Histórico do Schedule Management

Notebook a ponto de registrar o histórico de alterações do Schedule Management. Não é necessário executar este notebook.

Get Instance

Como o Schedule Management será implementado como um Singleton, é necessário utilizar o método `get_instance` para obter a instância do objeto. Para isso, desenvolvemos dois testes, um para verificar a instância e outro para verificar se a instância é única:

In [ ]:
```
class TestScheduleManagement(unittest.TestCase):
    def setUp(self):
        # Reset the singleton instance before each test
        ScheduleManagement._instance = None
        self.db_module = Mock()

    def test_get_instance_creates_instance(self):
        instance = ScheduleManagement.get_instance(self.db_module)
        self.assertIsInstance(instance, ScheduleManagement)

    def test_get_instance_returns_same_instance(self):
        instance1 = ScheduleManagement.get_instance(self.db_module)
        instance2 = ScheduleManagement.get_instance(self.db_module)
        self.assertIs(instance1, instance2)
```

Para isso, o código abaixo foi desenvolvido:

In [ ]:
```
class ScheduleManagement:
    """
    ScheduleManagement class
    Responsible for managing the schedules in the database

    Attributes:
        db: Database module
        schedules: Dictionary of schedules, where the key is the schedule ID
            and the value is the schedule instance
    """

    _instance = None

    @classmethod
    def get_instance(cls, database_module: MongoModule, schedules: dict = None):
        if cls._instance is None:
            cls._instance = cls(database_module, schedules)
        return cls._instance
```

O código foi suficiente para verificar se a instância é única e se a instância é a mesma.

Schedule Exists

Para verificar se um schedule existe, foi desenvolvido os seguintes testes:

In [ ]:
```
def test_schedule_exists(self):
    self.db_module.select_data = MagicMock(return_value=[{'_id': 'schedule1',\
                                                          'title': 'Schedule 1',\
                                                          'description': 'This is schedule 1',\
                                                          'permissions': {'user1': 'read', 'user2': 'write'},
                                                          'elements': ['element1', 'element2']
                                                          }])
    result = self.schedule_management.schedule_exists('schedule1')
    self.assertTrue(result)
    self.db_module.select_data.assert_called_with('schedules', {'_id': 'schedule1'})

def test_schedule_does_not_exist(self):
    self.db_module.select_data = MagicMock(return_value=[])
    result = self.schedule_management.schedule_exists('schedule1')
    self.assertFalse(result)
    self.db_module.select_data.assert_called_with('schedules', {'_id': 'schedule1'})
```

O código não foi suficiente para verificar se o schedule existe, pois o método `schedule_exists` não foi implementado. O código abaixo foi desenvolvido para implementar o método:

In [ ]:
```
def schedule_exists(self, schedule_id: str) -> bool:
    """
    Check if a schedule exists
    Args:
        schedule_id: Schedule ID

    Returns:
        True if the schedule exists, False otherwise
    """

    # Get the schedules from the database that match the given ID
    schedule = self.db_module.select_data('schedules', {'_id': schedule_id})

    # If the list is not empty, the schedule exists
    return bool(schedule)
```

O código foi suficiente.

Create Schedule

Para criar um schedule, foi desenvolvido um teste com três subtestes. O primeiro subteste verifica se a chamada de inserção dos dados no banco é chamada, o segundo subteste verifica se o retorno da função é o esperado e o terceiro subteste verifica se o schedule possui os atributos esperados:

In [ ]:
```
def test_create_schedule(self):
    # Arrange
    self.db_module.insert_data = MagicMock()
    schedule_id = "schedule1"
    title = "Schedule 1"
    description = "This is schedule 1"
    permissions = {"user1": "read", "user2": "write"}
    elements = ["element1", "element2"]
    self._test_create_schedule_insert_data(schedule_id, title, description, permissions, elements)
    self._test_create_schedule_return(schedule_id, title, description, permissions, elements)
    self._test_create_schedule_attributes(schedule_id, title, description, permissions, elements)

def _test_create_schedule_insert_data(self, schedule_id, title, description, permissions, elements):
    # Act
    result = self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
    # Assert
    self.db_module.insert_data.assert_called_with('schedules',
                                                  {'_id': schedule_id,
                                                  'title': title,
                                                  'description': description,
                                                  'permissions': permissions,
                                                      'elements': elements})

def _test_create_schedule_return(self, schedule_id, title, description, permissions, elements):
    # Act
    result = self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
    # Assert
    self.assertIsInstance(result, Schedule)

def _test_create_schedule_attributes(self, schedule_id, title, description, permissions, elements):
    # Act
    result = self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
    # Assert
    self.assertEqual(result.id, schedule_id)
    self.assertEqual(result.title, title)
    self.assertEqual(result.description, description)
    self.assertEqual(result.permissions, permissions)
    self.assertEqual(result.elements, elements)
```

Como o método `create_schedule` não foi implementado, o teste retorna erro. O código abaixo foi desenvolvido para implementar o método:

In [ ]:
```
def create_schedule(self, schedule_id: str, title: str, description: str,
        permissions: dict, elements: list) -> Schedule:
    """
    Create a new schedule

    Args:
        schedule_id: Schedule ID
        title: Title of the schedule
        description: Description of the schedule
        permissions: Dictionary of permissions, where the key is the user
        elements: List of elements IDs that are displayed in the schedule

    Returns:
        The created schedule instance
    """
```

```python
    schedule = Schedule(schedule_id, title, description, permissions, elements)
    self.db_module.insert_data('schedules', {'_id': schedule_id,
                                            'title': title,
                                            'description': description,
                                            'permissions': permissions,
                                            'elements': elements})
    return schedule
```

O código foi suficiente. Criamos dois subtestes adicionais: um para verificar se o schedule foi adicionado ao dicionário schedules do Schedule Management e outro para verificar se o erro DuplicatedIDError é lançado quando o schedule já existe:

In [ ]:
```python
def _test_create_schedule_adds_to_self_schedules(self, schedule_id):
    # Assert
    self.assertIn(schedule_id, self.schedule_management.schedules)
    self.assertIsInstance(self.schedule_management.schedules[schedule_id], Schedule)

def _test_create_schedule_raises_error_if_schedule_exists(self, schedule_id, title, description, permissions, e
    # Arrange
    self.schedule_management.schedule_exists = MagicMock(return_value=True)
    # Act & Assert
    with self.assertRaises(DuplicatedIDError):
        self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
```

O código não foi suficiente. O código foi adaptado, além de ser criado o erro DuplicatedIDError:

In [ ]:
```python
class DuplicatedIDError(Exception):
    """Raised when the ID already exists"""
    pass

def create_schedule(self, schedule_id: str, title: str, description: str,
        permissions: dict, elements: list) -> Schedule:
    """
    Create a new schedule

    Args:
        schedule_id: Schedule ID
        title: Title of the schedule
        description: Description of the schedule
        permissions: Dictionary of permissions, where the key is the user
        elements: List of elements IDs that are displayed in the schedule

    Returns:
        The created schedule instance
    """
    if self.schedule_exists(schedule_id):
        raise DuplicatedIDError(f"A schedule with ID {schedule_id} already exists")

    schedule = Schedule(schedule_id, title, description, permissions, elements)
    self.db_module.insert_data('schedules', {'_id': schedule_id,
                                            'title': title,
                                            'description': description,
                                            'permissions': permissions,
                                            'elements': elements})
    self.schedules[schedule_id] = schedule
    return schedule
```

O código foi suficiente. Em seguida, foi desenvolvido dois testes para verificar se erros das funções `set_title` e `set_description` do schedule são propagados para a função `create_schedule`:

In [ ]:
```python
def test_create_schedule_raises_error_with_invalid_title(self):
    # Arrange
    self.schedule_management.db_module.insert_data = MagicMock()
    self.schedule_management.db_module.select_data = MagicMock(return_value=[])
    invalid_titles = [None, 123, "", "    ", "a" * 51]  # Covers all restrictions
    schedule_id = "schedule10"
    description = "This is schedule 2"
    permissions = {"user1": "write", "user2": "read"}
    elements = ["element2", "element3"]
    # Act & Assert
    for title in invalid_titles:
        with self.assertRaises((ValueError, TypeError)):
            self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)

def test_create_schedule_raises_error_with_invalid_description(self):
    # Arrange
    self.schedule_management.db_module.insert_data = MagicMock()
    self.schedule_management.db_module.select_data = MagicMock(return_value=[])
    invalid_descriptions = [123, "a" * 501]  # Covers all restrictions
    schedule_id = "schedule10"
    title = "Schedule 2"
    permissions = {"user1": "write", "user2": "read"}
    elements = ["element2", "element3"]
    # Act & Assert
    for description in invalid_descriptions:
        with self.assertRaises((ValueError, TypeError)):
            self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
```

O código foi suficiente. Também foram desenvolvidos dois testes para impedir que o id não seja uma string e impedir que permissions esteja vazio:

```
In [ ]: def test_create_schedule_raises_error_with_non_string_id(self):
            # Arrange
            self.schedule_management.db_module.insert_data = MagicMock()
            self.schedule_management.db_module.select_data = MagicMock(return_value=[])
            schedule_id = 123  # Non-string ID
            title = "Schedule 2"
            description = "This is schedule 2"
            permissions = {"user1": "write", "user2": "read"}
            elements = ["element2", "element3"]
            # Act & Assert
            with self.assertRaises(TypeError):
                self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)

        def test_create_schedule_raises_error_with_empty_permissions(self):
            # Arrange
            self.schedule_management.db_module.insert_data = MagicMock()
            self.schedule_management.db_module.select_data = MagicMock(return_value=[])
            schedule_id = "schedule10"
            title = "Schedule 2"
            description = "This is schedule 2"
            permissions = {}  # Empty permissions
            elements = ["element2", "element3"]
            # Act & Assert
            with self.assertRaises(EmptyPermissionsError):
                self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
```

O código não foi suficiente. O código foi adaptado, e o erro EmptyPermissionsError foi utilizado:

```
In [ ]: class EmptyPermissionsError(Exception):
            """Raised when the permissions list is empty"""
            pass

        def create_schedule(self, schedule_id: str, title: str, description: str,
                permissions: dict, elements: list) -> Schedule:
            """
            Create a new schedule

            Args:
                schedule_id: Schedule ID
                title: Title of the schedule
                description: Description of the schedule
                permissions: Dictionary of permissions, where the key is the user
                elements: List of elements IDs that are displayed in the schedule

            Returns:
                The created schedule instance
            """
            # Possible errors:
            if self.schedule_exists(schedule_id):
                raise DuplicatedIDError(f"A schedule with ID {schedule_id} already exists")
            if not isinstance(schedule_id, str):
                raise TypeError("Schedule ID must be a string")
            if not permissions:
                raise EmptyPermissionsError("Permissions cannot be empty")
            # Create the schedule instance and insert it into the database
            schedule = Schedule(schedule_id, title, description, permissions, elements)
            self.db_module.insert_data('schedules', {'_id': schedule_id,
                                                      'title': title,
                                                      'description': description,
                                                      'permissions': permissions,
                                                      'elements': elements})
            # Add the schedule to the dictionary
            self.schedules[schedule_id] = schedule
            return schedule
```

O código foi suficiente. Criamos dois testes adicionais: um para verificar que cada element de elements é atualizado no banco e outro para levantar erro NonExistentIDError quando um elemento não existe:

```
In [ ]: def test_create_schedule_updates_elements(self):
            # Arrange
            schedule_id = "schedule1"
            title = "Test Title"
```

```
    description = "Test Description"
    permissions = {"user1": {}}
    elements = ["element1", "element2", "element3"]
    with patch.object(self.schedule_management, 'schedule_exists', return_value=False), \
        patch.object(ElementManagement, 'update_element', return_value=None) as mock_update_element:
        # Act
        self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
        # Assert
        assert mock_update_element.call_count == len(elements)
        for element_id in elements:
            mock_update_element.assert_any_call(element_id)

def test_create_schedule_raises_error_for_nonexistent_element(self):
    # Arrange
    schedule_id = "schedule1"
    title = "Test Title"
    description = "Test Description"
    permissions = {"user1": {}}
    elements = ["element1", "nonexistent_element"]
    with patch.object(self.schedule_management, 'schedule_exists', return_value=False), \
        patch.object(ElementManagement, 'element_exists', side_effect=[True, False]):
        # Act & Assert
        with self.assertRaises(NonExistentIDError):
            self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
```

O código não foi suficiente. Ele foi adaptado:

In [ ]:
```
def create_schedule(self, schedule_id: str, title: str, description: str,
                    permissions: dict, elements: list) -> Schedule:
    """
    Create a new schedule

    Args:
        schedule_id: Schedule ID
        title: Title of the schedule
        description: Description of the schedule
        permissions: Dictionary of permissions, where the key is the user
        elements: List of elements IDs that are displayed in the schedule

    Returns:
        The created schedule instance
    """
    # Possible errors:
    if self.schedule_exists(schedule_id):
        raise DuplicatedIDError(f"A schedule with ID {schedule_id} already exists")
    if not isinstance(schedule_id, str):
        raise TypeError("Schedule ID must be a string")
    if not permissions:
        raise EmptyPermissionsError("Permissions cannot be empty")

    # Check if each element exists
    element_manager = ElementManagement.get_instance()
    for element_id in elements:
        if not element_manager.element_exists(element_id):
            raise NonExistentIDError(f"No element found with ID {element_id}")

    # Create the schedule instance and insert it into the database
    schedule = Schedule(schedule_id, title, description, permissions, elements)
    self.db_module.insert_data('schedules', {'_id': schedule_id,
                                            'title': title,
                                            'description': description,
                                            'permissions': permissions,
                                            'elements': elements})
    # Add the schedule to the dictionary
    self.schedules[schedule_id] = schedule

    # Update each element
    for element_id in elements:
        element_manager.update_element(element_id)

    return schedule
```

O código foi suficiente. Um novo teste foi desenvolvido para verificar se os atributos schedules dos elementos são atualizados. Foi necessário adaptar outros dois testes:

In [ ]:
```
def test_create_schedule_updates_elements_schedules(self):
    # Arrange
    schedule_id = "schedule1"
    title = "Test Title"
    description = "Test Description"
    permissions = {"user1": {}}
    elements = ["element1", "element2", "element3"]
    mock_element = MagicMock()
    mock_element.schedules = []
    with patch.object(self.schedule_management, 'schedule_exists', return_value=False), \
         patch.object(ElementManagement, 'element_exists', return_value=True), \
         patch.object(ElementManagement, 'get_element', return_value=mock_element) as mock_get_element, \
         patch.object(ElementManagement, 'update_element', return_value=None) as mock_update_element:
        # Act
        self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
        # Assert
        assert mock_get_element.call_count == len(elements)
        assert mock_update_element.call_count == len(elements)
        for element_id in elements:
            mock_get_element.assert_any_call(element_id)
            mock_update_element.assert_any_call(element_id)

def test_create_schedule(self):
    # General test for create_schedule
    # Arrange
    self.db_module.insert_data = MagicMock()
    self.db_module.select_data = MagicMock(return_value=[])
    schedule_id = "schedule10"
    title = "Schedule 2"
    description = "This is schedule 2"
    permissions = {"user1": "write", "user2": "read"}
    elements = ["element2", "element3"]
    mock_element = MagicMock()
    with patch.object(ElementManagement, 'get_element', return_value=mock_element), \
    patch.object(ElementManagement, 'update_element', return_value=None):
        # Act
        result = self.schedule_management.create_schedule(schedule_id, title, description, permissions, element
        with self.subTest("Test insert_data is called with correct arguments"):
            self._test_create_schedule_insert_data(schedule_id, title, description, permissions, elements)
        with self.subTest("Test create_schedule returns a Schedule instance"):
            self._test_create_schedule_return(result)
        with self.subTest("Test Schedule instance has correct attributes"):
            self._test_create_schedule_attributes(result, schedule_id, title, description, permissions, element
        with self.subTest("Test create_schedule adds to self.schedules"):
            self._test_create_schedule_adds_to_self_schedules(schedule_id)
        with self.subTest("Test create_schedule raises error if schedule exists"):
            self._test_create_schedule_raises_error_if_schedule_exists(schedule_id, title, description, permiss

def test_create_schedule_updates_elements_schedules(self):
    # Arrange
    schedule_id = "schedule1"
    title = "Test Title"
    description = "Test Description"
    permissions = {"user1": {}}
    elements = ["element1", "element2", "element3"]
    mock_element = MagicMock()
    mock_element.schedules = []
    with patch.object(self.schedule_management, 'schedule_exists', return_value=False), \
        patch.object(ElementManagement, 'element_exists', return_value=True), \
        patch.object(ElementManagement, 'get_element', return_value=mock_element) as mock_get_element, \
        patch.object(ElementManagement, 'update_element', return_value=None) as mock_update_element:
        # Act
        self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
        # Assert
        assert mock_get_element.call_count == len(elements)
        assert mock_update_element.call_count == len(elements)
        for element_id in elements:
            mock_get_element.assert_any_call(element_id)
            mock_update_element.assert_any_call(element_id)
```

O código teve de ser adaptado:

In [ ]:
```
def create_schedule(self, schedule_id: str, title: str, description: str,
                    permissions: dict, elements: list) -> Schedule:
    """
    Create a new schedule

    Args:
        schedule_id: Schedule ID
        title: Title of the schedule
        description: Description of the schedule
        permissions: Dictionary of permissions, where the key is the user
        elements: List of elements IDs that are displayed in the schedule

    Returns:
        The created schedule instance
    """
    # Possible errors:
    if self.schedule_exists(schedule_id):
        raise DuplicatedIDError(f"A schedule with ID {schedule_id} already exists")
```

```python
    if not isinstance(schedule_id, str):
        raise TypeError("Schedule ID must be a string")
    if not permissions:
        raise EmptyPermissionsError("Permissions cannot be empty")

    # Check if each element exists
    element_manager = ElementManagement.get_instance()
    for element_id in elements:
        if not element_manager.element_exists(element_id):
            raise NonExistentIDError(f"No element found with ID {element_id}")

    # Create the schedule instance and insert it into the database
    schedule = Schedule(schedule_id, title, description, permissions, elements)
    self.db_module.insert_data('schedules', {'_id': schedule_id,
                                             'title': title,
                                             'description': description,
                                             'permissions': permissions,
                                             'elements': elements})
    # Add the schedule to the dictionary
    self.schedules[schedule_id] = schedule

    # Update each element and add the schedule to its schedules attribute
    for element_id in elements:
        element = element_manager.get_element(element_id)
        element.schedules.append(schedule)
        element_manager.update_element(element_id)

    return schedule
```

O código foi suficiente. Analogamente, foram desenvolvidos os mesmos três testes para atualização no banco, verificação e atualização dos atributos dos users:

In [ ]:
```python
def test_create_schedule_updates_users(self):
    # Arrange
    schedule_id = "schedule1"
    title = "Test Title"
    description = "Test Description"
    permissions = {"user1": {}, "user2": {}, "user3": {}}
    elements = ["element1", "element2", "element3"]
    mock_user = MagicMock()
    mock_element = MagicMock()
    with patch.object(self.schedule_management, 'schedule_exists', return_value=False), \
        patch.object(UserManagement, 'get_user', return_value=mock_user), \
        patch.object(UserManagement, 'update_user', return_value=None) as mock_update_user,\
        patch.object(ElementManagement, 'get_element', return_value=mock_element), \
        patch.object(ElementManagement, 'update_element', return_value=None) as mock_update_element:
        # Act
        self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
        # Assert
        assert mock_update_user.call_count == len(permissions)
        for user_id in permissions:
            mock_update_user.assert_any_call(user_id)

def test_create_schedule_raises_error_for_nonexistent_user(self):
    # Arrange
    schedule_id = "schedule1"
    title = "Test Title"
    description = "Test Description"
    permissions = {"user1": {}, "nonexistent_user": {}}
    elements = ["element1", "element2"]
    with patch.object(self.schedule_management, 'schedule_exists', return_value=False), \
         patch.object(UserManagement, 'user_exists', side_effect=[True, False]):
        # Act & Assert
        with self.assertRaises(NonExistentIDError):
            self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)

def test_create_schedule_updates_users_schedules(self):
    # Arrange
    schedule_id = "schedule1"
    title = "Test Title"
    description = "Test Description"
    permissions = {"user1": {}, "user2": {}, "user3": {}}
    elements = ["element1", "element2", "element3"]
    mock_user = MagicMock()
    mock_user.schedules = []
    mock_element = MagicMock()
    mock_element.schedules = []
    with patch.object(self.schedule_management, 'schedule_exists', return_value=False), \
        patch.object(UserManagement, 'user_exists', return_value=True), \
        patch.object(UserManagement, 'get_user', return_value=mock_user) as mock_get_user, \
        patch.object(UserManagement, 'update_user', return_value=None) as mock_update_user, \
        patch.object(ElementManagement, 'get_element', return_value=mock_element) as mock_get_element, \
        patch.object(ElementManagement, 'update_element', return_value=None) as mock_update_element:
```

```
        # Act
        self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
        # Assert
        assert mock_get_user.call_count == len(permissions)
        assert mock_update_user.call_count == len(permissions)
        for user_id in permissions:
            mock_get_user.assert_any_call(user_id)
            mock_update_user.assert_any_call(user_id)
```

Três outros testes também tiveram de ser adaptados:

In [ ]:
```
def test_create_schedule_updates_elements_schedules(self):
    # Arrange
    schedule_id = "schedule1"
    title = "Test Title"
    description = "Test Description"
    permissions = {"user1": {}}
    elements = ["element1", "element2", "element3"]
    mock_user = MagicMock()
    mock_user.schedules = []
    mock_element = MagicMock()
    mock_element.schedules = []
    with patch.object(self.schedule_management, 'schedule_exists', return_value=False), \
        patch.object(UserManagement, 'user_exists', return_value=True), \
        patch.object(UserManagement, 'get_user', return_value=mock_user) as mock_get_user, \
        patch.object(UserManagement, 'update_user', return_value=None) as mock_update_user, \
        patch.object(ElementManagement, 'get_element', return_value=mock_element) as mock_get_element, \
        patch.object(ElementManagement, 'update_element', return_value=None) as mock_update_element:
        self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
        # Assert
        assert mock_get_element.call_count == len(elements)
        assert mock_update_element.call_count == len(elements)
        for element_id in elements:
            mock_get_element.assert_any_call(element_id)
            mock_update_element.assert_any_call(element_id)

def test_create_schedule_updates_elements(self):
    # Arrange
    schedule_id = "schedule1"
    title = "Test Title"
    description = "Test Description"
    permissions = {"user1": {}}
    elements = ["element1", "element2", "element3"]
    mock_user = MagicMock()
    mock_element = MagicMock()
    with patch.object(self.schedule_management, 'schedule_exists', return_value=False), \
        patch.object(UserManagement, 'get_user', return_value=mock_user), \
        patch.object(UserManagement, 'update_user', return_value=None) as mock_update_user,\
        patch.object(ElementManagement, 'get_element', return_value=mock_element), \
        patch.object(ElementManagement, 'update_element', return_value=None) as mock_update_element:
        # Act
        self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)
        # Assert
        assert mock_update_element.call_count == len(elements)
        for element_id in elements:
            mock_update_element.assert_any_call(element_id)

def test_create_schedule(self):
    # General test for create_schedule
    # Arrange
    self.db_module.insert_data = MagicMock()
    self.db_module.select_data = MagicMock(return_value=[])
    schedule_id = "schedule10"
    title = "Schedule 2"
    description = "This is schedule 2"
    permissions = {"user1": "write", "user2": "read"}
    elements = ["element2", "element3"]
    mock_user = MagicMock()
    mock_element = MagicMock()
    with patch.object(UserManagement, 'get_user', return_value=mock_user), \
    patch.object(UserManagement, 'update_user', return_value=None), \
    patch.object(ElementManagement, 'get_element', return_value=mock_element), \
    patch.object(ElementManagement, 'update_element', return_value=None):
        # Act
        result = self.schedule_management.create_schedule(schedule_id, title, description, permissions, element
        with self.subTest("Test insert_data is called with correct arguments"):
            self._test_create_schedule_insert_data(schedule_id, title, description, permissions, elements)
        with self.subTest("Test create_schedule returns a Schedule instance"):
            self._test_create_schedule_return(result)
        with self.subTest("Test Schedule instance has correct attributes"):
            self._test_create_schedule_attributes(result, schedule_id, title, description, permissions, element
        with self.subTest("Test create_schedule adds to self.schedules"):
            self._test_create_schedule_adds_to_self_schedules(schedule_id)
        with self.subTest("Test create_schedule raises error if schedule exists"):
            self._test_create_schedule_raises_error_if_schedule_exists(schedule_id, title, description, permiss
```

O código foi adaptado para ateender os testes:

In [ ]:
```
def create_schedule(self, schedule_id: str, title: str, description: str,
                    permissions: dict, elements: list) -> Schedule:
    """
    Create a new schedule

    Args:
        schedule_id: Schedule ID
        title: Title of the schedule
        description: Description of the schedule
        permissions: Dictionary of permissions, where the key is the user
        elements: List of elements IDs that are displayed in the schedule

    Returns:
        The created schedule instance
    """
    # Possible errors:
    if self.schedule_exists(schedule_id):
        raise DuplicatedIDError(f"A schedule with ID {schedule_id} already exists")
    if not isinstance(schedule_id, str):
        raise TypeError("Schedule ID must be a string")
    if not permissions:
        raise EmptyPermissionsError("Permissions cannot be empty")

    # Check if each element exists
    element_manager = ElementManagement.get_instance()
    for element_id in elements:
        if not element_manager.element_exists(element_id):
            raise NonExistentIDError(f"No element found with ID {element_id}")

    # Check if each user exists
    user_manager = UserManagement.get_instance()
    for user_id in permissions.keys():
        if not user_manager.user_exists(user_id):
            raise NonExistentIDError(f"No user found with ID {user_id}")

    # Create the schedule instance and insert it into the database
    schedule = Schedule(schedule_id, title, description, permissions, elements)
    self.db_module.insert_data('schedules', {'_id': schedule_id,
                                             'title': title,
                                             'description': description,
                                             'permissions': permissions,
                                             'elements': elements})
    # Add the schedule to the dictionary
    self.schedules[schedule_id] = schedule

    # Update each element and add the schedule to its schedules attribute
    for element_id in elements:
        element = element_manager.get_element(element_id)
        element.schedules.append(schedule)
        element_manager.update_element(element_id)

    # Update each user and add the schedule to its schedules attribute
    for user_id in permissions.keys():
        user = user_manager.get_user(user_id)
        user.schedules.append(schedule)
        user_manager.update_user(user_id)

    return schedule
```

O código foi suficiente.

Get Schedule

Para obter um schedule, foi desenvolvido um teste inicial para um caso onde o schedule existe e está no dicionário schedules do Schedule Management:

In [ ]:
```
def test_get_schedule_id_exists_on_dict(self):
    # Arrange
    schedule_id = "schedule10"
    title = "Schedule 2"
    description = "This is schedule 2"
    permissions = {"user1": "write", "user2": "read"}
    elements = ["element2", "element3"]
    schedule = Schedule(schedule_id, title, description, permissions, elements)
    self.schedule_management.schedules[schedule_id] = schedule
    # Act
    result = self.schedule_management.get_schedule(schedule_id)
    # Assert
    self.assertEqual(result, schedule)
```

O código inicial foi o seguinte:

```
In [ ]:
def get_schedule(self, schedule_id: str) -> Schedule:
    """
    Get a schedule by its ID

    Args:
        schedule_id: Schedule ID

    Returns:
        The schedule instance
    """

    if schedule_id in self.schedules:
        return self.schedules[schedule_id]
```

O código foi suficiente. Foram criados mais dois testes: um para que seja retornado um schedule quando ele não está no dicionário schedules do Schedule Management porém está no banco de dados e outro para que seja retornado um erro quando o schedule não está no dicionário schedules do Schedule Management e nem no banco de dados:

```
In [ ]:
def test_get_schedule_id_exists_in_database(self):
    # Arrange
    schedule_id = "schedule10"
    title = "Schedule 2"
    description = "This is schedule 2"
    permissions = {"user1": "write", "user2": "read"}
    elements = ["element2", "element3"]
    self.schedule_management.db_module.select_data = MagicMock(return_value={
        '_id': schedule_id,
        'title': title,
        'description': description,
        'permissions': permissions,
        'elements': elements
    })
    self.schedule_management.schedule_exists = MagicMock(return_value=True)
    # Act
    result = self.schedule_management.get_schedule(schedule_id)
    # Assert
    self.assertEqual(result.title, title)
    self.assertEqual(result.description, description)
    self.assertEqual(result.permissions, permissions)
    self.assertEqual(result.elements, elements)

def test_get_schedule_id_doesnt_exist(self):
    # Arrange
    schedule_id = "schedule10"
    self.schedule_management.schedule_exists = MagicMock(return_value=False)
    # Act & Assert
    with self.assertRaises(NonExistentIDError):
        self.schedule_management.get_schedule(schedule_id)
```

Para isso, foi necessário criar o erro NonExistentIDError:

```
In [ ]:
class NonExistentIDError(Exception):
    """Raised when the ID does not exist"""
    pass
```

O código não fui suficiente. O código foi adaptado:

```
In [ ]:
def get_schedule(self, schedule_id: str) -> Schedule:
    """
    Get a schedule by its ID

    Args:
        schedule_id: Schedule ID

    Returns:
        The schedule instance
    """
    if schedule_id in self.schedules:
        return self.schedules[schedule_id]
    elif self.schedule_exists(schedule_id):
        schedule_data = self.db_module.select_data('schedules', {'_id': schedule_id})
        schedule = Schedule(schedule_id,
                            schedule_data['title'],
                            schedule_data['description'],
                            schedule_data['permissions'],
                            schedule_data['elements'])
        self.schedules[schedule_id] = schedule
        return schedule
    else:
        raise NonExistentIDError(f"No schedule found with ID {schedule_id}")
```

O código foi suficiente.

Update Schedule

Para atualizar um schedule, foi desenvolvido um teste inicial para um caso onde o schedule existe e está no dicionário schedules do Schedule Management:

In [ ]:
```
def test_update_schedule_id_exists(self):
    # Arrange
    schedule_id = "schedule10"
    title = "Schedule 2"
    description = "This is schedule 2"
    permissions = {"user1": "write", "user2": "read"}
    elements = ["element2", "element3"]
    schedule = Schedule(schedule_id, title, description, permissions, elements)
    self.schedule_management.schedules[schedule_id] = schedule
    self.schedule_management.db_module.update_data = MagicMock()
    # Act
    self.schedule_management.update_schedule(schedule_id)
    # Assert
    self.schedule_management.db_module.update_data.assert_called_once_with(
        'schedules',
        {'_id': schedule_id},
        schedule.to_dict()
    )
```

Como o método `update_schedule` não foi implementado, o teste retorna erro. O código abaixo foi desenvolvido para implementar o método:

In [ ]:
```
def update_schedule(self, schedule_id: str) -> None:
    """
    Updates a schedule in the database

    Args:
        schedule_id: Schedule ID
    """
    schedule = self.schedules[schedule_id]
    new_data = schedule.to_dict()
    self.db_module.update_data('schedules', {'_id': schedule_id}, new_data)
```

O código foi suficiente. Foi criado mais um teste para verificar se o erro NonExistentIDError é lançado quando o schedule não está no database. Não é necessário dois casos como no método anterior, pois não faz sentido rodar o método `update_schedule` se o schedule não está no dictionary schedules do Schedule Management:

In [ ]:
```
def test_update_schedule_id_doesnt_exist(self):
    # Arrange
    schedule_id = "schedule10"
    self.schedule_management.schedule_exists = MagicMock(return_value=False)
    # Act & Assert
    with self.assertRaises(NonExistentIDError):
        self.schedule_management.update_schedule(schedule_id)
```

O código não foi suficiente. O código foi adaptado:

In [ ]:
```
def test_update_schedule_id_doesnt_exist(self):
    # Arrange
    schedule_id = "schedule10"
    self.schedule_management.schedule_exists = MagicMock(return_value=False)
    # Act & Assert
    with self.assertRaises(NonExistentIDError):
        self.schedule_management.update_schedule(schedule_id)
```

O código foi suficiente.

Delete Schedule

Para deletar um schedule, foi desenvolvido um teste inicial para verificar se o delete é chamado para o banco:

In [ ]:
```python
def test_delete_schedule_deletes_from_database(self):
    # Check that delete_schedule deletes the schedule from the database
    # Arrange
    schedule_id = "schedule10"
    self.schedule_management.db_module.delete_data = MagicMock()
    # Act
    self.schedule_management.delete_schedule(schedule_id)
    # Assert
    self.schedule_management.db_module.delete_data.assert_called_once_with(
        'schedules',
        {'_id': schedule_id}
    )
```

Como o método `delete_schedule` não foi implementado, o teste retorna erro. O código abaixo foi desenvolvido para implementar o método:

In [ ]:
```python
def delete_schedule(self, schedule_id: str) -> None:
    """
    Deletes a schedule from the database

    Args:
        schedule_id: Schedule ID

    """

    self.db_module.delete_data('schedules', {'_id': schedule_id})
```

O código foi suficiente. Adicionamos dois novos testes: um para verificar se o schedule é removido do dicionário schedules do Schedule Management e outro para verificar se o erro NonExistentIDError é lançado quando o schedule não existe:

In [ ]:
```python
def test_delete_schedule_deletes_schedule_from_dictionary(self):
    # Check that delete_schedule deletes the schedule from the dictionary
    # Arrange
    schedule_id = "schedule10"
    self.schedule_management.schedules[schedule_id] = MagicMock()
    self.schedule_management.schedule_exists = MagicMock(return_value=True)
    self.schedule_management.db_module.delete_data = MagicMock()
    # Act
    self.schedule_management.delete_schedule(schedule_id)
    # Assert
    self.schedule_management.db_module.delete_data.assert_called_once_with(
        'schedules',
        {'_id': schedule_id}
    )
    self.assertNotIn(schedule_id, self.schedule_management.schedules)

def test_delete_schedule_id_doesnt_exist(self):
    # Check that delete_schedule raises an error when the schedule does not exist
    # Arrange
    schedule_id = "schedule10"
    self.schedule_management.schedule_exists = MagicMock(return_value=False)
    # Act & Assert
    with self.assertRaises(NonExistentIDError):
        self.schedule_management.delete_schedule(schedule_id)
```

O código não foi suficiente. O código foi adaptado:

In [ ]:
```python
def delete_schedule(self, schedule_id: str) -> None:
    """
    Deletes a schedule from the database and the schedules dictionary

    Args:
        schedule_id: Schedule ID
    """
    if not self.schedule_exists(schedule_id):
        raise NonExistentIDError(f"No schedule found with ID {schedule_id}")

    self.db_module.delete_data('schedules', {'_id': schedule_id})
    if schedule_id in self.schedules:
        del self.schedules[schedule_id]
```

O código foi suficiente. Outro teste foi realizado para verificar se os elements do schedule são atualizados. Para isso, também foi necessário atualizar o teste `test_delete_schedule_deletes_from_database`:

In [ ]:
```python
def test_delete_schedule_deletes_from_database(self):
    # Check that delete_schedule deletes the schedule from the database
    # Arrange
    schedule_id = "schedule10"
    self.schedule_management.db_module.delete_data = MagicMock()
    # Mock the return value of select_data
    self.schedule_management.db_module.select_data.return_value = {
        '_id': schedule_id,
        'title': 'Test Title',
        'description': 'Test Description',
        'permissions': {},
        'elements': []
    }
    # Act
    self.schedule_management.delete_schedule(schedule_id)
    # Assert
    self.schedule_management.db_module.delete_data.assert_called_once_with('schedules', {'_id': schedule_id})

def test_delete_schedule_updates_elements(self):
    # Arrange
    schedule_id = "schedule1"
    element_ids = ["element1", "element2", "element3"]
    mock_schedule = MagicMock()
    mock_schedule.elements = element_ids
    self.schedule_management.schedules[schedule_id] = mock_schedule
    with patch.object(self.schedule_management, 'get_schedule', return_value=mock_schedule), \
        patch.object(ElementManagement, 'update_element', return_value=None) as mock_update_element:
        mock_element_manager = MagicMock()
        # Act
        self.schedule_management.delete_schedule(schedule_id)
        # Assert
        assert mock_update_element.call_count == len(element_ids)
        for element_id in element_ids:
            mock_update_element.assert_any_call(element_id)
```

O código não foi suficiente. O código foi adaptado:

In [ ]:
```
def delete_schedule(self, schedule_id: str) -> None:
    """
    Deletes a schedule from the database and the schedules dictionary

    Args:
        schedule_id: Schedule ID
    """
    if not self.schedule_exists(schedule_id):
        raise NonExistentIDError(f"No schedule found with ID {schedule_id}")

    # Update each element
    schedule = self.get_schedule(schedule_id)
    element_manager = ElementManagement.get_instance()
    for element_id in schedule.elements:
        element_manager.update_element(element_id)

    self.db_module.delete_data('schedules', {'_id': schedule_id})
    if schedule_id in self.schedules:
        del self.schedules[schedule_id]
```

Também foi feito um teste para atualizar os users do schedule:

In [ ]:
```
def test_delete_schedule_updates_users(self):
    # Arrange
    schedule_id = "schedule1"
    user_ids = ["user1", "user2", "user3"]
    mock_schedule = MagicMock()
    mock_schedule.permissions = {user_id: {} for user_id in user_ids}
    self.schedule_management.schedules[schedule_id] = mock_schedule
    with patch.object(self.schedule_management, 'get_schedule', return_value=mock_schedule), \
        patch.object(UserManagement,'update_user', return_value=None) as mock_update_user:
        # Act
        self.schedule_management.delete_schedule(schedule_id)
        # Assert
        assert mock_update_user.call_count == len(user_ids)
        for user_id in user_ids:
            mock_update_user.assert_any_call(user_id)
```

O código não foi suficiente. O código foi adaptado:

In [ ]:
```
def delete_schedule(self, schedule_id: str) -> None:
    """
    Deletes a schedule from the database and the schedules dictionary

    Args:
        schedule_id: Schedule ID
    """
    if not self.schedule_exists(schedule_id):
        raise NonExistentIDError(f"No schedule found with ID {schedule_id}")

    # Update each element
    schedule = self.get_schedule(schedule_id)
    element_manager = ElementManagement.get_instance()
    for element_id in schedule.elements:
        element_manager.update_element(element_id)

    # Update each user
    user_ids = schedule.permissions.keys()
    user_manager = UserManagement.get_instance()
    for user_id in user_ids:
        user_manager.update_user(user_id)

    self.db_module.delete_data('schedules', {'_id': schedule_id})
    if schedule_id in self.schedules:
        del self.schedules[schedule_id]
```

O código foi suficiente.

Add Element to Schedule

Para adicionar um elemento a um schedule, foi desenvolvido um teste inicial para verificar se a lista elements do schedule é atualizada:

In [ ]:
```
def test_add_element_to_schedule_updates_schedule_elements(self):
    # Check that add_element_to_schedule updates the elements list of the schedule
    # Arrange
    schedule_id = "schedule10"
    element_id = "element1"
    self.schedule_management.schedules[schedule_id] = Schedule(schedule_id, "Title", "Description", {"user1": "
    # Act
    self.schedule_management.add_element_to_schedule(schedule_id, element_id)
    # Assert
    self.assertIn(element_id, self.schedule_management.schedules[schedule_id].elements)
```

Como o método `add_element_to_schedule` não foi implementado, o teste retorna erro. O código abaixo foi desenvolvido para implementar o método:

In [ ]:
```
def add_element_to_schedule(self, schedule_id: str, element_id: str) -> None:
    """
    Add an element to a schedule

    Args:
        schedule_id: Schedule ID
        element_id: Element ID
    """

    schedule = self.schedules[schedule_id]
    if element_id not in schedule.elements:
        schedule.elements = schedule.elements + [element_id]
```

O código foi suficiente. Em seguida, foi desenvolvido um teste para verificar se o erro NonExistentIDError é lançado quando o element não existe:

In [ ]:
```
@patch.object(ElementManagement, 'get_instance')
def test_add_element_to_schedule_invalid_element(self, mock_get_instance):
    # Arrange
    mock_element_manager = Mock()
    mock_element_manager.element_exists.return_value = False
    mock_get_instance.return_value = mock_element_manager
    schedule_id = "schedule1"
    element_id = "nonexistent_element"
    # Act & Assert
    with self.assertRaises(NonExistentIDError):
        self.schedule_management.add_element_to_schedule(schedule_id, element_id)
```

O código não foi suficiente. O código foi adaptado:

In [ ]:
```
def add_element_to_schedule(self, schedule_id: str, element_id: str) -> None:
    """
    Add an element to a schedule

    Args:
        schedule_id: Schedule ID
        element_id: Element ID
    """
    element_manager = ElementManagement.get_instance()
    if not element_manager.element_exists(element_id):
        raise NonExistentIDError(f"No element found with ID {element_id}")

    schedule = self.schedules[schedule_id]
    if element_id not in schedule.elements:
        schedule.elements = schedule.elements + [element_id]
```

O código foi suficiente. Em seguida, foi desenvolvido um teste para verificar se o erro NonExistentIDError é lançado quando o schedule não existe:

```
In [ ]: def test_add_element_to_schedule_invalid_schedule(self):
            # Check that add_element_to_schedule raises an error when the schedule does not exist
            # Arrange
            schedule_id = "nonexistent_schedule"
            element_id = "element1"
            self.schedule_management.schedule_exists = MagicMock(return_value=False)
            # Act & Assert
            with self.assertRaises(NonExistentIDError):
                self.schedule_management.add_element_to_schedule(schedule_id, element_id)
```

O código não foi suficiente. O código foi adaptado:

```
In [ ]: def add_element_to_schedule(self, schedule_id: str, element_id: str) -> None:
            """
            Add an element to a schedule

            Args:
                schedule_id: Schedule ID
                element_id: Element ID
            """
            element_manager = ElementManagement.get_instance()
            if not element_manager.element_exists(element_id):
                raise NonExistentIDError(f"No element found with ID {element_id}")

            if not self.schedule_exists(schedule_id):
                raise NonExistentIDError(f"No schedule found with ID {schedule_id}")

            schedule = self.schedules[schedule_id]
            if element_id not in schedule.elements:
                schedule.elements = schedule.elements + [element_id]
```

O código foi suficiente. Em seguida, foi desenvolvido um teste para verificar se o erro DuplicatedElementError é lançado quando o element já está no schedule:

```
In [ ]: def test_add_element_to_schedule_duplicated_element(self):
            # Arrange
            schedule_id = "schedule1"
            element_id = "element1"
            self.schedule_management.schedules[schedule_id] = Schedule(schedule_id, "Title", "Description", {"user1": "
            self.schedule_management.schedule_exists = MagicMock(return_value=True)
            # Act & Assert
            with self.assertRaises(DuplicatedIDError):
                self.schedule_management.add_element_to_schedule(schedule_id, element_id)
```

O código não foi suficiente. O código foi adaptado:

```
In [ ]: def add_element_to_schedule(self, schedule_id: str, element_id: str) -> None:
            """
            Add an element to a schedule

            Args:
                schedule_id: Schedule ID
                element_id: Element ID
            """
            element_manager = ElementManagement.get_instance()
            if not element_manager.element_exists(element_id):
                raise NonExistentIDError(f"No element found with ID {element_id}")

            if not self.schedule_exists(schedule_id):
                raise NonExistentIDError(f"No schedule found with ID {schedule_id}")

            schedule = self.schedules[schedule_id]
            if element_id not in schedule.elements:
                schedule.elements = schedule.elements + [element_id]
            else:
                raise DuplicatedIDError(f"Element with ID {element_id} already exists in schedule {schedule_id}")
```

O código foi suficiente. Adicionamos mais um teste para verificar se o schedule está sendo atualizado no banco:

```
In [ ]: def test_add_element_to_schedule_calls_update_database_schedule(self):
            # Check that add_element_to_schedule calls update_schedule
            # Arrange
            schedule_id = "schedule1"
            element_id = "element1"
            self.schedule_management.schedules[schedule_id] = Schedule(schedule_id, "Title", "Description", {"user1": "
```

```python
    self.schedule_management.update_schedule = MagicMock()
    # Act
    self.schedule_management.add_element_to_schedule(schedule_id, element_id)
    # Assert
    self.schedule_management.update_schedule.assert_called_once_with(schedule_id)
```

O código não foi suficiente. O código foi adaptado:

In [ ]:
```python
def add_element_to_schedule(self, schedule_id: str, element_id: str) -> None:
    """
    Add an element to a schedule

    Args:
        schedule_id: Schedule ID
        element_id: Element ID
    """
    element_manager = ElementManagement.get_instance()
    if not element_manager.element_exists(element_id):
        raise NonExistentIDError(f"No element found with ID {element_id}")

    if not self.schedule_exists(schedule_id):
        raise NonExistentIDError(f"No schedule found with ID {schedule_id}")

    schedule = self.schedules[schedule_id]
    if element_id not in schedule.elements:
        schedule.elements = schedule.elements + [element_id]
        self.update_schedule(schedule_id)
    else:
        raise DuplicatedIDError(f"Element with ID {element_id} already exists in schedule {schedule_id}")
```

O código foi suficiente. A próxima etapa foi desenvolver um teste para atualizar a lista de schedules do element. Contudo, a implementação do método get_element do Element Management gerou erro em outros dois testes. O novo teste e os testes atualizados são:

In [ ]:
```python
def test_add_element_to_schedule_updates_schedule_elements(self):
    # Check that add_element_to_schedule updates the elements list of the schedule
    # Arrange
    schedule_id = "schedule10"
    element_id = "element1"
    mock_element = MagicMock()
    self.schedule_management.schedules[schedule_id] = Schedule(schedule_id, "Title", "Description", {"user1": "
    with patch.object(ElementManagement, 'get_element', return_value=mock_element):
        # Act
        self.schedule_management.add_element_to_schedule(schedule_id, element_id)
        # Assert
        self.assertIn(element_id, self.schedule_management.schedules[schedule_id].elements)

def test_add_element_to_schedule_calls_update_schedule(self):
    # Check that add_element_to_schedule calls update_schedule
    # Arrange
    schedule_id = "schedule1"
    element_id = "element1"
    mock_element = MagicMock()
    self.schedule_management.schedules[schedule_id] = Schedule(schedule_id, "Title", "Description", {"user1": "
    self.schedule_management.update_schedule = MagicMock()
    with patch.object(ElementManagement, 'get_element', return_value=mock_element):
        # Act
        self.schedule_management.add_element_to_schedule(schedule_id, element_id)
        # Assert
        self.schedule_management.update_schedule.assert_called_once_with(schedule_id)

def test_add_element_to_schedule_updates_element_schedules(self):
    # Arrange
    schedule_id = "schedule10"
    element_id = "element1"
    # Create a mock schedule with 'element2' as an element
    mock_schedule = MagicMock(spec=Schedule)
    mock_schedule.elements = ["element2"]
    # Create a mock element with no schedules
    mock_element = MagicMock()
    mock_element.schedules = []
    # Mock the get_element method to return our mock element when called with 'element1'
    with patch.object(ElementManagement, 'get_element', return_value=mock_element):
        # Add the mock schedule to the schedules dictionary
        self.schedule_management.schedules[schedule_id] = mock_schedule
        # Act
        self.schedule_management.add_element_to_schedule(schedule_id, element_id)
        # Assert
        self.assertIn(schedule_id, mock_element.schedules)
```

Esse código foi suficiente:

In [ ]:
```python
def add_element_to_schedule(self, schedule_id: str, element_id: str) -> None:
    """
    Add an element to a schedule

    Args:
        schedule_id: Schedule ID
        element_id: Element ID
    """
    element_manager = ElementManagement.get_instance()
    if not element_manager.element_exists(element_id):
        raise NonExistentIDError(f"No element found with ID {element_id}")

    if not self.schedule_exists(schedule_id):
        raise NonExistentIDError(f"No schedule found with ID {schedule_id}")

    schedule = self.schedules[schedule_id]
    if element_id not in schedule.elements:
        schedule.elements = schedule.elements + [element_id]
        self.update_schedule(schedule_id)
        element = element_manager.get_element(element_id)
        element.schedules = element.schedules + [schedule_id]
    else:
        raise DuplicatedIDError(f"Element with ID {element_id} already exists in schedule {schedule_id}")
```

Por fim, adicionamos um teste para verificar se o element está sendo atualizado no banco:

In [ ]:
```python
def test_add_element_to_schedule_calls_update_element(self):
    # Check that add_element_to_schedule calls update_element
    # Arrange
    schedule_id = "schedule1"
    element_id = "element1"
    mock_element = MagicMock()
    mock_element.schedules = []
    self.schedule_management.schedules[schedule_id] = Schedule(schedule_id, "Title", "Description", {"user1": "
    with patch.object(ElementManagement, 'get_element', return_value=mock_element), \
        patch.object(ElementManagement, 'update_element', return_value=None) as mock_update_element:
        # Act
        self.schedule_management.add_element_to_schedule(schedule_id, element_id)
        # Assert
        mock_update_element.assert_called_once_with(element_id)
```

O código não foi suficiente. O código foi adaptado:

In [ ]:
```python
def add_element_to_schedule(self, schedule_id: str, element_id: str) -> None:
    """
    Add an element to a schedule

    Args:
        schedule_id: Schedule ID
        element_id: Element ID
    """
    element_manager = ElementManagement.get_instance()
    if not element_manager.element_exists(element_id):
        raise NonExistentIDError(f"No element found with ID {element_id}")

    if not self.schedule_exists(schedule_id):
        raise NonExistentIDError(f"No schedule found with ID {schedule_id}")

    schedule = self.schedules[schedule_id]
    if element_id not in schedule.elements:
        schedule.elements = schedule.elements + [element_id]
        self.update_schedule(schedule_id)
        element = element_manager.get_element(element_id)
        element.schedules = element.schedules + [schedule_id]
        element_manager.update_element(element_id)
    else:
        raise DuplicatedIDError(f"Element with ID {element_id} already exists in schedule {schedule_id}")
```

O código foi suficiente.

Refactor: Observer

Com a implementação do Observer, foi necessário refatorar o código do Schedule Management. Para isso, o seguinte teste foi modificado:

In [ ]:
```python
def test_add_element_to_schedule_calls_update_schedule(self):
    # Check that add_element_to_schedule calls update_schedule
    # Arrange
    schedule_id = "schedule1"
    element_id = "element1"
    mock_element = MagicMock()
    test_schedule = Schedule(schedule_id, "Title", "Description", {"user1": "read"}, ["element2"])
    test_schedule.attach(self.schedule_management)
    self.schedule_management.schedules[schedule_id] = test_schedule
    self.schedule_management.update = MagicMock()
    with patch.object(ElementManagement, 'get_element', return_value=mock_element):
        # Act
        self.schedule_management.add_element_to_schedule(schedule_id, element_id)
        # Assert
        self.schedule_management.update.assert_called_once_with(test_schedule)
```

O código adaptado é o seguinte:

In [ ]:
```python
def add_element_to_schedule(self, schedule_id: str, element_id: str) -> None:
    """
    Add an element to a schedule

    Args:
        schedule_id: Schedule ID
        element_id: Element ID
    """
    element_manager = ElementManagement.get_instance()
    if not element_manager.element_exists(element_id):
        raise NonExistentIDError(f"No element found with ID {element_id}")

    if not self.schedule_exists(schedule_id):
        raise NonExistentIDError(f"No schedule found with ID {schedule_id}")

    schedule = self.schedules[schedule_id]
    if element_id not in schedule.elements:
        schedule.elements = schedule.elements + [element_id]
        element = element_manager.get_element(element_id)
        element.schedules = element.schedules + [schedule_id]
        element_manager.update_element(element_id)
    else:
        raise DuplicatedIDError(f"Element with ID {element_id} already exists in schedule {schedule_id}")
```

Outros testes vieram a ser modificados. Como com o Observer temos que os métodos `update_user`, `update_schedule` e `update_element` não necessitam mais ser chamados, pois o Observer já faz isso ao alterarmos os atributos dos objetos. Dessa forma, alguns testes foram removidos: `test_add_element_to_schedule_calls_update_element`, `test_create_schedule_updates_users_schedules` e `test_create_schedule_updates_elements_schedules`. Os testes referentes à `create_schedule` e `delete_schedule` sofreram alterações:

In [ ]:
```python
def test_create_schedule(self):
    """ General test for create_schedule """
    # Arrange
    self.db_module.insert_data = MagicMock()
    self.db_module.select_data = MagicMock(return_value=[])
    schedule_id = "schedule10"
    title = "Schedule 2"
    description = "This is schedule 2"
    permissions = {"user1": "write", "user2": "read"}
    elements = ["element2", "element3"]
    mock_user = MagicMock()
    mock_element = MagicMock()
    with patch.object(self.user_management, 'get_user', return_value=mock_user), \
    patch.object(self.user_management, 'update_user', return_value=None), \
    patch.object(self.user_management, 'user_exists', return_value=True), \
    patch.object(self.element_management, 'get_element', return_value=mock_element), \
    patch.object(self.element_management, 'update_element', return_value=None), \
    patch.object(self.element_management, 'element_exists', return_value=True):
        # Act
        result = self.schedule_management.create_schedule(schedule_id,
        title, description, permissions, elements)
        with self.subTest():
            self._test_create_schedule_insert_data(schedule_id,
        title, description, permissions, elements)
        with self.subTest():
            self._test_create_schedule_return(result)
        with self.subTest():
            self._test_create_schedule_attributes(result, schedule_id,
            title, description, permissions, elements)
        with self.subTest():
            self._test_create_schedule_adds_to_self_schedules(schedule_id)
        with self.subTest():
            self._test_create_schedule_raises_error_if_schedule_exists(
                schedule_id, title, description, permissions, elements)

def test_create_schedule_raises_error_with_invalid_title(self):
    """Test that create_schedule raises an error when the title is invalid"""
    # Arrange
    self.schedule_management.db_module.insert_data = MagicMock()
    self.schedule_management.db_module.select_data = MagicMock(return_value=[])
    invalid_titles = [None, 123, "", "   ", "a" * 51]  # Covers all restrictions
    schedule_id = "schedule10"
    description = "This is schedule 2"
    permissions = {"user1": "write", "user2": "read"}
    elements = ["element2", "element3"]
    # Act & Assert
    with patch.object(self.user_management, 'user_exists', return_value=True), \
    patch.object(self.element_management, 'element_exists', return_value=True):
        for title in invalid_titles:
            with self.assertRaises((ValueError, TypeError)):
                self.schedule_management.create_schedule(schedule_id, title,
                        description, permissions, elements)

def test_create_schedule_raises_error_with_invalid_description(self):
    """Test that create_schedule raises an error when the description is invalid"""
    # Arrange
    self.schedule_management.db_module.insert_data = MagicMock()
    self.schedule_management.db_module.select_data = MagicMock(return_value=[])
    invalid_descriptions = [123, "a" * 501]  # Covers all restrictions
    schedule_id = "schedule10"
    title = "Schedule 2"
    permissions = {"user1": "write", "user2": "read"}
    elements = ["element2", "element3"]
    # Act & Assert
    with patch.object(self.user_management, 'user_exists', return_value=True), \
    patch.object(self.element_management, 'element_exists', return_value=True):
        for description in invalid_descriptions:
            with self.assertRaises((ValueError, TypeError)):
                self.schedule_management.create_schedule(schedule_id, title,
                        description, permissions, elements)

def test_create_schedule_raises_error_with_non_string_id(self):
    """
    Test that create_schedule raises an error when the ID is not a string
    """
    # Arrange
    self.schedule_management.db_module.insert_data = MagicMock()
    self.schedule_management.db_module.select_data = MagicMock(return_value=[])
    schedule_id = 123  # Non-string ID
    title = "Schedule 2"
    description = "This is schedule 2"
    permissions = {"user1": "write", "user2": "read"}
    elements = ["element2", "element3"]
    # Act & Assert
    with patch.object(self.user_management, 'user_exists', return_value=True), \
    patch.object(self.element_management, 'element_exists', return_value=True):
        with self.assertRaises(TypeError):
            self.schedule_management.create_schedule(schedule_id, title,
                    description, permissions, elements)

def test_create_schedule_raises_error_with_empty_permissions(self):
    """
    Test that create_schedule raises an error when the permissions are empty
    """
    # Arrange
    self.schedule_management.db_module.insert_data = MagicMock()
    self.schedule_management.db_module.select_data = MagicMock(
            return_value=[])
    schedule_id = "schedule10"
    title = "Schedule 2"
    description = "This is schedule 2"
    permissions = {}  # Empty permissions
    elements = ["element2", "element3"]
    # Act & Assert
    with patch.object(self.user_management, 'user_exists', return_value=True), \
    patch.object(self.element_management, 'element_exists', return_value=True):
        with self.assertRaises(EmptyPermissionsError):
            self.schedule_management.create_schedule(schedule_id, title,
                    description, permissions, elements)


def test_create_schedule_updates_elements(self):
    # Arrange
    schedule_id = "schedule1"
    title = "Test Title"
    description = "Test Description"
    permissions = {"user1": {}}
    elements = ["element1", "element2", "element3"]
    mock_user = MagicMock()
```

```python
        mock_element = MagicMock()
        mock_element.schedules = []

        with patch.object(self.user_management, 'user_exists', return_value=True), \
                patch.object(self.user_management, 'get_user', return_value=mock_user), \
                patch.object(self.element_management, 'element_exists', return_value=True), \
                patch.object(self.element_management, 'get_element', return_value=mock_element), \
                patch.object(self.schedule_management, 'schedule_exists', return_value=False):

            # Act
            self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)

            # Assert
            for _ in elements:
                self.assertIn(schedule_id, mock_element.schedules)


    def test_create_schedule_raises_error_for_nonexistent_element(self):
        """
        Test that create_schedule raises an error
        when the element does not exist
        """
        # Arrange
        schedule_id = "schedule1"
        title = "Test Title"
        description = "Test Description"
        permissions = {"user1": {}}
        elements = ["element1", "nonexistent_element"]
        with patch.object(self.schedule_management, 'schedule_exists',
                return_value=False), \
            patch.object(ElementManagement, 'element_exists',
                    side_effect=[True, False]):
            # Act & Assert
            with self.assertRaises(NonExistentIDError):
                self.schedule_management.create_schedule(schedule_id, title,
                        description, permissions, elements)


    def test_create_schedule_updates_users(self):
        # Arrange
        schedule_id = "schedule1"
        title = "Test Title"
        description = "Test Description"
        permissions = {"user1": {}, "user2": {}, "user3": {}}
        elements = ["element1", "element2", "element3"]
        mock_element = MagicMock()
        mock_element.schedules = []
        mock_user = MagicMock()
        mock_user.schedules = []

        with patch.object(self.user_management, 'user_exists', return_value=True), \
            patch.object(self.user_management, 'get_user', return_value=mock_user), \
            patch.object(self.element_management, 'element_exists', return_value=True), \
            patch.object(self.element_management, 'get_element', return_value=mock_element), \
            patch.object(self.schedule_management, 'schedule_exists', return_value=False):

            # Act
            self.schedule_management.create_schedule(schedule_id, title, description, permissions, elements)

            # Assert
            for user_id in permissions:
                self.assertIn(schedule_id, mock_user.schedules)

    def test_create_schedule_raises_error_for_nonexistent_user(self):
        """
        Test that create_schedule raises an error when the user does not exist
        """
        # Arrange
        schedule_id = "schedule1"
        title = "Test Title"
        description = "Test Description"
        permissions = {"user1": {}, "nonexistent_user": {}}
        elements = ["element1", "element2"]
        with patch.object(self.schedule_management, 'schedule_exists',
                return_value=False), \
            patch.object(UserManagement, 'user_exists',
                    side_effect=[True, False]):
            # Act & Assert
            with self.assertRaises(NonExistentIDError):
                self.schedule_management.create_schedule(schedule_id, title,
                        description, permissions, elements)

    def test_delete_schedule_deletes_from_database(self):
        """Check that delete_schedule deletes the schedule from the database"""
        # Arrange
        schedule_id = "schedule10"
        self.schedule_management.db_module.delete_data = MagicMock()
        # Mock the return value of select_data
        self.schedule_management.db_module.select_data.return_value = [{
            '_id': schedule_id,
            'title': 'Test Title',
            'description': 'Test Description',
            'permissions': {},
            'elements': []
        }]
        # Act
        self.schedule_management.delete_schedule(schedule_id)
        # Assert
        self.schedule_management.db_module.delete_data.assert_called_once_with(
                'schedules', {'_id': schedule_id})

    def test_delete_schedule_deletes_schedule_from_dictionary(self):
        """
        Check that delete_schedule deletes the schedule from the dictionary
        """
        # Arrange
        schedule_id = "schedule10"
        self.schedule_management.schedules[schedule_id] = MagicMock()
        self.schedule_management.schedule_exists = MagicMock(return_value=True)
        self.schedule_management.db_module.delete_data = MagicMock()
        # Act
        self.schedule_management.delete_schedule(schedule_id)
        # Assert
        self.schedule_management.db_module.delete_data.assert_called_once_with(
            'schedules',
            {'_id': schedule_id}
        )
        self.assertNotIn(schedule_id, self.schedule_management.schedules)

    def test_delete_schedule_id_doesnt_exist(self):
        """
        Test that delete_schedule raises an error
        when the schedule does not exist
        """
        # Arrange
        schedule_id = "schedule10"
        self.schedule_management.schedule_exists = MagicMock(return_value=False)
        # Act & Assert
        with self.assertRaises(NonExistentIDError):
            self.schedule_management.delete_schedule(schedule_id)

    def test_delete_schedule_updates_elements(self):
        """Test that delete_schedule updates the schedules of the elements"""
        # Arrange
        schedule_id = "schedule1"
        element_ids = ["element1", "element2", "element3"]
        mock_schedule = MagicMock()
        mock_schedule.elements = element_ids
        self.schedule_management.schedules[schedule_id] = mock_schedule
        mock_element = MagicMock()
        mock_element.schedules = [schedule_id]

        with patch.object(self.schedule_management, 'get_schedule', return_value=mock_schedule), \
            patch.object(self.element_management, 'get_element', return_value=mock_element):

            # Act
            self.schedule_management.delete_schedule(schedule_id)

            # Assert
            for element_id in element_ids:
                self.assertNotIn(schedule_id, mock_element.schedules)

    def test_delete_schedule_updates_users(self):
        """Test that delete_schedule updates the schedules of the users"""
        # Arrange
        schedule_id = "schedule1"
        user_ids = ["user1", "user2", "user3"]
        mock_schedule = MagicMock()
        mock_schedule.permissions = {user_id: {} for user_id in user_ids}
        self.schedule_management.schedules[schedule_id] = mock_schedule
        mock_user = MagicMock()
        mock_user.schedules = [schedule_id]

        with patch.object(self.schedule_management, 'get_schedule', return_value=mock_schedule), \
            patch.object(self.user_management, 'get_user', return_value=mock_user):
```

```
        # Act
        self.schedule_management.delete_schedule(schedule_id)

        # Assert
        for user_id in user_ids:
            self.assertNotIn(schedule_id, mock_user.schedules)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js