Group Project

(Cherepanova Millena, Arkhipova Alexandra, Selifanov Tikhon, Abgaryan Artur, Nedbay Pavel)

1. Description of the product (Planned end users, users' needs)

Team: БДШКИ

Project name: "Café Insight Manager" (Уютное кафе)

Short introduction: In today's data-driven business landscape, cafes must adopt advanced technologies for enhanced service management. Our project offers a bespoke solution tailored for cafes, emphasizing improved menu and order management. Utilizing sophisticated data analytics tools through key components like the Menu Table and the Order History Table, this initiative is designed to refine cafe operations, boost customer engagement, and propel business growth.

Overview: The project employs a new strategy in cafe management by optimizing menu and order processes. It utilizes two primary databases: the Menu Table, which provides detailed pricing and, optionally, promotional details for each item, and the Order History Table, which captures intricate details of every transaction. This structure is crafted to boost operational efficiency, deliver crucial customer insights, and cultivate loyalty through personalized engagement and incentives.

Purpose: The project's goal is to create a streamlined database for cafes, integrating a detailed Menu Table with an Order History Table, to enhance operational efficiency and facilitate informed decision-making.

Users: Cafe Managers, Staff

User's Needs: Cafe Managers and Staff, including Waiters, require a system that streamlines order management and inventory tracking, simplifies menu updates. They need an intuitive interface for efficient order processing and real-time monitoring of stock levels to ensure menu availability. Also, tools for analyzing customer behavior and sales data are essential for strategic decision-making, menu optimization, and enhancing overall customer service. The system should support these operational needs while being user-friendly and adaptable to the dynamic cafe environment.

Implementation details: In order to effectively gather statistics for the project, we decided to outsource the development of an automatic data collection system to programmers from ФКН НИУ ВШЭ. The system they develop will automatically capture transactional data from every sale, including detailed order information. This strategic partnership is expected to result in a robust and efficient data collection system, providing us with valuable insights for optimizing our cafe management processes and enhancing customer experiences.

Additional features: This database system, while initially designed for cafe management, is also highly adaptable for use in small restaurant networks. Its comprehensive features, including automated inventory management, customer feedback integration, employee performance tracking,

and in-depth financial reporting, are scalable and versatile. This makes it an ideal solution for small restaurant chains looking to streamline operations, gain valuable insights into customer behavior, and efficiently manage multiple locations with a unified, data-driven approach.

2. Functional requirements and data restrictions in textual form

Functional requirements:

1. The system should allow cafe administrators to add, edit, and exclude menu items.
2. Each menu item entry should include details such as name, description, and price.
3. Menu items should be categorizable for easy navigation and management.
4. The database should be scalable.
5. The system should be able to handle concurrent interactions from at least 20-30 users simultaneously.
6. All technical manipulations on the database should be documented.
7. The system should facilitate real-time order processing, allowing customers to place orders electronically.
8. Order confirmation, updates and status tracking should be sent to both customers and cafe staff.
9. Access to different system functionalities should be role-based, with roles like administrator, staff, and customer.
10. User authentication should be secure and comply with industry best practices.
11. The database must work without interruptions. In case of a problem, the resuscitation window for the database should not be more than a fixed period.

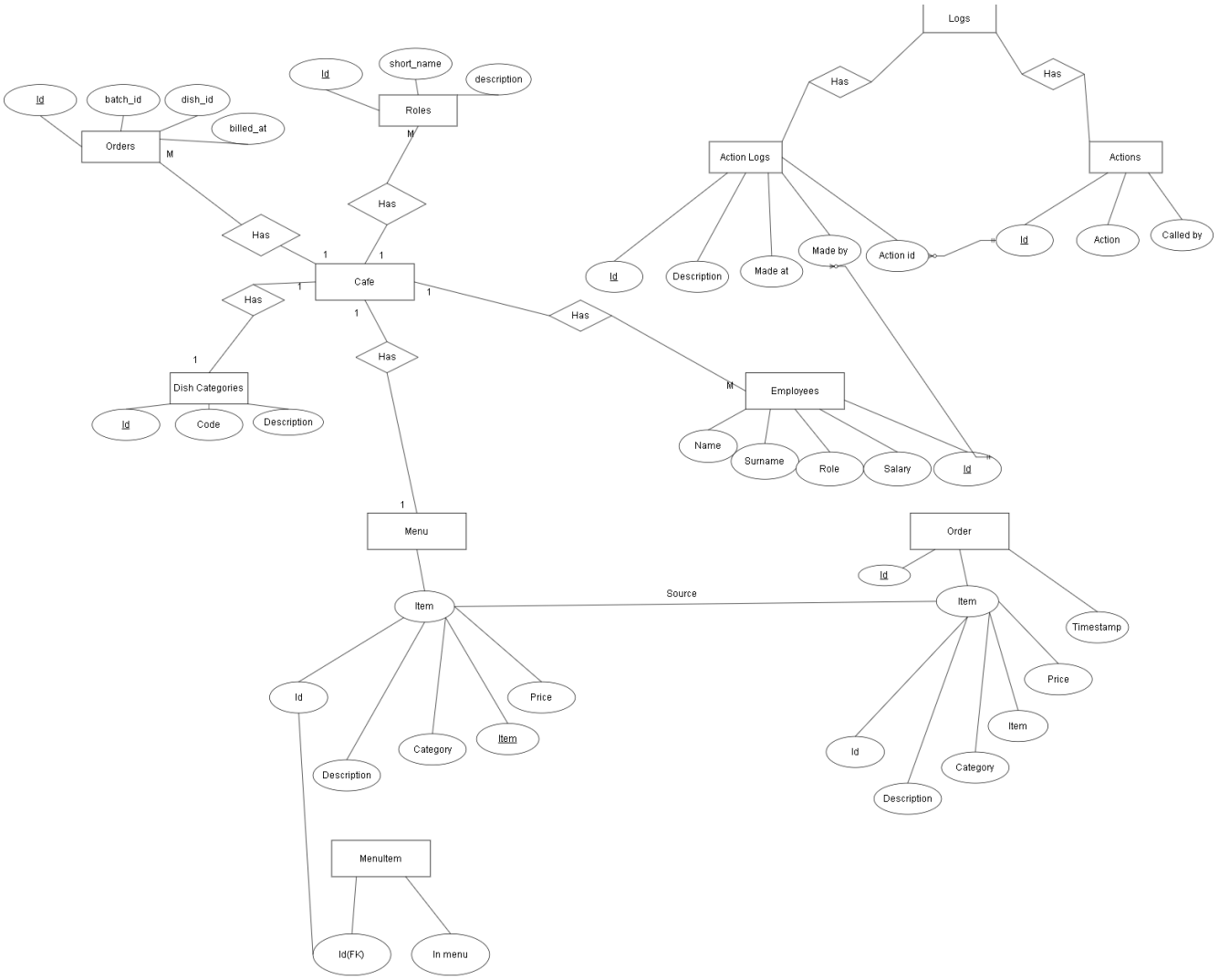
Data Restrictions:

1. Customer personal information, including names and contact details, should be stored securely and comply with data protection regulations.
2. Access to sensitive customer data should be restricted to authorized personnel.
3. Data includes information such as item names, prices, and customer orders.
4. Recipe and composition details of menu items should be treated as confidential information.
5. Access to menu management functionalities should be restricted to authorized administrators.
6. Order details, including customer preferences and transaction history, should be stored securely.
7. Data encryption should be implemented to protect sensitive information during transmission.
8. Data storage and processing practices should align with legal requirements concerning customer data and financial transactions.
9. Regular backups of critical data, especially Order History data, should be performed to ensure data recovery in case of system failures or data loss events.

3. Non-functional requirements

1. Users interact with the system through a frontend app, sending data (e.g., menu updates, orders) via API to the server.
2. Each user request should be processed within 3 seconds after the data is sent from the frontend app to ensure a responsive user experience.
3. Private data, including customer information and transaction details, should be securely stored and transmitted.
4. Database administration tasks should require the consent of both administrators. No single administrator should be able to tamper with the database alone
5. Permission changes or critical operations, such as data deletion, should require authorization from both administrators.
6. During the testing phase, the system should operate within the specified hardware constraints: Core i5 processor, 8GB RAM, and a 60GB disk.
7. Auditing mechanisms should be in place to track changes to the database and identify any suspicious activities.
8. The database architecture should be designed to scale seamlessly to accommodate future growth beyond the initial project scope.
9. The system should provide meaningful error messages to users in case of data input errors or system issues.
10. Robust error-handling mechanisms should be in place to prevent data corruption and ensure system stability.

4. A preliminary database schema (ER diagram)



5. Functional and multi-valued (optional) dependencies

Order is in BCNF:
{Idl} → {Item, Timestamp, Id}

Menu is in BCNF:
{Item} → {Id, Description, Category, item, Price}

Dish Categories is in BCNF:
{Id} → {Code, Description}

Orders is in BCNF:
{Id} → {batch_id, dish_id, billed_at}

Roles is in BCNF:
{Idl} → {short_name, description}

Action Logs is in BCNF:
{Id} → {Description, Made at, Made by, Action id}

Actions is in BCNF:
{Id} → {Action, Called by}

Employees is in BCNF:
{Id} → {Name, Surname, Role, Salary}

6. Normalized database scheme (Relation model in textual form*)

After doing BCNF, table Menu changed from this form: Menu: (Item -> Id, description, Category, Item, Price, In menu), to form with two tables Menu and MenuItem: (Id(FK), In menu)

7. SQL DDL script for database creation based on normalized schema

```
CREATE TABLE IF NOT EXISTS cafe.t_dish_categories
(
    id integer NOT NULL DEFAULT nextval('cafe.t_dish_categories_id_seq'::regclass),
    code character varying(50) COLLATE pg_catalog."default" NOT NULL,
    description text COLLATE pg_catalog."default",
    CONSTRAINT t_dish_categories_pkey PRIMARY KEY (id)
)
```

```
CREATE TABLE IF NOT EXISTS cafe.t_employees
(
    id integer NOT NULL DEFAULT nextval('cafe.t_employees_id_seq'::regclass),
    name character varying(80) COLLATE pg_catalog."default" NOT NULL,
    surname character varying(80) COLLATE pg_catalog."default" NOT NULL,
    role integer NOT NULL,
    salary bigint NOT NULL,
    CONSTRAINT t_employees_pkey PRIMARY KEY (id),
    CONSTRAINT t_employees_role_fkey FOREIGN KEY (role)
        REFERENCES cafe.t_roles (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

```
CREATE TABLE IF NOT EXISTS cafe.t_menu
(
    id bigint NOT NULL DEFAULT nextval('cafe.menu_id_seq'::regclass),
    price bigint NOT NULL,
    name character varying(100) COLLATE pg_catalog."default" NOT NULL,
    description text COLLATE pg_catalog."default",
    in_menu boolean NOT NULL DEFAULT true,
    category integer NOT NULL DEFAULT 100,
    CONSTRAINT menu_pkey PRIMARY KEY (id),
    CONSTRAINT t_menu_category_fkey FOREIGN KEY (category)
        REFERENCES cafe.t_dish_categories (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)
```

```
CREATE TABLE IF NOT EXISTS cafe.t_orders
(
    id bigint NOT NULL DEFAULT nextval('cafe.t_orders_id_seq'::regclass),
    batch_id bigint NOT NULL DEFAULT nextval('cafe.t_orders_batch_id_seq'::regclass),
    dish_id integer NOT NULL,
    billed_at time with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
    billed_by integer NOT NULL,
    CONSTRAINT t_orders_pkey PRIMARY KEY (id),
    CONSTRAINT t_orders_billed_by_fkey FOREIGN KEY (billed_by)
        REFERENCES cafe.t_employees (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT t_orders_dish_id_fkey FOREIGN KEY (dish_id)
        REFERENCES cafe.t_menu (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

```
CREATE TABLE IF NOT EXISTS cafe.t_roles
(
    id integer NOT NULL DEFAULT nextval('cafe.t_roles_id_seq'::regclass),
    short_name character varying(10) COLLATE pg_catalog."default" NOT NULL,
    description text COLLATE pg_catalog."default",
    CONSTRAINT t_roles_pkey PRIMARY KEY (id)
)
```

```sql
CREATE OR REPLACE VIEW cafe.v_menu
 AS
 SELECT tm.name,
    tm.description,
    tm.price,
    tdc.code
   FROM cafe.t_menu tm
     JOIN cafe.t_dish_categories tdc ON tm.category = tdc.id
  WHERE tm.in_menu = true
  ORDER BY tm.name;
```

```sql
CREATE OR REPLACE VIEW cafe.v_orders_with_price
 AS
 SELECT o.batch_id,
    min(o.billed_at) AS billed_at,
    sum(tm.price) AS total_price
   FROM cafe.t_orders o
     JOIN cafe.t_menu tm ON tm.id = o.dish_id
  GROUP BY o.batch_id
  ORDER BY o.batch_id;
```

```sql
CREATE OR REPLACE VIEW cafe.v_pretty_t_orders
 AS
 SELECT o.id,
    o.batch_id,
    tm.name,
    o.billed_at,
    (te.name::text || ' '::text) || te.surname::text AS billed_by
   FROM cafe.t_orders o
     JOIN cafe.t_menu tm ON tm.id = o.dish_id
     JOIN cafe.t_employees te ON te.id = o.billed_by;
```

```sql
CREATE TABLE IF NOT EXISTS log.t_actions
(
    id bigint NOT NULL DEFAULT nextval('log.t_actions_id_seq'::regclass),
    action character varying(50) COLLATE pg_catalog."default" NOT NULL,
    called_by text COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT t_actions_pkey PRIMARY KEY (id)
)
```

```sql
CREATE TABLE IF NOT EXISTS log.t_action_logs
(
    id bigint NOT NULL DEFAULT nextval('log.t_action_logs_id_seq'::regclass),
    description text COLLATE pg_catalog."default" NOT NULL,
    made_at timestamp with time zone NOT NULL DEFAULT CURRENT_TIMESTAMP,
    made_by integer NOT NULL,
    action_id integer,
    CONSTRAINT t_action_logs_pkey PRIMARY KEY (id),
    CONSTRAINT t_action_logs_action_id_fkey FOREIGN KEY (action_id)
        REFERENCES log.t_actions (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT t_action_logs_made_by_fkey FOREIGN KEY (made_by)
        REFERENCES cafe.t_employees (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

8. SQL DML queries that implement functional requirements

```
CREATE OR REPLACE FUNCTION cafe.f_add_dish(
    _name character varying,
    _description text,
    _price bigint,
    _employee_id integer)
    RETURNS void
    LANGUAGE 'sql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
INSERT INTO cafe.t_menu(name, description, price)
VALUES(_name, _description, _price);
INSERT INTO log.t_action_logs(action_id, description, made_by)
VALUES (1, 'Added dish named ' || _name || ' to the menu', _employee_id);
$BODY$;
```

```
CREATE OR REPLACE FUNCTION cafe.f_add_order(
    _batch_id bigint,
    _dish_id bigint,
    _billed_at timestamp with time zone,
    _billed_by integer,
    _called_by integer)
    RETURNS void
    LANGUAGE 'sql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
INSERT INTO cafe.t_orders(batch_id, dish_id, billed_at, billed_by)
VALUES(_batch_id, _dish_id, _billed_at, _billed_by);

INSERT INTO log.t_action_logs(action_id, description, made_by)
VALUES(3, 'Added order to batch with id {' || _batch_id
       ||
       '} and dish id {' || _dish_id,
      _called_by);
$BODY$;
```

```sql
CREATE OR REPLACE FUNCTION cafe.f_set_dish_state(
    _dish_id integer,
    _state boolean,
    _employee_id integer)
    RETURNS void
    LANGUAGE 'sql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
UPDATE cafe.t_menu
SET in_menu = _state
WHERE id = _dish_id;
INSERT INTO log.t_action_logs(action_id, description, made_by)
VALUES(2,
       'Changed state of dish with id {' || _dish_id || '} to ' || _state,
      _employee_id);
$BODY$;
```

9. BONUS TASK: Undernormalization problems

Undernormalization issues may arise if, for instance, detailed customer information is stored directly within the "Order History Table" instead of maintaining a separate "Customers Table." In this scenario, the system may face challenges when dealing with changes to customer details. For example, if a customer updates their contact information, such as a phone number or email address, each corresponding entry in the "Order History Table" would need to be individually modified. Given the dynamic nature of customer data and the volume of transactions, this approach could lead to inefficiencies and data redundancy.

Furthermore, undernormalization may pose problems related to security and compliance with data protection regulations. If sensitive customer information, such as names and contact details, is duplicated across multiple entries in the "Order History Table," ensuring consistent and secure management of this data becomes complex. Access control and data protection measures may become harder to enforce, potentially leading to breaches in privacy and non-compliance with legal requirements.

In addressing undernormalization challenges, adopting a normalized structure with a dedicated "Customers Table" would be beneficial. This approach allows for centralized management of customer information, reducing redundancy, and facilitating updates with minimal impact on the system. It enhances data integrity, simplifies data maintenance tasks, and ensures better compliance with security and privacy regulations.

10. BONUS TASK: correctness of accounting for non-functional requirements

It's imperative to meticulously account for non-functional requirements (NFRs) to ensure the robustness, efficiency, and sustainability of the system. NFRs, which dictate how the system operates, encompass various crucial aspects for the project's success. Interface requirements, including understandability and user-friendliness, are paramount for cafe managers and staff who rely on an intuitive system for streamlined order processing and menu management. Performance requirements, such as security and efficiency, are critical for safeguarding customer data and ensuring real-time order processing within the stipulated time limits. Economic requirements, including cost considerations and future development plans, are essential for the project's financial viability and long-term success.

Operating requirements, addressing system accessibility for maintenance and memory constraints, are pivotal for sustaining smooth operations in the dynamic cafe environment. Environmental requirements play a role in understanding the project's impact on its surroundings, be it nature or human aspects. These considerations not only shape the system's functionality but also provide insights into its implementation, aiding in constructing a comprehensive development plan and optimizing general system performance.

Furthermore, by incorporating NFRs, the project can identify parameters influencing system qualities, ranging from social and economic factors to political and security considerations. This comprehensive understanding helps in anticipating user responses, calculating potential losses or required investments, and proactively addressing errors or delays in system operations. This foresight is invaluable when scaling the product, ensuring continual improvement and responsiveness to external factors, ultimately enhancing the overall quality and adaptability of the cafe management system.