

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій

Кафедра програмних систем і технологій

Індивідуальне завдання № 1
Тема: «Створення SQL database Azure»

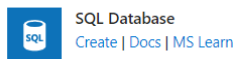
Дисципліна «Хмарні технології»

Підготував:
студент гр. ПІЗ-43(1)
Мішак Максим

Для виконання роботи було обрано хмарний провайдер Azure, та базу даних SQL Azure. Для використання бази даних потрібно наслідувати декілька правил. Перш за все база даних повинна бути інкапсульована, що означає її незалежність та недоступність для зміни з будь якого місця. Також потрібно налаштувати правила авторизації, щоб тільки довірені джерела могли виконувати операції по її редагуванню. База даних та вся супутня інфраструктура буде розгортатись в хмарі. Для цього потрібно створити її екземпляр.



Щоб створити її екземпляр потрібно перейти в Create a resource, після чого обрати SQL databases



Після цього нам відкриється форма, яку потрібно заповнити. Форма включає в себе всі параметри об'єму, потужності, захищеності та правил підключення.

Basics Networking Security Additional settings Tags Review + create

Create a SQL database with your preferred configurations. Complete the basic tab then go to Review + Create to provision with exact defaults, or visit each tab to customize. [Learn more](#) of

Want to try Azure SQL Database for free? Create a free developer database with the first 100,000 vCore seconds, 10GB of data, and 10GB of backup storage free per month for the lifetime of the subscription. [Learn more](#) of

[Apply offer \(free tier\)](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * () Azure subscription 1

Resource group * () mssql_db_resource

[Create new](#)

Database details

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources.

Database name *

Server * () sql-db-server (East US)

[Create new](#)

Want to use SQL elastic pool? () ☒ Yes ☐ No

Workload environment ☐ Development ☒ Production

Compute + storage * ()

General Purpose
Standard server (Gen2), 1 vCore, 10 GB storage, zone-redundant disabled
[Configure database](#)

Backup storage redundancy

Choose how your PITR and TLA backups are replicated. Geo restore or ability to recover from regional outage is only available when geo-redundant storage is selected.

Backup storage redundancy () ☐ Locally-redundant backup storage ☐ Zone-redundant backup storage ☒ Geo-redundant backup storage

Warning Selected value for backup storage redundancy is Geo-redundant backup storage. Standard backup will be geo-replicated which might impact your data residency requirements. [Learn more](#) of

Cost summary

General Purpose (GP_Gen2_1)	
Provisioned capacity (vCores)	104,031 x 2
Provisioned capacity (vCores)	x 2
Provisioned capacity (vCores)	x 132
Max storage (vCores)	x 41.6
Estimated cost / resource	372.937 USD

Важливими параметрами є:

- Resource group

- Server
- Backup location

Також потрібно заповнити поля з назвою бази даних, налаштувати середовище роботи ,використання elastic pool. Для своєї бази даних я обрав такі параметри: назва бази даних sql_db, середовище роботи: development, elastic pool - false. Параметри середовища:

Home > Create a resource >

Configure

Feedback

Service and compute tier

Select from the available tiers based on the needs of your workload. The vCore model provides a wide range of configuration controls and offers Hyperscale and Serverless to automatically scale your database based on your workload needs. Alternately, the DTU model provides set price/performance packages to choose from for easy configuration. [Learn more](#)

Service tier: General Purpose (Most budget friendly)

Compute tier: ☐ Provisioned - Compute resources are pre-allocated. Billed per hour based on vCores configured. ☒ Serverless - Compute resources are auto-scaled. Billed per second based on vCores used.

Compute Hardware

Select the hardware configuration based on your workload requirements. Availability of compute optimized, memory optimized, and confidential computing hardware depends on the region, service tier, and compute tier.

Hardware Configuration: Standard-series (Gen5)
up to 80 vCores, up to 240 GB memory
[Change configuration](#)

Max vCores: 1

Min vCores: 0.5 vCores

2.02 GB MIN MEMORY 3 GB MAX MEMORY

Auto-pause delay

The database automatically pauses if it is inactive for the time period specified here, and automatically resumes when database activity recurs. Alternatively, auto pausing can be disabled.

☒ Enable auto-pause

Days: 0 Hours: 1 Minutes: 0

Data max size (GB): 1

307.2 MB LOG SPACE ALLOCATED

Would you like to make this database zone redundant? ☐ Yes ☒ No

Cost summary	
General Purpose (GP_S_Gen5_1)	
Cost per GB (in USD)	0.12
Max storage selected (in GB)	x 1.3
ESTIMATED STORAGE COST / MONTH	0.15 USD
COMPUTE COST / VCORE SECOND	0.000145 USD

NOTES

¹ Serverless databases are billed in vCore seconds based on a combination of CPU and memory utilization. [Learn more about serverless billing](#)

Тут я обрав без серверну архітектуру, віртуальне ядро: 0,5 - 1, автовимкнення: через годину, розмір бази даних: 1 гігабайт, решту налаштувань залишив стандартними. Після цього було налаштовано сервер.

[Home](#) > [Create a resource](#) > [Create SQL Database](#) >

Create SQL Database Server

Microsoft

Server details

Enter required settings for this server, including providing a name and location. This server will be created in the same subscription and resource group as your database.


Server name *

Enter server name
...database.windows.net

Location *

(US) East US

Authentication

 Azure Active Directory (Azure AD) is now Microsoft Entra ID. [Learn more](#)

Select your preferred authentication methods for accessing this server. Create a server admin login and password to access your server with SQL authentication, select only Microsoft Entra authentication [Learn more](#) using an existing Microsoft Entra user, group, or application as Microsoft Entra admin [Learn more](#), or select both SQL and Microsoft Entra authentication.

Authentication method

- ☒ Use Microsoft Entra-only authentication
☐ Use both SQL and Microsoft Entra authentication
☐ Use SQL authentication

Set Microsoft Entra admin *

Not Selected
[Set admin](#)

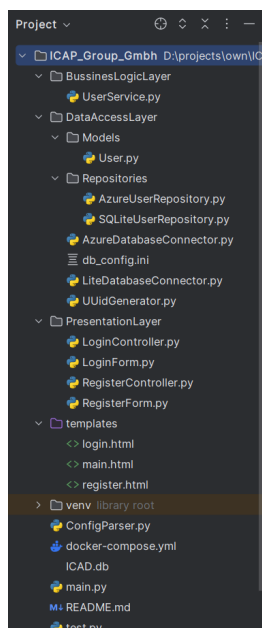
Назва сервера: sql-db-server, локація West EU, авторизація SQL and Entra authentication. Для ролі адміністратор обрав тільки власний акаунт. Після цього потрібно провести налаштування правил доступу. Для цього на вкладці Networking було обрано доступ з публічного ендпоінта, дозволено іншим сервісам інтегрувати з базою даних, та додано мій зовнішній IP адрес. Після цього було розгорнуто базу даних з усією супутньою архітектурою. Для її використання з додатку потрібно отримати connection string. В залежності від обраного методу авторизації нюанси використання connection string можуть бути різними. В моєму випадку я використовував авторизацію по SQL. Для отримання рядка переходимо на вкладку

Connections. Я буду використовувати odbc драйвер для, python, тому копіюю відповідний рядок. На порталі надано стрічку без пароля. Для доступу він потрібен. Щоб не зберігати його у відкритому вигляді, я буду використовувати файл конфігурації, та клас, який буде зчитувати значення з конфігурації.

```
1 import configparser
2
3
4 class ConfigReader:
5
6     def __init__(self):
7         self.config = configparser.ConfigParser()
8         self.config.read('DataAccessLayer/db_config.ini')
9         self.__sql_lite_string = self.config.get(section='SQLite_connection', option='sqlite_connection_string')
10        self.__azure_db = self.config.get(section='Azure_SQL_connection', option='connection_string')
11
12    def get_sqlite_string(self):
13        return self.__sql_lite_string
14
15    def get_azure_string(self):
16        return self.__azure_db
17
```

Цей клас зчитує значення з ini файлу, завдяки цьому приватна та чутлива інформація залишається поза кодом, через що стає складніше її отримати третім лицам.

Архітектуру додатку я обрах трьохрівневу, тому, що вона гарно підходить для нарощування функціоналу, та використання різних баз даних з однією моделлю. Структура проекту виглядає має вигляд класичної трирівневої архітектури та поділена на три рівні



Кожен рівень відповідальний за конкретний аспект, та виконує тільки певний спектр дій.

Рівень представлення складається з контролера користувача, контролера реєстрації, та двох форм для них.

```
1  from flask_wtf import FlaskForm
2  from wtforms import StringField, PasswordField
3  from wtforms.validators import DataRequired
4
5
6  class Login(FlaskForm):
7      login = StringField(label="login", validators=[DataRequired()])
8      password = PasswordField(label="password", validators=[DataRequired()])
9
```

1. Форма логування

```
1  from flask_restful import Resource
2  from flask import request, render_template, make_response, redirect
3  from PresentationLayer.RegisterForm import Register
4  from BussinesLogicLayer.UserService import UserService
5
6
7  class RegisterRequest(Resource):
8      @staticmethod
9      def get():
10         form = Register()
11         template = render_template(template_name_or_list='register.html', form=form)
12         response = make_response(template)
13         response.headers['Content-Type'] = 'text/html'
14         return response
15
16     @staticmethod
17     def post():
18         form = Register(request.form)
19         userService = UserService
20         userService.store_data(form.data)
21         userService.store_data_azure(form.data)
22         return redirect('/login')
```

2. Контролер логування

```

1  from flask_wtf import FlaskForm
2  from wtforms import StringField, PasswordField, SubmitField
3  from wtforms.validators import DataRequired
4
5
6  class Register(FlaskForm):
7      name = StringField(label='name', validators=[DataRequired()])
8      surname = StringField(label='surname', validators=[DataRequired()])
9      login = StringField(label='login', validators=[DataRequired()])
10     password = PasswordField(label='password', validators=[DataRequired()])
11

```

3. Форма реєстрації

```

1  from flask_restful import Resource
2  from flask import request, render_template, make_response, redirect
3  from PresentationLayer.LoginForm import Login
4  from BussinesLogicLayer.UserService import UserService
5
6
7  class LoginRequest(Resource):
8      @staticmethod
9      def get():
10         form = Login()
11         template = render_template('template_name_or_list:login.html', form=form)
12         response = make_response(template)
13         response.headers['Content-Type'] = 'text/html'
14         return response
15
16     @staticmethod
17     def post():
18         form = Login(request.form)
19         userservice = UserService
20         if userservice.compare_passwords(form.data['password'], userservice.get_password_sqlite(form.data['login'])):
21             return "pass is ok in sqlite and azure"
22         if userservice.compare_passwords(form.data['password'], userservice.get_password_sqlite(form.data['login'])):
23             return "pass is ok in sqlite"
24         else:
25             return "pass not ok not in sqlite nor azure"
26

```

4. Контролер реєстрації

Ці компоненти відповідають первинний вигляд даних, та їх отримання з користувацького інтерфейсу. Також за їх передачу на рівень бізнес логіки. Через те, що проект реалізовує тільки реєстрацію та логування - рівень бізнес логіки досить простий.

```

1 import bcrypt
2 from DataAccessLayer.LiteDatabaseConnector import LiteDatabaseConnector
3 from DataAccessLayer.AzureDatabaseConnector import AzureDatabaseConnector
4 from DataAccessLayer.Repositories.SQLiteUserRepository import UserRepository
5 from DataAccessLayer.Repositories.AzureUserRepository import AzureRepository
6 from DataAccessLayer.UuidGenerator import UuidGenerator
7
8
9 7 usages  ± Expand
10 class UserService:
11     2 usages  ± Expand
12     @staticmethod
13     def hash_password(password):
14         hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
15         return hashed_password
16
17     1 usage  ± Expand
18     @staticmethod
19     def check_password(plain_password, hashed_password):
20         return bcrypt.checkpw(plain_password.encode('utf-8'), hashed_password)

```

```

19 1 usage  ± Expand
20 @staticmethod
21 def store_data(form_data):
22     user_uuid = UuidGenerator
23     sqlite_db = LiteDatabaseConnector()
24     sqlite_db.create_tables()
25     user_repository = UserRepository(sqlite_db.session)
26     user_repository.sqlite_add_user(user_uuid.generate(),
27                                     form_data['name'],
28                                     form_data['surname'],
29                                     form_data['login'],
30                                     UserService.hash_password(form_data['password']))
31     sqlite_db.close_session()
32
33 1 usage  ± Expand
34 @staticmethod
35 def store_data_azure(form_data):
36     user_uuid = UuidGenerator
37     az_db = AzureDatabaseConnector()
38     azure = AzureRepository(az_db.return_connector())
39     azure.azure_add_user(user_uuid.generate(),
40                           form_data['name'],
41                           form_data['surname'],
42                           form_data['login'],
43                           UserService.hash_password(form_data['password']))

```

```

43 2 usages  ± Expand
44 ⚡ @staticmethod
45 def get_password_sqlite(login):
46     db = LiteDatabaseConnector()
47     user_repository = UserRepository(db.session)
48     stored_user = user_repository.sqlite_get_user_by_login(login)
49     return stored_user.password
50
51 1 usage  ± Expand
52 @staticmethod
53 def get_password_azure(login):
54     az_db = AzureDatabaseConnector()
55     azure = AzureRepository(az_db.return_connector())
56     return azure.azure_get_user(login)[0][4]
57
58 3 usages  ± Expand
59 @staticmethod
60 def compare_passwords(stored_password, current_password):
61     if UserService.check_password(stored_password, current_password):
62         return True
63     else: return False

```


Сервіс отримує данні. Після чого в різних випадках веде себе по різному. При реєстрації на рівні сервісу генерується uuid. Я використовую uuid для того, щоб не було проблем в разі заміни якихось записів, їх видалень і так далі. Для того, щоб уникнути конфліктів, також гарантовано мати унікальний ID, я використовую 36-ти значний рядок в якості ідентифікатора. Також, якщо користувач реєструєця, потрібно зберігати його пароль. Зберігати пароль у відкритому вигляді, так само, як і будь яку чутливу інформацію, у відкритому вигляді не можна. Тому я використовую хешування паролю за допомогою bcrypt. При логуванні користувача, ми викликаємо метод, який отримує по логіну користувача так звану “сіль” паролю. Сіль пародю це пароль в хешованому вигляді. Після чого, пароль який надав користувач порівнюється з сіллю паролю, яка збережена в базі даних для цього користувача. В разі їх співпадіння - користувач отримує повідомлення pass is ok in sqlite and azure, якщо в обох базах даних пароль співпадає, або pass is ok in sqlite, якщо співпадає тільки в локальній базі даних.

Наступним в ієрархії є рівень представлення даних. Він відповідає за взаємодію з базами даних та застосування моделей.

Для підключення до обох баз даних я використовую відповідні класи-конектори.

```
1 from sqlalchemy import create_engine
2 from sqlalchemy.orm import sessionmaker
3 from DataAccessLayer.Models.User import Base
4 from ConfigParser import ConfigReader
5
6
7 class LiteDatabaseConnector:
8     def __init__(self):
9         self.conf_reader = ConfigReader()
10        self.connection_string = self.conf_reader.get_sqlite_string()
11        self.engine = create_engine(self.connection_string, echo=True)
12        self.Session = sessionmaker(bind=self.engine)
13        self.session = self.Session()
14
15    def create_tables(self):
16        Base.metadata.create_all(self.engine)
17
18    def close_session(self):
19        self.session.close()
20
```

1. Коннектор для SQLite

```
1 from ConfigParser import ConfigReader
2 import pyodbc as odbc
3
4
5 3 usages  ▴ Expband *
6 class AzureDatabaseConnector:
7     ▴ Expband
8     def __init__(self):
9         self.config = ConfigReader()
10        self.connection = odbc.connect(self.config.get_azure_string())
11        self.cursor = self.connection.cursor()
12
13    2 usages  ▴ Expband
14    def return_connector(self):
15        return self.cursor
16
17    ▴ Expband
18    def close_session(self):
19        self.cursor.close()
20
21 16
```

2. Коннектор для Azure

Ці класи відповідають за ініціацію підключення до бази даних. Для збереження даних я використовую модель користувача. Модель говорить, як повинен структуруватися об'єкт користувач.

```
1 from sqlalchemy import Column, String, Sequence
2 from sqlalchemy.ext.declarative import declarative_base
3
4 Base = declarative_base()
5
6
7 5 usages  ▴ Expband
8 class User(Base):
9     """User data model"""
10    __tablename__ = 'users'
11    id = Column(String(), Sequence("user_id_seq"), primary_key=True)
12    name = Column(String(50))
13    surname = Column(String(50))
14    login = Column(String(50))
15    password = Column(String(50))
16
17    ▴ Expband
18    def __init__(self, id, name, surname, login, password):
19        self.id = id
20        self.name = name
21        self.surname = surname
22        self.login = login
23        self.password = password
24
25    ▴ Expband
26    def __repr__(self):
27        return f"{self.id}/{self.name}/{self.surname}/{self.login}/{self.password}"
28
29
```

1. Модель користувача

Після цього виконуються саме операції з базами даних. Виконується дві операції запису та читання.

```
1 from DataAccessLayer.Models.User import User
2
3
4 class AzureRepository:
5     def __init__(self, cursor):
6         self.cursor = cursor
7         self.insert_query = f"""insert into users(ID, name, surname, login, password)
8             values(?, ?, ?, ?, ?)
9         """
10        self.get_query = f"""select * from users where login = ?"""
11
12    def azure_add_user(self, id, name, surname, login, password):
13        new_user = User(id=id, name=name, surname=surname, login=login, password=password)
14        self.cursor.execute(self.insert_query,
15                             new_user.id,
16                             new_user.name,
17                             new_user.surname,
18                             new_user.login,
19                             new_user.password)
20        self.cursor.commit()
21
22    def azure_get_user(self, login):
23        self.cursor.execute(self.get_query, login)
24        return self.cursor.fetchall()
25
```

1. Репозиторій для Azure. Тут можна побачити, яким саме чином виконуються запити. В полях класу записані запити, а в функціях підставляються значення

```
1 from DataAccessLayer.Models.User import User
2
3
4 class UserRepository:
5     def __init__(self, session):
6         self.session = session
7
8     def sqlite_add_user(self, id, name, surname, login, password):
9         new_user = User(id=id, name=name, surname=surname, login=login, password=password)
10        self.session.add(new_user)
11        self.session.commit()
12
13    def sqlite_get_user_by_login(self, login):
14        user = self.session.query(User).filter_by(login=login).first()
15        return user
16
```

2. З базою даних SQLite виконуються запити за допомогою SQLalchemy, тому не потрібно писати запити на SQL