



MongoDB projekt

Mikołaj Gosztyła, Michał Dydek

27.05.2024

Spis treści

1	Opis zadania i technik do jego realizacji	3
1.1	Techniki wykorzystane do realizacji	3
2	Kolekcje	4
2.1	Kolekcja employees	4
2.1.1	Opis pól w kolekcji	4
2.1.2	Przykładowe dane	4
2.2	Kolekcja res	5
2.2.1	Opis pól w kolekcji	5
2.2.2	Przykładowe dane	5
2.3	Kolekcja expenses	6
2.3.1	Opis pól w kolekcji	6
2.3.2	Przykładowe dane	6
2.4	Kolekcja incomes	7
2.4.1	Opis pól w kolekcji	7
2.4.2	Przykładowe dane	7
2.5	Kolekcja properties	8
2.5.1	Opis pól w kolekcji	8
2.5.2	Przykładowe dane	8
3	Endpointy	9
3.1	GET	9
3.2	POST	9
3.3	PUT	11
3.4	DELETE	11

4	Raport	12
5	Używanie oprogramowania	14
5.1	Strona główna	14
5.2	Reservations	15
5.3	Login	21
5.4	Expenses	25
5.5	Incomes	31
5.6	Report	32
5.7	Niezałogowany pracownik	35
6	Dyskusja zrealizowanych technik	36
6.1	Trigger przy dodawaniu nowej rezerwacji	36
6.2	Możliwość zapisu raportu	36

1 Opis zadania i technik do jego realizacji

Aplikacja obsługuje restaurację i niektóre działania, które mogłyby być przydatne dla jej pracowników. Operacje, które wspiera nasze oprogramowanie to:

- rezerwacje - klient dzwoni do pracownika, który następnie wprowadza dane do systemu
- dodawanie wydatków - bieżące wydatki przez firmę mogą być wprowadzane przez pracownika
- dodawanie dochodów - po każdej transakcji pracownik również może wprowadzić dochody wraz z niektórymi informacjami odnośnie nich

Zdecydowaliśmy się na stworzenie oprogramowania od strony pracowniczej, a nie dla klientów ponieważ jako pracownik mamy większą kontrolę nad wprowadzanymi danymi.

1.1 Techniki wykorzystane do realizacji

- Front-End - do realizacji front-endu wykorzystaliśmy język JavaScript, a także wykorzystaliśmy framework React
- Backend - backend realizuje skrypt napisany również w JavaScriptcie przy użyciu frameworka Express, a do komunikacji z bazą danych został użyty framework Mongoose
- Baza danych - rodzaj bazy danych na jaką się zdecydowaliśmy to nierelacyjna baza MongoDB

2 Kolekcje

2.1 Kolekcja employees

W tej kolekcji przechowujemy dane na temat każdego pracownika. Aplikacja umożliwia logowanie każdego z użytkowników w systemie. Dodatkowo można tworzyć nowe konta.

```
1 const employeeSchema = new mongoose.Schema({
2   name: { type: String, required: true },
3   surname: { type: String, required: true },
4   employee_number: { type: Number, required: true, unique: true },
5   password: { type: String, required: true }
6 });
```

Listing 1: Employees

2.1.1 Opis pól w kolekcji

- **name** - imie pracownika
- **surname** - nazwisko pracownika
- **employee number** - numer pracownika
- **password** - hasło konta pracownika

2.1.2 Przykładowe dane

```
1 {
2   _id: ObjectId('66546fb68e7f68d20e761563'),
3   name: 'Michał',
4   surname: 'Dydek',
5   employee_number: 1,
6   password: 'test',
7   __v: 0
8 }
```

2.2 Kolekcja res

W tej kolekcji znajdują się aktualne rezerwacje na stoliki w restauracji. Na tej kolekcji jest również nałożony dodatkowo trigger, który usuwa stare rezerwacje.

```
1 const resSchema = new mongoose.Schema({
2   employee_id: { type: String, required: true },
3   date: { type: String, required: true },
4   time: { type: String, required: true },
5   duration: { type: Number, required: true },
6   table: { type: String, required: true },
7 });
```

Listing 2: Reservations

2.2.1 Opis pól w kolekcji

- **employee id** - numer pracownika, który dokonał rezerwacji
- **date** - dzień, miesiąc i rok rezerwacji
- **time number** - godzina rozpoczęcia rezerwacji
- **duration** - długość rezerwacji
- **table** - numer stolika, którego dotyczy rezerwacja

2.2.2 Przykładowe dane

```
1 {
2   _id: ObjectId('665d9a1589f261518c10c452'),
3   employee_id: '1',
4   client: 'Gosztyla',
5   date: '2024-06-20',
6   time: '09:00',
7   duration: 0.5,
8   table: '1',
9   __v: 0
10 }
```

2.3 Kolekcja expenses

Kolekcja, w której przechowywane są wydatki firmowe i każdy z pracowników ma możliwość wprowadzenia danych, które są wykorzystywane w raporcie.

```
1 const expenseSchema = new mongoose.Schema({
2   employee_number: { type: Number, required: true },
3   item: { type: String, required: true },
4   quantity: { type: Number, required: true },
5   unit_price: { type: Number, required: true },
6   date: { type: String, required: true },
7 });
```

Listing 3: Expenses

Jest to kolekcja, na której wykonywane są wszystkie operacje CRUD.

- create - można dodawać nowe dane o wydatkach
- read - z panelu pracownika możemy wypisać dane z tabeli
- update - każdy wydatek jesteśmy w stanie edytować
- delete - istnieje również możliwość usunięcia wydatku

2.3.1 Opis pól w kolekcji

- **employee number** - numer pracownika, który zarejestrował wydatek
- **item** - rzecz, której dotyczy dodany wydatek
- **quantity** - ilość
- **unit_price** - cena jednostkowa
- **date** - data wydatku

2.3.2 Przykładowe dane

```
1 {
2   _id: ObjectId('665475238e7f68d20e76158a'),
3   employee_number: 3,
4   item: 'paluszki',
5   quantity: 10,
6   price: 20,
7   date: '2024-06-02',
8   __v: 0
9 }
```

2.4 Kolekcja incomes

Kolekcja, w której przechowywane są przychody dla firmy i każdy z pracowników ma również możliwość wprowadzenia danych, które są później wykorzystywane w raporcie.

```
1 const incomeSchema = new mongoose.Schema({
2   employee_number: { type: Number, required: true },
3   order_id: { type: String, required: true },
4   price: { type: Number, required: true },
5   date: { type: String, required: true },
6 });
```

Listing 4: Incomes

Jest to kolekcja, na której wykonywane są wszystkie operacje CRUD.

- create - można dodawać nowe dane o przychodach
- read - z panelu pracownika możemy mieć dostęp do danych z kolekcji
- update - każdy przychód jesteśmy w stanie edytować
- delete - istnieje również możliwość usunięcia przychodu

2.4.1 Opis pól w kolekcji

- **employee number** - numer pracownika, który zarejestrował wpływ
- **order id** - rzecz, której dotyczy dodany wpływ
- **price** - cena
- **date** - data wpływu

2.4.2 Przykładowe dane

```
1 {
2   _id: ObjectId('6654c95943ee7b58959a5aba'),
3   employee_number: 2,
4   order_id: '29283',
5   price: 30,
6   date: '2024-05-30',
7   __v: 0
8 }
```

2.5 Kolekcja properties

```
1 const propertiesSchema = new mongoose.Schema({
2   numberOfTables: { type: Number, required: true },
3   openingTime: { type: String, required: true },
4   closingTime: { type: String, required: true },
5   closedDays: [{ type: String, required: true }]
6 });
```

Listing 5: Properties

2.5.1 Opis pól w kolekcji

- **numberOfTables** - ilość stolików w restauracji
- **openingTime** - godzina otwarcia restauracji
- **closingTime** - godzina zamknięcia restauracji
- **closedDays** - dni, w które restauracja będzie zamknięta

2.5.2 Przykładowe dane

```
1 {
2   _id: ObjectId('6662b86ce7563e909d776e99'),
3   numberOfTables: 20,
4   openingTime: '9:00',
5   closingTime: '22:00',
6   closedDays: [ '2024-12-24', '2024-12-25' ],
7   __v: 0
8 }
```


3 Endpointy

3.1 GET

- `\config` - pobiera konfigurację restauracji, która znajduje się w folderze `Config` i pliku `restaurantProperties.json`

```
1 app.get('/config', async (req, res) => {  
2   try {  
3     const config = await Properties.find();  
4     res.json(config);  
5   } catch (error) {  
6     res.status(500).json({ error: 'Server error' });  
7   }  
8 });
```

- `\employees` - pobiera listę wszystkich pracowników, bez hasła należącego do nich konta
- `\reservations` - pobiera listę wszystkich rezerwacji
- `\expensesList` - pobiera listę wszystkich wydatków
- `\incomesList` - pobiera listę wszystkich dochodów

3.2 POST

- `\employeeAdd` - dodaje nowego pracownika
- `\login` - loguje pracownika na jego konto

```
1 app.post("/login", async (req, res) => {  
2   try {  
3     const { employee_number, password } = req.body;  
4     const employee = await Employee.findOne({ employee_number });  
5  
6     if (!employee) {  
7       return res.status(404).json({ message: 'User not found' });  
8     }  
9  
10    if (employee.password === password) {  
11      res.status(200).json({ id: employee.employee_number });  
12    } else {  
13      res.status(401).json({ message: 'Invalid credentials' });  
14    }  
15  } catch (error) {  
16    console.error(error);  
17    res.status(500).json({ message: 'Internal Server Error' });  
18  }  
19 });
```

- `\deleteAccount` - usuwa konto pracownika

- \res - dodaje nową rezerwację

```

1 app.post("/res", async (req, res) => {
2   try {
3     const { employee_id, date, time, duration, table } = req.body;
4
5     if (!employee_id || !date || !time || !duration || !table) {
6       return res.status(400).json({ message: 'All fields are required' });
7     }
8
9     const reservationDateTime = new Date(date);
10    const [hour, minute] = time.split(':').map(Number);
11    reservationDateTime.setHours(hour, minute);
12    const endTime = new Date(reservationDateTime);
13    const additionalMinutes = duration * 60;
14    endTime.setMinutes(endTime.getMinutes() + additionalMinutes);
15
16    const currentDate = new Date();
17    if (reservationDateTime <= currentDate) {
18      return res.status(400).json({ message: 'Please select a future date for
19        reservation.' });
20    }
21
22    const properties = await Properties.findOne({});
23    if (properties) {
24      const [closingHour, closingMinute] = properties.closingTime.split(':').map(
25        Number);
26      const closingDate = new Date(reservationDateTime);
27      closingDate.setHours(closingHour);
28      closingDate.setMinutes(closingMinute);
29
30      if (closingDate < endTime) {
31        return res.status(400).json({ message: 'End time cannot exceed closing
32          time: ${properties.closingTime}' });
33      }
34    }
35
36    const existingReservations = await Res.find({ date, table });
37
38    for (const existingRes of existingReservations) {
39      const existingStart = new Date(existingRes.date);
40      const [existingHour, existingMinute] = existingRes.time.split(':').map(
41        Number);
42      existingStart.setHours(existingHour, existingMinute);
43      const existingEnd = new Date(existingStart);
44      existingEnd.setMinutes(existingEnd.getMinutes() + existingRes.duration *
45        60);
46
47      if (
48        (reservationDateTime >= existingStart && reservationDateTime <
49          existingEnd) ||
50        (endTime > existingStart && endTime <= existingEnd) ||
51        (reservationDateTime <= existingStart && endTime >= existingEnd)
52      ) {
53        return res.status(400).json({ message: 'The selected time overlaps with
54          an existing reservation.' });
55      }
56    }
57
58    const newRes = new Res(req.body);
59    await newRes.save();
60    res.status(201).json({ message: 'Reservation successful' });
61  } catch (error) {
62    console.error(error);
63    res.status(400).json({ message: 'Reservation not successful', error: error.
64      message });
65  }
66 }

```

- \expense - dodaje nowy wydatek
- \income - dodaje nowy dochód

- `\saveTurnoverData` - zapisuje raport do pliku data.csv

3.3 PUT

- `\expenses` \ - aktualizuje wydatek o podanym ID

```

1 app.put('/expenses/:id', async (req, res) => {
2   try {
3     const expense = await Expense.findById(req.params.id);
4     if (!expense) {
5       return res.status(404).json({ message: 'Expense not found' });
6     }
7
8     expense.employee_number = req.body.employee_number || expense.
      employee_number;
9     expense.item = req.body.item || expense.item;
10    expense.quantity = req.body.quantity || expense.quantity;
11    expense.unit_price = req.body.price || expense.price;
12    expense.date = req.body.date || expense.date;
13
14    const updatedExpense = await expense.save();
15    res.json(updatedExpense);
16  } catch (err) {
17    res.status(400).json({ message: err.message });
18  }
19 });

```

- `\incomes` \ - aktualizuje dochód o podanym ID

3.4 DELETE

- `\reservations` \ - usuwa rezerwację o podanym ID

- `\expenses` \ - usuwa wydatek o podanym ID

```

1 app.delete('/expenses/:id', async (req, res) => {
2   const { id } = req.params;
3   try {
4     const deletedExpense = await Expense.findByIdAndDelete(id);
5     if (!deletedExpense) {
6       return res.status(404).json({ message: 'Expense not found' });
7     }
8     res.status(200).json({ message: 'Expense deleted successfully' });
9   } catch (error) {
10    console.error('Error deleting expense:', error);
11    res.status(500).json({ message: 'Internal Server Error' });
12  }
13 });

```

- `\incomes` \ - usuwa dochód o podanym ID

4 Raport

Raport, który umożliwia nasza aplikacja łączy 3 kolekcje: incomes, expenses, employees. Polega on na wygenerowaniu całkowitego obrotu i ilości dodanych rekordów do dwóch pierwszych kolekcji. Zdecydowaliśmy się na taki wybór, ponieważ uważamy, że jest to przydatna i potrzebna informacja dla pracodawcy, który pewnie chciałby mieć wgląd w niektóre dane konkretnych pracowników. Dodatkowo można uwzględnić konkretny rok i miesiąc, by wygenerować raport dla konkretnej daty.

```
1 db.incomes.aggregate([
2 {
3   $addFields: {
4     year: { $year: { $toDate: "$date" } },
5     month: { $month: { $toDate: "$date" } }
6   }
7 },
8 {
9   $project: {
10     employee_number: 1,
11     price: 1
12   }
13 },
14 {
15   $unionWith: {
16     coll: "expenses",
17     pipeline: [
18       {
19         $addFields: {
20           year: { $year: { $toDate: "$date" } },
21           month: { $month: { $toDate: "$date" } }
22         }
23       },
24       {
25         $project: {
26           employee_number: 1,
27           price: { $multiply: ["$unit_price", "$quantity"] }
28         }
29       }
30     ]
31   }
32 },
33 {
34   $group: {
35     _id: "$employee_number",
36     count: { $sum: 1 },
37     monetary_turnover: { $sum: "$price" }
38   }
39 },
40 {
41   $sort: {
42     count: -1
43   }
44 },
45 {
46   $lookup: {
47     from: "employees",
48     localField: "_id",
```

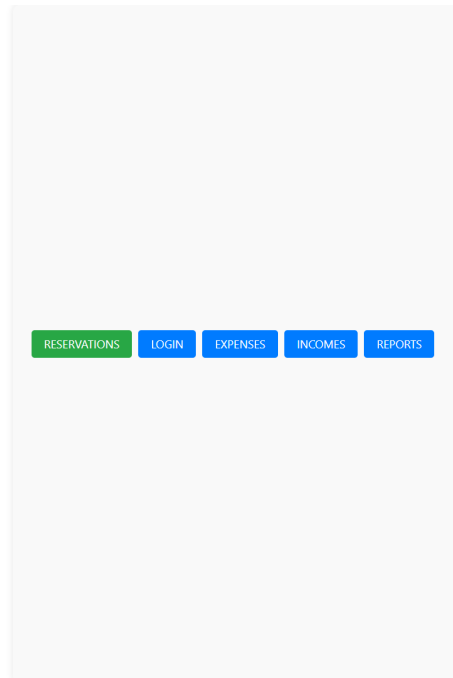
```
49     foreignField: "employee_number",
50     as: "employee_info"
51   }
52 },
53 {
54   $unwind: "$employee_info"
55 },
56 {
57   $project: {
58     name: "$employee_info.name",
59     surname: "$employee_info.surname",
60     _id: 0,
61     count: 1,
62     monetary_turnover: 1
63   }
64 }
65 ])
```

Listing 6: Zapytanie do bazy danych realizujące raport

5 Używanie oprogramowania

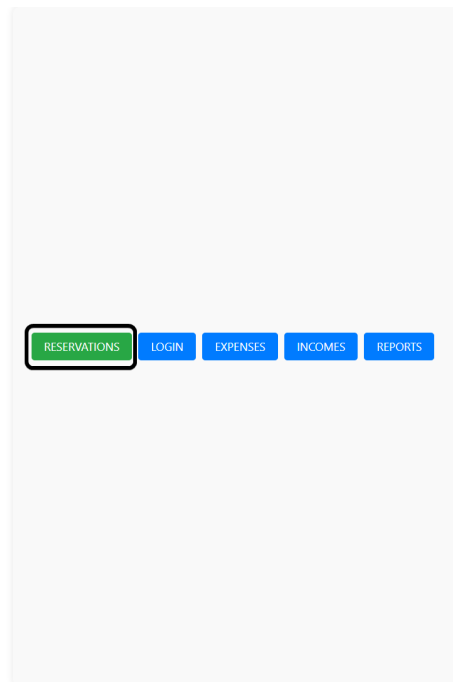
5.1 Strona główna

Widok strony głównej:



5.2 Reservations

Po wciśnięciu przycisku **Reservations**:



Przenosimy się do panelu rezerwacji:

BACK

Make a Reservation

Table number: 1

Reservation date: 07.06.2024

Reservation time: 09:00

Reservation duration (hours): 0.5

Client:

Make Reservation

Find reservations for client:

Reservations for table number: 1

Client	Date	Time	Duration	Action
dd	2024-06-07	09:00	0.5 hours	Delete
g	2024-06-14	09:00	0.5 hours	Delete
ssss	2024-06-19	09:00	0.5 hours	Delete
test	2024-06-19	16:30	3 hours	Delete
Gosztyla	2024-06-20	09:00	0.5 hours	Delete

Po wypełnieniu danych o rezerwacji klienta, możemy spróbować dodać taką rezerwację, za pomocą przycisku **Make Reservation**:

BACK

Make a Reservation

Table number: 2

Reservation date: 15.06.2024

Reservation time: 10:30

Reservation duration (hours): 2

Client: Dydek

Make Reservation

Find reservations for client:

Reservations for table number: 2

Client	Date	Time	Duration	Action
dd	2024-06-07	19:30	1 hours	Delete
dd	2024-06-07	20:30	1 hours	Delete
dd	2024-06-07	21:30	0.5 hours	Delete

Jeśli dane rezerwacji są poprawne, dostaniemy komunikat o poprawnym dodaniu rezerwacji:

Komunikat ze strony localhost:3000
Reservation successful

OK

Table number: 2

Reservation date: 15.06.2024

Reservation time: 10:30

Reservation duration (hours): 2

Client: Dydek

Make Reservation

Find reservations for client:

Reservations for table number: 2

Client	Date	Time	Duration	Action
dd	2024-06-07	19:30	1 hours	Delete
dd	2024-06-07	20:30	1 hours	Delete
dd	2024-06-07	21:30	0.5 hours	Delete

Rezerwacja następnie będzie dostępna w widoku rezerwacji dla wybranego stolika:

BACK

Make a Reservation

Table number: 2

Reservation date: 15.06.2024

Reservation time: 10:30

Reservation duration (hours): 2

Client: Dydek

Make Reservation

Find reservations for client:

Reservations for table number: 2

Client	Date	Time	Duration	Action
dd	2024-06-07	19:30	1 hours	Delete
dd	2024-06-07	20:30	1 hours	Delete
dd	2024-06-07	21:30	0.5 hours	Delete
Dydek	2024-06-15	10:30	2 hours	Delete

Jeśli wybraliśmy termin rezerwacji w przeszłości, to nie będziemy mogli dokonać rezerwacji:

Komunikat ze strony localhost:3000
Please select a future date for reservation.
OK

Make a Reservation

Table number: 2

Reservation date: 03.06.2024

Reservation time: 10:30

Reservation duration (hours): 2

Client: Dydek

Make Reservation

Find reservations for client:

Reservations for table number: 2

Client	Date	Time	Duration	Action
dd	2024-06-07	19:30	1 hours	Delete
dd	2024-06-07	20:30	1 hours	Delete
dd	2024-06-07	21:30	0.5 hours	Delete

Jeśli wybraliśmy termin rezerwacji, który kończy się po zamknięciu rezerwacji, dostaniemy komunikat:

Komunikat ze strony localhost:3000

End time cannot exceed closing time: 22:00

OK

Table number: 2

Reservation date: 12.06.2024

Reservation time: 18:30

Reservation duration (hours): 11

Client: Dydek

Make Reservation

Find reservations for client:

Reservations for table number: 2

Client	Date	Time	Duration	Action
dd	2024-06-07	19:30	1 hours	Delete
dd	2024-06-07	20:30	1 hours	Delete
dd	2024-06-07	21:30	0.5 hours	Delete

Jeśli wybraliśmy termin rezerwacji, który jakkolwiek nachodzi na inną rezerwację, dokonaną na ten sam stół, dostaniemy komunikat:

Komunikat ze strony localhost:3000

The selected time overlaps with an existing reservation.

OK

Table number: 2

Reservation date: 07.06.2024

Reservation time: 18:30

Reservation duration (hours): 2

Client: Gosztyla

Make Reservation

Find reservations for client:

Reservations for table number: 2

Client	Date	Time	Duration	Action
dd	2024-06-07	19:30	1 hours	Delete
dd	2024-06-07	20:30	1 hours	Delete
dd	2024-06-07	21:30	0.5 hours	Delete

Mamy również możliwość usunięcia rezerwacji:

BACK

Make a Reservation

Table number: 2

Reservation date: 15.06.2024

Reservation time: 10:30

Reservation duration (hours): 2

Client: Dydek

Make Reservation

Find reservations for client:

Reservations for table number: 2

Client	Date	Time	Duration	Action
dd	2024-06-07	19:30	1 hours	Delete
dd	2024-06-07	20:30	1 hours	Delete
dd	2024-06-07	21:30	0.5 hours	Delete
Dydek	2024-06-15	10:30	2 hours	Delete

Po usunięciu dostaniemy komunikat:

Komunikat ze strony localhost:3000
Reservation deleted successfully

OK

Table number: 2

Reservation date: 15.06.2024

Reservation time: 10:30

Reservation duration (hours): 2

Client: Dydek

Make Reservation

Find reservations for client:

Reservations for table number: 2

Client	Date	Time	Duration	Action
dd	2024-06-07	19:30	1 hours	Delete
dd	2024-06-07	20:30	1 hours	Delete
dd	2024-06-07	21:30	0.5 hours	Delete
Dydek	2024-06-15	10:30	2 hours	Delete

Mamy również możliwość wyszukania wszystkich rezerwacji, dla danego klienta:

[BACK](#)

Make a Reservation

Table number:

Reservation date:

Reservation time:

Reservation duration (hours):

Client:

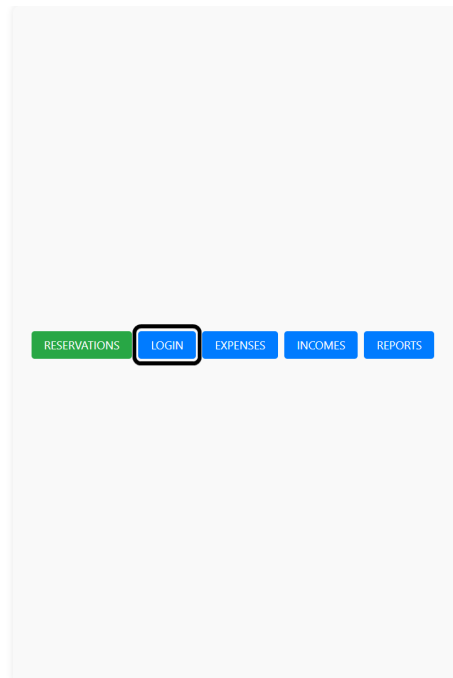
[Make Reservation](#)

Find reservations for client:

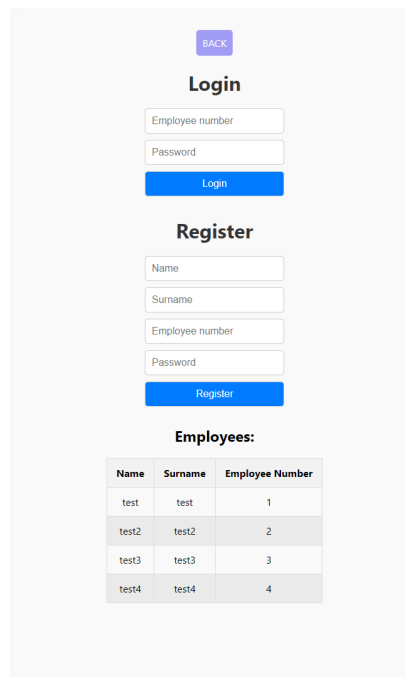
Client	Date	Time	Duration	Action
Gosztyla	2024-06-20	09:00	0.5 hours	Delete

5.3 Login

Po wciśnięciu przycisku **Login**:



Przenosimy się do panelu logowania:



BACK

Login

Employee number
Password
Login

Register

Name
Surname
Employee number
Password
Register

Employees:

Name	Surname	Employee Number
test	test	1
test2	test2	2
test3	test3	3
test4	test4	4

Możemy zalogować się na istniejące konto pracownicze:

The screenshot displays the application's login and registration interface. At the top, there is a 'Login' section with a 'BACK' button, a title 'Login', and two input fields: one containing the number '1' and another with masked characters '....'. Below these is a blue 'Login' button. Underneath is a 'Register' section with a title 'Register' and four input fields labeled 'Name', 'Surname', 'Employee number', and 'Password'. A blue 'Register' button is at the bottom of this section. Below the registration form is a table titled 'Employees:' containing four rows of test data.

Name	Surname	Employee Number
test	test	1
test2	test2	2
test3	test3	3
test4	test4	4

Po podaniu poprawnych danych, zostajemy zalogowani:

The screenshot shows the application after a successful login. A box at the top contains the text 'You are logged in as' followed by 'employee number: 1' and a blue 'Logout' button. Below this box is the same 'Employees:' table as seen in the previous screenshot.

Name	Surname	Employee Number
test	test	1
test2	test2	2
test3	test3	3
test4	test4	4

Jeśli podamy złe dane logowania, dostajemy komunikat:

The screenshot shows a web application interface. At the top, a light blue notification box contains the text "Komunikat ze strony localhost:3000" and "Invalid credentials", with an "OK" button. Below this is a login form with a username field containing "1" and a password field with masked characters, followed by a "Login" button. Underneath is a "Register" section with fields for Name, Surname, Employee number, and Password, and a "Register" button. At the bottom, a table titled "Employees:" lists four test users.

Name	Surname	Employee Number
test	test	1
test2	test2	2
test3	test3	3
test4	test4	4

Możemy również zarejestrować nowego pracownika:

The screenshot shows the same web application interface as before, but with the "Register" form highlighted by a black border. The "Employees:" table remains at the bottom.

Name	Surname	Employee Number
test	test	1
test2	test2	2
test3	test3	3
test4	test4	4

Po podaniu poprawnych danych, pracownik zostaje dodany do listy wszystkich zarejestrowanych pracowników:

[BACK](#)

Login

Login

Register

Register

Employees:

Name	Surname	Employee Number
test	test	1
test2	test2	2
test3	test3	3
test4	test4	4
Mikolaj	Gosztyla	10

W przypadku, gdy dane nie są poprawne, dostajemy komunikat:

Komunikat ze strony localhost:3000
Numer pracownika musi być unikalny

OK

Login

Register

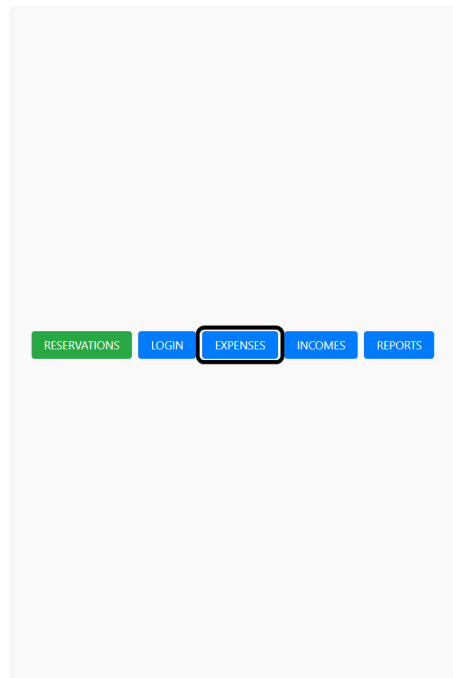
Register

Employees:

Name	Surname	Employee Number
test	test	1
test2	test2	2
test3	test3	3
test4	test4	4
Mikolaj	Gosztyla	10

5.4 Expenses

Po wciśnięciu przycisku **Expenses**:



Przenosimy się do panelu wydatków:

BACK

Expense Manager

Item

Quantity

Unit Price

dd.mm.yyyy

Add

Expenses List

Employee Number	Item	Quantity	Price	Date	Actions
1	a	1	\$	2024-06-02	<div>Edit</div> <div>Delete</div>
1	fd	22	\$	2024-05-25	<div>Edit</div> <div>Delete</div>

Mamy możliwość dodania nowego wydatku:

BACK

Expense Manager

Add

Expenses List

Employee Number	Item	Quantity	Price	Date	Actions
1	a	1	\$	2024-06-02	<div>Edit</div> <div>Delete</div>
1	fd	22	\$	2024-05-25	<div>Edit</div> <div>Delete</div>
1	ziemniaki	100	2\$	2024-07-02	<div>Edit</div> <div>Delete</div>

Po wciśnięciu przycisku **Add**, wydatek zostaje dodany do listy:

BACK

Expense Manager

Add

Expenses List

Employee Number	Item	Quantity	Price	Date	Actions
1	a	1	\$	2024-06-02	<div>Edit</div> <div>Delete</div>
1	fd	22	\$	2024-05-25	<div>Edit</div> <div>Delete</div>
1	ziemniaki	100	2\$	2024-07-02	<div>Edit</div> <div>Delete</div>
1	paluszki	1000	20\$	2024-07-24	<div>Edit</div> <div>Delete</div>

Mamy możliwość edycji dodanego wcześniej wydatku:

BACK

Expense Manager

Item

Quantity

20

dd.mm.yyyy

Add

Expenses List

Employee Number	Item	Quantity	Price	Date	Actions
1	a	1	\$	2024-06-02	<div>Edit</div> <div>Delete</div>
1	fd	22	\$	2024-05-25	<div>Edit</div> <div>Delete</div>
1	ziemniaki	100	2\$	2024-07-02	<div>Edit</div> <div>Delete</div>
1	paluszki	1000	20\$	2024-07-24	<div>Edit</div> <div>Delete</div>

Po wciśnięciu przycisku **Edit**, dane o wybranych wydatku zostają automatycznie uzupełnione w panelu na górze:

BACK

Expense Manager

ziemniaki

100

2

02.07.2024

Edit

Cancel Edit

Expenses List

Employee Number	Item	Quantity	Price	Date	Actions
1	a	1	\$	2024-06-02	<div>Edit</div> <div>Delete</div>
1	fd	22	\$	2024-05-25	<div>Edit</div> <div>Delete</div>
1	ziemniaki	100	2\$	2024-07-02	<div>Edit</div> <div>Delete</div>
1	paluszki	1000	20\$	2024-07-24	<div>Edit</div> <div>Delete</div>

Mamy teraz możliwość edycji danych o wybranym wydatku:

[BACK](#)

Expense Manager

[Edit](#)
[Cancel Edit](#)

Expenses List

Employee Number	Item	Quantity	Price	Date	Actions
1	a	1	\$	2024-06-02	Edit Delete
1	fd	22	\$	2024-05-25	Edit Delete
1	ziemniaki	100	2\$	2024-07-02	Edit Delete
1	paluszki	1000	20\$	2024-07-24	Edit Delete

Po ponownym wciśnięciu przysku **Edit**, nowe dane zostają zapisane:

[BACK](#)

Expense Manager

[Add](#)

Expenses List

Employee Number	Item	Quantity	Price	Date	Actions
1	a	1	\$	2024-06-02	Edit Delete
1	fd	22	\$	2024-05-25	Edit Delete
1	ziemniaki	30000	4\$	2024-07-18	Edit Delete
1	paluszki	1000	20\$	2024-07-24	Edit Delete

Jeśli jednak chcemy anulować edycję danych o wybranym wydatku, możemy wcisnąć przycisk **Cancel Edit**:

BACK

Expense Manager

paluszki

1000

20

24. 07. 2024

Edit

Cancel Edit

Expenses List

Employee Number	Item	Quantity	Price	Date	Actions
1	a	1	\$	2024-06-02	<div>Edit</div> <div>Delete</div>
1	fd	22	\$	2024-05-25	<div>Edit</div> <div>Delete</div>
1	ziemniaki	30000	4\$	2024-07-18	<div>Edit</div> <div>Delete</div>
1	paluszki	1000	20\$	2024-07-24	<div>Edit</div> <div>Delete</div>

Mamy również możliwość usunięcia wydatku:

BACK

Expense Manager

Item

Quantity

20

dd. mm. rrrr

Add

Expenses List

Employee Number	Item	Quantity	Price	Date	Actions
1	a	1	\$	2024-06-02	<div>Edit</div> <div>Delete</div>
1	fd	22	\$	2024-05-25	<div>Edit</div> <div>Delete</div>
1	ziemniaki	30000	4\$	2024-07-18	<div>Edit</div> <div>Delete</div>
1	paluszki	1000	20\$	2024-07-24	<div>Edit</div> <div>Delete</div>

Po wciśnięciu przycisku **Delete**, wydatek zostaje usunięty:

BACK

Expense Manager

Item

Quantity

20

dd.mm.yyyy

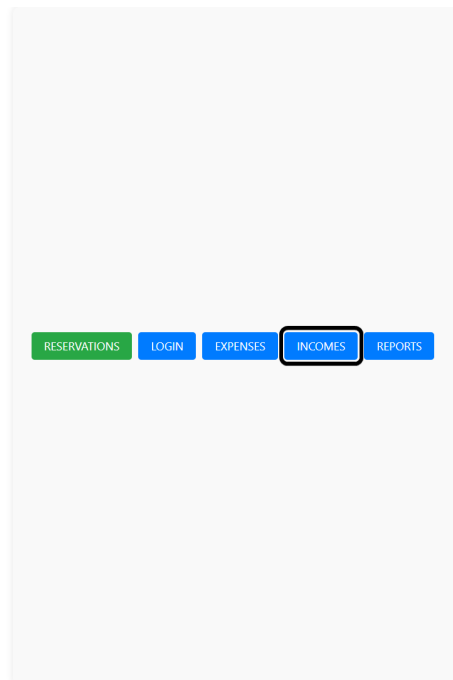
Add

Expenses List

Employee Number	Item	Quantity	Price	Date	Actions
1	a	1	\$	2024-06-02	<div>Edit</div> <div>Delete</div>
1	ziemniaki	30000	4\$	2024-07-18	<div>Edit</div> <div>Delete</div>
1	paluszki	1000	20\$	2024-07-24	<div>Edit</div> <div>Delete</div>

5.5 Incomes

Po wciśnięciu przycisku **Incomes**:



Przenosimy się do panelu wpływów:

BACK

Income Manager

Order ID

Price

dd.mm.yyyy

Add

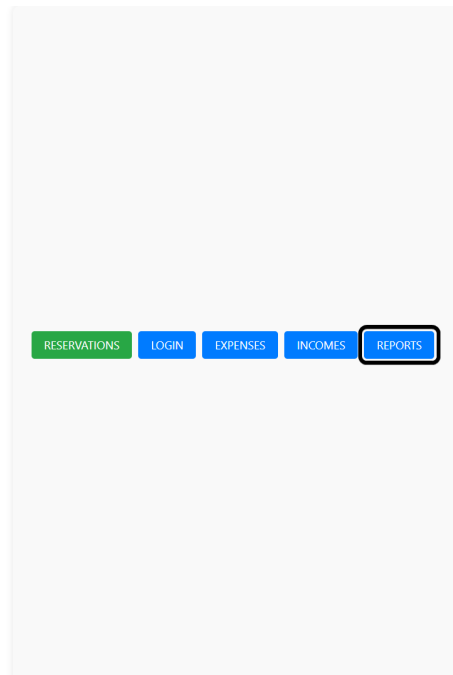
Incomes List

Order ID	Price	Date	Actions
2	\$2	2024-05-30	<div>Edit</div> <div>Delete</div>

Panel wpływów oferuje takie same funkcjonalności, jak panel wydatków i jego obsługa jest taka sama.

5.6 Report

Po wciśnięciu przycisku **Report**:



Przenosimy się do panelu, gdzie mamy możliwość podglądu wygenerowanego raportu:

[BACK](#)

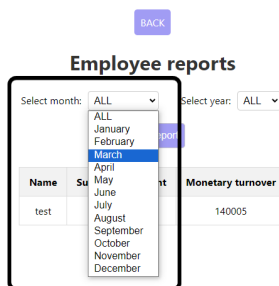
Employee reports

Select month: Select year:

[Save report](#)

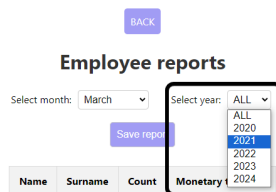
Name	Surname	Count	Monetary turnover
test	test	5	140005

Mamy możliwość sprecyzowania, z którego miesiąca chcemy wygenerować raport (albo ze wszystkich, opcja **ALL**):



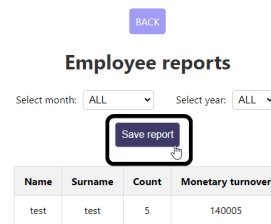
The screenshot shows the 'Employee reports' form. At the top left is a purple 'BACK' button. Below it, the title 'Employee reports' is centered. To the left of the title is a 'Select month:' dropdown menu, which is open, showing a list of months from 'ALL' to 'December'. 'March' is highlighted in blue. To the right of the title is a 'Select year:' dropdown menu, which is closed and shows 'ALL'. Below these dropdowns is a table with columns: 'Name', 'Surname', 'Count', and 'Monetary turnover'. The first row of data shows 'test' in the 'Name' column and '140005' in the 'Monetary turnover' column.

Tak jak i analogiczną możliwość sprecyzowania roku:



The screenshot shows the 'Employee reports' form. At the top left is a purple 'BACK' button. Below it, the title 'Employee reports' is centered. To the left of the title is a 'Select month:' dropdown menu, which is closed and shows 'March'. To the right of the title is a 'Select year:' dropdown menu, which is open, showing a list of years from 'ALL' to '2024'. '2021' is highlighted in blue. Below these dropdowns is a 'Save report' button. Below the button is a table with columns: 'Name', 'Surname', 'Count', and 'Monetary'. The first row of data shows 'test' in the 'Name' column and '140005' in the 'Monetary' column.

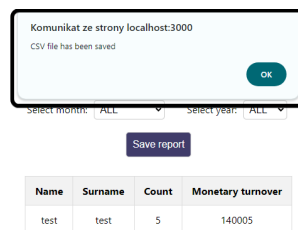
Mamy również możliwość zapisania wyników raportu do pliku csv, za pomocą przysku **Save**:



The screenshot shows the 'Employee reports' section of a web application. At the top, there is a purple 'BACK' button. Below it, the title 'Employee reports' is centered. Under the title, there are two dropdown menus: 'Select month:' with 'ALL' selected and 'Select year:' with 'ALL' selected. A red rectangular box highlights a purple 'Save report' button. Below the button is a table with four columns: 'Name', 'Surname', 'Count', and 'Monetary turnover'. The table contains one data row with the values 'test', 'test', '5', and '140005'.

Name	Surname	Count	Monetary turnover
test	test	5	140005

Po jego wciśnięciu, dostajemy komunikat:



This screenshot shows the same 'Employee reports' interface as the previous one, but with a light blue success message overlay at the top. The message reads 'Komunikat ze strony localhost:3000' and 'CSV file has been saved'. There is an 'OK' button in the bottom right corner of the message box. The 'Save report' button is still highlighted with a red box. The table below remains the same.

Name	Surname	Count	Monetary turnover
test	test	5	140005

5.7 Niezalogowany pracownik

Jeśli pracownik nie zaloguje się na żadne konto, nie będzie miał on dostępu do funkcjonalności oprogramowania:



6 Dyskusja zrealizowanych technik

6.1 Trigger przy dodawaniu nowej rezerwacji

Podczas dodawania nowej rezerwacji, by nie przechowywać zbędnych starych rezerwacji, usuwane są wszystkie rezerwacje, których data jest wcześniejsza niż aktualny dzień. Clean-Reservations to funkcja, która odpowiada za wyszukanie i usunięcie starych rezerwacji

```
1 resSchema.post('save', function(doc) {  
2   cleanReservations();  
3 })
```

Listing 7: Trigger

6.2 Możliwość zapisu raportu

Nasza aplikacja wspiera zapisywanie wygenerowanego raportu do pliku .csv, który można potem łatwo analizować i obrabiać. Zdecydowaliśmy się na to, ponieważ uważamy, że samo wyświetlanie się danych nie byłoby wystarczające dla klienta i na pewno takie usprawnienie zwiększa wygodę używania aplikacji.