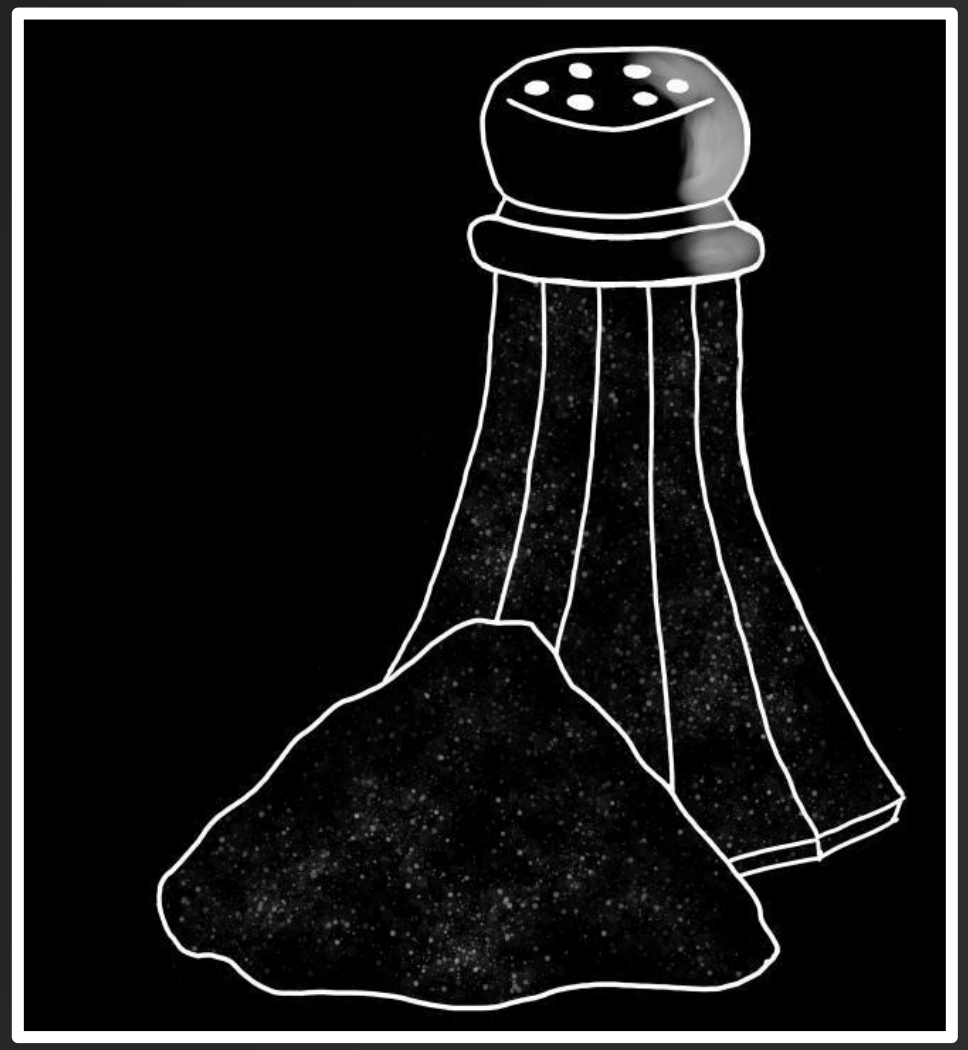
Software Assignment and License Tracking System (S.A.L.T.S.)



TEAM RED

Tyson Lucas, Noelle Stark, Frederick Bumgarner, Kara Lynch, Oluwasegun Durosinmi

Rocket® software

PRODUCT OVERVIEW & MISSION

NextGen
Rocket software

FLASK WEBSERVER

```python
@app.post("/addLicense/")
def addLicense():
    log.log("INFO", "Request to add license received.")
    license_request = request.json
    success, auth_response = authentication.authorize(request.headers)
    if success:
        # INTEGRATION: Call backend function
        # credentials = Credentials(auth_response)
        # request_info = AddLicReq(auth_response)
        # LicenseDatabase.addLicense(request_info, credentials)
        return f'<p>License added'
    else:
        abort(401)
```

AUTHENTICATION MODULE

```python
def authorize(self, headers) -> tuple[bool, str]:
    """
    Public method for calling authorization server.

    PARAS:
        headers:: headers from incoming HTTP request, should contain authorization token

    RETURN:
        bool:: True if successfully authenticated, False otherwise
        str:: The credentials associated with the token
    """
    log.log("INFO", "Beginning authorization process.")
    valid_token, token = self._validate_token(headers)
    if not valid_token:
        log.log("ERROR", "Invalid authorization token provided.")
        return False, None

    body = {"token": token}
    log.log("INFO", "Calling authentication server.")
    auth_response = requests.post(self._auth_url, json=body)
    if auth_response.status_code == 201:
        log.log("INFO", "Token authenticatd successfully.")
        return True, auth_response.text
    else:
        log.log("ERROR", "Token failed authorization.")
        return False, None
```

AUTHENTICATION MODULE

```python
def _validate_token(self, headers) -> tuple[bool, str]:
    """
    Internal method for sanity checking a provided token.

    Tests:
        1. Is token present (not NULL)?
        2. Is token too large?
        3. Is token too small?

    PARAS:
        headers:: headers stripped from HTTP request

    RETURN:
        bool:: if true, token was validated
        str::  contains the stripped authorization token
    """
    # Extract token
    token = headers.get("Bearer")
    if token is None:
        log.log("WARNING", "Request has no authorization token attached.")
        return False, None

    # TODO: ADD LENGTH CHECKING
    if len(token) < 250:
        log.log("WARNING", "Provided security token was too short.")
        return False, None
    elif len(token) > 400:
        log.log("WARNING", "Provided security token was too long.")
        return False, None

    return True, token
```

API TEST SCRIPT

```python
if __name__ == "__main__":
    # Test 1: Sending a valid token.
    test1()
    print("\n")

    # Test 2: Sending an expired token. Token is from 10/29
    test2()
    print("\n")

    # Test 3: Sending a fake token that is too long. (>400 chars)
    test3()
    print("\n")

    # Test 4: Sending a fake token that is too short. (<250 chars)
    test4()
    print("\n")

    # Test 5: Sending empty token.
    test5()
    print("\n")

    # Test 6: Sending no Bearer header along with request.
    test6()
    print("\n")
```

```python
# Test 3: Sending a fake token that is too long. (>400 chars)
def test3() -> bool:
    print("TEST 3: Sending a token that is too long.")
    long_token = ""
    for i in range(81):
        long_token += "AAAAA"
    print(f'Length of long token: {len(long_token)}')

    long_response = requests.post(API_URL, headers={"Bearer": long_token})
    if long_response.status_code == 401:
        print("TEST SUCCEEDED")
    else:
        print("TEST FAILED")
        print(long_response.text)
        print(long_response.status_code)
```

USER CREDENTIALS CLASS

```
@dataclass
class UserCredentials:
    """
    This class handles the credentials for a user, including their department and title.

    Methods:

    id(), first_name(), last_name(), location(), department(), title(): Will return the respective info field.

    name(): Will return the user's first and last name.

    is_manager(): Will return True if the user's title is "Manager", False otherwise.

    validate(): Checks each of the user info fields to ensure its validity. Will throw an exception if any are invalid.
    """
```

USER CREDENTIALS CLASS

```
36          _id: int
37          _first_name: str
38          _last_name: str
39          _location: str
40          _department: str
41          _title: str
```

```
91          def first_name(self):
92              return self._first_name
93
94          def last_name(self):
95              return self._last_name
96
97          def employee_id(self):
98              return self._id
```

USER CREDENTIALS CLASS

```
def validate(self):
    """Checks all of the information entered for a user for validity. Will return an error if any of the fields are too long,
       too short, or have invalid characters.
    """
    global config_dict
    if len(config_dict.keys()) == 0:
        config_dict = get_configs()

    _field_name = ""
    try:
        _field_name = "First name"
        check_data_type(self._first_name, str, "a string")
        self._first_name = self._first_name.strip()
        check_field_size(self._first_name, config_dict["FIRST_NAME_MAX_SIZE"], config_dict["FIRST_NAME_MIN_SIZE"])
        is_alpha_or_hyphen(self._first_name)

        _field_name = "Last name"
        check_data_type(self._last_name, str, "a string")
        self._last_name = self._last_name.strip()
        check_field_size(self._last_name, config_dict["LAST_NAME_MAX_SIZE"], config_dict["LAST_NAME_MIN_SIZE"])
        is_alpha_or_hyphen(self._last_name)

        _field_name = "Employee ID number"
```

USER CREDENTIALS CLASS

```
77                _field_name = "Job title"
78                check_data_type(self._title, str, "a string")
79                self._title = self._title.strip()
80                list_check(self._title, config_dict["TITLES_LIST"])
81
82
83            except Exception as e:
84                error_msg = _field_name + " " + e.args[0]
85                log.log("WARNING", f"Credentials validation failed for {self.name()}: {error_msg}")
86                raise Exception(error_msg)
```

```
[WARNING] - 2025-10-30 15:34:13,582 - license-tracker/src/credentials/credentials_manager/validate:85 - Credentials
 validation failed for Ava'); DROP TABLE Employee; -- Fishbie: First name must contain only letters, spaces, and up
 to one hyphen
```

USER CREDENTIALS CLASS - TESTING

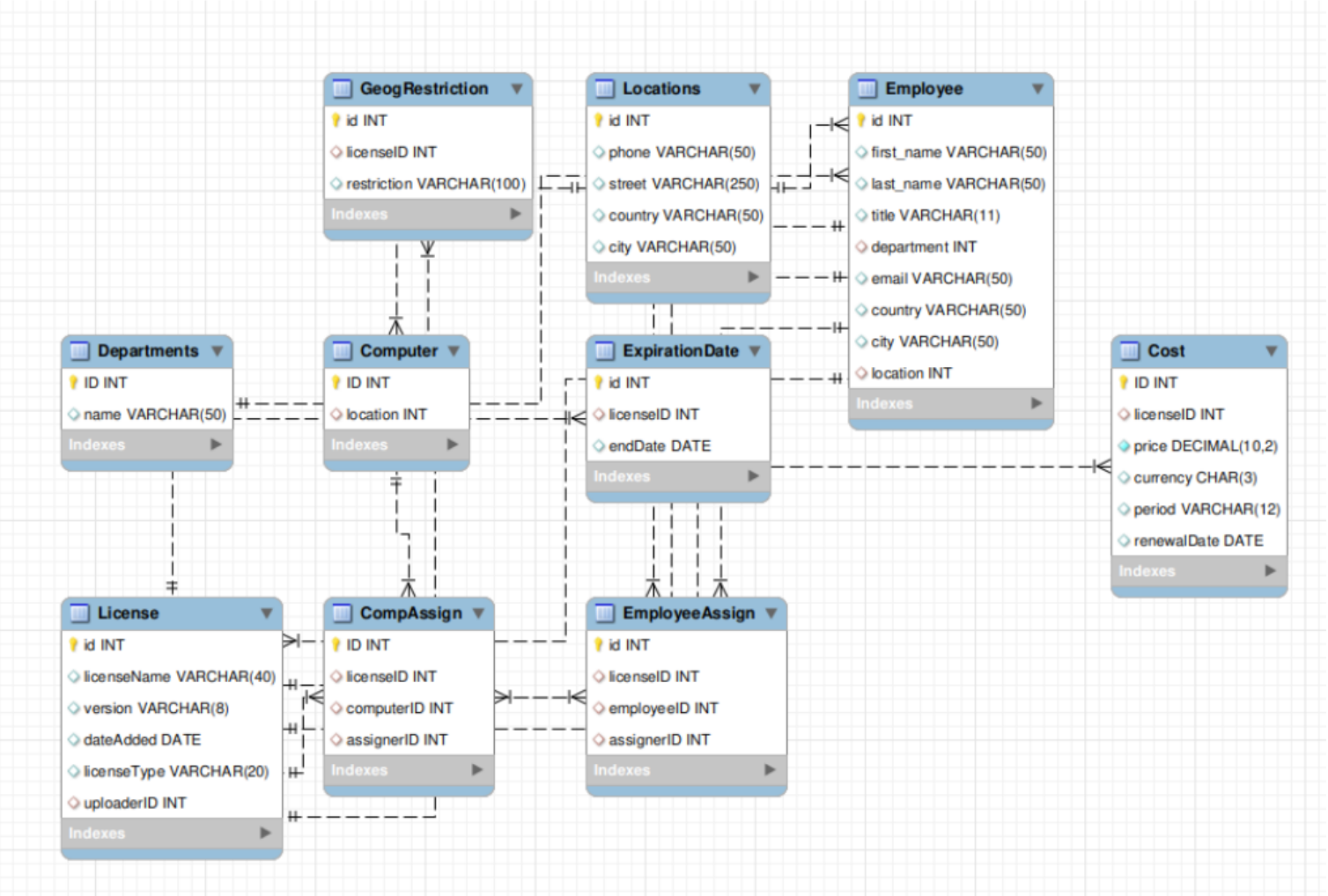
```
test_data = [
    (
        263,
        "Ava'); DROP TABLE Employee; --",
        "Fishbie",
        "Japan",
        "Legal",
        "Manager"
    ),
    (
```

USER CREDENTIALS CLASS - TESTING

```
users = []
for test in test_data:
    new_user = UserCredentials(test[0], test[1], test[2], test[3], test[4], test[5])

    try:
        new_user.validate()
        users.append(new_user)
        print(f"{new_user.name()} added successfully")
    except Exception as e:
        pass
print("List of users:")
for user in users:
    print(user.name())
```

ER DIAGRAM


GeogRestriction
id INT
licenseID INT
restriction VARCHAR(100)
Indexes
Locations
id INT
phone VARCHAR(50)
street VARCHAR(250)
country VARCHAR(50)
city VARCHAR(50)
Indexes
Employee
id INT
first_name VARCHAR(50)
last_name VARCHAR(50)
title VARCHAR(11)
department INT
email VARCHAR(50)
country VARCHAR(50)
city VARCHAR(50)
location INT
Indexes
Departments
ID INT
name VARCHAR(50)
Indexes
Computer
ID INT
location INT
Indexes
ExpirationDate
id INT
licenseID INT
endDate DATE
Indexes
Cost
ID INT
licenseID INT
price DECIMAL(10,2)
currency CHAR(3)
period VARCHAR(12)
renewalDate DATE
Indexes
License
id INT
licenseName VARCHAR(40)
version VARCHAR(8)
dateAdded DATE
licenseType VARCHAR(20)
uploaderID INT
Indexes
CompAssign
ID INT
licenseID INT
computerID INT
assignerID INT
Indexes
EmployeeAssign
id INT
licenseID INT
employeeID INT
assignerID INT
Indexes

DATABASE DEMONSTRATION

Q & A

NextGen
Rocket software