МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

Кафедра «Програмної інженерії»

Звіт з лабораторної роботи №2

З дисципліни «Архітектура програмного забезпечення»

Виконав:
ст. гр. ПЗПІ-19-10
Ковальов В. О.

Перевірила:
Доц: Сокорчук І П.

Харків 2022

Вступ

Метою роботи є розробити серверну/back-end частину програмної системи.

1 АРХІТЕКТУРНІ РІШЕННЯ

Серверну частину написано на мові JavaScript з використанням фрейм ворку Node.js Також використані наступні пакети/додаткові фрейм ворки:

- Node express,
- MongoDB,
- Node cors,

Архітектуру побудовано за принципами Onion архітектури, але без використання інтерфейсів між кожним рівнем програми (див. рис. 1 та додатки А - Г).

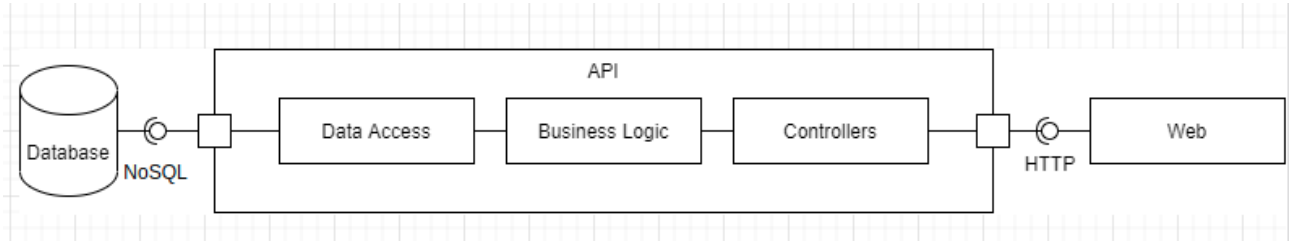

Рисунок 1 – Діаграма компонентів

Дана архітектура найкраще підходить до мого проекту, бо вона найбільш проста та має увесь необхідний функціонал.

2 РОЗГОРТАННЯ ПРОЕКТУ

Проект буде розгорнуто локально та матиме 5 функціональних частин,які можуть розгортатися незалежно одна від одної:

- База даних,
- Серверна частина,
- Веб сторінка,
- Мобільний додаток,
- Модуль для відправки даних на сервер (IoT)

Обмін даними здійснюватиметься здебільшого за протоколом HTTP та HTTPS. Взаємодія між базою даних та сервером здійснюється SQL запитами (див. рис. 2).

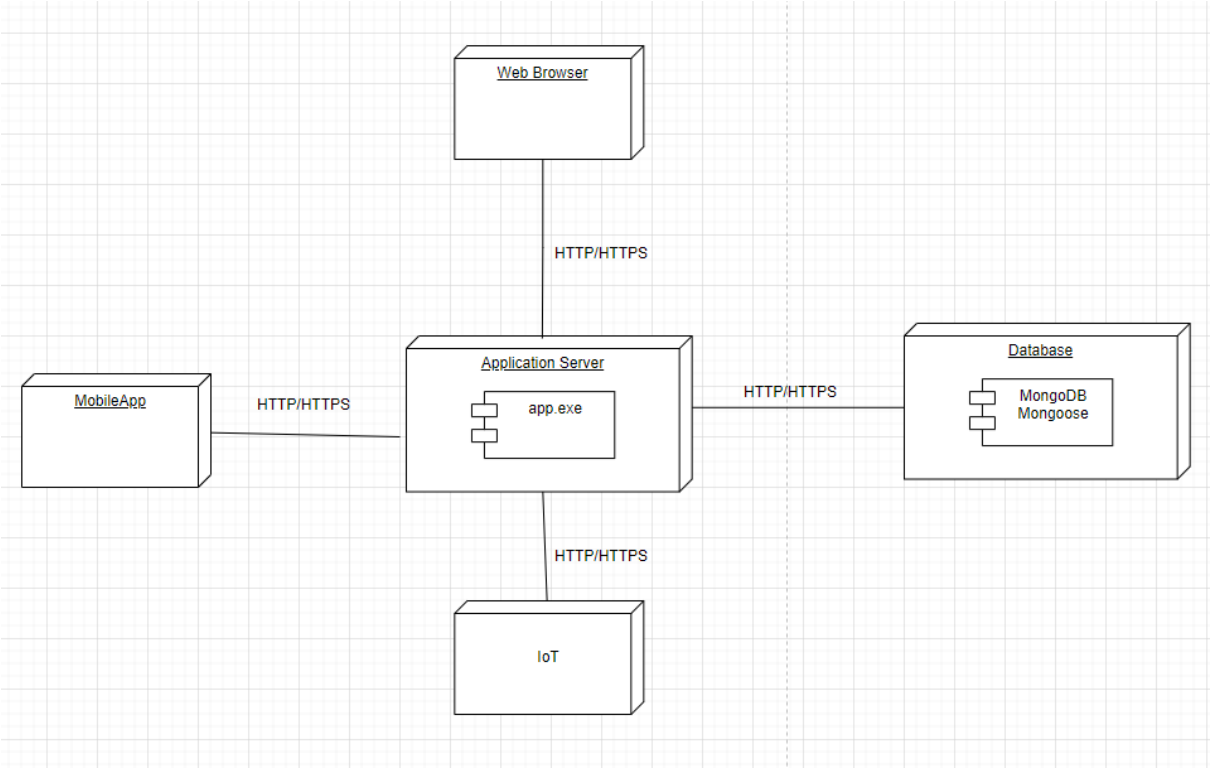

Рисунок 2 – Діаграма розгортання

Як можна побачити усі додатки залежать від серверної частини,тому на ній будуть робитися правки у подальшому.

3 ФУНКЦІОНАЛ СИСТЕМИ

Система матиме 3 ролі :

- Адміністратор
- Менеджер
- Офіціант

Такий розподіл ролей дозволить мінімізувати час розробки та повністю покриває необхідний функціонал. Детальніший розподіл обов'язків зображений на рисунку 3.

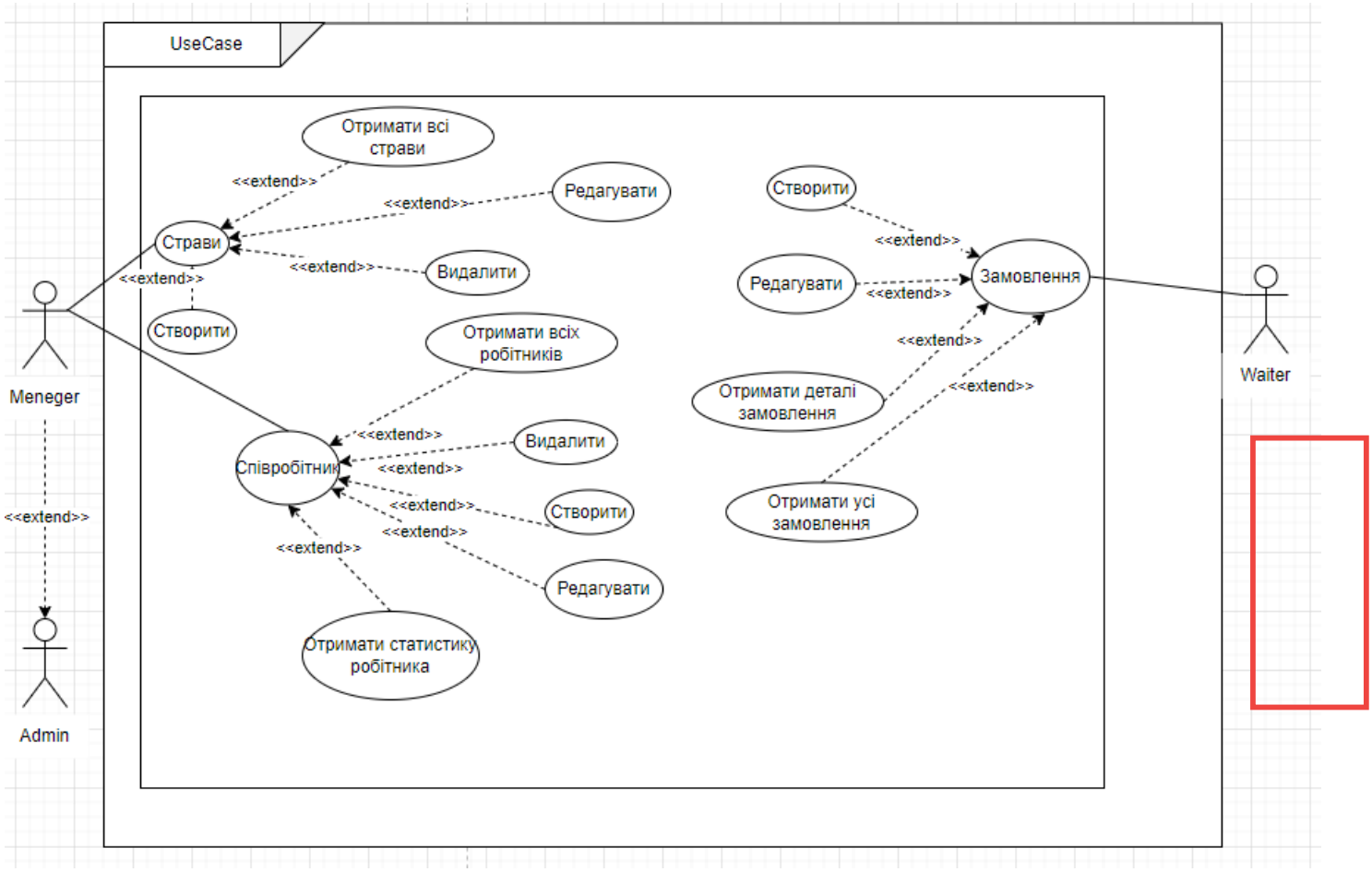

Рисунок 3 – Use Case діаграма

Як можна побачити у системі Адміністратор має функції над такими сутностями як страви та робітник, а робітник може взаємодіяти з замовленням.

4 СТРУКТУРА БАЗИ ДАННИХ

На проекті використовується СУБД Mongoose. База даних містить 5 основних сутностей (моделей):

- Персонал,
- Страва,
- Категорія,
- Замовлення,
- Організація

Через те, що використовується документо-орієнтовану базу даних, система містить документи, необхідні для функціювання цього пакету (див. рис. 4). Для кожної сутності розроблено REST специфікацію (див. таблиці 1 – 3).

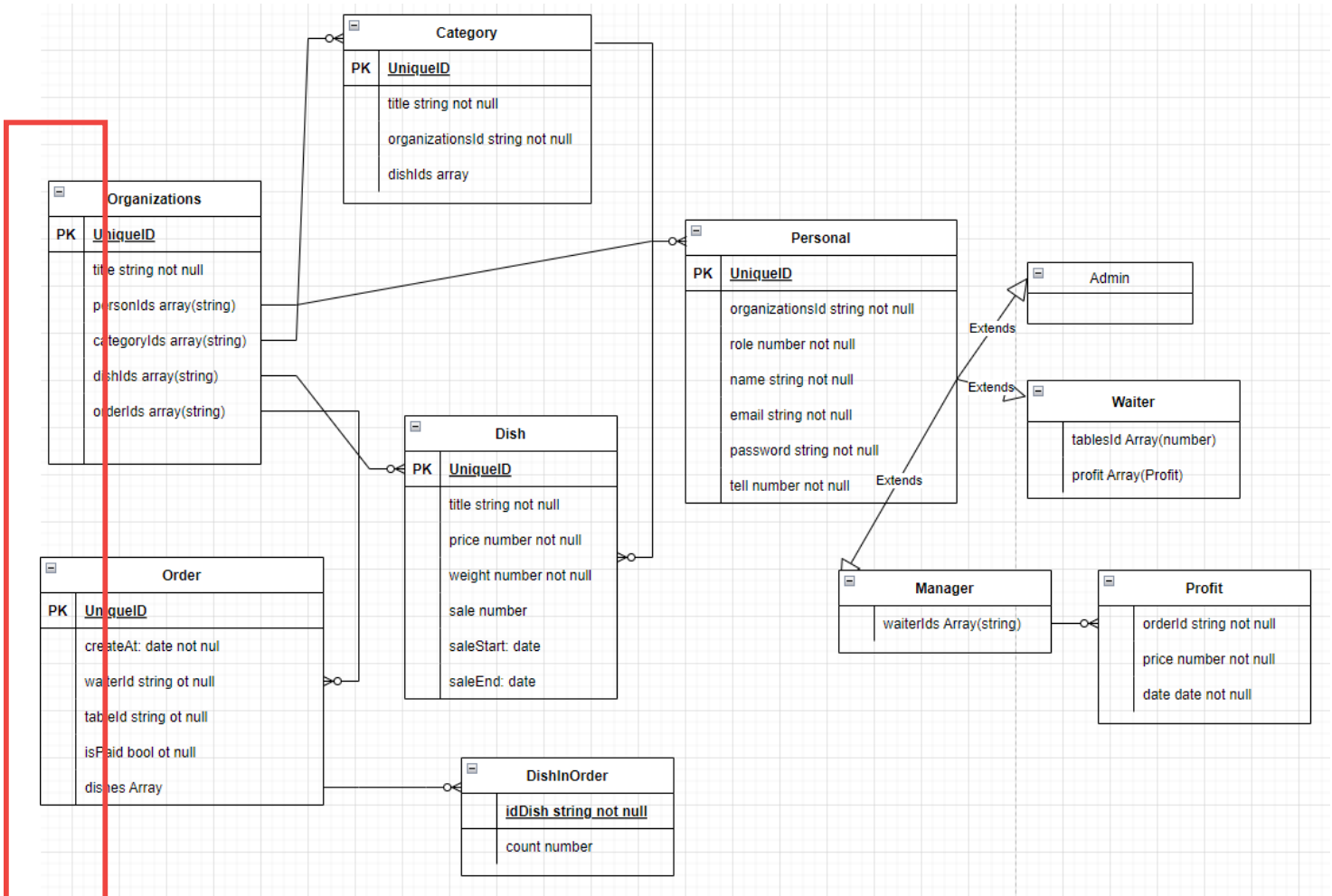

Рисунок 4 – ER діаграма

Хоча в використовується документо-орієнтована база даних, але завдяки такій базі даних відсутні дублювання.

1 REST специфікація

Таблиця 1 – Контролер Category

Тип запиту	Маршрут	Вхідні дані	Вихідні дані
Post	/category	{ "id": 0, "title": ‘’ }	-
Get	/category	-	Список створених категорій
Get	/category/:id		Одна категорія за id
Put	/category/:id	{ "id": 0, "title": ‘’ }	-
Delete	/category/:id	-	-

Таблиця 2 – Контролер User

Тип запиту	Маршрут	Вхідні дані	Вихідні дані
Post	/users	{ email: "" firstName : " " lastName: " "phone: “" }	-
Get	/users	-	Список усіх робітників
Get	/users/:id	-	{ email: "" firstName : " " lastName: " "phone: “" }
Put	/users/:id	{ email: "" firstName : " "	-

		lastName: " "phone: “" }	
Delete	/users/:id	-	 -

Таблиця 3 – Контролер Dish

Тип запиту	**Маршрут**	**Вхідні дані**	**Вихідні дані**
Post	/dish	{ category: ""price: 0 title: "" }	-
Get	/dish	-	Список страв
Get	/dish/:id	-	{ category: ""price: 0 title: "" }
Put	/dish/:id	{ category: ""price: 0 title: "" }	-
Delete	/dish/:id	-	-

Висновок

Результатом роботи є повноцінна серверна частина продукту (програмна), яка може бути розміщена незалежно від бази даних.

ДОДАТОК А

Код Клас user.js

```
import {deleteDocWithId, getCollectionInDB, getOneDocInDB, setDataToDBWithId} from
"../firebase/bataBase.js";
import {createUserWithEmailAndPassword} from "firebase/auth";
import {auth} from "../firebase/firebase.js";

class UserController{
    async register(req, res) {
        try{
            const {email, password} = req.body;
            const response = await createUserWithEmailAndPassword(auth, email,
password);
            await setDataToDBWithId('users', response.user.uid, {
                email: email
            });
            res.json(response);
        } catch(error) {
            res.status(500).json(error)
        }
    }

    async getOne(req, res) {
        try {
            const {id} = req.params;
            if (!id)
                res.status(400).json({"message": 'Not user'})
            const user = await getOneDocInDB('users', id)
            res.status(200).json(user);
        } catch (error) {
            res.status(500).json(error)
        }
    }
    async getAll(req, res) {
        try {
            const dishesResponse = await getCollectionInDB('users');
            const users = [];
            dishesResponse.forEach(doc => users.push({...doc.data(), id: doc.id}));
            res.status(200).json(users);
        } catch (error) {
            res.status(500).json(error)
        }
    }
    async update(req, res) {
        try {
            const {id} = req.params;
            if (!id)
                res.status(400).json({"message": 'Not user'});
            const user = await setDataToDBWithId('users', id, req.body);
            res.status(200).json(user);
        } catch (error) {
            res.status(500).json(error)
        }
    }
    async delete(req, res) {
```

```
54          try {
55              const {id} = req.params;
56              if (!id)
57                  res.status(400).json({"message": 'Not user'});
58              await deleteDocWithId('users', id);
59              res.status(200).json('delete');
60          } catch (error) {
61              res.status(500).json(error)
62          }
63      }
64  }
65
66
```

ДОДАТОК Б

Код Клас dish.js

```
67  import { getOneDocInDB, setDataToDBOutId, getCollectionInDB, setDataToDBWithId,
68  deleteDocWithId } from "../firebase/bataBase.js";
69
70  class DishControllers {
71      async create(req, res){
72          try{
73              const dish = await setDataToDBOutId('dishes', req.body);
74              res.status(200).json(dish);
75          } catch(error) {
76              res.status(500).json(error)
77          }
78      }
79
80      async getOne(req, res) {
81          try {
82              const {id} = req.params;
83              if (!id)
84                  res.status(400).json({"message": 'Not dish'})
85              const dish = await getOneDocInDB('dishes', id)
86              res.status(200).json(dish);
87          } catch (error) {
88              res.status(500).json(error)
89          }
90      }
91      async getAll(req, res) {
92          try {
93              const dishesResponse = await getCollectionInDB('dishes')
94              const dishes = [];
95              dishesResponse.forEach(doc => dishes.push({...doc.data(), id: doc.id}));
96              res.status(200).json(dishes);
97          } catch (error) {
98              res.status(500).json(error)
99          }
100     }
101     async update(req, res) {
102         try {
103             const {id} = req.params;
104             if (!id)
105                 res.status(400).json({"message": 'Not dish'})
106             const dish = await setDataToDBWithId('dishes', id, req.body)
107             res.status(200).json(dish);
108         } catch (error) {
109             res.status(500).json(error)
110         }
111     }
112     async delete(req, res) {
113         try {
114             const {id} = req.params;
115             if (!id)
116                 res.status(400).json({"message": 'Not dish'})
117             await deleteDocWithId('dishes', id)
118             res.status(200).json('delete');
119         } catch (error) {
```

```
            res.status(500).json(error)
        }
    }
}

export default new DishControllers();
```

ДОДАТОК В

Код Клас category.js

```
126  import {
127      deleteDocWithId,
128      getCollectionInDB,
129      getOneDocInDB,
130      setDataToDBOutId,
131      setDataToDBWithId
132  } from "../firebase/bataBase.js";
133
134
135  class CategoryControllers {
136      async create(req, res){
137          try{
138              const category = await setDataToDBOutId('category', req.body);
139              res.status(200).json(category);
140          } catch(error) {
141              res.status(500).json(error)
142          }
143      }
144      async getOne(req, res) {
145          try {
146              const {id} = req.params;
147              if (!id)
148                  res.status(400).json({"message": 'Not category'})
149              const category = await getOneDocInDB('category', id)
150              res.status(200).json(category);
151          } catch (error) {
152              res.status(500).json(error)
153          }
154      }
155      async getAll(req, res) {
156          try {
157              const dishesResponse = await getCollectionInDB('category')
158              const categories = [];
159              dishesResponse.forEach(doc => categories.push({...doc.data(), id: doc.id}));
160              res.status(200).json(categories);
161          } catch (error) {
162              res.status(500).json(error)
163          }
164      }
165      async update(req, res) {
166          try {
167              const {id} = req.params;
168              if (!id)
169                  res.status(400).json({"message": 'Not category'})
170              const category = await setDataToDBWithId('category', id, req.body)
171              res.status(200).json(category);
172          } catch (error) {
173              res.status(500).json(error)
174          }
175      }
176      async delete(req, res) {
177          try {
178              const {id} = req.params;
```

```
            if (!id)
                res.status(400).json({"message": 'Not category'})
            await deleteDocWithId('category', id)
            res.status(200).json('delete');
        } catch (error) {
            res.status(500).json(error)
        }
    }
```

ДОДАТОК Г

Код Клас order.js

```
import {
    deleteDocWithId,
    getCollectionInDB,
    getOneDocInDB,
    setDataToDBOutId,
    setDataToDBWithId
} from "../firebase/bataBase.js";

class OrderControllers {
    async create(req, res){
        try{
            const order = await setDataToDBOutId('orders', req.body);
            res.status(200).json(order);
        } catch(error) {
            res.status(500).json(error)
        }
    }

    async getOne(req, res) {
        try {
            const {id} = req.params;
            if (!id)
                res.status(400).json({"message": 'Not dish'})
            const order = await getOneDocInDB('orders', id)
            res.status(200).json(order);
        } catch (error) {
            res.status(500).json(error)
        }
    }
    async getAll(req, res) {
        try {
            const dishesResponse = await getCollectionInDB('orders')
            const orders = [];
            dishesResponse.forEach(doc => orders.push({...doc.data(), id: doc.id}));
            res.status(200).json(orders);
        } catch (error) {
            res.status(500).json(error)
        }
    }
    async update(req, res) {
        try {
            const {id} = req.params;
            if (!id)
                res.status(400).json({"message": 'Not orders'})
            const order = await setDataToDBWithId('orders', id, req.body)
            res.status(200).json(order);
        } catch (error) {
            res.status(500).json(error)
        }
    }
    async delete(req, res) {
        try {
            const {id} = req.params;
```

```
            if (!id)
                res.status(400).json({"message": 'Not orders'})
            await deleteDocWithId('orders', id)
            res.status(200).json('delete');
        } catch (error) {
            res.status(500).json(error)
        }
    }
}
```