Soviet_Architecture_DataBase

SADB

web project made as structured storage of data about soviet type architecture all across the world There will be information about soviet housebuilding series and their specific types

By: Plachotnick_Yaroslav
Torlakiyan Kirill

MANY_LIBRARY

A large number of libraries were used for the project (Required installation)

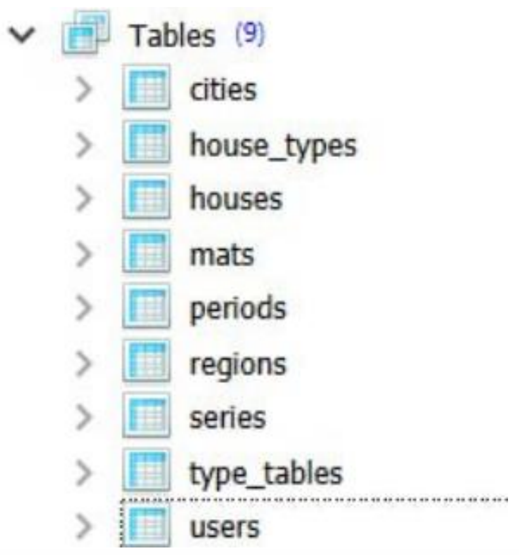
LIST_OF_LIBRARIES

- SERVER PART (FLASK's modules) - Flask, flask-wtf, flask-login, flask-restful
- HTML, CSS
- System – OS, HashLib
- DataBase – SQLalchemy

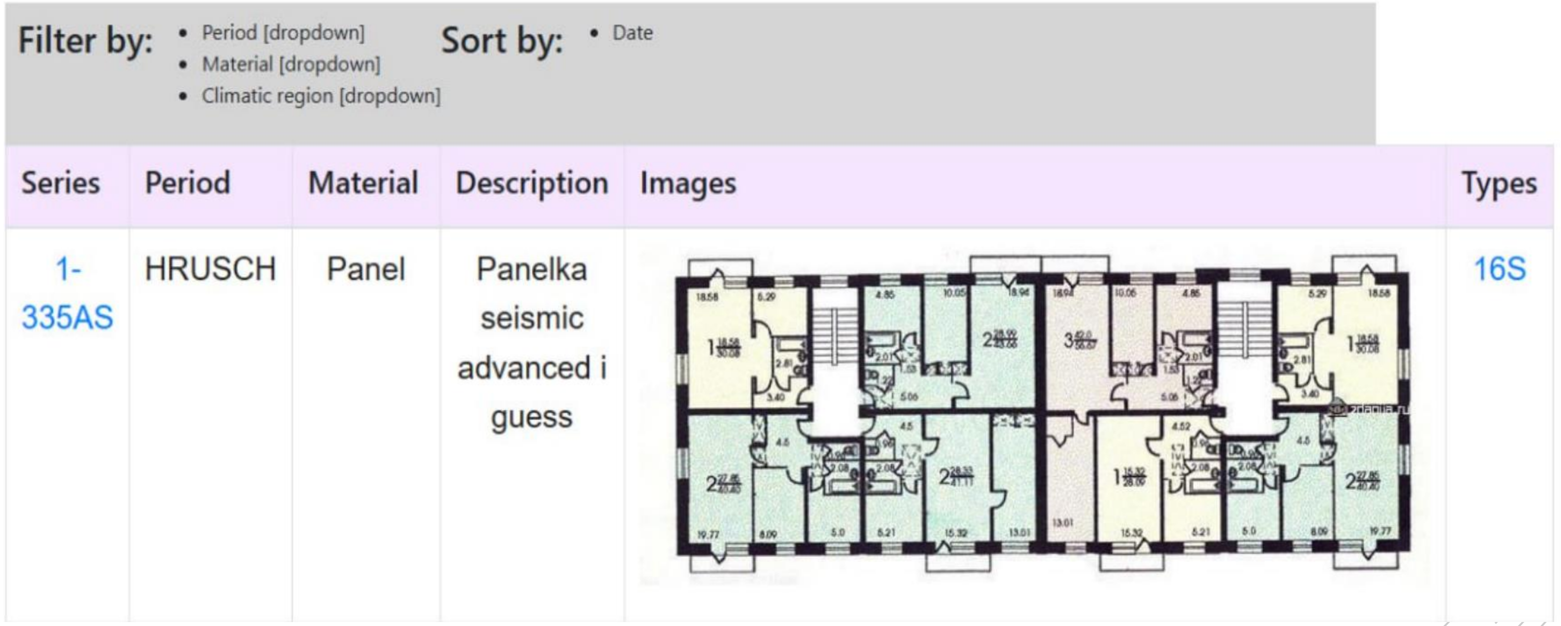
COMPLEX OF DATABASES AND USER's VIEW OF IT

HISTORICAL_ACCURACY

The aim of the project is to collect, store and organize historically accurate information about the architecture of the Soviet Union.

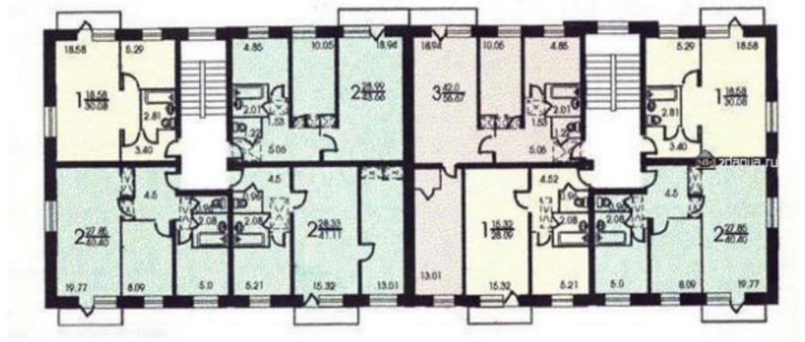


Tables (9)
- cities
- house_types
- houses
- mats
- periods
- regions
- series
- type_tables
- users



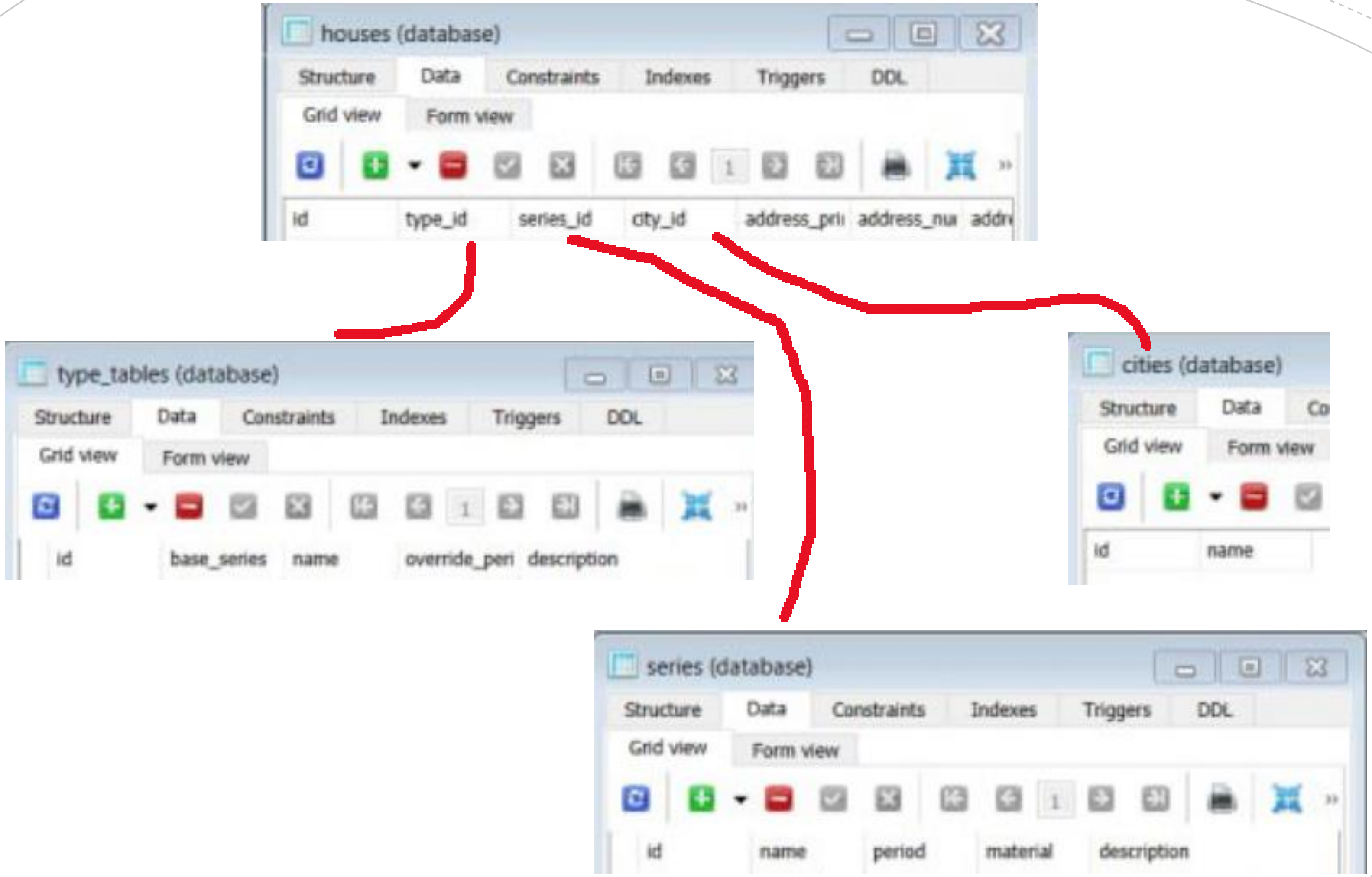
Filter by:
- Period [dropdown]
- Material [dropdown]
- Climatic region [dropdown]

Sort by:
- Date

Series	Period	Material	Description	Images	Types
1-335AS	HRUSCH	Panel	Panelka seismic advanced i guess		16S

DATA_SYSTEM

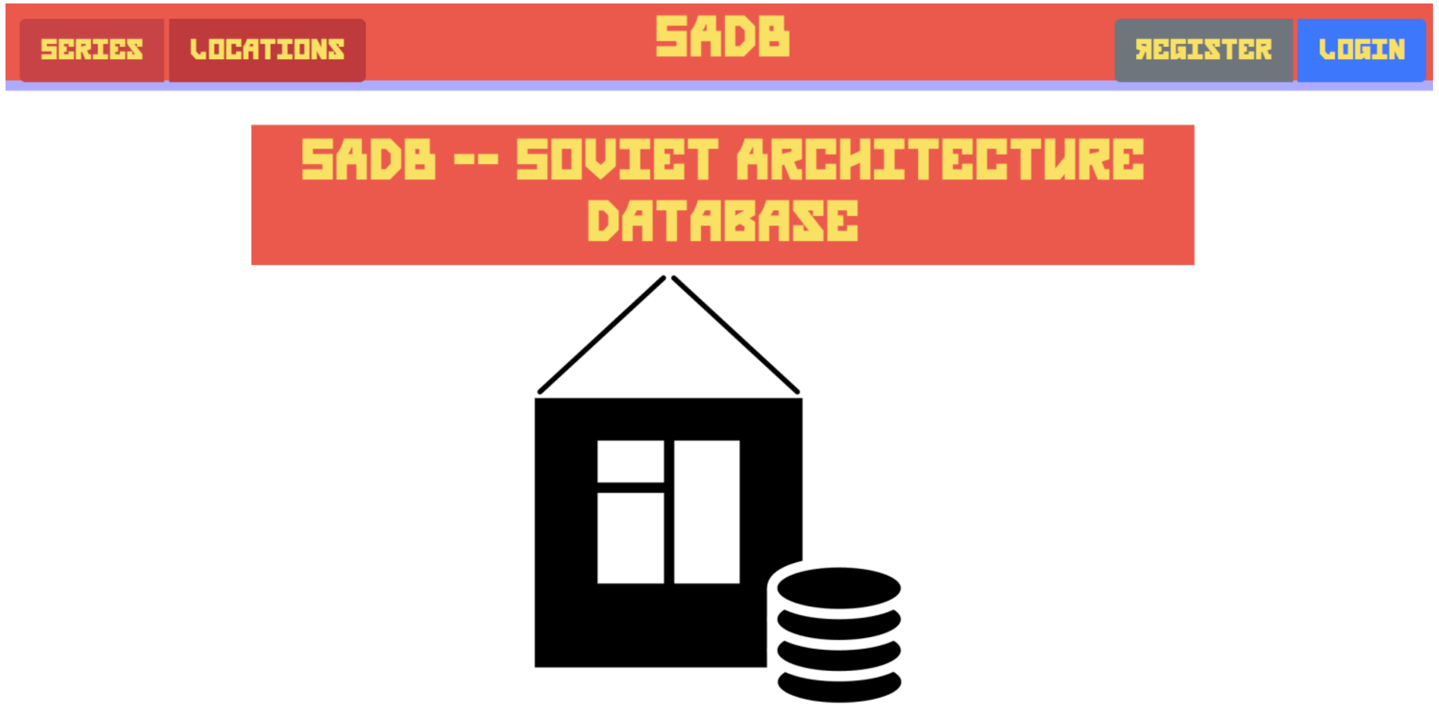
A variety of interconnected databases were used to simplify and streamline data storage.


houses (database)
Structure Data Constraints Indexes Triggers DDL
Grid view Form view
id type_id series_id city_id address_pri address_nu addr
type_tables (database)
Structure Data Constraints Indexes Triggers DDL
Grid view Form view
id base_series name override_peri description
cities (database)
Structure Data Co
Grid view Form view
id name
series (database)
Structure Data Constraints Indexes Triggers DDL
Grid view Form view
id name period material description

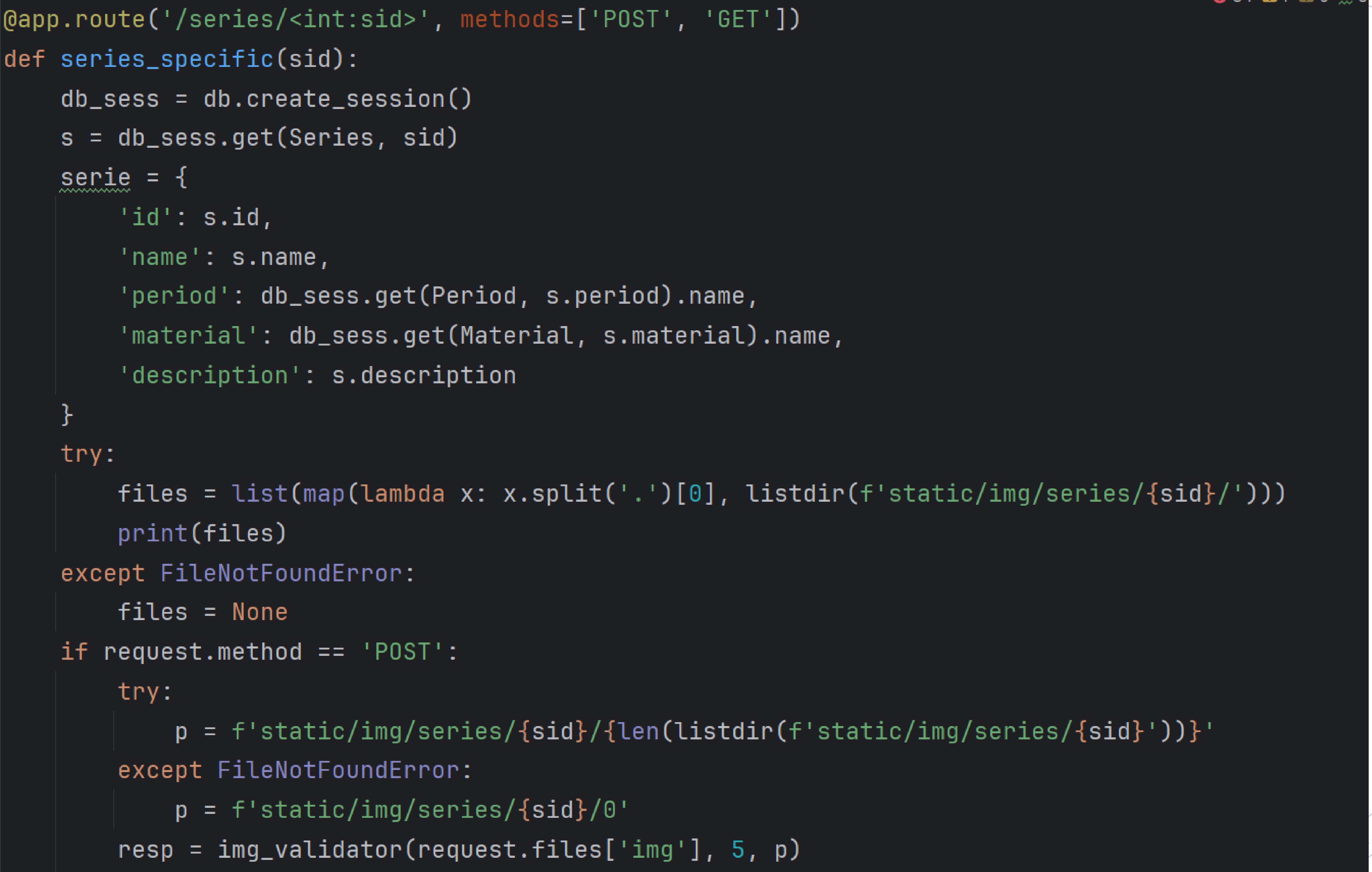
User-friendly_and_stylized

The interface was created similar to the style of the Soviet Union, while remaining user-friendly.

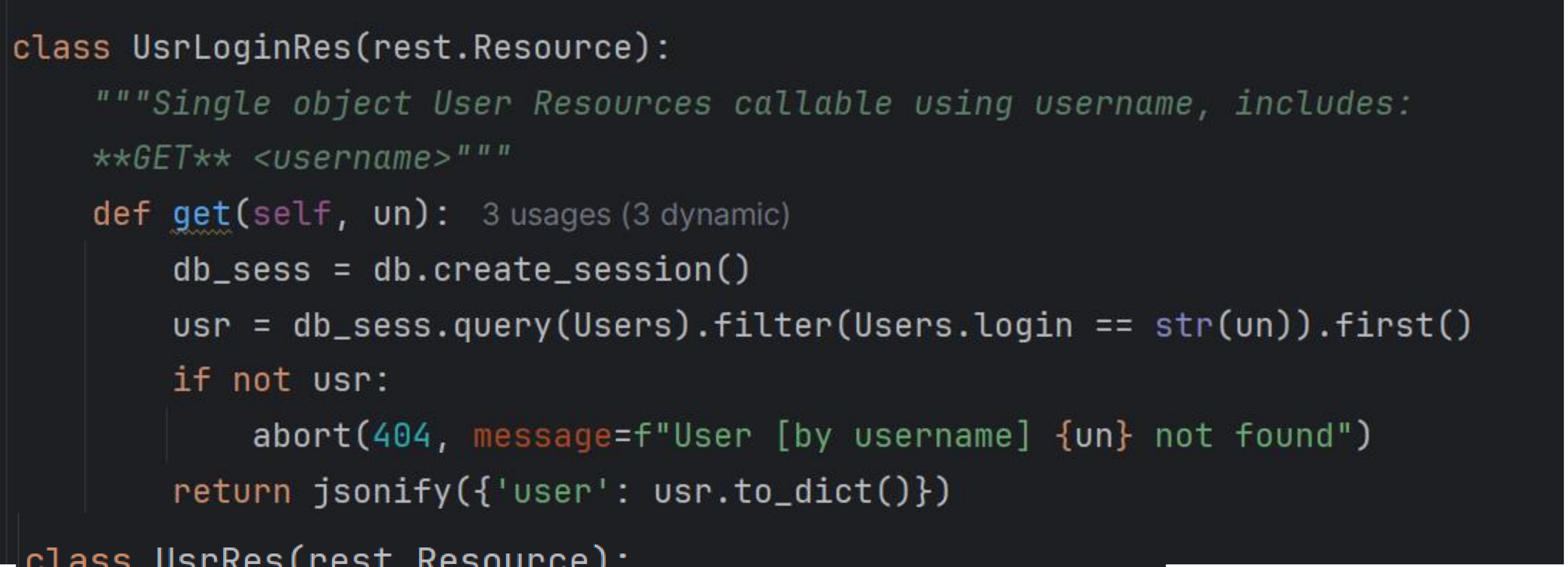
MAIN_PAGE



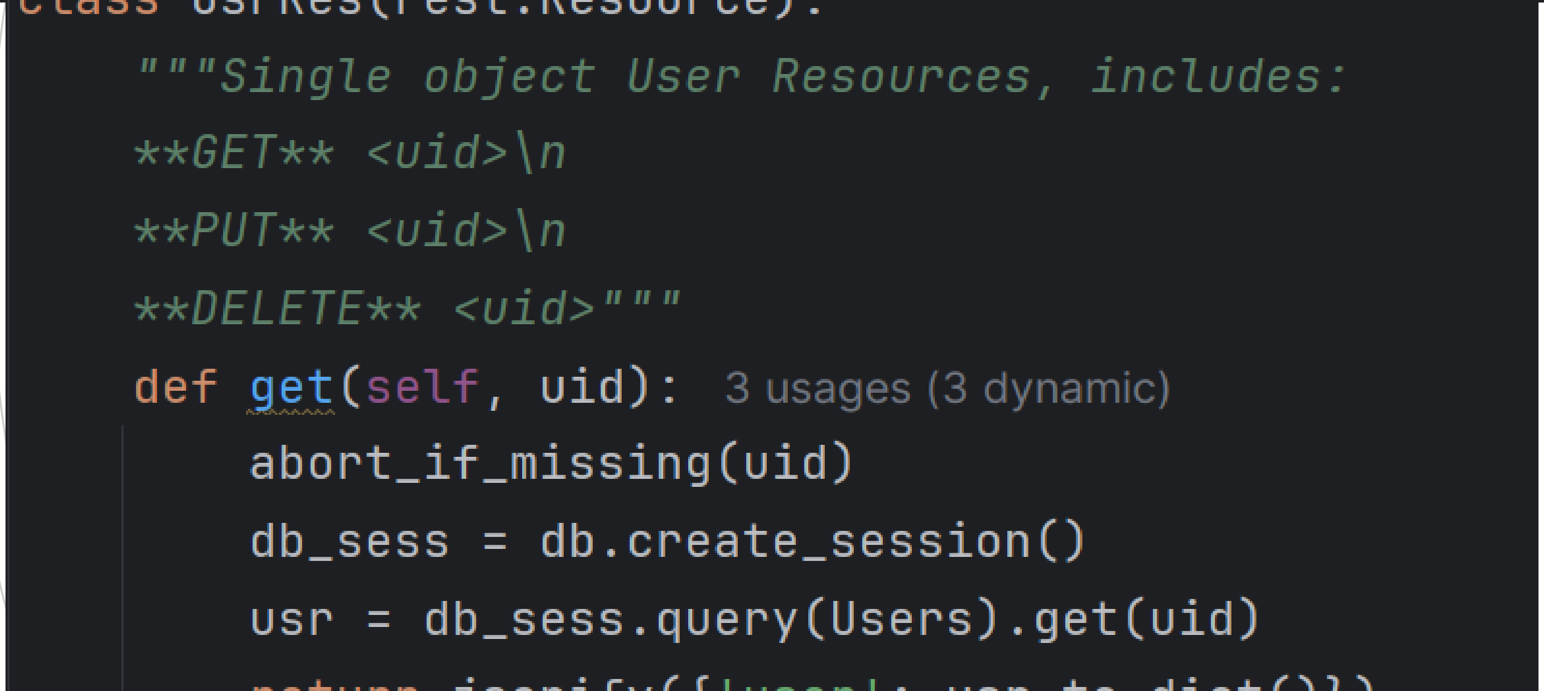
SADB -- SOVIET ARCHITECTURE DATABASE



```python
@app.route('/series/<int:sid>', methods=['POST', 'GET'])
def series_specific(sid):
    db_sess = db.create_session()
    s = db_sess.get(Series, sid)
    serie = {
        'id': s.id,
        'name': s.name,
        'period': db_sess.get(Period, s.period).name,
        'material': db_sess.get(Material, s.material).name,
        'description': s.description
    }
    try:
        files = list(map(lambda x: x.split('.')[0], listdir(f'static/img/series/{sid}/')))
        print(files)
    except FileNotFoundError:
        files = None
    if request.method == 'POST':
        try:
            p = f'static/img/series/{sid}/{len(listdir(f'static/img/series/{sid}'))}'
        except FileNotFoundError:
            p = f'static/img/series/{sid}/0'
        resp = img_validator(request.files['img'], 5, p)
```



```
class UsrLoginRes(rest.Resource):
    """Single object User Resources callable using username, includes:
    **GET** <username>"""
    def get(self, un):  3 usages (3 dynamic)
        db_sess = db.create_session()
        usr = db_sess.query(Users).filter(Users.login == str(un)).first()
        if not usr:
            abort(404, message=f"User [by username] {un} not found")
        return jsonify({'user': usr.to_dict()})
```



```
class UsrRes(rest.Resource):
    """Single object User Resources, includes:
    **GET** <uid>\n
    **PUT** <uid>\n
    **DELETE** <uid>"""
    def get(self, uid):  3 usages (3 dynamic)
        abort_if_missing(uid)
        db_sess = db.create_session()
        usr = db_sess.query(Users).get(uid)
```