

Database 1.2

Unique index; error handling and responses

Creating a unique index

- In your UserManager class, create a unique index for username
- Do this in init AND reset

```
class UserManager:
    # [...]
    def __init__(self, conn_str):
        # [...]
        self.col.create_index("username", unique=True)
        # [...] etc.
    # same in reset
```

Handling exceptions

- In `create_user`, use a try-except statement to handle `DuplicateKeyError`
- You'll have to add to your imports:
 - `from pymongo.errors import DuplicateKeyError`

```
data1 = {'username': 'joe', 'password': 'schmoe'}  
x = mycol.insert_one(data1)  
try:  
    x = mycol.insert_one(data1)  
except DuplicateKeyError:  
    print('username is already taken')
```

Unique index in UserManager

- Update your init and reset to create a unique index
- Update your create_user to handle duplicate keys
- If there is a duplicate key error, what will the return be?

```
class UserManager:  
    # ...  
    def create_user(self, user_dict:dict) -> str | None:  
        '''insert and return _id'''
```

Providing error information

- We should not just return None.
- Return error information.
- If we return an error message, how will the user (of the class) know whether the string is the id or the message?

Providing error information

- We can use a tuple, list, or dict.
- For example:

```
class UserManager:
    # [...]
    def create_user(self, user_dict:dict) -> dict:
        '''insert and return _id'''

        try:
            x = mycol.insert_one(data1)
            return {'status':1, 'inserted_id':x.inserted_id}
        except DuplicateKeyError:
            msg = 'username is already taken'
            return {'status':0, 'message':msg}
```

Updating your tests

- We need to update our tests to match

```
# [...]
class TestUserManager(unittest.TestCase):

    def test_duplicate(self):
        ''' reset, try to create same username twice'''
        # [...]
        # result of second create should be 0 or False
        self.assertFalse(result.get('status'))

        # # optional printing
        # if result.get('status'):
        #     print(result.get('inserted_id'))
        # else:
        #     print(result.get('message'))
```

More fail cases

- What are some other ways that `create_user` could fail?
- What are some ways that `update_user` could fail?

More fail cases

- `create_user` should fail if the wrong type of data is provided.
- we want to ensure:
 - correct field names
 - correct types
- The same is true with update data provided to `update_user`
 - additionally: specify changeable fields
- This would take a lot of logic to handle
 - is there a better way?