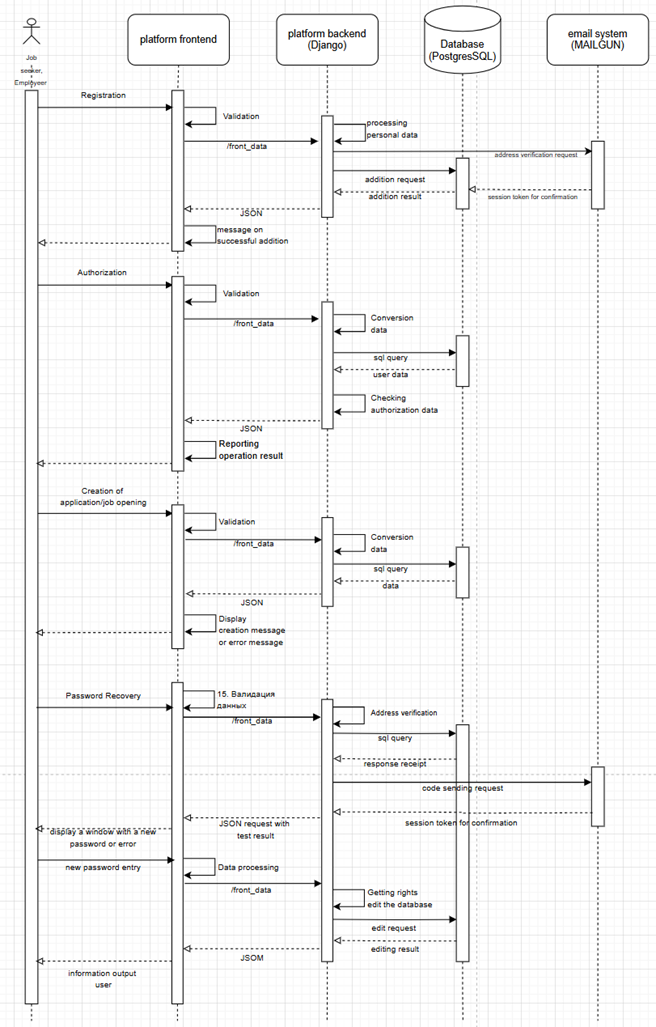
* An error message is sent to the user.

**3. If the token is invalid:**

* An error message is sent to the user.

**4. If the new password does not meet criteria:**

* An error message is sent to the user.



9. Class diagram

Class Diagram

Classes and Relationships:

Class: Users

Description:

The base class for all users of the system.

Attributes:

UserID: The unique identifier of the user.

Username: The username used to log in.

Email: The user's email address.

Password: The user's encrypted password.

Role: The user's role in the system (e.g. JobSeeker, Employer).

Methods:

Register(): Registers a new user.

Login(): Verifies credentials and logs in to the system.

Logout(): Logs the user out of the system.

Class: Admin (inherits User)

Description:

System administrator, has advanced privileges to manage the system.

Attributes:

AdminID: The unique identifier of the administrator.

Name: The name of the administrator.

Email: The administrator's email address.

AccessLevel: The administrator's access level.

Methods:

ViewLogs(): View system logs.

ManageUsers(UserID): Manage users.

Class: JobSeeker (inherits User)

Description:

A user looking for a job on the system.

Attributes:

Resume: The job seeker's resume.

SearchHistory: The job seeker's job search history.

AppliedJobs: List of jobs for which the user has applied.

Methods:

SearchJob(): SearchJob(JobJob).

ApplyForJob(JobID): Applying for a job.

UpdateResume(): Updating a resume.

Class: Employer (inherits User)

Description:

The employer or company representative posting job openings.

Attributes:

CompanyProfile: The employer's company profile.

PostedJobs: A list of posted job openings.

Methods:

PostJob(): Posting a new job posting.

ReviewApplications(JobID): Reviewing applications for job vacancies.

SendInvitation(JobSeekerID): Sending an invitation to a job seeker.

Class: JobPosting

Description:

Information about a job posting by an employer.

Attributes:

JobID: The unique identifier of the job posting.

EmployerID: The identifier of the employer who posted the job.

Title: The title of the vacancy.

Description: Description of the vacancy.

Requirements: The requirements for applicants.

ApplicationDeadline: The deadline to apply for the job vacancy.

Methods:

Create(): Creates a new vacancy.

Update(): Updates the information about the vacancy.

Delete(): Deleting a vacancy.

Class: Application

Description:

A job application from a job seeker containing information about their desire to work in a particular position.

Attributes:

ApplicationID: - The unique identifier of the application.

JobID: - The identifier of the vacancy for which the application is being submitted.

JobSeekerID: - The identifier of the job seeker who submitted the application.

SubmissionDate: - The date the application was submitted.

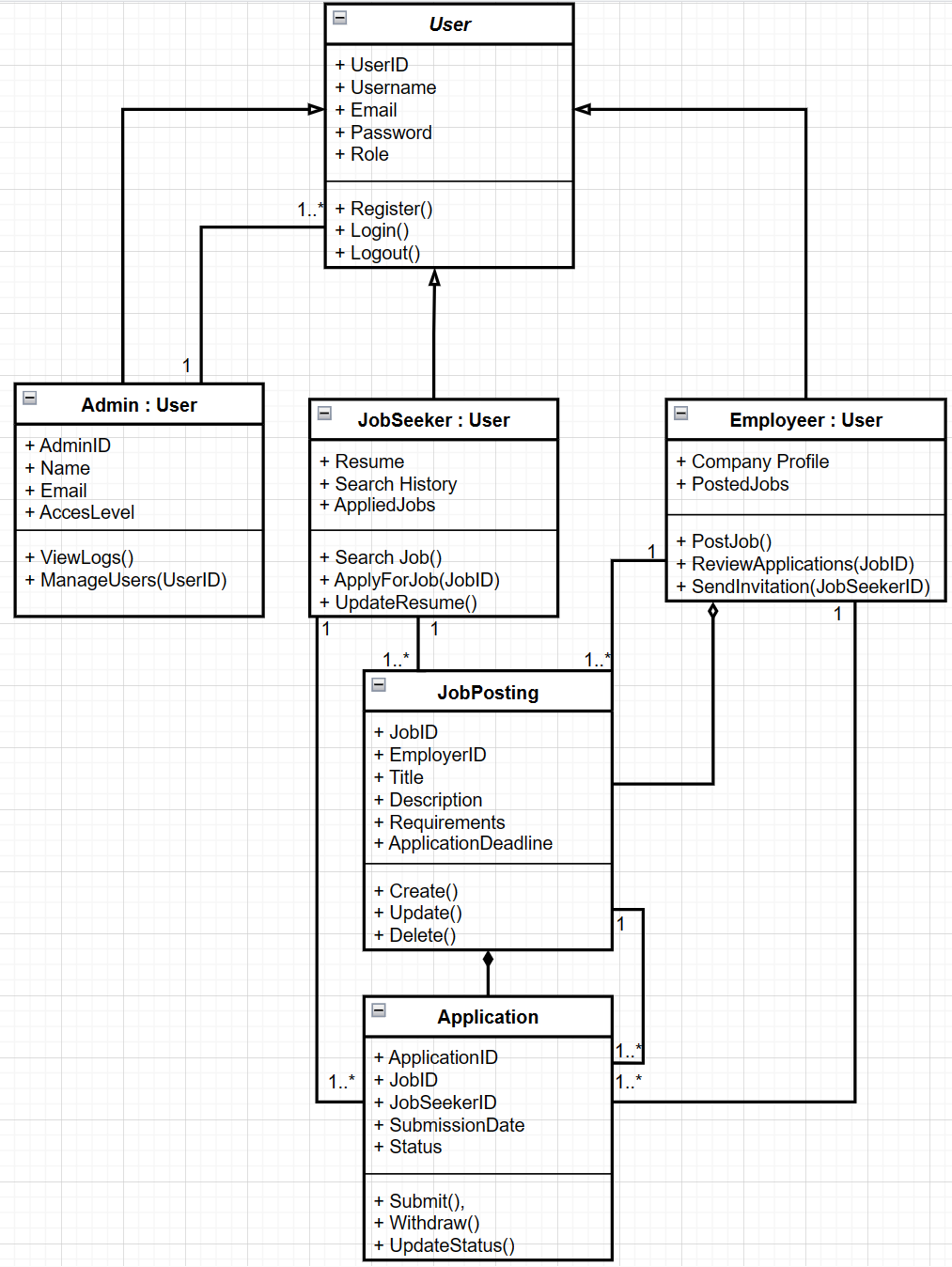
Status: - Status of the application (e.g., 'accepted', 'under consideration', 'rejected').

Methods:

Submit(): - Submit the application for review by the employer.

Withdraw(): - Withdrawing the application by the applicant.

UpdateStatus(Status: String): - Updating the status of an application.



10. Context diagram

Components:

1) Input:

* Applicant Data: Information submitted by job seekers, including personal details, resumes, and job applications.
* Vacancies: Job postings submitted by employers, including job descriptions, requirements, and application deadlines.
* Support Requests: Inquiries and issues submitted by users (both job seekers and employers) requiring assistance from the support team.

2) Control:

* Platform Rules: Policies and regulations that govern the operations of the JSE system, ensuring compliance and proper usage.
* Administering: Administrative tasks performed by platform administrators, such as user management, data monitoring, and system maintenance.

3) Mechanism:

* JSE Servers: The backend infrastructure that hosts the JSE system, including databases and application servers.
* User Interface: The frontend interface through which users interact with the JSE system, including web and mobile applications.
* Analytics System: Tools and systems used for analyzing data, generating reports, and providing insights into system performance and user behavior.

4) Output:

* Selected Vacancies for Job Seekers: Customized job listings and recommendations provided to job seekers based on their profiles and preferences.
* Selection of Candidates for Employers: Lists of suitable candidates for employers, generated based on job applications and filtering criteria.
* Administrator Reports: Reports and analytics generated for platform administrators to monitor system performance, user activities, and other key metrics.

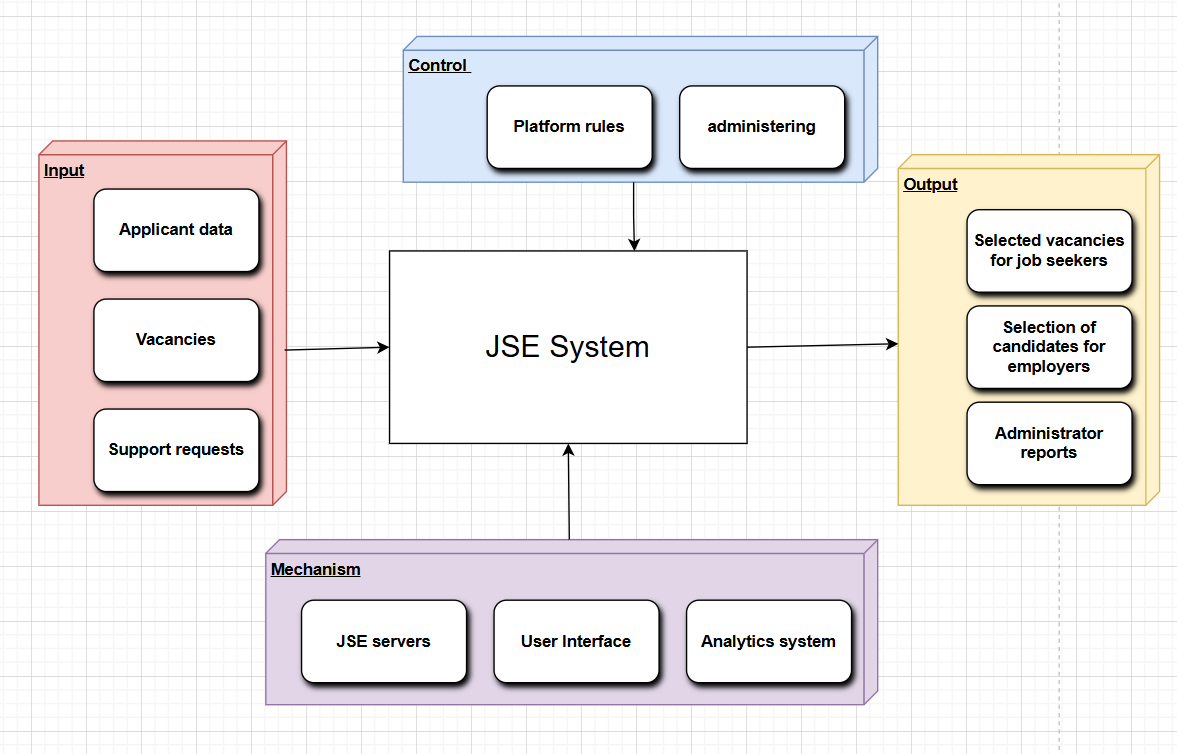
**Interactions:**

Input to JSE System: Applicant data, vacancies, and support requests are submitted to the JSE system for processing and handling.

Control to JSE System: Platform rules and administrative actions are applied to manage the operations and ensure proper functioning of the system.

Mechanism to JSE System: The backend servers, user interface, and analytics system support the JSE system's operations, providing the necessary infrastructure and tools.

Output from JSE System: The system generates selected vacancies for job seekers, candidate selections for employers, and reports for administrators, delivering valuable information and insights to the respective users.



11. Decomposition diagram (more about the processes)

**Components:**

1) Registration:

* User Data: Information provided by job seekers (Seeker) or employers (Employer) during the registration process.
* Input Validation: The administrator checks the incoming data to ensure it is correct and complete.
* Support Requests: Users can submit support requests if they encounter issues during registration.

2) Authorization:

* User Authentication: Verification of user credentials to allow access to the system.
* MFA Mechanism: Multi-Factor Authentication (MFA) for enhanced security during the login process.
* Problem Resolution: Handling of authorization issues, with feedback from the support team if needed.

3) Profile Management:

* Profile Editing: Users can update their resumes or company profiles.
* Verification: Re-verification of profiles with assistance from the support team, if necessary.
* Data Collection and Analytics: Collection and analysis of user data to improve system functionality and compliance.

4) Job Search and Candidate Selection:

* Job Search Panel (for Seekers): Interface for job seekers to find job vacancies.
* Candidate Search Panel (for Employers): Interface for employers to find suitable candidates.
* Filtering Mechanisms: Tools for selecting relevant jobs for job seekers and suitable candidates for employers.
* Analytics System: Analyzes and refines search results to enhance user experience.

5) Support and Administration:

* Support Requests Handling: Users can request help from the support team.
* System Rules: Platform rules governing the system operations and ensuring compliance.
* Data Administration: Database administrators manage and oversee data integrity and security.
* Report Generation: Creating system reports and ensuring compliance with GDPR and other protocols.

**Interactions:**

Registration Process:

* Users (Seekers and Employers) provide their data for registration.
* Data is validated by the administrator.
* If issues arise, users can request help from the support team.

Authorization Process:

* Users log in using their credentials.
* MFA is used for additional security.
* Authorization issues are resolved with feedback from the support team.

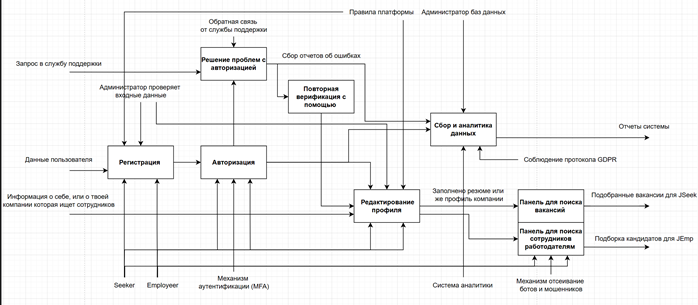
Profile Management:

* Users can edit their profiles.
* Re-verification is done if necessary.
* Data is collected and analyzed for system improvement and compliance.

Job and Candidate Search:

* Job seekers use the search panel to find vacancies.
* Employers use the search panel to find candidates.
* Analytics system refines the search results.

Support and Administration:

* Users can request support for any issues.
* Administrators ensure data security and compliance with platform rules.
* Reports are generated for system monitoring and compliance.

12. Deployment diagram

**Nodes and Components:**

1) Web Server:

* Components:

1. Load Balancer: Distributes incoming traffic across multiple servers to ensure reliability and availability.

2. Web Frontend: Handles user interactions, containing HTML, CSS, and JavaScript artifacts.

2) Security Node:

* Components:

1. FireWall: Provides network security by monitoring and controlling incoming and outgoing network traffic.

2. Authorization Server: Manages user authentication and authorization.

3) Application Server:

* Components:

1. Backend: The core logic of the JSE system, which processes requests and interacts with other components.

Artifacts:

`backend.jar` for starting the system.

2. Application Management Component: Manages job applications.

Artifacts:

`Submit applications`

`Application status`

3. API Service: Provides REST API for frontend-backend communication.

4. Employer Response Component: Manages employer interactions with job applications.

Artifacts:

`View/manage application`

5. Job Application Component: Handles job postings and applications.

4) Database Server:

* Components:

1. PostgreSQL Controller: Manages the PostgreSQL database.

Artifacts:

`Database scripts` for managing database schema and data.

2. Elasticsearch Service: Provides search functionality within the system.

5) Additional Services:

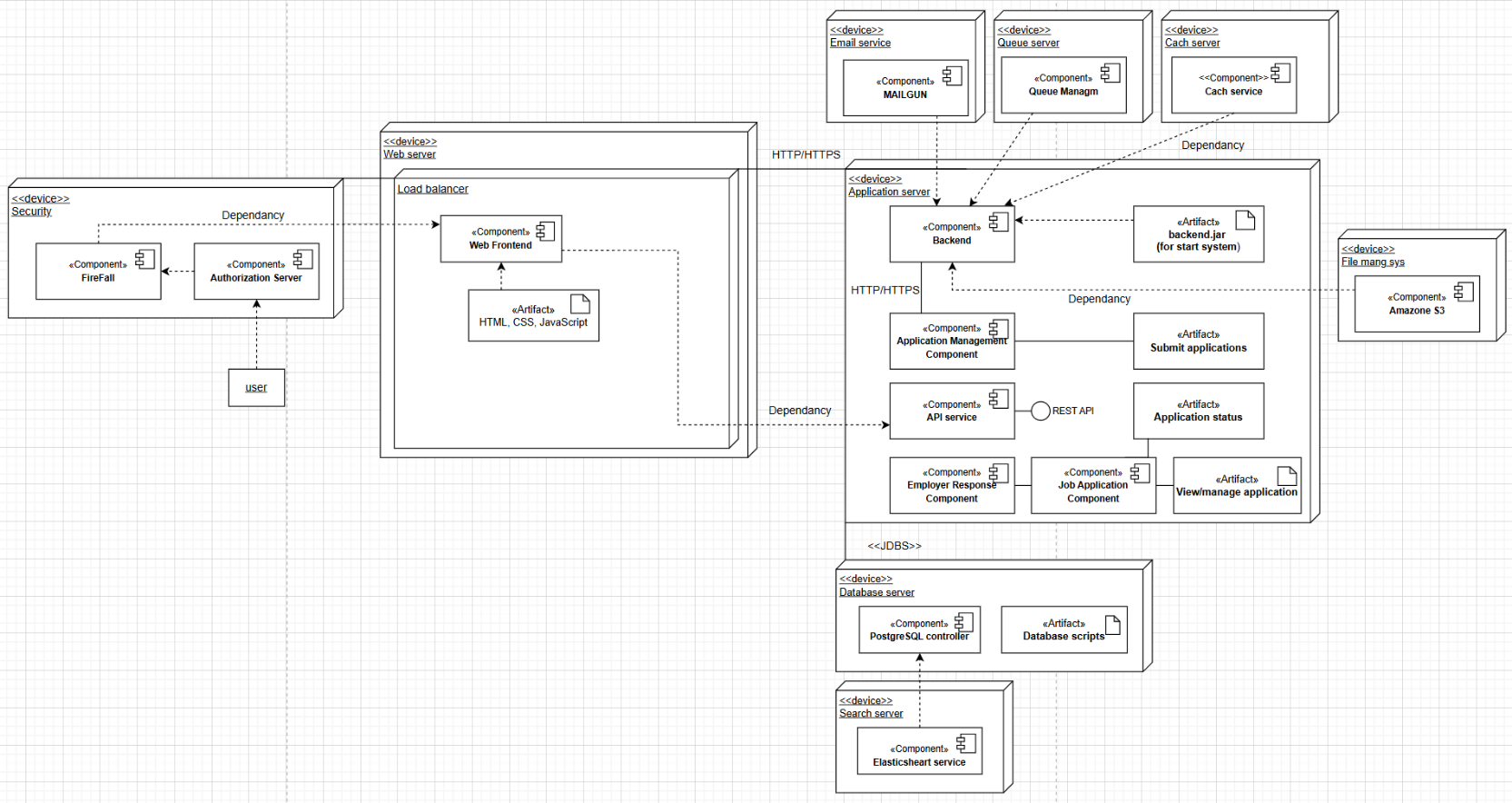
* Email Service (MAILGUN): Handles sending emails for notifications and confirmations.
* Queue Server: Manages asynchronous tasks and job queues.
* Cache Server: Provides caching to enhance performance.
* File Management System (Amazon S3): Manages file storage and backups.

**Interactions:**

* Web Frontend interacts with the Backend via the API Service using HTTP/HTTPS protocols.
* Backend communicates with the Database Server to store and retrieve data.
* Backend also interacts with the Email Service for sending emails, the Queue Server for handling background tasks, and the Cache Server for improving performance.
* Security Node ensures that all communications are secure, and the Authorization Server handles user authentication.
* Elasticsearch Service provides enhanced search capabilities by indexing the data and processing search queries.

**Dependencies:**

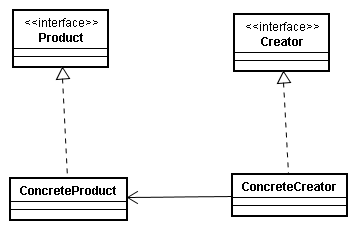
* The Web Frontend relies on the Load Balancer for distributing traffic.
* The Backend depends on the API Service for processing requests from the Web Frontend.
* The Database Server and its components are essential for data storage and retrieval.
* The Email Service, Queue Server, Cache Server, and File Management System are auxiliary services that support the primary functions of the JSE system.



13. Choose the best pattern

1. Creational pattern name: **Factory Method**

* Problem description: Мне необходимо создавать разные типы объектов, такие как "Job Seeker", "Employer", "Admin", без жесткой привязки к конкретным классам. Жесткие связи затрудняют гибкость и расширяемость проекта.
* Problem solution: Использую Factory Method, который позволяет создать общий фабричный метод для создания объектов, не указывая конкретный класс. Например, при регистрации нового пользователя фабричный метод возвращает объект нужного типа в зависимости от выбранной роли.
* Advantage of this pattern for your problem: Это упрощает добавление новых типов пользователей в будущем и делает систему более гибкой, поскольку не нужно менять основной код для добавления новых классов.
* Disadvantage of this pattern for your problem: При необходимости добавления новых типов объектов может возникнуть сложность в поддержке кода. Также использование фабричного метода может затруднить понимание кода для новых разработчиков.
* Graphical representation (class diagram):



1. Creational pattern name: **Abstract Factory**

* Problem description: В моем проекте нужно создавать семейства связанных объектов, таких как аккаунты для разных ролей (Job Seeker, Employer, Admin), при этом сохраняя согласованность между ними.
* Problem solution: Применяю Abstract Factory, который позволяет создать фабрику для создания семейств связанных объектов. Это дает возможность создавать аккаунты с согласованными интерфейсами, что обеспечивает целостность и унификацию.
* Advantage of this pattern for your problem: Abstract Factory обеспечивает согласованность между связанными объектами и позволяет легко добавлять новые семейства. Он также способствует поддержанию хорошей структуры кода.
* Disadvantage of this pattern for your problem: Увеличивается сложность при добавлении новых семейств объектов, и поддержка кода может стать сложнее. Кроме того, Abstract Factory требует большего числа классов, что может увеличить сложность проекта.
* Graphical representation (class diagram):

Изображение выглядит как текст, диаграмма, линия, План

Автоматически созданное описание

1. Structural patterns name: **Adapter**

* Problem description: Мой проект включает разные модули и системы, которые могут быть несовместимы из-за различий в интерфейсах. Например, если я хочу интегрировать модуль обработки заявлений с базой данных, интерфейсы могут не совпадать.
* Problem solution: Применяю шаблон Adapter, который создает промежуточный класс для согласования несовместимых интерфейсов. Это позволяет интегрировать разные системы, обеспечивая гибкость и унификацию.
* Advantage of this pattern for your problem: Adapter упрощает интеграцию различных модулей и систем, делая их совместимыми. Это также позволяет использовать существующие классы без необходимости их изменения.
* Disadvantage of this pattern for your problem: Adapter может усложнить структуру проекта, увеличив число классов и связей. Также это может замедлить производительность, поскольку адаптеры добавляют дополнительный уровень обработки.
* Graphical representation (class diagram):

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

1. Structural patterns name: **Decorator**

* Problem description: В моем проекте возникает необходимость добавлять новые функции к существующим классам, не изменяя основной код. Например, можно добавить дополнительные привилегии к аккаунтам пользователей.
* Problem solution: Применяю шаблон Decorator, который позволяет оборачивать базовый класс, добавляя новые функции. Это обеспечивает динамичное расширение без изменения базового кода.
* Advantage of this pattern for your problem: Decorator позволяет добавлять функционал к существующим классам, сохраняя основной код неизменным. Это обеспечивает гибкость и упрощает расширение системы.
* Disadvantage of this pattern for your problem: Decorator может усложнить структуру кода, особенно при большом числе оберток. Кроме того, он может затруднить понимание кода для новых разработчиков из-за дополнительного уровня оборачивания.
* Graphical representation (class diagram):

Изображение выглядит как текст, диаграмма, линия, Параллельный

Автоматически созданное описание

1. Behavioral patterns name: **Observer**

* Problem description: В моем проекте требуется уведомлять об изменениях в системе, например, когда изменяется статус вакансии или приходит новое сообщение.
* Problem solution: Использую шаблон Observer, который позволяет создавать список наблюдателей и уведомлять их об изменениях. Это обеспечивает реактивность системы и уменьшает жесткие зависимости.
* Advantage of this pattern for your problem: Observer позволяет уведомлять о событиях и изменениях без жестких связей между классами, что делает систему более гибкой. Это также упрощает добавление новых наблюдателей или событий.
* Disadvantage of this pattern for your problem: Observer может увеличить сложность структуры, а при большом количестве наблюдателей может возникнуть проблема с производительностью. Кроме того, управление наблюдателями может потребовать дополнительного внимания.
* Graphical representation (class diagram):

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

1. Behavioral patterns name: **State**

* Problem description: В проекте требуется изменение поведения в зависимости от состояния, например, при переходе заявлений между разными этапами обработки.
* Problem solution: Применяю шаблон State, который позволяет изменять поведение системы в зависимости от текущего состояния. Это обеспечивает гибкость и упрощает переходы между состояниями.
* Advantage of this pattern for your problem: State позволяет управлять поведением системы, делая переходы между состояниями более управляемыми. Это также облегчает добавление новых состояний и переходов.
* Disadvantage of this pattern for your problem: State может усложнить код при большом числе состояний или переходов между ними. Кроме того, поддержка кода может стать сложнее из-за увеличения числа состояний.
* Graphical representation (class diagram):

Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание

14. APPLYING architectural styles

**Selection of Architectural Styles**

The microservice style is chosen for the JSE project because it provides:

* **Modularity:** Each service is responsible for a specific piece of functionality, making it easier to develop, test and deploy.
* **Scalability:** The system can be easily scaled by adding or replacing services as needed.
* **Flexibility:** Components can communicate through asynchronous mechanisms such as NATS Streaming, allowing the development of independent services.

**Description of Architecture**

The diagram shows a JSE architecture that utilizes microservices for various tasks:

1. **JSE UI**: This is the frontend of the project. It can be implemented as a web or mobile application that provides users with access to JSE functionality. It interacts with the JSE Service to process requests.
2. **JSE Service**: This service serves as a central point for processing requests from the JSE UI and coordinates communication between other microservices. It receives requests from the frontend and distributes them to the corresponding services, such as User Management Service or Jobs Service.
3. **User Management Service**: This service is responsible for user management, including registration, authentication, and profile management. It interacts with JSE DB to store user data.
4. **Jobs Service**: This service is responsible for managing vacancies. It can handle the addition, editing and deletion of vacancies, as well as interact with JSE DB to store information about vacancies.
5. **Admin Service**: This service is designed for administrative tasks such as access rights management and system monitoring. It can interact with other services to perform its functions.
6. **Auth Service**: This service is responsible for authenticating users. It can issue tokens or other means of authentication, ensuring the security of the system.
7. **JSE DB**: It is a database used to store user, job and administrative information. It provides a secure repository for the JSE system.
8. **NATS Streaming**: It is a messaging system that allows services to communicate asynchronously. Services can send and subscribe to messages, which provides flexibility and reliability.

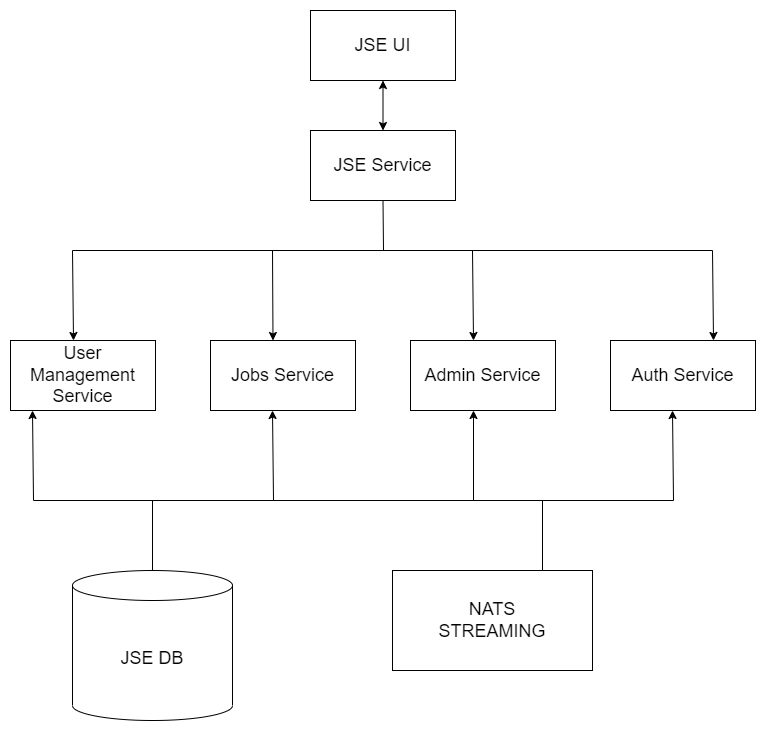
**Interaction between Components**

The diagram shows the interaction between the different services:

* **The JSE UI** sends requests to the JSE Service, which coordinates interaction with other services.
* **JSE** Service interacts with User Management Service, Jobs Service, Admin Service and Auth Service to handle requests related to users, jobs, administration and authentication.
* **JSE DB** is used for data storage and NATS Streaming provides asynchronous communication between services.

**Graphical Representation of Architecture**

The above diagram shows the microservice architecture of JSE:



**Conclusion**

Applying a microservice style to a JSE project provides modularity, flexibility and scalability. Services can be developed and deployed independently of each other, and communication is accomplished through asynchronous mechanisms such as NATS Streaming and REST APIs. The JSE DB provides a robust data repository.