TAB projekt
Firma Szkoleniowa

Rok akademicki: 2021/2022

Informatyka Sem. 6, Gr. ISMIP1
Section composition:
Szymon Borończyk
Zuzanna Erd
Dawid Furs
Kacper Kozak
Tomasz Wąsiewicz

1. Cel projektu

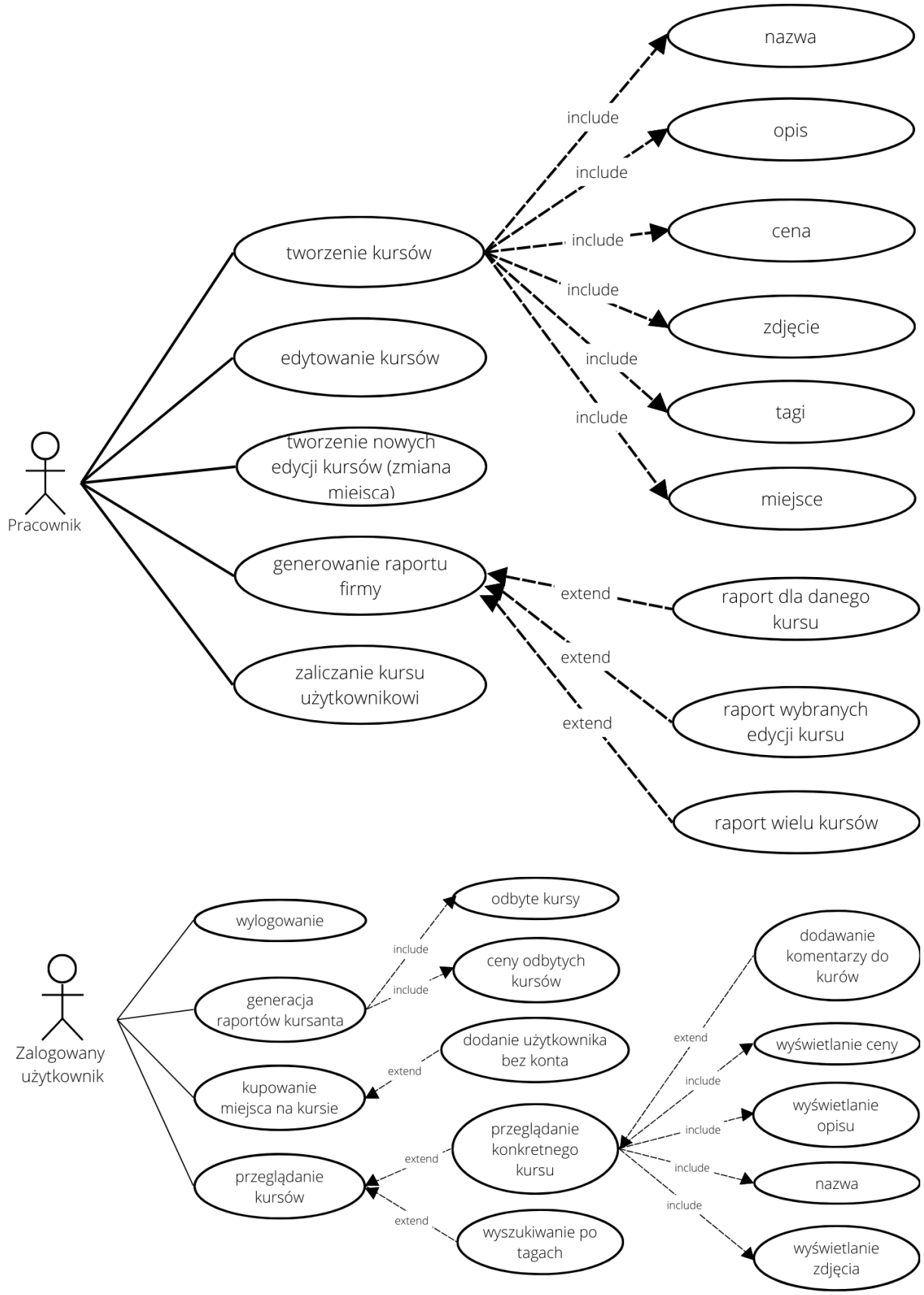
Celem projektu było stworzenie strony www - sklepu internetowego dla firmy prowadzącej szkolenia. System został zrealizowany zgodnie z wzorcem projektowym Model-Widok-Kontroler. W systemie oprócz podstawowej funkcjonalności sklepu internetowego można dodawać komentarze do odbytego już kursu. Ważną funkcjonalnością jest też pobieranie raportów: klienta (na jakich kursach był) oraz pracownika (jakie dochody ma firma).

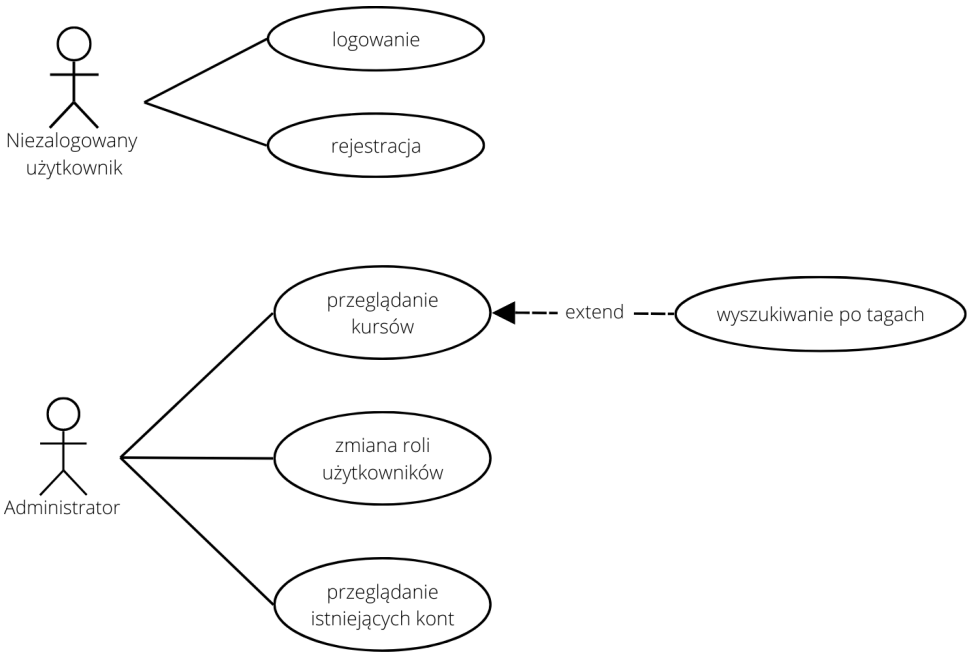
2. Analiza zadania

a. Funkcjonalności

Funkcjonalność	**Administrator**	**Użytkownik niezalogowany**	**Użytkownik zalogowany**	**Pracownik**
Rejestracja i logowanie		x		
Przeglądanie ofert	x	x	x	x
Zmiana roli użytkownika	x			
Wyszukiwanie kursów po tagach	x	x	x	x
Tworzenie kursów				x
Edytowanie kursów (możliwe, dopóki nikt się nie zapisał)				x
Tworzenie nowych edycji kursów (miejsce, czas zapisów i czasy rozpoczęcia/zakończenia)				x
Zapisy na edycję kursu (z możliwością kupienia miejsc dla bezkontowców)			x	
Przeglądanie danej edycji kursu	x	x	x	x
Dodawanie komentarzy do danego kursu			x	x
Ustawianie ceny pojedynczej edycji kursu				x
Ustawianie kosztu jaki ponosi firma za każdą edycję danego kursu				x
Generowanie raportu firmy (dla danego kursu/wybranych edycji/wielu kursów) o ilości sprzedanych miejsc, cenach, kosztach i wyliczonym zysku				x
Generowanie raportu kursanta (jakie kursy odbył i w jakich były cenach)			x	
Zaliczanie kursu użytkownikowi				x
Przeglądanie istniejących kont	x			

b. Model przypadków użycia





3. Część główna projektu

a. Opis instalacji:

1) Instrukcja instalacji projektu znajduje się na githubie, pod adresem: https://github.com/tomawas455/tabproject/blob/master/INSTALLATION.md

b. Najważniejsze klasy, metody i funkcje

1. Za każdy endpoint odpowiedzialna jest metoda z modułu routes. Metody te są udekorowane przez obiekt klasy Flask. Zostały one podzielone na 4 typy metod:

a. GET - do pobierania danych z bazy np.:

```python
@bp.route('/', methods=['GET'])
@only_worker
def get_courses():
    courses_page = (Course.query.order_by(Course.id)
                    .paginate(error_out=False, max_per_page=9999))
    return {
        "courses": [course.to_dict() for course in courses_page.items],
        "pages": courses_page.pages,
        "page": courses_page.page,
        "page_size": courses_page.per_page,
        "is_last": not courses_page.has_next
    }
```

b. POST - do dodawania nowych danych np.:

```python
@bp.route('/', methods=['POST'])
@only_worker
def create_city():
    name = request.form.get('city')

    if not name:
        raise BadRequest('Parameter should not be an empty string')

    if City.query.filter(City.city.ilike(name)).count() > 0:
        raise BadRequest('This City arleady exist!')

    city = City(name)
    db.session.add(city)
    db.session.commit()
    return city.to_dict()
```

c. DELETE - do usuwania danych np.:

```python
@bp.route('/<int:multimedia_id>', methods=['DELETE'])
@only_worker
def delete_multimedia(multimedia_id):
    multimedia = Multimedia.query.filter_by(id=multimedia_id).first()

    if not multimedia:
        raise NotFound('Multimedia with this id does not exist!')

    db.session.delete(multimedia)
    db.session.commit()

    return multimedia.to_dict()
```

d. PATCH - do edycji istniejących danych np.:

```python
@bp.route('/', methods=['PATCH'])
@only_worker
def edit_participation():
    data_json = request.get_json()
    if any(parameter not in data_json for parameter in ("user_id", "training_id", "passed")):
        raise BadRequest("Required parameters: user_id, training_id")
    participation = Participation.query.filter_by(
        user_id=data_json["user_id"], training_id=data_json["training_id"]).first()
    if participation is None:
        raise NotFound(
            "Participation with this user id and training id does not exist!")
    participation.passed = data_json["passed"]
    db.session.commit()
    return participation.to_dict()
```

2. Modele ORM:
   a. Do komunikacji z bazą danych skorzystaliśmy z biblioteki SQLAlchemy, która realizuje mapowanie obiektowo-relacyjne. Pozwala nam to tworzyć modele reprezentujące tabele w bazie danych oraz komunikować się z bazą danych z poziomu kodu.

   b. Modele znajdują się w module *backend/src/models*. Przykład modelu użytkownika:

```python
class User(BaseModel):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.Unicode(200), nullable=False, unique=True)
    name = db.Column(db.Unicode(100), nullable=False)
    surname = db.Column(db.Unicode(100), nullable=False)
    password = db.Column(db.UnicodeText(), nullable=False)
    role_id = db.Column(
        db.SmallInteger, db.ForeignKey(Role.id), nullable=False)

    role = db.relationship(Role)

    _default_fields = ['email', 'name', 'surname', 'role']
    _hidden_fields = ['password']

    def __init__(self, email, name, surname, password, role):
        self.email = email
        self.name = name
        self.surname = surname
        self.password = password
        self.role = role
```

3. Funkcje pomocnicze
   a. Ograniczenie dostępu, realizowane w postaci dekoratorów używanych przy endpointach. Dekorator sprawdza czy użytkownik wywołujący daną metodę ma rolę uprawniającą go do wywoływania danego endpointu.

```python
def check_roles(role_names):
    if g.user is None or g.user.role.name not in role_names:
        raise Unauthorized(
            "You need to have different permissions to see this"
        )


def only_user(view):
    @functools.wraps(view)
    def wrapped_view(**kwargs):
        check_roles(['user'])
        return view(**kwargs)

    return wrapped_view
```

b. Ładowanie informacji o zalogowanym użytkowniku. Dzięki tej funkcji, udekorowanej metodą *before_request* podczas obsługi zapytań, endpointy mają informację o zalogowanym użytkowniku, z bazy danych.

```python
@app.before_request
def load_logged_in_user():
    session_id = session.get('session_id')
    if session_id is None:
        g.user = None
    else:
        g.user = User.query.filter_by(id=session_id).first()
```

c. Tworzenie aplikacji. Funkcja ta importuje endpointy z modułu *routes* i rejestruje blueprinty w instancji Flaska, tak że będą przez niego obsługiwane.

```python
def create_app():
    from routes import (
        auth, users, trainings, courses, tags,
        places, cities, multimedia, meetings, comments, participations, raports

    )
    app.register_blueprint(auth.bp)
    app.register_blueprint(users.bp)
    app.register_blueprint(tags.bp)
    app.register_blueprint(places.bp)
    app.register_blueprint(cities.bp)
    app.register_blueprint(trainings.bp)
    app.register_blueprint(courses.bp)
    app.register_blueprint(multimedia.bp)
    app.register_blueprint(meetings.bp)
    app.register_blueprint(comments.bp)
    app.register_blueprint(participations.bp)
    app.register_blueprint(raports.bp)
```

4. Podział ról przy realizowaniu projektu

Osoba	**Zadania**
Tomasz Wąsiewicz	DevOps (pliki konfiguracyjne, boilerplate), backend endpoints, modele do BD
Szymon Brończyk	backend endpoints, modele do BD
Kacper Kozak	backend endpoints, modele do BD
Zuzanna Erd	frontend, dokumentacja, obrazki
Dawid Furs	frontend