Todo list - simple task tracker

Project repository

Created by: Matvei Morenkov, Egor Greb, Dmytro Kovalov, Grigorii Voronchikhin

Summary

Summary.. 1
Application description.. 2
Motivation.. 2
Project strategic plan.. 3
Business benefit.. 3
SWOT analysis.. 4
Porter's Five Forces (5F).. 5
PEST analysis.. 7
Functional Requirements.. 8
Non-functional Requirements.. 8
Users.. 9
Use case diagram.. 10
UML diagrams.. 11
Architecture selection.. 14

Application description

The Todo List application is a comprehensive task management system designed to facilitate efficient organization and tracking of tasks for individuals and teams. Inspired by the functionality of popular platforms like Trello, this project employs a microservices architecture to provide robust features while ensuring scalability and flexibility.

The main component, todo-list-api, serves as the backbone of the system, offering a RESTful API for managing users and tasks securely. Users can register, log in, and utilize various task-related functionalities such as task creation, editing, reading, and deletion. Authentication is implemented using JWT tokens, ensuring secure access to the system's functionalities.

Complementing the core functionality are two microservices: todo-list-email-sender and todo-list-scheduler. The email sender service integrates with RabbitMQ to receive messages from the scheduler and backend service. It leverages Spring Mail to send notifications and updates to users via email, enhancing communication and keeping users informed about task-related activities. Meanwhile, the scheduler service, powered by Spring Scheduler and Spring AMQP, automates the generation of daily reports for users and triggers email notifications accordingly.

Motivation

In summary, the motivation for creating this application is to empower individuals and teams with a user-friendly, feature-rich task management solution that fosters collaboration, enhances productivity, and simplifies the complexities associated with organizing and tracking tasks effectively.

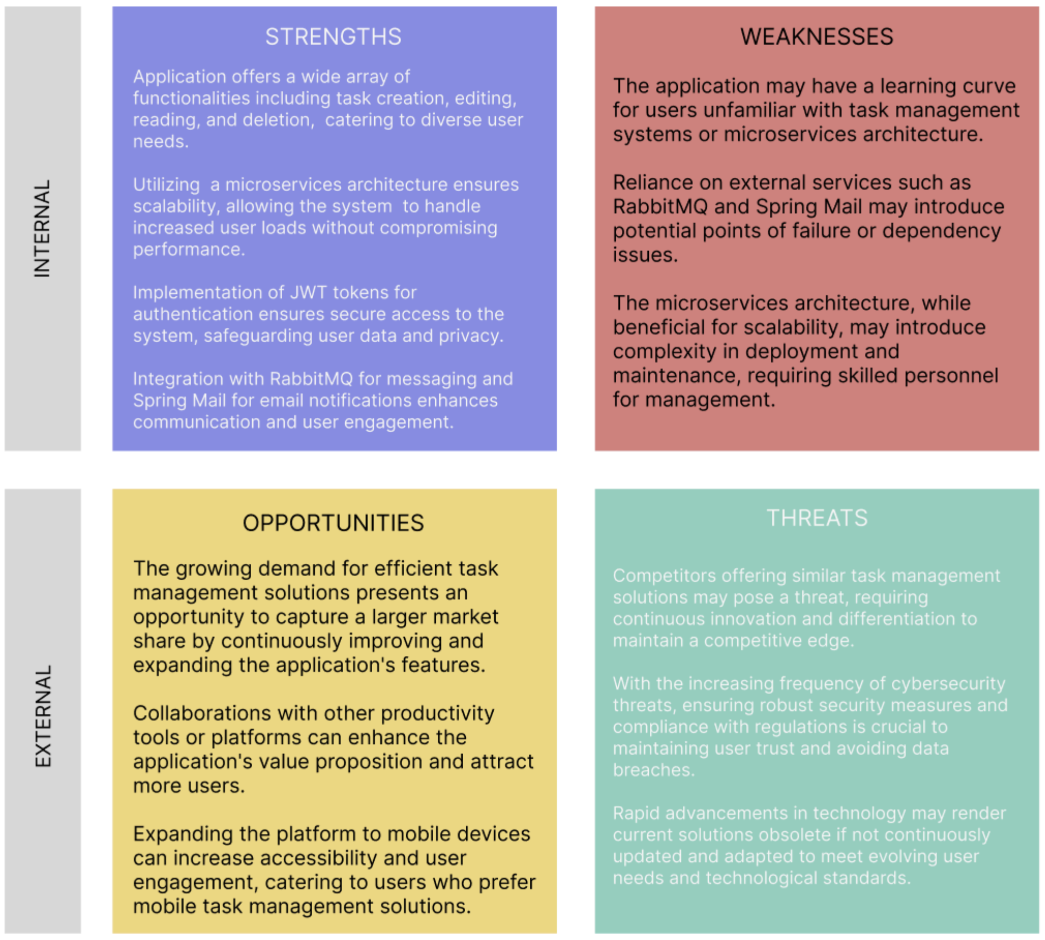
Project strategic plan

Our project aims to develop a leading task management platform, empowering users to organize tasks efficiently and collaborate seamlessly. We will achieve this mission by building a scalable and reliable system using modern technologies such as Java, Spring Boot, and Docker. Our goals include providing a user-friendly interface, implementing robust security measures, and fostering collaboration through real-time updates and task sharing. Market analysis will inform our feature development and marketing strategies to ensure we meet user needs and stay ahead of competitors.

Business benefit

Our platform increases productivity by streamlining task management, saving time and effort for users. Enhanced collaboration features facilitate better communication and coordination among team members, driving efficiency. By automating task reminders and notifications, our platform helps organizations save costs associated with manual follow-ups. We offer a competitive advantage by providing a feature-rich and user-friendly solution, attracting and retaining users. Robust security measures ensure the safety and privacy of user data, fostering trust and compliance with regulations.

SWOT analysis



INTERNAL

STRENGTHS

Application offers a wide array of functionalities including task creation, editing, reading, and deletion, catering to diverse user needs.

Utilizing a microservices architecture ensures scalability, allowing the system to handle increased user loads without compromising performance.

Implementation of JWT tokens for authentication ensures secure access to the system, safeguarding user data and privacy.

Integration with RabbitMQ for messaging and Spring Mail for email notifications enhances communication and user engagement.

WEAKNESSES

The application may have a learning curve for users unfamiliar with task management systems or microservices architecture.

Reliance on external services such as RabbitMQ and Spring Mail may introduce potential points of failure or dependency issues.

The microservices architecture, while beneficial for scalability, may introduce complexity in deployment and maintenance, requiring skilled personnel for management.

EXTERNAL

OPPORTUNITIES

The growing demand for efficient task management solutions presents an opportunity to capture a larger market share by continuously improving and expanding the application's features.

Collaborations with other productivity tools or platforms can enhance the application's value proposition and attract more users.

Expanding the platform to mobile devices can increase accessibility and user engagement, catering to users who prefer mobile task management solutions.

THREATS

Competitors offering similar task management solutions may pose a threat, requiring continuous innovation and differentiation to maintain a competitive edge.

With the increasing frequency of cybersecurity threats, ensuring robust security measures and compliance with regulations is crucial to maintaining user trust and avoiding data breaches.

Rapid advancements in technology may render current solutions obsolete if not continuously updated and adapted to meet evolving user needs and technological standards.

Picture 1 - SWOT analysis

Porter's Five Forces (5F)

Parameter	Significance	Description	Solution
Threats of substitutes emergence	Insignificant	There is no substitute for ToDo lists.	There is no need.
Existing competitors	High	The ToDo app market is highly competitive with many existing players competing for market share. Competition can lead to lower revenues or increased investment in innovation and features to differentiate from competitors.	Focus on building a strong brand reputation and customer base primarily through the development of new productivity methods, targeted marketing efforts and partnerships with existing companies. Differentiate ourselves by providing unique features or benefits to customers and exploring partnerships to provide more comprehensive solutions.
Threats of new competitors entering the market	High	The ToDo app market is already established with many well-known players, but the ToDo app is such a trivial idea that almost anyone can create a similar one for little cost.	
Bargaining power of customers	High	Many free ToDo apps are available to buyers, making it easy for them to switch to competitors. Buyers may have the bargaining power to negotiate lower fees or better service.	Implement flexible pricing strategies and provide added value to customers to increase loyalty. Focus on building a strong brand and reputation to attract and retain customers.

Bargaining power of suppliers	Low	There are many technology vendors and service providers from which ToDo app developers can choose, giving them bargaining power to negotiate better prices and services.	Work to build strong relationships with key suppliers and negotiate favourable contracts to reduce costs. Explore alternative suppliers to reduce dependence on one supplier.

PEST analysis

Political factors	Changes in laws and regulations concerning personal data protection, such as the GDPR regulation in Europe, can impact how to-do applications collect, store, and process user data.
Economical factors	The ToDo system project is likely to be influenced by economic conditions such as inflation, interest rates, and exchange rates, which can have a significant impact on the project. Low inflation and a stable economic environment may signify favorable conditions for the development and operation of the project, while high inflation and economic uncertainty may pose challenges.
Sociocultural factors	Trends in work methods and lifestyles can impact the demand for ToDo systems and the requirements for their features (e.g., integration with mobile applications, task sharing capabilities, etc.).
Technological factors	The success of the ToDo system project will largely depend on technology and innovation. Rapid advancements in technology can influence the development of the ToDo system and its ability to integrate with new technologies and platforms. Additionally, changes in security standards and technologies can affect the ways in which data in the ToDo system are protected.

Functional Requirements

- FR01 - The system must allow the user to create a new account

  New users must be able to create an account and use the features of the application.

- FR02 - The system must allow the user to log in

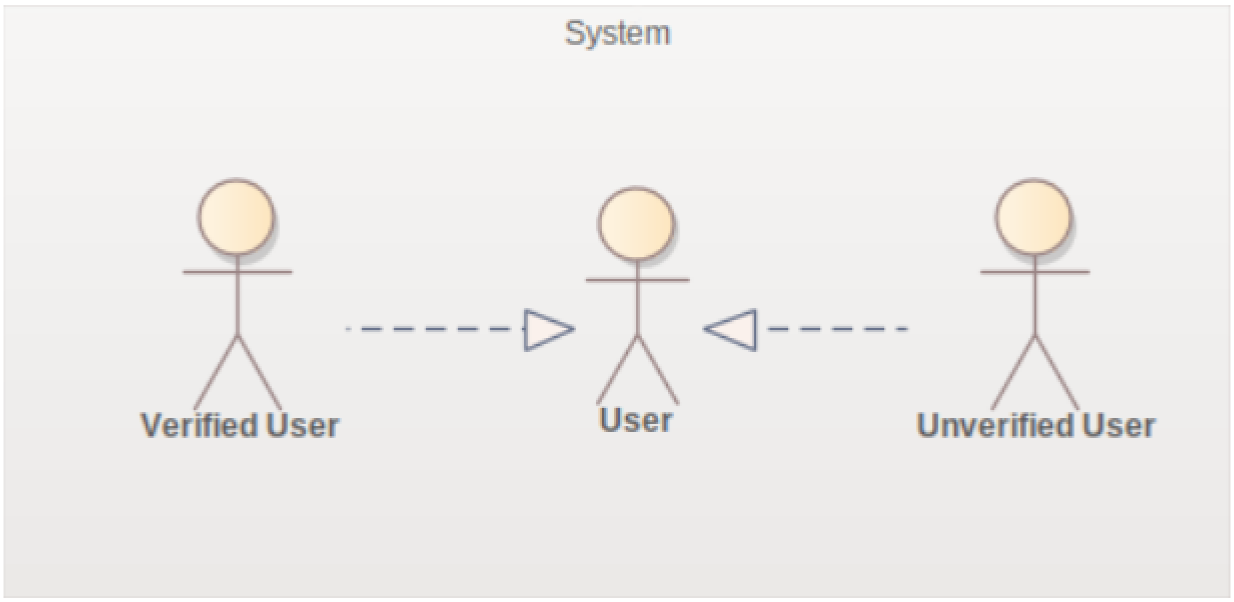
  Users who already have an account must be able to log in.

- FR03 - The system must allow a logged in user to log out.
- FR04 - The system must allow the logged in user to create tasks.
- FR05 - The system must allow the logged in user to edit the tasks.
- FR06 - The system must allow the logged in user to view existing tasks.
- FR07 - The system must allow the logged in user to delete tasks.
- FR08 - The system must allow the user to set up email notifications.

Non-functional Requirements

- NFR01 - The system must have a modern and pleasant appearance.
- NFR02 - The system should be easy to use and intuitive.
- NFR03 - The system should have a fast response time.
- NFR04 - The system should securely protect user data.
- NFR05 - The system should be compatible with different devices, platforms and browsers.

Users



Picture No. 2 – System users

The system will support 2 types of users: Verified user and Unverified user. A verified user has the ability to create, edit, view tasks, mark tasks as complete. The unverified user is limited in functionality. He has the opportunity to only register.

Verified user:

- Log In
- Make New ToDo
- View ToDo History
- Edit ToDo
- Mark ToDo as Complete

Unverified user:

- Register

Use case diagram

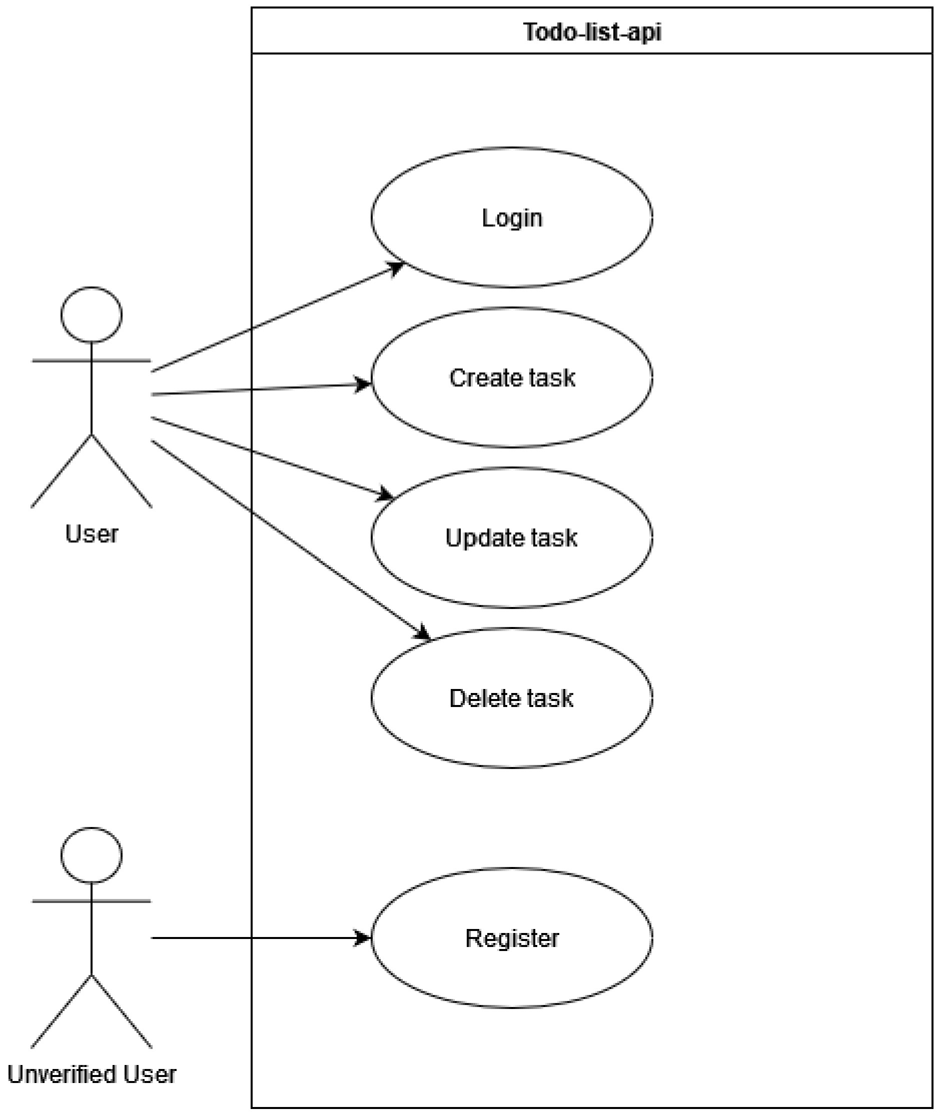


Figure 4.1 – Use case diagram

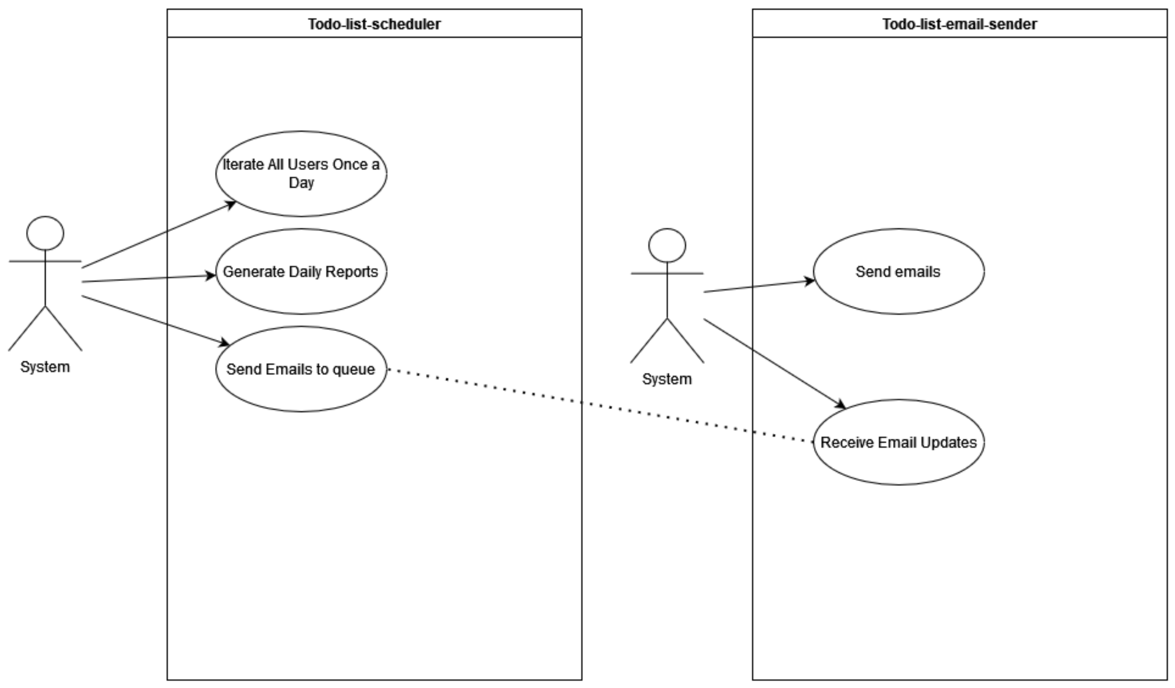
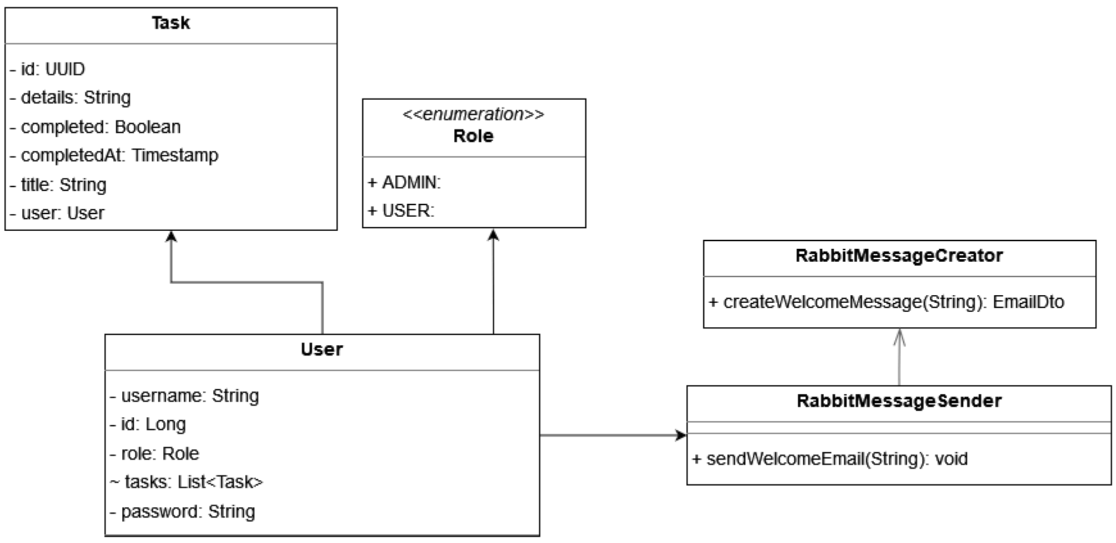
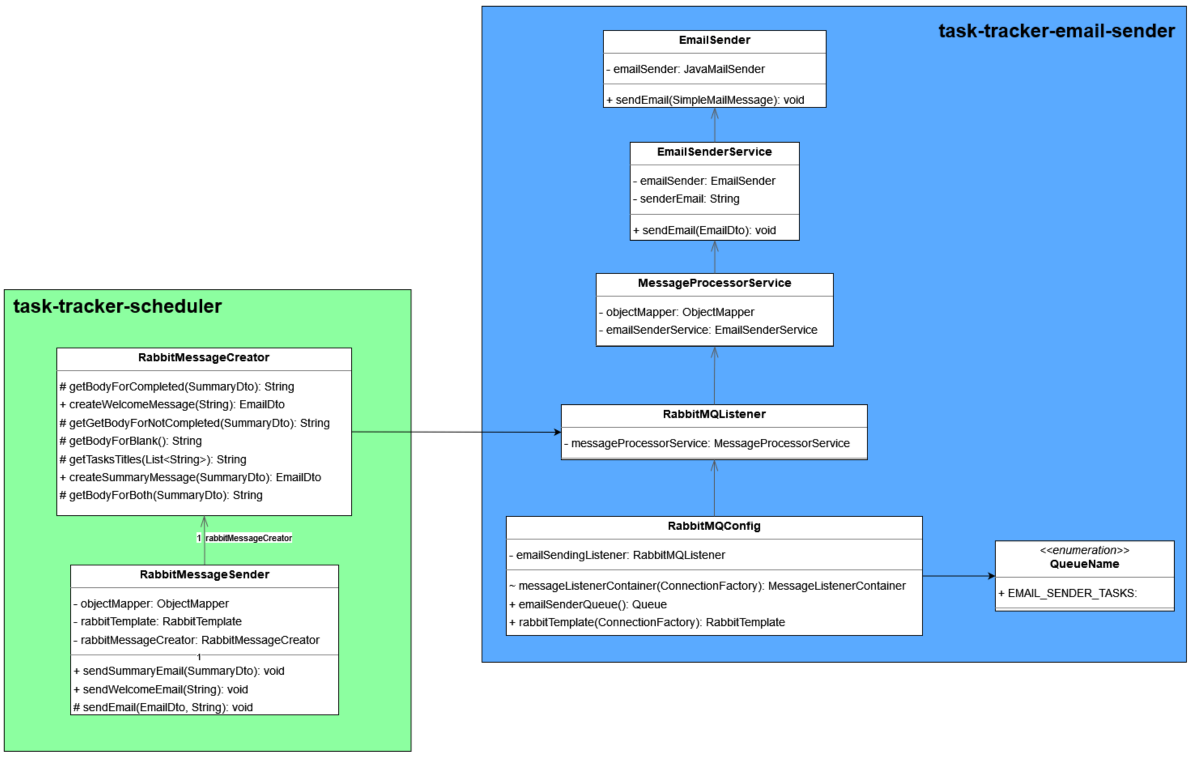


Figure 4.2 – Use case diagram

UML diagrams



Picture 4 – Api class diagram


task-tracker-email-sender
EmailSender
- emailSender: JavaMailSender
+ sendEmail(SimpleMailMessage): void
EmailSenderService
- emailSender: EmailSender
- senderEmail: String
+ sendEmail(EmailDto): void
MessageProcessorService
- objectMapper: ObjectMapper
- emailSenderService: EmailSenderService
RabbitMQListener
- messageProcessorService: MessageProcessorService
RabbitMQConfig
- emailSendingListener: RabbitMQListener
~ messageListenerContainer(ConnectionFactory): MessageListenerContainer
+ emailSenderQueue(): Queue
+ rabbitTemplate(ConnectionFactory): RabbitTemplate
<<enumeration>>
QueueName
+ EMAIL_SENDER_TASKS:
task-tracker-scheduler
RabbitMessageCreator
# getBodyForCompleted(SummaryDto): String
+ createWelcomeMessage(String): EmailDto
# getGetBodyForNotCompleted(SummaryDto): String
# getBodyForBlank(): String
# getTasksTitles(List<String>): String
+ createSummaryMessage(SummaryDto): EmailDto
# getBodyForBoth(SummaryDto): String
1 rabbitMessageCreator
RabbitMessageSender
- objectMapper: ObjectMapper
- rabbitTemplate: RabbitTemplate
- rabbitMessageCreator: RabbitMessageCreator
+ sendSummaryEmail(SummaryDto): void
+ sendWelcomeEmail(String): void
# sendEmail(EmailDto, String): void

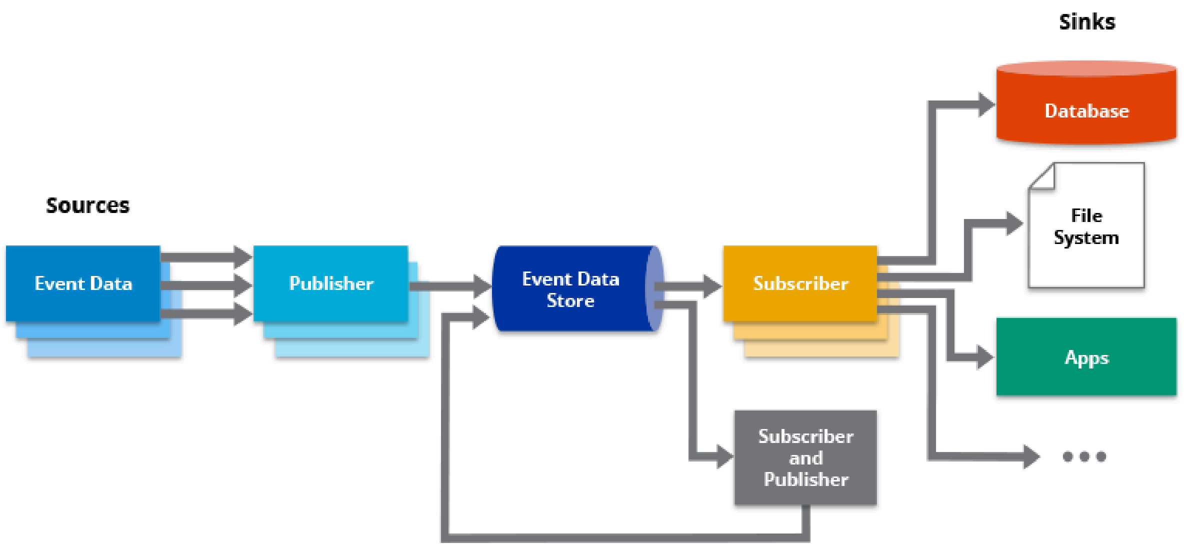
Picture 5 – Email sender and Scheduler class diagram

Architecture selection

The application's complexity and requirements suggest a **microservices architecture** as the foundation. This approach breaks down the system into smaller, manageable services, each responsible for specific tasks or functionalities. With microservices, scalability becomes easier as individual components can be scaled independently based on demand. Additionally, microservices offer flexibility in technology choices, allowing for the adoption of different tools and frameworks suited to each service's needs.

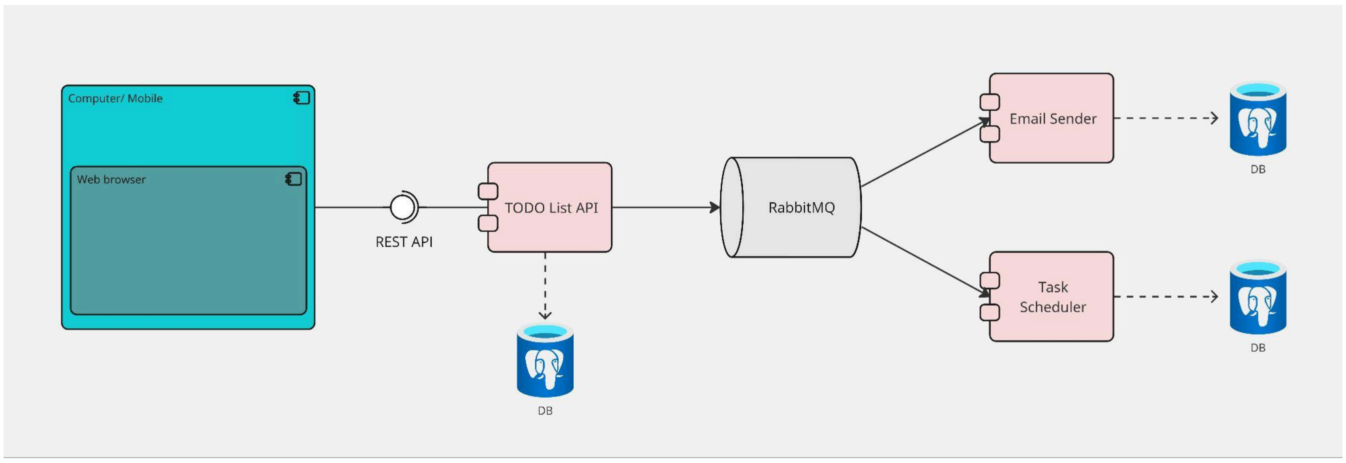
Complementing the microservices architecture, an **event-driven approach** using RabbitMQ or Kafka enhances system resilience and real-time capabilities. Events can be produced and consumed asynchronously, enabling services to communicate without direct dependencies. This decoupling improves fault tolerance and allows for efficient handling of tasks such as notifications and updates.

Furthermore, event-driven architecture facilitates real-time updates, which are crucial for features like task reminders. By broadcasting events across the system, users can receive timely notifications and stay updated on task-related activity



Picture 6 - Event Driven Architecture

Component Diagram


Computer/ Mobile
Web browser
REST API
TODO List API
DB
RabbitMQ
Email Sender
DB
Task Scheduler
DB