SDT Project details

You can work in teams of 3. Only one person has to make an account on https://sdt.rusudinu.ro/ to upload each milestone.

The milestone will be uploaded as a link to a ***public*** github repository, each milestone being on a different branch named milestone-number-milestone-name. (example: 1-teams-and-project-description) You can pick a fitting milestone name, just make sure you put the number as a prefix. Make sure your repository is public, otherwise it won't be graded.

For the first project meeting you must choose your project team. You're free to finish milestones ahead of schedule and present them during project hours. Online submission alone will not count for grading — all team members must be present and explain their individual contributions to the project.

Milestones starting from week 3:

Milestone 1

Choose your project theme and the design patterns you will use:

- For this first milestone you must pick your project theme. Write a readme.md file containing the name of each team member and a description of the project you want to implement. (For a reference, you can consult the projects on page 5 for inspiration, but you cannot choose one of them directly. Every team must come up with its own unique project idea.) Make sure your project description is well written and includes all the details needed in order to implement the milestones.
- Identify at least four design patterns that will be utilized in the project.*
- For each selected design pattern, provide a brief explanation justifying its use. This should detail how the pattern solves a specific problem and the advantages it offers over simpler alternatives.

Grading Criteria:

Grade 5:

- A readme.md file is present with team member names and a basic project description.
- At least two design patterns are listed.
- A minimal attempt is made to justify the use of the design patterns.

Grade 7:

- The readme.md file contains a clear and understandable project description.
- At least three appropriate design patterns are identified.
- The justification for each pattern explains its purpose within the project, but may lack a detailed comparison to alternatives.

Grade 10:

- The readme.md provides a well-written, detailed project description that clearly outlines all necessary aspects for implementation.
- At least four relevant and non-trivial design patterns are selected.
- The justification for each pattern is thorough, clearly linking it to specific project requirements and effectively arguing its benefits over simpler approaches.

Milestone 2

Design and the implementation of the chosen project, using at least 4 non-trivial design patterns.

- Design a single, comprehensive UML class diagram for the entire project.
- To highlight the pattern, you can use a note in UML, connected to all the classes involved in that pattern. Try to have a single design class diagram, do not treat the design patterns separately, instead they should be integrated in the entire app design.
- Create two detailed UML sequence diagrams.
- These diagrams should illustrate the main use cases of the application.

- Implement the design patterns discussed previously. Implementation in Java should be based on the design.
- The implementation will be limited in scope, focusing on selected functionalities to showcase design patterns. Focus on showcasing core functionalities rather than building the complete system. Limit the implementation to a proof-of-concept scope to demonstrate how the patterns interact with one another.

Grading Criteria:

Grade 5:

- A basic UML class diagram is provided, but it may be incomplete or contain inaccuracies.
- An attempt has been made to identify design patterns in the class diagram.
- One or two simple UML sequence diagrams are created.
- A partial Java implementation exists, but the demonstration of design patterns is unclear or flawed.

Grade 7:

- A complete and mostly correct UML class diagram.
- The design patterns are correctly identified in the class diagram.
- Two UML sequence diagrams correctly depict the main use cases.
- The Java implementation correctly demonstrates at least two of the chosen design patterns with some interaction.

Grade 10:

- A detailed and well-structured UML class diagram that covers the entire proposed system.
- All four design patterns are clearly and correctly annotated in the UML diagram.
- Two comprehensive and accurate UML sequence diagrams illustrate user interactions.
- The Java implementation is well-structured and effectively showcases the implementation and interaction of all four design patterns in a proof-of-concept.

Milestone 3

Investigate, describe and evaluate three different software architectures for your system, each one using different architectural styles:

- Monolithic architecture
- Microservices architecture
- Another distributed architecture style of your choice (e.g. peer-to-peer, event-driven etc.)

For each architecture, you must:

- Describe how the system would be structured in that style (main components, interactions, and data flow).
- Provide a deployment and component to illustrate the architecture.
- Discuss the pros and cons of using this style for your project.
- Finally, compare the three alternatives and explain which one you consider the most suitable for your project, and why.

Grading Criteria:

Grade 5:

- A brief description of the three architectures is provided.
- Diagrams are either missing or very simplistic.
- A superficial discussion of pros and cons is included.
- A choice of architecture is stated but with minimal justification.

Grade 7:

- Each architecture is described with a reasonable level of detail.
- Clear component and deployment diagrams are provided for each style.
- The pros and cons for each architecture are relevant to the project.
- The final selection is justified with some valid points.

Grade 10:

- The descriptions of all three architectures are detailed, accurate, and tailored to the specifics of the project.
- The diagrams are clear, professional, and accurately reflect the described structures.
- The pros and cons analysis is in-depth, insightful, and directly relates to the project's requirements.
- The final comparison is thorough, and the choice of the most suitable architecture is convincingly argued.

Milestone 4

Implement your project using a microservices architecture. The implementation should:

- Contain at least three independent services, each with a clearly defined responsibility.
- Demonstrate inter-service communication (e.g., via REST APIs, gRPC, or message queues).

- Use Postman to test the microservices, provide in your submission a collection of postman requests covering all features.
- Use Docker to package and run the services. Provide in Readme a clear instruction on how to run the system.

Focus on implementing core functionalities that showcase how the system works in a microservices style, rather than building every feature in full detail.

Grading Criteria:

Grade 5:
- At least two services are implemented.
- Basic inter-service communication is present but may be unreliable.
- A partial Postman collection is provided.
- Docker is used, but the setup instructions are incomplete or contain errors.
- The application compiles and starts without errors.

Grade 7:
- Three independent services with clear responsibilities are implemented.
- Inter-service communication is functional and correctly implemented.
- A Postman collection that tests most of the features is provided.
- The system can be run successfully using the provided Docker instructions.

Grade 10:
- Three or more well-defined and independent services are implemented.
- Inter-service communication is robust and efficiently implemented.
- A comprehensive Postman collection covers all functionalities and edge cases.
- The Docker setup is clean, efficient, and the instructions in the Readme.md are clear and easy to follow.

Milestone 5

Extend your microservices implementation with:
Message Queue Integration
- Use a message queue (e.g., RabbitMQ, Kafka, or another suitable technology) for asynchronous communication between at least two services.
- Demonstrate how the queue improves scalability, decoupling, or fault tolerance in your system.

CI/CD Pipeline

- Set up a simple continuous integration and deployment (CI/CD) pipeline (e.g., using GitHub Actions, GitLab CI, or Jenkins).
- The pipeline should automatically build, test, and deploy your services (at least to a local Docker environment).

Provide documentation in your README.md showing:

- How the message queue is integrated and what role it plays in the architecture.
- How the CI/CD pipeline is configured and how to run/observe it.

Grading Criteria:

Grade 5:
- A message queue is included, but its role and benefits are not clearly demonstrated.
- A basic CI/CD pipeline is configured that may only perform a build or test step.
- Minimal documentation is provided for the new features.

Grade 7:
- A message queue is successfully integrated for asynchronous communication between two services.
- A CI/CD pipeline is set up that automatically builds and tests the services.
- The README.md provides a decent explanation of the message queue and CI/CD setup.

Grade 10:
- The message queue is effectively used for asynchronous communication, and the benefits (e.g., improved scalability, fault tolerance) are clearly demonstrated and explained.

- A complete CI/CD pipeline is implemented that automatically builds, tests, and deploys the services to a local Docker environment upon code changes.
- The documentation in the README.md is clearly explaining the message queue architecture and providing easy-to-follow instructions for the CI/CD pipeline.

* Hint: Write down the list of all OOP design patterns then try to find matches with your project statement.
https://refactoring.guru/design-patterns/catalog
More than likely you will use one or two creational design patterns (Factory, Builder, Singleton) and a Behavioral pattern such as Strategy or Observer.

1. Healthcare Management System

Problem Statement: The goal of this project is to develop a comprehensive National Healthcare Management System that integrates various components to streamline healthcare processes, enhance patient care, and ensure efficient data management. The project aims to compare three architectural approaches - monolithic, containerized architecture and microservices- evaluating them based on data security measures, interoperability, and deployment flexibility.

- Create a healthcare distributed system that integrates various applications of the healthcare providers in the territory, including electronic medical records (EMRs) in the GP's offices*), pharmacies, analysis laboratories, and patient portals.

- Patients' electronic healthcare records (EHRs), including health issues and prescriptions history, as well as medication data should be securely stored, continually updated from peripheral applications, and shared across healthcare providers.
- Assure robust data security measures and explore interoperability with external healthcare systems.
- Implement functionalities for patient records, appointment scheduling, and billing such as in the followings:

Functionalities:

1. **Patient Records:**
   - Create, update, and retrieve patients' electronic health records.
   - Ensure secure storage and sharing of patient records across healthcare providers and pharmacies.
   - Implement authentication and authorization mechanisms to control access.
2. **Appointment Scheduling:**
   - Develop a user-friendly interface for scheduling and managing appointments.
   - Enable real-time updates and notifications for appointments.
   - Ensure the system supports different types of appointments and services.
3. **Billing:**
   - Implement billing functionalities, including invoicing, payment tracking, and financial reporting.
   - Adhere to healthcare billing standards and regulations to ensure compliance.
   - Provide a secure and transparent billing process for patients and healthcare providers.
4. **Prescription History and Medication Data:**
   - Track and maintain a comprehensive prescription history for each patient.
   - Include functionalities for managing medication data, including dosage and instructions.
   - Ensure data accuracy and integrity in prescription history and medication records.

2. Online E-Commerce Platform

Problem Statement: The project objective is to develop a comprehensive and robust platform that manages online selling costs in a strategic way, establishes deeper business relationships, provides a unique customer experience, improves customer loyalty, and refines service efficiency. The platform implements user authentication, product catalog management, and order processing functionalities. The project aims to compare three architectural approaches - monolithic, microservices, and event-driven - evaluating them based on user experience, transaction processing and integration with external services.

- The platform may optionally implement common e-commerce features:
  - Product recommendations
  - User reviews and ratings
  - Wishlist functionality
  - Discounts and promotions
  - Account management and order history.
- The platform software has to emphasize scalability and fault tolerance.
- The platform software implements functionalities such as in the followings:

Functionalities:

1. **User Registration**:
   - Implement user registration and authentication functionalities.
   - Ensure secure storage of user data and sensitive information.
   - Provide a seamless and user-friendly registration process.
2. **Product Catalog**:
   - Develop a comprehensive product catalog with categories and product details.
   - Implement search and filter functionalities for users to easily find products.
   - Ensure efficient handling of product data, including images and descriptions.
3. **Shopping Cart**:
   - Implement a shopping cart system for users to add, remove, and manage selected items.
   - Ensure real-time updates on product availability and pricing.
   - Provide a smooth and intuitive shopping experience.
4. **Order Processing**:
   - Develop a robust order processing system, including order confirmation and payment processing.
   - Implement secure transaction handling and integrate payment gateways.
   - Provide order tracking functionalities for users.

2. Travel Booking System

Problem Statement: The objective of this project is to create a comprehensive Travel Booking System that allows users to search, book, and manage travel reservations. The system should implement secure payment processing and integrate seamlessly with external travel APIs. The project aims to compare three architectural approaches- monolithic, microservices, and event-driven- evaluating them based on system responsiveness, transaction processing, and ease of integration with external services.

Functionalities:

1. **User Registration and Profile:**
   - Implement user registration and profile management for travelers.
   - Provide a personalized user dashboard for managing bookings and preferences.
   - Ensure secure storage of user information.
2. **Travel Reservation:**
   - Develop a user-friendly interface for searching and booking travel reservations.
   - Implement features for filtering search results based on preferences.
   - Provide real-time availability and pricing information.
3. **Secure Payment Processing:**
   - Implement secure payment processing for booking transactions.
   - Integrate with reputable payment gateways to ensure financial transactions are secure.
   - Comply with industry standards for online payment security.
4. **Integration with External Travel APIs:**
   - Integrate with external travel APIs for fetching real-time data on flights, hotels, and other travel services.
   - Ensure seamless communication and data exchange with external services.
   - Explore APIs for travel itinerary recommendations and additional services.

3. Project Management Tool

Problem statement: The objective of this project is to design a comprehensive Project Management Tool with functionalities for task management, collaboration, and reporting. The project aims to investigate three architectural choices - monolithic, microservices, and containerized architecture - implementing features that enhance team collaboration, ensure data consistency, and streamline deployment efficiency. The impact on project management workflows will be analyzed for each architecture.

Evaluate each architecture based on team collaboration, data consistency, and deployment efficiency. Analyze the impact on project management workflows.

Functionalities:

1. **Task Management:**
   - Implement features for creating, assigning, and tracking tasks within projects.
   - Provide task prioritization, status tracking, and deadline management functionalities.
   - Explore task dependencies and subtasks for complex project structures.
2. **Collaboration Features:**
   - Develop collaboration features to facilitate communication within project teams.
   - Implement real-time messaging, commenting, and file sharing functionalities.
   - Explore integration with communication tools for seamless collaboration.
3. **Reporting and Analytics:**
   - Implement reporting features for tracking project progress, milestones, and timelines.
   - Provide analytics on team performance, task completion rates, and project timelines.
   - Ensure customizable reporting options to meet various project management needs.

4. Inventory Management System

Problem statement: The objective of this project is to build a robust Inventory Management System with functionalities for inventory tracking, order processing, and reporting. The project aims to implement user authentication, inventory CRUD (Create, Read, Update, Delete) operations, and order fulfillment workflows. Additionally, three architectural choices—monolithic, microservices, and event-driven—will be compared based on system reliability, real-time updates, and ease of adapting to changing business requirements. The impact on inventory management workflows will be thoroughly analyzed.

Functionalities

1.User Authentication:

Implement a secure user authentication system with role-based access control.
Define roles such as admin, manager, and staff, each with specific permissions.
Ensure data security and privacy through robust authentication mechanisms.

2.**Inventory CRUD Operations:**

Develop CRUD functionalities for managing inventory items.
Enable features for adding, updating, viewing, and deleting items.
Implement categorization and tagging for efficient inventory organization.

3.Order Processing and Fulfillment:

Implement order processing workflows, including order creation, status tracking, and fulfillment. Enable real-time updates on order status and inventory levels.
Implement features for order cancellation and refunds.

4.Reporting and Analytics:

Implement reporting features for monitoring inventory levels, sales, and order fulfillment.
Provide analytics on popular products, slow-moving items, and order trends.
Ensure customizable reporting options to meet business intelligence needs.