Jan Gniedziejko 193633

Franciszek Gajda 193200

Olga Granecka 193389

KEY-VALUE DATABASE PROJECT

System for order management for an e-commerce platform

BUSINESS DOMAIN

- **Order placement** – the system is given the information about the new order (products, the recipient)
- **Order Validation -** the system verifies order accuracy, stock availability
- **Stock Updates -** the system synchronizes with inventory systems to track product availability in real-time. This helps avoid issues such as overselling
- **Backordering and Pre-ordering -** if a product is temporarily unavailable, the system can manage backorders or facilitate pre-orders for upcoming stock.
- **Shipping Coordination -** once an order is confirmed, the system communicates with delivery partners to arrange shipping, track packages, and notify customers of delivery date.
- **Multi-Warehouse Management** - the system can route orders to different warehouses based on location, stock levels, and delivery speed requirements.
- **Notifications and Tracking** - automated emails and SMS updates keep customers informed at every step, from order confirmation to shipment and delivery.
- **Order Insights**: Detailed reports on order volumes, processing times, and fulfillment rates provide insights to optimize operations.

STAKEHOLDERS

- **Customers**: users placing orders and expecting efficient delivery.
- **E-commerce Business Owners**: oversee sales, inventory, and customer satisfaction.
- **Warehouse management**: handle physical product packing and shipping.
- **Customer Service Representatives**: address customer issues and assist with them.

- **Logistics Partners**: manage shipping and delivery of products.

CAP THEOREM

The system must be **available** so that users can continue placing orders, receive confirmations and interact with the system.
There might be temporary inconsistencies in data due to the lack of real-time synchronization during a partition.
Consistency is much more needed in case of handling financial transactions, which are not the topic of our system.

USAGE SCENARIO

An order is placed on the platform. The location of products that are in the order is determined. If the products are at different locations, it must be decided to which location the products must be relocated in order to assemble the order. Once every product is in one location, the delivery company is notified. The delivery man picks up the order and delivers it to the recipient.

TYPES OF OPERATIONS USED

Hash keys – adding key, removing key, indexing

Hset order:123 recipient Kowalski order_date 09.11.24 product_list products

order_history list

List – adding/removing values of keys, modifying values

Lrem prod.list.warehouse3 1 prod1

Transactions – ensuring that data is consistent in the database, ex. Status of the orders is changed only if all products are included in the package and ready to be sent

WATCH order:123:status

MULTI

GET order:123:status # verify the order status

SET order:123:status "sent"

EXEC

Sub/pub - communicating beetween warehouses, as well as with delivery company and customers

Agent 3: subscribe warehouse.channel

Agent2 (warehouse2): publish warehouse(2).channel “prod1 needed in warehouse2”

COMMANDS

order	OrderID
recipient	name
Order_date	
Product_list	[set] prod1,prod2,prod3
Order_history	[list/set]

Orders samples:

//ORDER 1

Hset order:123 recipient "Kowalski" order_date "2024-11-09" product_list "products" status "placed"

RPUSH placed.orders order:123

RPUSH order:123:products prod1 prod6 prod9

//ORDER 2

HSET order:124 recipient "Nowak" order_date "2024-11-10" product_list "products" status "placed"

RPUSH placed.orders order:124

```
RPUSH order:124:products prod4 prod5


//ORDER 3

HSET order:125 recipient "Lewandowski" order_date "2024-11-11" product_list "products" status "placed"

RPUSH placed.orders order:125

RPUSH order:125:products prod9 prod2 prod3
```

Warehouse product lists initialization (products needed in a warehouse)[Delivery]:

```
SADD warehouse1:product_list prod1 prod4 prod9

SADD warehouse2:product_list prod2 prod3 prod6 prod8

SADD warehouse3:product_list prod5 prod7
```

Warehouse product deletion (when preparing orders, or sending products to another warehouse)

```
SREM warehouse2:product_list prod6
```

Channels initialization:

```
//Agent 1 subscribes to warehouse1 channel

SUBSCRIBE warehouse1

//Agent 2 subscribes to warehouse2 channel

SUBSCRIBE warehouse2

//Agent 3 subscribes to warehouse3 channel

SUBSCRIBE warehouse3
```

Exemplary agents messages :

PUBLISH warehouse2.channel "prod1 needed in warehouse2"

PUBLISH warehouse1.channel "no prod1 in warehouse1"

PUBLISH warehouse3.channel "prod1 available in warehouse3"

//Products can travel between warehouses:

PUBLISH warehouse3.channel "prod1 sent to warehouse2"

//Products delivery:

PUBLISH warehouse2.channel "prod1 delivered to warehouse2"

Products list updates (ordered completed so remove some products, products delivery so add some products etc):

SREM warehouse3:product_list 1 prod1

SADD warehouse2:product_list prod1

Order status will change during realization process of the order by our platform so:

//After some steps order is ready for pickup so we will remove them from placed.orders list
//and move it to ready.orders list:

```
SET order:123:status "ready"          #order status changed

LREM placed.orders 0 order:123      # Remove from ` placed.orders`

RPUSH ready.orders order:123       # Add to ` ready.orders`
```

//Order sent (picked up by delivery company) so analogically like previous one:

```
SET order:123:status "sent"                        #order status changed

LREM ready.orders 0 order:123       # Remove from ` ready.orders`
```

```
RPUSH sent.orders order:123        # Add to `sent.orders`


//Order delivered:

SET order:123:status "delivered"     # order status changed

LREM sent.orders 0 order:123       # Remove from `sent.orders`

RPUSH delivered.orders order:123    # Add to `delivered.orders`
```

Let's assume that from time to time (e.g. when synchronizing data in the warehouse) we want to check whether an order with the ready status has not suddenly changed to sent (i.e. it has already been picked up by the courier). If such a change occurs, we want to react and update the lists:

```
WATCH order:123:status

MULTI

GET order:123:status # verify the order status

SET order:123:status "sent"

EXEC
```

If the status has changed between WATCH and EXEC, the transaction will fail. If EXEC returns nil, it means that the status changed before the transaction was completed (e.g. another process has already set order:123:status to sent). In this case, we can retrieve the status again and take further action.

If the status has changed to "sent" (e.g. the order was received earlier), we remove the order from ready.orders and move it to sent.orders:

```
//Check status again to make sure that it is "sent"

GET order:123:status
```

```
//If it is "sent" move order to proper list:

LREM ready.orders 0 order:123

RPUSH sent.orders order:123
```