**Backend**

**Preferred processes and methodologies:**

* **Strongly typed programming languages:** These languages help detect errors during compilation rather than runtime, leading to more reliable code.
* **Microservices architecture:** This approach breaks down the application into smaller, independent services, making it easier to manage, scale, and deploy.
* **Vertical slicing:** This design pattern organizes services around business capabilities, ensuring each service has a clear and focused responsibility.
* **Event sourcing:** This pattern captures application state changes as events, enabling a more resilient and scalable event-driven architecture.
* **Containerization with Docker and Kubernetes:** This combination allows for easy deployment and management of applications in a containerized environment.

**Preferred tools and platforms:**

* **Docker:** A containerization platform for building, running, and managing applications in containers.
* **Kubernetes:** An open-source container orchestration platform for automating deployment, scaling, and management of containerized applications.
* **RabbitMQ or Service Bus:** Message brokers for asynchronous communication between microservices.

**Frontend**

**Preferred processes and methodologies:**

* **Component-based frontend frameworks:** Frameworks like React, Vue.js, or Angular provide a structured approach to building UI components, promoting modularity and reusability.

**Preferred tools and platforms:**

* **React, Vue.js, or Angular:** Popular JavaScript frameworks for building single-page applications.

**Evolution of existing products/apps**

The proposed approach, with its emphasis on microservices and decoupling the frontend from the backend, facilitates future evolution of existing products/apps. By adopting a modular architecture, new features and updates can be introduced without disrupting the entire system. Additionally, the use of standard technologies and open-source tools makes it easier to adopt new frameworks or services as they emerge.

**Additional considerations:**

* **Continuous integration and continuous delivery (CI/CD):** Implementing a CI/CD pipeline automates the build, testing, and deployment process, ensuring frequent and reliable delivery of updates.
* **Monitoring and observability:** Implementing monitoring tools and practices allows for continuous tracking of system health, performance, and potential issues.
* **Security:** Prioritizing security throughout the development lifecycle is crucial to protect applications from vulnerabilities and cyberattacks.

By adopting these processes, methodologies, tools, and platforms, you can build and evolve robust, scalable, and maintainable products/apps.