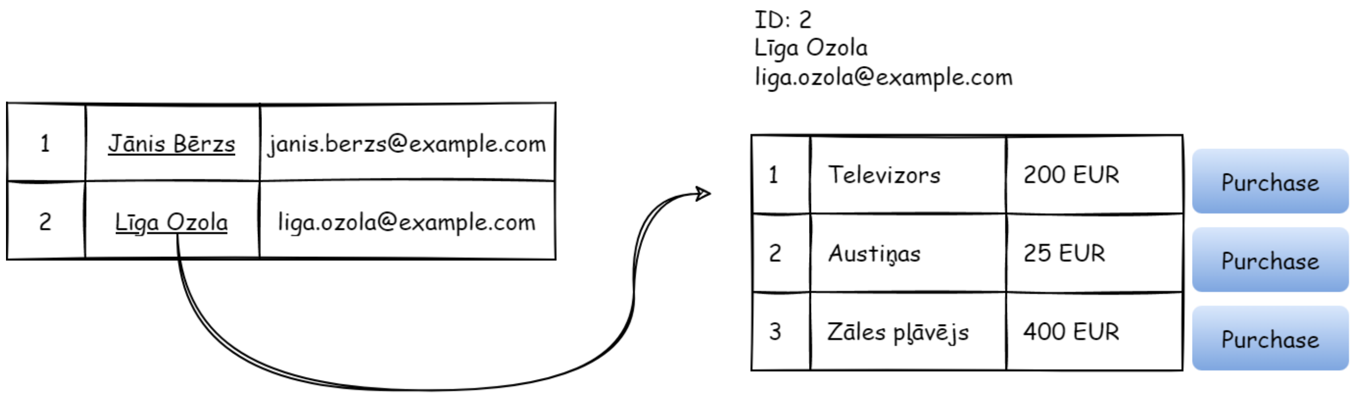
Developer Task



Build a small e-commerce type of application as a distributed system with the following services:

- User service
- Purchase service
- Email service

The data should be stored in persistent storage with a caching mechanism.

The **user** service should have the following API endpoints:

- **`GET /users`** that returns an array of user objects, each including the following user data: id, first name, last name, and e-mail address.
- **`GET /users/:id`** that returns the object with the data for the specific user.

The **purchase** service should have the following API endpoints:

- **`GET /items`** that returns an array of item objects, each including the following item data: id, name, and price.
- **`POST /purchase`** that accepts a payload with a reference to the user and the item to be purchased. The purchase is considered to be successful if the price of the item is lower than 50. The fact of purchase should be asynchronously published in a message broker or a distributed event system where it would be picked up by the **email** service.

The **email** service should be able to:

- Send a confirmation email to the user about a successful purchase if the price of the item is lower than 50.
- Send a rejection email to the user if the price of the item equals or is higher than 50.

Note that the service doesn't really need to send a real e-mail, but it should be possible to acknowledge the fact that the respective purchase has been processed.

The **frontend** of the application should consist of two pages:

- The home page with the list of users. It should be possible to view the details of each user in a different view.
- The user page with the user details and the list of available purchase items. It should be possible to purchase a single item.

Recommended technologies:

- Backend: PHP (Laravel, Symfony), Kafka, RabbitMQ, Redis, PostgreSQL (or any other persistent storage).
- Frontend: Angular, Vue.js, React, or any other SPA framework.

It should be possible to set up the entire system with a single Docker Compose file.