

Informatyka Stosowana gr.1 s.6 2021/22

Mateusz Birkholz



**POLITECHNIKA
BYDGOSKA**

im. Jana i Jędrzeja Śniadeckich

Projekt – Lista zadań z wykorzystaniem frameworków React i Flask

PROGRAMOWANIE SERWISÓW SIECIOWYCH

1. Założenia projektu

Projekt zakłada zbudowanie aplikacji client-server udostępniającej użytkownikom dostęp do listy zadań, możliwość dodawania elementów do listy, usuwania ich, oznaczania czy edytowania. Każdy z elementów na liście posiada swój opis, datę, oraz priorytet. Strona pozwala na rejestrację oraz logowanie użytkowników dzięki czemu każdy użytkownik ma dostęp do swojej własnej listy. Strona komunikuje się z API serwerowym a przesłane do niego dane zostają sprawdzone i zapisane w bazie danych. Front-end aplikacji został przygotowany we frameworku React natomiast server w Pythonie z wykorzystaniem frameworku Flask.

2. Opis działania serwera

Server jest prostym API napisanym w Pythonie z wykorzystaniem Flaska. Do obsługi bazy danych wykorzystuje SQLAlchemy w wersji flaskowej.

```
app = Flask(__name__)

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///db.sqlite3'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)
```

Do obsługi bazy danych przygotowane zostały 2 klasy User i Todo. Posiadają odpowiednie dla nich pola:

- User
 - Id
 - Name
 - Password
 - Todos – relacja 1 do wielu (lista todo danego uzytkownika)
- Todo
 - Id
 - Text
 - isComplete
 - date
 - prio

```

class Todo(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    text = db.Column(db.String())
    isComplete = db.Column(db.Boolean, default=False)
    date = db.Column(db.Date, default=db.func.now())
    prio = db.Column(db.String())
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    def to_json(self):
        return {
            'id': self.id,
            'text': self.text,
            'isComplete': self.isComplete,
            'date': self.date,
            'prio': self.prio
        }

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String())
    password = db.Column(db.String())
    todos = db.relationship('Todo', backref='user')
    def to_json(self):
        return {
            'id': self.id,
            'name': self.name,
            'password': self.password,
            'todos': [todo.to_json() for todo in self.todos]
        }

```

Poza polami oba posiadają metodę pozwalającą na przekonwertowanie obiektu do formatu JSON.

API zawiera metody:

- login – na podstawie loginu i hasła
- register – po sprawdzeniu czy dany użytkownik już nie istnieje
- todos – zwraca listę zadań danego użytkownika na podstawie jego id
- completeTodo – odznaczenie zadania jako wykonanego lub nie
- deleteTodo – usunięcie zadania z listy
- addTodo – dodanie zadania do listy
- editTodo – edycję zadania z listy

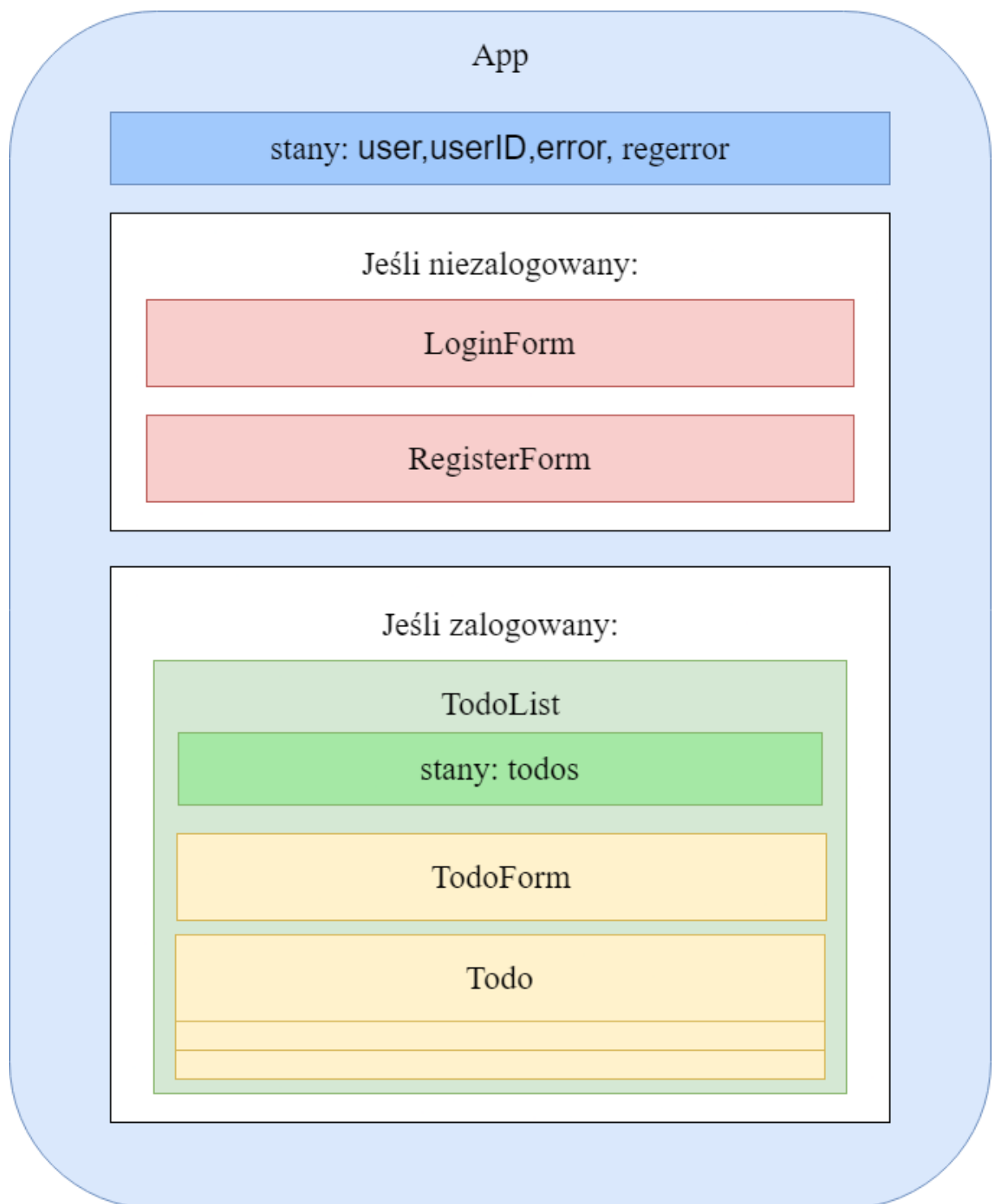
Przykład (editTodo):

```

@app.route('/editTodo/<int:ident>/<string:text>/<string:prio>/<string:data>/<int:user_id>')
def editTodo(ident, text, prio, data, user_id):
    todo = Todo.query.filter_by(id=ident).first()
    todo.text = text
    todo.prio = prio
    todo.date = datetime.strptime(data, '%Y-%m-%d')
    db.session.commit()
    li = Todo.query.filter_by(user_id=user_id).all()
    return json.dumps([o.to_json() for o in li], indent=4, sort_keys=True, default=str)

```

3. Układ Komponentów



4. Opis komponentów

App

App jest podstawowym komponentem. Zawiera informacje o zalogowanym użytkowniku na podstawie których definiowana jest jego zawartość. Posiada metody takie jak login czy register przekazywane do komponentów LoginForm i RegisterForm. Funkcje te pozwalają na manipulację stanami zawierającymi informacje o zalogowanym użytkowniku a także odpowiedzialne są za aktualizowanie informacji o błędach podczas rejestracji lub logowania.

Jeśli użytkownik jest zalogowany zamiast formularzy do logowania i rejestracji wyświetli się komponent TodoList zawierający indywidualną listę zadań użytkownika.

Render komponentu:

```
return (  
  <div className='todo-app'>  
    {userID==='?' ? (  
      <>  
        <h1 style={{color: "white", fontSize: "40px"}}>Todo App</h1>  
        <LoginForm error={error} setError={setError} handleChangePassword={handleChangePassword}  
          handleChangeUsername={handleChangeUsername} login={login}/>  
        <hr className='hr-form'></hr>  
        <RegisterForm regerror={regerror} setRegerror={setRegerror} handleChangePassword={handleChangePassword}  
          handleChangeUsername={handleChangeUsername} register={register}/>  
      </>  
    ) : (  
      <TodoList data={passData()} setUser={setUser} setUserID={setUserID}/>  
    )  
  }  
  </div>  
)
```

Funkcja logowania wysyła zapytanie POST do API serwerowego z zapytaniem o sprawdzenie czy użytkownik o takim hasle i loginie istnieje. Jeśli serwer zwróci status success dane użytkownika zostają zapisane w stanach a strona przechodzi do widoku listy. Jeśli nie wyświetlony zostanie błąd. Strona wykorzystuje również mechanizm sessionStorage dzięki któremu po odświeżeniu strony użytkownik dalej pozostaje zalogowany:

```
const [user, setUser] = useState(sessionStorage.getItem('user')===null?'':sessionStorage.getItem('user'));
const [userID, setUserID] = useState(sessionStorage.getItem('userID')===null?'':sessionStorage.getItem('userID'));
```

```
const login = () => {
  var requestOptions = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({'username': username, 'password': password})
  };
  fetch('/login', requestOptions).then(
    res => res.json()
  ).then(
    data => {
      if (data.success) {
        setUser(data.user);
        setUserID(data.id);
        setError('');
        sessionStorage.setItem('user', data.user);
        sessionStorage.setItem('userID', data.id);
      }
      else{
        setError(data.error);
      }
    }
  )
}
```

LoginFrom i RegisterForm

LoginFrom i RegisterForm są prawie tym samym komponentem. Różnią się jednym polem danych, komunikatami błędów oraz wykonywaną przez formularz funkcją. Oba są proste w budowie więc nie ma co tutaj zbyt dużo o nich pisać.

```

<h2 style={{color: "white", padding: "0 0 15px 0", textDecoration: "underline"}}>Rejestracja</h2>
<div style={{color: "white"}}>
  <form>
    <div className='form-login'>
      <div className='div-login'>
        <div className='label-login'>Nazwa użytkownika:</div>
        <input className='input-login' type='text' name='username' id='regnazwa' onChange={handleChangeUsername}/>
      </div>
      <div className='div-login'>
        <div className='label-login'>Hasło:</div>
        <input className='input-login' type='password' name='password' id='regpass' onChange={handleChangePassword}/>
      </div>
      <div className='div-login'>
        <div className='label-login'>Powtórz hasło:</div>
        <input className='input-login' type='password' name='password2' id='regpass2' />
      </div>
      <button className='login-btn' onClick={(e) => {
        e.preventDefault();
        let check = document.getElementById('regpass').value === document.getElementById('regpass2').value;

        let emptyNameOrPass = document.getElementById('regpass').value !== ''
        && document.getElementById('regnazwa').value !== ''
        && document.getElementById('regpass2').value !== '';

        if(check && emptyNameOrPass){
          register();
        }
        else{
          if(!emptyNameOrPass){
            setRegerror("Nie wypełniono wszystkich pól");
          }
          else{
            setRegerror("Hasła nie są takie same");
          }
        }
      }}>Zarejestruj</button>
      <div className='error-log'>{regerror}</div>
    </form>
  </div>

```

TodoList

TodoList jest komponentem wyświetlanym po zalogowaniu użytkownika i głównym elementem sterującym widoku, ponieważ to wewnątrz niego znajdują się wszystkie funkcje odpowiadające za komunikowanie się z API serwera. Posiada stan todos który jest listą zadań danego użytkownika. W momencie wywołania dowolnej metody komunikującej się z API serwer po wprowadzeniu ewentualnych zmian w bazie wysyła aktualną listę zadań która jest aktualizowana automatycznie przez mechanizm stanu React. Na początku pobierana jest lista zadań która później w przypadku akcji użytkownika jest aktualizowana:

```

const fetchTodos = () => {
  fetch('/todos/'+id).then(
    res => res.json()
  ).then(
    data => {
      console.log(data);
      setTodos(data);
    }
  )
}

useEffect(fetchTodos, []);

```

Przykładowa funkcja komunikująca się z API wygląda w następujący sposób:

```
const updateTodo = (todoId, currentValue, newValue, oldPrio, newPrio, oldDate, newDate) => {
  if(newPrio === oldPrio){
    if(newDate === oldDate){
      if (newValue === '' || /\s*$/.test(newValue) || currentValue === newValue) {
        return;
      }
    }
  }
  console.log(todoId, currentValue, newValue, oldPrio, newPrio, oldDate, newDate);
  fetch("/editTodo/"+todoId+"/"+newValue+"/"+newPrio+"/"+newDate+"/"+passedData.data.id).then(
    res => res.json()
  ).then(
    data => {
      setTodos(data);
    }
  )
};
```

Najpierw sprawdza czy zadania zostało zmienione lub czy nie zostały wprowadzone puste lub błędne dane. Następnie jeśli wszystko jest w porządku wysyła zapytanie GET do serwera i czeka na odpowiedź. Odpowiedzią jest lista zadań danego użytkownika.

Render komponentu najpierw tworzy obiekt TodoForm a następnie mapuje listę zadań generując obiekt Todo dla każdego zadania. Na końcu dodawany jest przycisk Wyloguj który usuwa informacje o użytkowniku ze stanów oraz z sessionStorage:

```
return (
  <>
    <h1>Twoja lista zadań {passedData.data.user}</h1>
    <TodoForm onSubmit={addTodo} />
    {todos.map((todo, index) => (
      <Todo todo={todo} index={index} key={index} completeTodo={completeTodo} removeTodo={removeTodo} updateTodo={updateTodo}/>
    ))}
    <div className='logout-div'><form onSubmit={() => {
      sessionStorage.clear();
      setUser('');
      setUserID('');
    }}><button className='logoutBtn'>Wyloguj</button></form></div>
  </>
);
```


Todo

Todo jest komponentem odpowiadającym jednemu zadaniu. Wyświetla on wszystkie informacje o zadaniu a także zawiera 2 przyciski służące do usunięcia zadaniu lub przejścia do jego edycji. Todo zawiera stan edit który określa w jaki stanie w danej chwili jest zadanie. Jeśli zadanie nie jest edytowane wyświetla się standardowy kafelek z informacjami oraz przyciskami. Jeśli jest w trybie edycji zamiast kafelka generowany jest TodoForm. Dzięki niemu użytkownik ma możliwość prowadzenia innych danych do zadania.

```
const [edit, setEdit] = useState({
  id: null,
  value: '',
  prio: '',
  date: ''
})

const submitUpdate = (value, prio, date) => {
  updateTodo(edit.id, edit.value, value, edit.prio, prio, edit.date, date);
  setEdit({
    id: null,
    value: '',
    prio: '',
    date: ''
  });
};

if (edit.id) {
  return <TodoForm edit={edit} onSubmit={submitUpdate} />;
}
```

```
return (
  <div
    className={todo.isComplete ? 'todo-row complete' : 'todo-row'} key={index}>
    <div key={todo.id} onClick={() => completeTodo(todo.id)}>
      {todo.text}
    </div>
    <div className='todo-info'>
      <div className='todo-space'>
        Termin: {todo.date} | Priorytet: {prioToString(todo.prio)}
      </div>
      <div className='icons'>
        <TiEdit onClick={() => setEdit({id: todo.id, value: todo.text, prio: todo.prio, date: todo.date})} className='edit-icon' />
        <RiCloseCircleLine onClick={() => removeTodo(todo.id)} className='delete-icon' />
      </div>
    </div>
  </div>
);
```

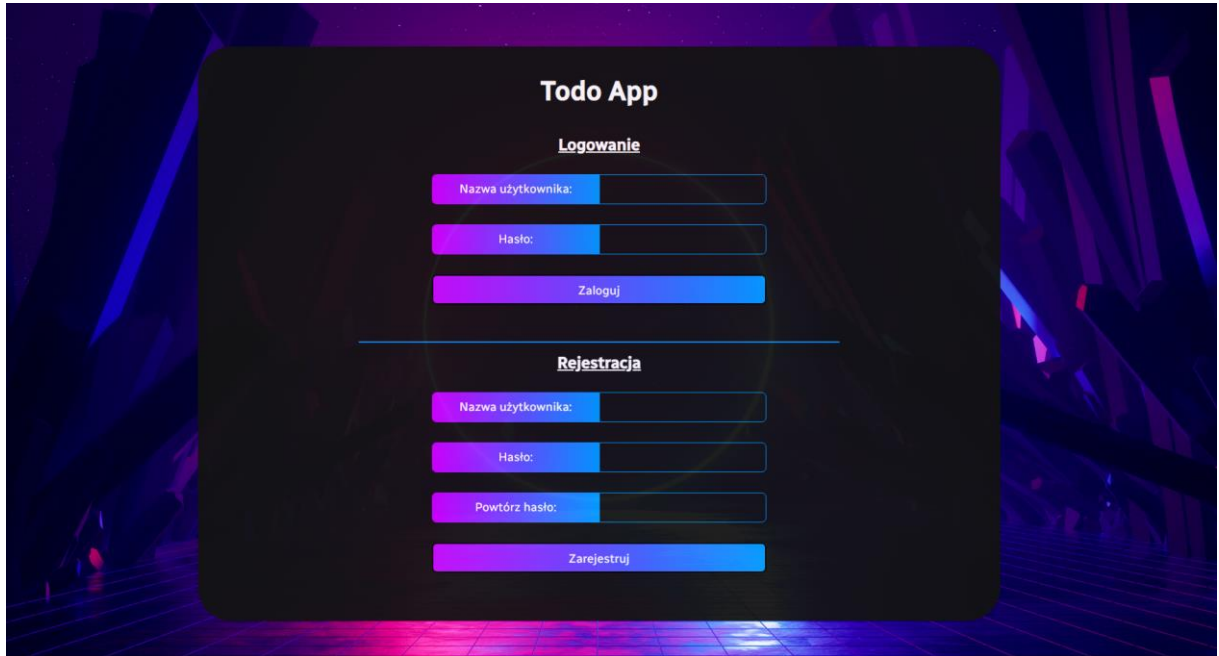
TodoForm

TodoForm jest formularzem umożliwiającym dodawanie lub wcześniej omawiane edytowanie zadania. Funkcja którą ma realizować jest mu przekazywana z komponentu nadrzędnego. Zawiera on stany dzięki którym zapamiętuje wartość pól wejściowych. Render różni się jedynie wykonana funkcją oraz klasami CSS w zależności od tego czy zadaniem jest edycja czy dodawania zadania.

```
return (
  <form className="todo-form" onSubmit={handleSubmit}>
    {props.edit ? (
      <div className="flex-hor">
        <input type="text" placeholder='...' value={input} name="text" className='todo-input edit' onChange={handleChange} ref={inputRef}/>
        <input type="date" className='date-pic-edit' defaultValue={date} onChange={handleDate}/>
        <div className='selectdiv-edit'>
          <label>
            <select name='prio' value={prio} className='todo-prio' onChange={handlePrio}>
              <option value="low">Opcjonalne</option>
              <option value="medium">Normalne</option>
              <option value="high">Ważne</option>
            </select>
          </label>
        </div>
        <button className='todo-button edit'>Zapisz</button>
      </div>
    ) : (
      <div className="flex-hor">
        <input type="text" placeholder='np. Posprzątaj garaż' value={input} name="text" className='todo-input' onChange={handleChange} ref={inputRef}/>
        <input type="date" className='date-pic' defaultValue={"2022-01-01"} onChange={handleDate}/>
        <div className='selectdiv'>
          <label>
            <select name='prio' value={prio} className='todo-prio' onChange={handlePrio}>
              <option value="low">Opcjonalne</option>
              <option defaultValue value="medium">Normalne</option>
              <option value="high">Ważne</option>
            </select>
          </label>
        </div>
        <button className='todo-button'>Dodaj zadanie</button>
      </div>
    )}
  </form>
)
```

5. Prezentacja strony

Strona dla użytkownika niezalogowanego:



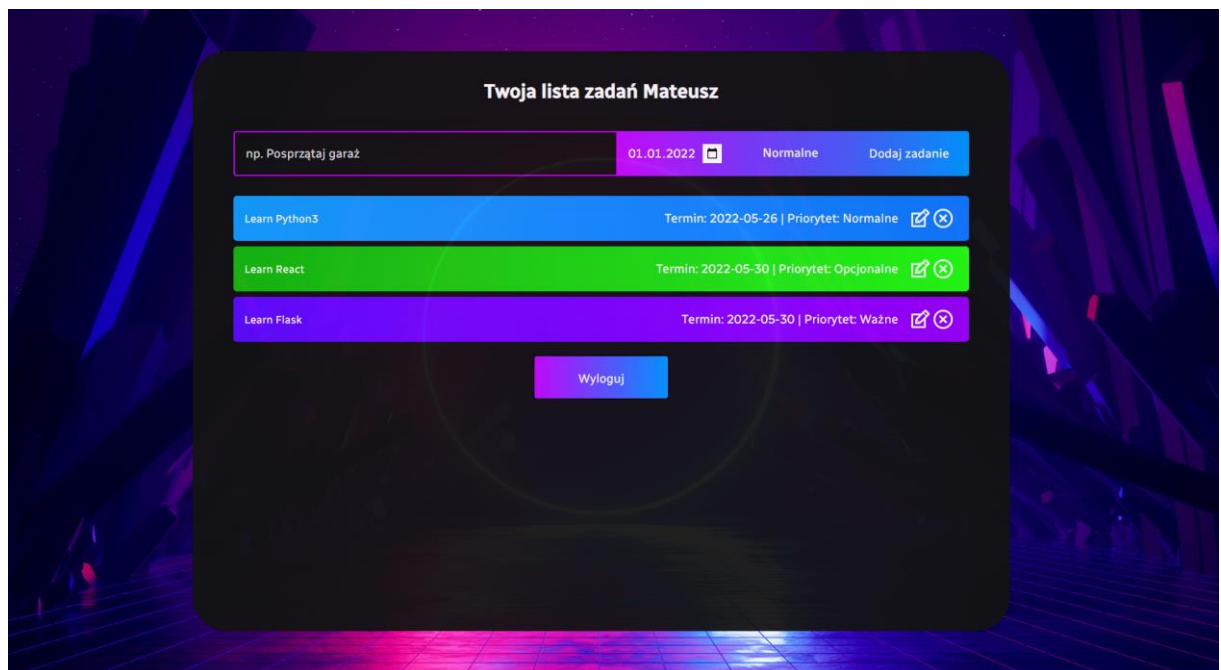
The screenshot shows a dark-themed web interface for a 'Todo App'. It features two main sections: 'Logowanie' (Login) and 'Rejestracja' (Registration). The 'Logowanie' section has input fields for 'Nazwa użytkownika:' and 'Hasło:', followed by a 'Zaloguj' button. The 'Rejestracja' section has input fields for 'Nazwa użytkownika:', 'Hasło:', and 'Powtórz hasło:', followed by a 'Zarejestruj' button. The background is a dark, abstract image with purple and blue light effects.

Błędy przy rejestracji lub logowaniu:

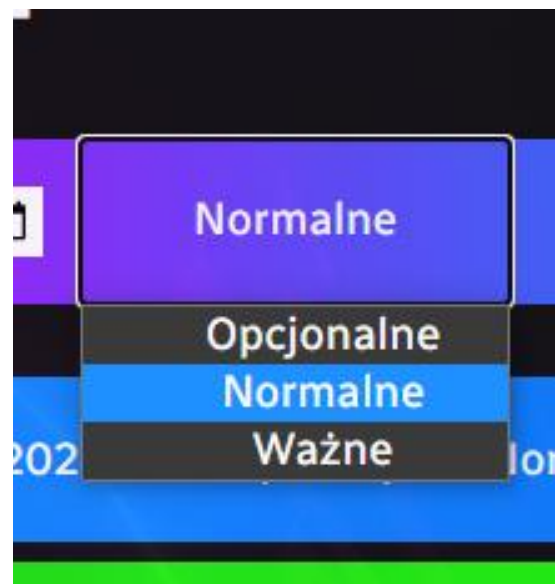
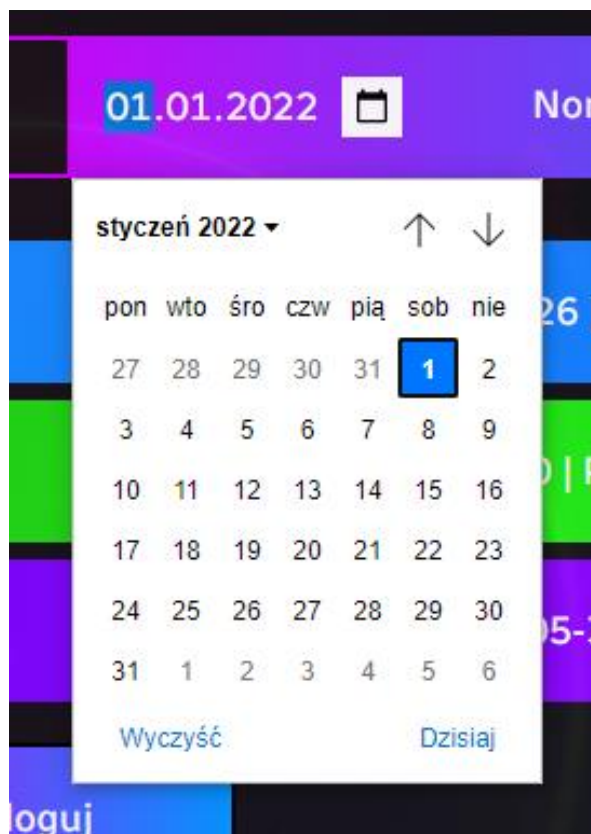


This screenshot shows the same 'Todo App' interface but with error messages. In the 'Logowanie' section, the 'Nazwa użytkownika:' field has a red error message 'Błąd'. Below the 'Zaloguj' button, a red message reads 'Nie wypełniono wszystkich pól'. In the 'Rejestracja' section, the 'Nazwa użytkownika:' field contains the text 'Mateusz'. The 'Hasło:' field contains seven dots '.....'. The 'Powtórz hasło:' field contains three dots '...'. Below the 'Zarejestruj' button, a red message reads 'Hasła nie są takie same'.





Strona po zalogowaniu:









Wybór daty i priorytetu:



Edycja pierwszego zadania:





np. Posprzątaj garaż	01.01.2022	Normalne	Dodaj zadanie
Learn Python3	26.05.2022	Normalne	Zapisz
Learn React	Termin: 2022-05-30 Priorytet: Opcjonalne		 
Learn Flask	Termin: 2022-05-30 Priorytet: Ważne		 

Zadania odznaczone jako wykonane:

np. Posprzątaj garaż	01.01.2022	Normalne	Dodaj zadanie
Learn Python3	Termin: 2022-05-26 Priorytet: Normalne		 
Learn React	Termin: 2022-05-30 Priorytet: Opcjonalne		 
Learn Flask	Termin: 2022-05-30 Priorytet: Ważne		 

Usunięte zadanie:

Twoja lista zadań Mateusz

np. Posprzątaj garaż	01.01.2022	Normalne	Dodaj zadanie
Learn Python3	Termin: 2022-05-26 Priorytet: Normalne		 
Learn React	Termin: 2022-05-30 Priorytet: Opcjonalne		 

Wyloguj