Kanboard

Create your own Kanban Board

Kanboard is a simple visual task board software.
From scratch to finish, manage your projects with ease.

Get Started

Requisiti - User Stories

Ruoli, in sintesi

- **Non Registered User**: non ha un account registrato e non è autenticato.
- **Non Logged In User:** ha un account registrato ma non è autenticato.
- **Logged In User**: è autenticato, ha accesso alle proprie board e può crearne di nuove.
- **Board Owner**: gestisce la board, le colonne, le card e gli utenti.
- **Board Guest**: partecipa alle board a cui è invitato, può gestire solo le card.

Alcune proposte

In qualità di [*Non Logged In User]*
voglio *effettuare l'accesso al mio account*
in modo da *accedere alla mia dashboard personale e gestire i miei progetti*

Acceptance test: test_login

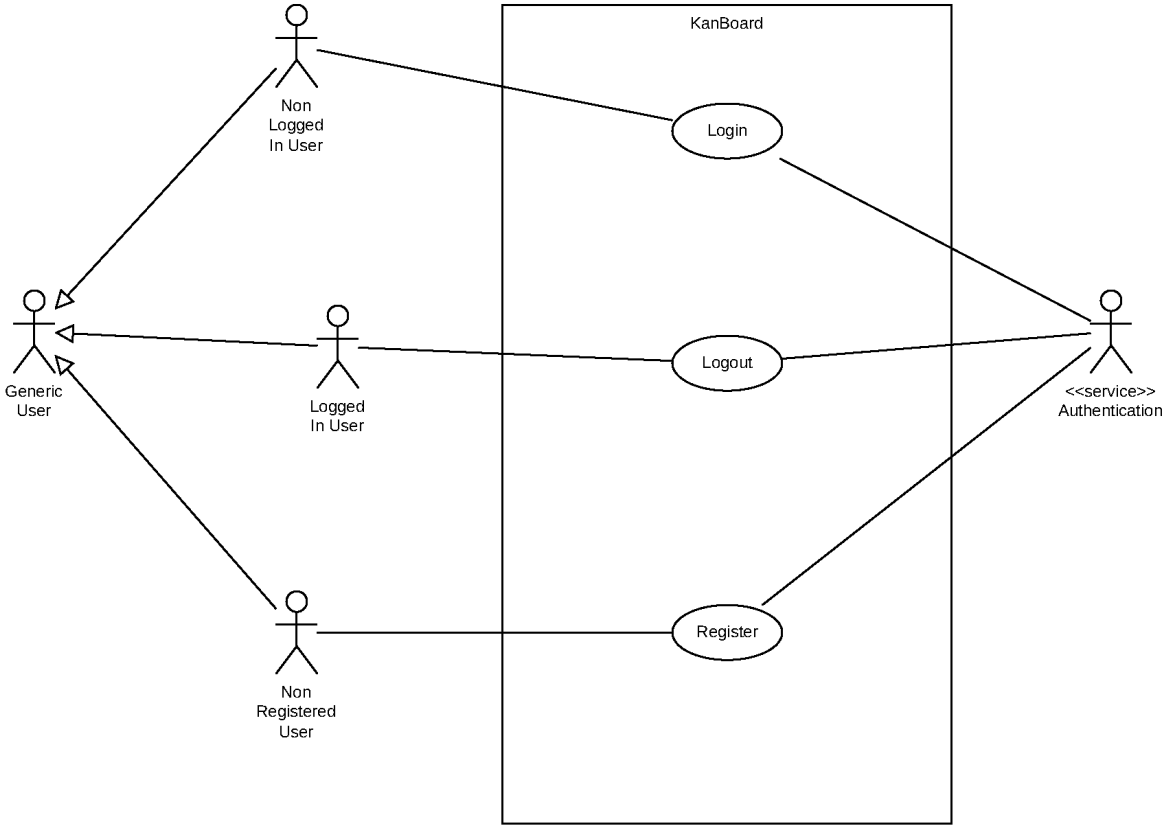
In qualità di *[Logged In User]*
voglio *creare una nuova Board con un titolo*
in modo da *organizzare i progetti in spazi separati e ben definiti all'interno della mia dashboard*

Acceptance test: test_create_board

In qualità di *[Board Owner]*
voglio *creare una nuova Colonna con un titolo nella Board corrente*
in modo da *separare le attivita' in categorie e migliorare la gestione del progetto*

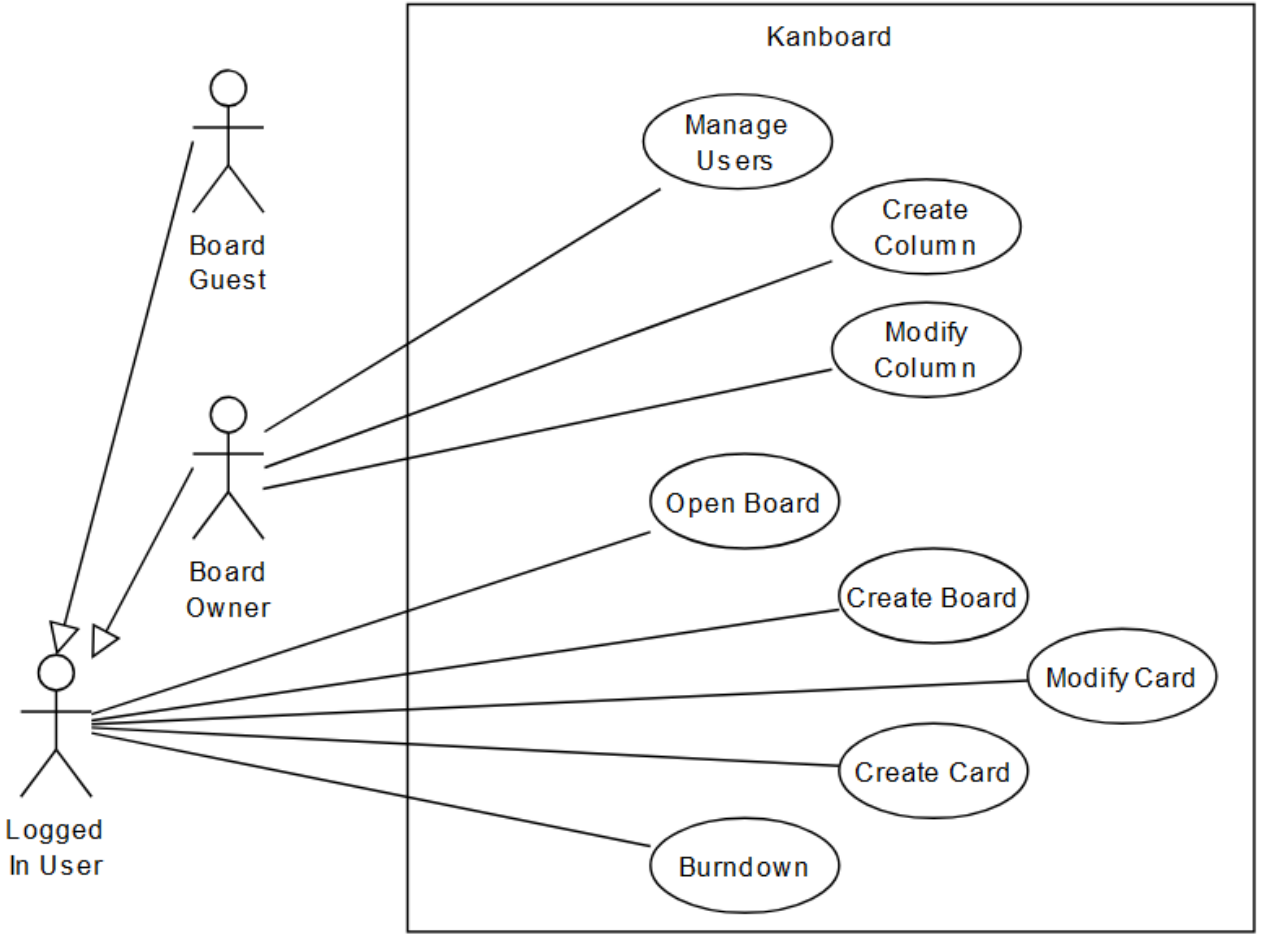
Acceptance test: test_create_column

Requisiti – Diagramma delle User Stories

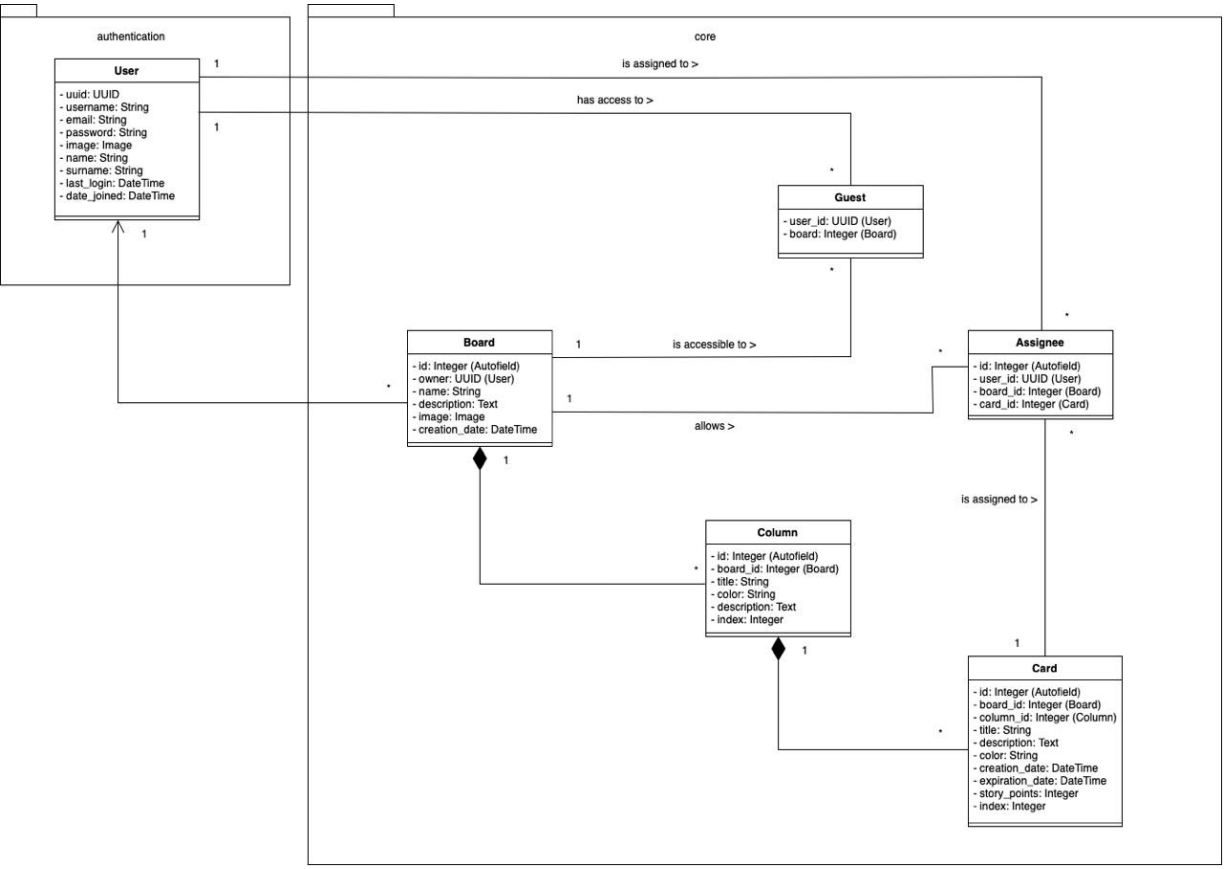


Autenticazione

Core

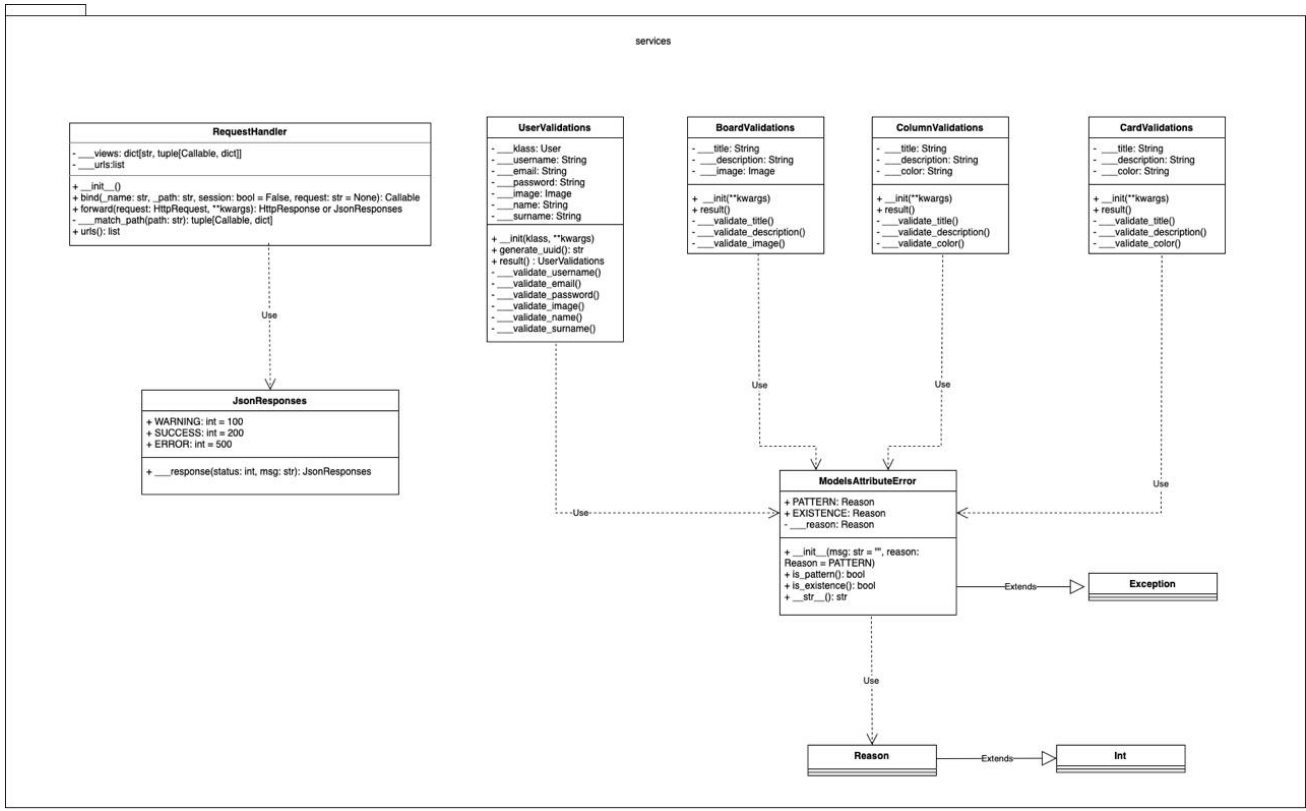


Requisiti – Diagramma delle Classi

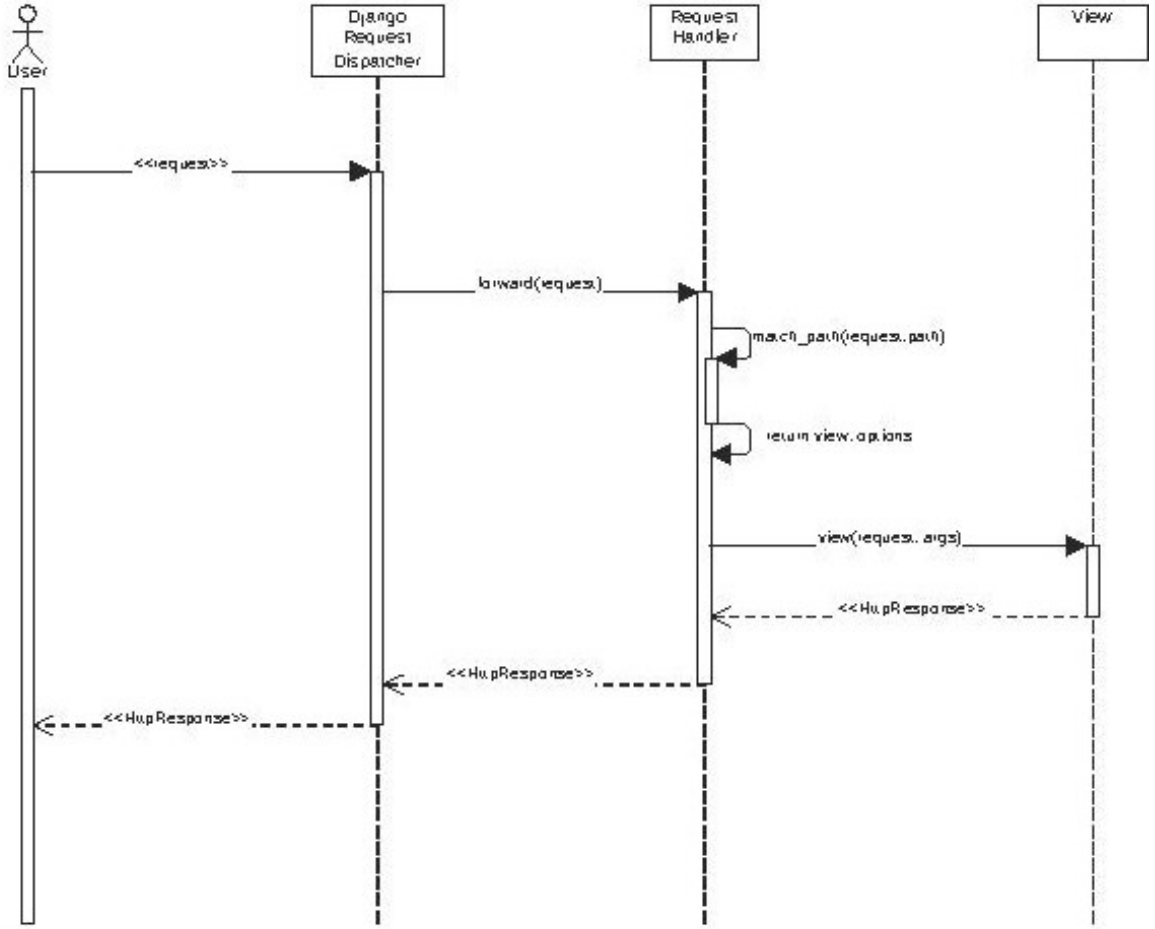


Core

Services

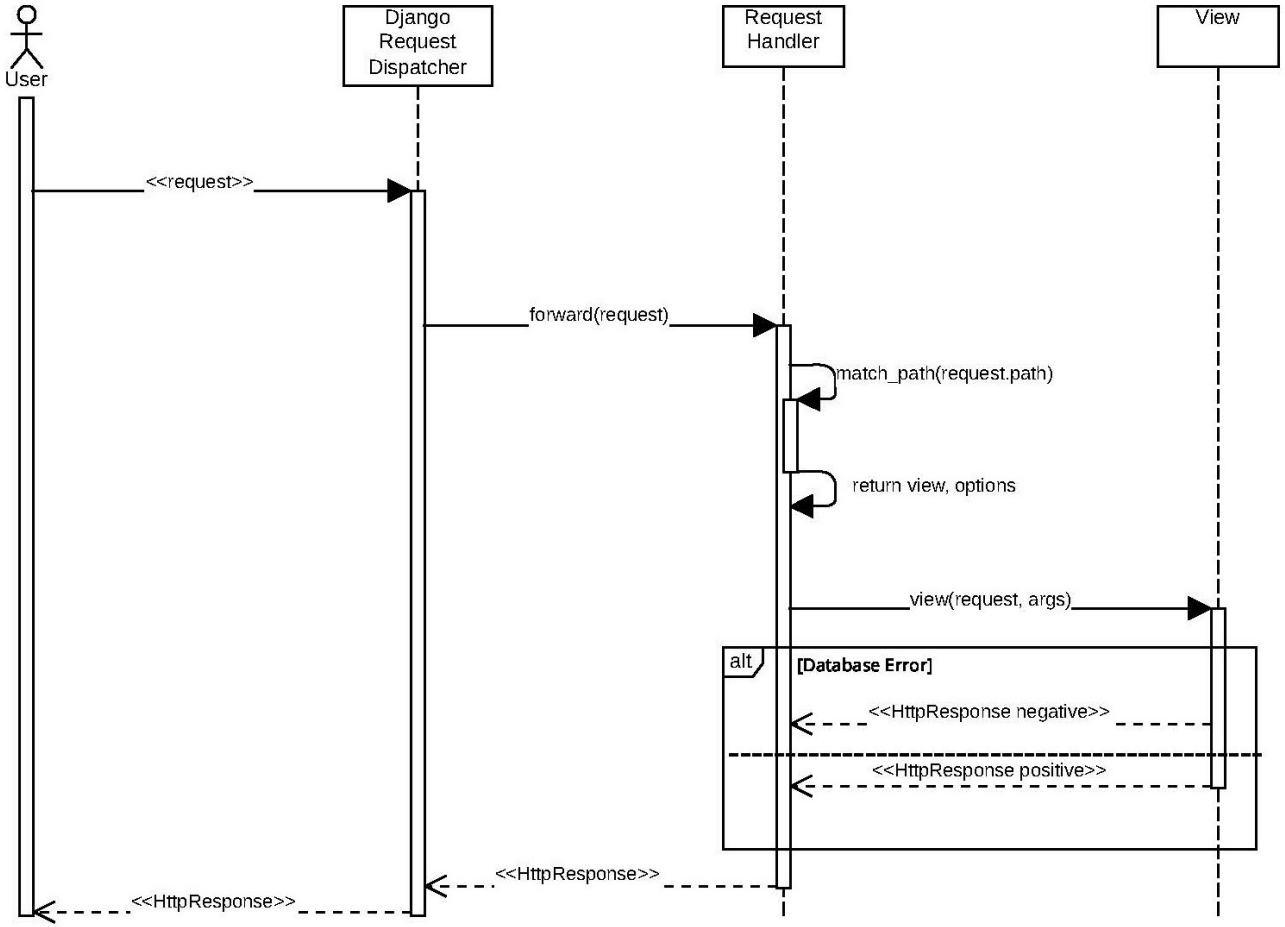


Requisiti – Diagrammi di Sequenza

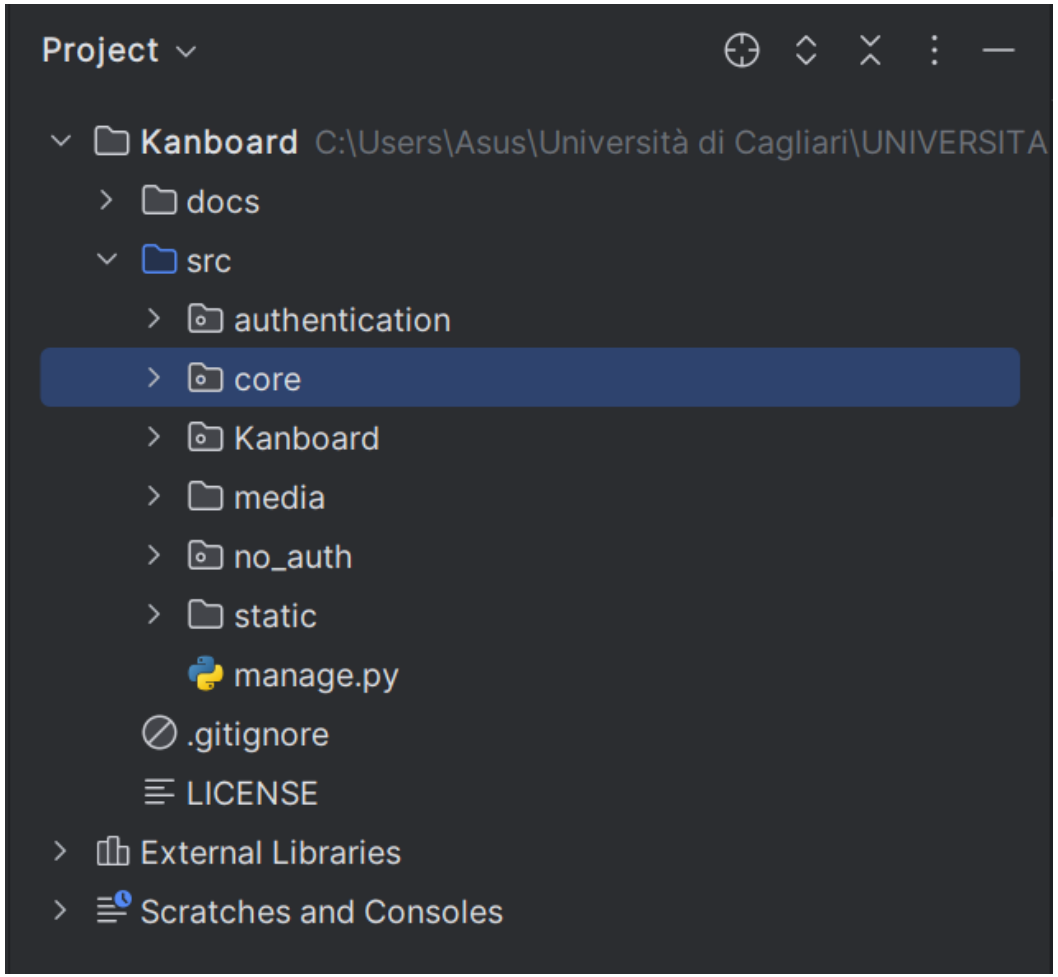


Retrieve

Store



Organizzazione del Codice



- «authentication»: gestisce registrazione, login, modifica profilo e logout, con validazioni personalizzate.
- «core»: include gestione e visualizzazione di board, colonne e card, con controllo accessi.
- «no_auth»: gestisce pagine statiche dove non è richiesta l'autenticazione.

Modelli - Implementazione

```python
class User(models.Model):
    uuid = models.UUIDField(primary_key=True, editable=False)
    username = models.CharField(max_length=16, unique=True)
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=32)
    image = models.ImageField(null=True, blank=True)
    name = models.CharField(max_length=32)
    surname = models.CharField(max_length=32)
    last_login = models.DateTimeField()
    date_joined = models.DateTimeField()
```

«User»

- Rappresenta gli utenti registrati nel sistema
- Include informazioni essenziali e dettagli relativi all'attività dell'utente

```python
class Guest(models.Model):  27 usages
    id = models.AutoField(primary_key=True)
    user_id = models.ForeignKey(User, on_delete=models.CASCADE, db_column="user_id")
    board_id = models.ForeignKey(Board, on_delete=models.CASCADE, db_column="board_id")
```

«Guest»

- Rappresenta gli utenti ospite registrati ad una Board.

Modelli - Implementazione

```python
class Board(models.Model):
    id = models.AutoField(primary_key=True)
    owner = models.ForeignKey(User, on_delete=models.CASCADE, db_column="owner")
    name = models.CharField(max_length=100)
    description = models.TextField(default="")
    image = models.ImageField(blank=True, null=True)
    creation_date = models.DateTimeField()
```

«Board»

- Rappresenta una Board all'interno del sistema, gestita da un utente.

```python
class Column(models.Model):  18 usages
    id = models.AutoField(primary_key=True)
    board_id = models.ForeignKey(Board, on_delete=models.CASCADE, db_column="board_id")
    title = models.CharField(max_length=20)
    description = models.TextField(max_length=256)
    color = models.CharField(default="#808080", max_length=7)
    index = models.IntegerField()
```

«Column»

- Rappresenta una colonna associata a una specifica Board.

Modelli - Implementazione

```python
class Card(models.Model):  22 usages
    id = models.AutoField(primary_key=True)
    board_id = models.ForeignKey(Board, on_delete=models.CASCADE, db_column="board_id")
    column_id = models.ForeignKey(Column, on_delete=models.CASCADE, db_column="column_id")
    title = models.CharField(max_length=20)
    description = models.TextField(max_length=256)
    color = models.CharField(default="#808080", max_length=7)
    creation_date = models.DateTimeField()
    expiration_date = models.DateTimeField(null=True, blank=True, default=None)
    completion_date = models.DateTimeField(null=True, blank=True, default=None)
    story_points = models.IntegerField(default=0)
    index = models.IntegerField()
```

«Card»

- Rappresenta una Card all'interno di una colonna di una Board.

```python
class Assignee(models.Model):  11 usages
    id = models.AutoField(primary_key=True)
    user_id = models.ForeignKey(User, on_delete=models.CASCADE, db_column="user_id")
    board_id = models.ForeignKey(Board, on_delete=models.CASCADE, db_column="board_id")
    card_id = models.ForeignKey(Card, on_delete=models.CASCADE, db_column="card_id")
```

« Assignee»

- Rappresenta un utente assegnato ad una Card.

Views - Implementazione

```python
@HANDLER.bind(_name:"board", _path:"board/<int:board_id>/", request="GET", session=True)
@requires_csrf_token
def board(request, board_id):
    """
    Renders the board page with the board details and columns.
    Requires the method to be GET and the user to be authenticated.

    :param request: HttpRequest - The HTTP request object.
    :param board_id: int - The ID of the board to display.
    :return: HttpResponse - The rendered HTML page with the board details.
    """
    uuid = get_user_from(request)
    board = get_board(Board, board_id)

    if check_board_invalid(board):
        return response_error("Board not found.")

    if check_user_not_owner_or_guest(Board, Guest, board_id, uuid):
        return response_error("You do not have access to this board.")

    board_info = {
        'id': board.id,
        'name': board.name,
        'description': board.description,
        'image': board.image,
        'creation_date': board.creation_date
    }

    columns = get_board_elements(Column, Card, Assignee, User, board_id)
    return render(request, template_name:"boards.html", context:{
        "board": board_info,
        "columns": columns
    })
```

- View binding:
  - Name of view
  - Path
  - Request type (eg. GET)
  - Session (eg. True)
- Django models
- Render templates

Views – Request Handler

```python
class RequestHandler:  8 usages
    """
    This class is a singleton class that handles requests
    before they are sent to the views.

    This class is responsible for:
        - Validating requests
        - Handling errors
        - Sending the responses to the client

    How to use:
        - use the bind decorator to bind a view to a specific path
        - use the forward method inside the urls.py file to
          dynamically forward requests to the appropriate view

    Methods:
        - bind: Binds a view to a specific path.
        - forward: Forwards a request to the appropriate view.
        - match_path: Matches the request path to the bound view.
    """
```

```python
def forward(self, request: HttpRequest, **kwargs):  1 usage
    """
    Forwards a request to the appropriate view.

    :param request: HttpRequest - The request object.
    :param kwargs: tuple - Additional arguments to pass to the view.
    :return: HttpResponse - The response from the view or an error message.
    """

    view, options = None, None
    try:
        view, options = self.___match_path(request.path)
    except Exception:
        return HttpResponse("404 Not Found")

    uuid = request.session.get(key:'uuid', default:None)

    if options['session'] and uuid is None:
        return HttpResponse("401 Unauthorized")
    if options['request'] is not None and request.method != options['request']:
        return HttpResponse("405 Method Not Allowed")

    return view(request, **kwargs)
```

Test Unitari

```python
def test_validate_user_username_ok(self):
    validator = UserValidations(username="valid_username")
    try:
        validator.result()
    except ModelsAttributeError as e:
        self.fail(f"result() raised ModelsAttributeError unexpectedly: {e}")

def test_validate_user_username_too_long(self):
    validator = UserValidations(username="this_username_is_definitely_too_long")
    self.assertRaises(ModelsAttributeError, validator.result)

def test_validate_user_username_invalid_characters(self):
    validator = UserValidations(username="invalid#username")
    self.assertRaises(ModelsAttributeError, validator.result)
```

- Libreria unittest
- Testiamo le interfacce
- Testing a singolo punto di uscita
- Non eseguiamo Mocking

Test di Accettazione – Le basi

```python
def setUp(self):
    self.driver = webdriver.Firefox()
    self.wait = WebDriverWait(self.driver, timeout: 5)
    self.server_url = "http://localhost:8000/"
    self.dummy_username = "acceptancetest"
    self.dummy_password = "acceptancetest"
    self.dummy_email = "acceptance@test.com"
    self.dummy_name = "Acceptance"
    self.dummy_surname = "Test"
    self.dummy_board = "Acceptance Board"
```

```python
def tearDown(self):
    driver = self.driver
    try:
        driver.get(self.server_url + 'acceptance/delete/')
    except Exception as e:
        print(f"Cleanup failed: {e}")
    finally:
        self.driver.close()
        print("Test completed.")
```

- Libreria Selenium
- «setUp»: utilizza dati «dummy» come input nei test (predicibilità) e inizializza il time-out di risposta massimo ottenibile (evita le *sleep(…)* esplicite!)
- «tearDown»: rimuove le risorse residue create dai test in questione.

Test di Accettazione

```python
def test_register(self):  1 usage
    driver = self.driver
    driver.get(self.server_url + 'register')
    self.assertEqual(first: "Kanboard - Register", driver.title)

    name = driver.find_element(By.ID, value: "name")
    surname = driver.find_element(By.ID, value: "surname")
    username = driver.find_element(By.ID, value: "username")
    email = driver.find_element(By.ID, value: "email")
    password = driver.find_element(By.ID, value: "password")
    repeat_password = driver.find_element(By.ID, value: "repeat-password")
    submit = driver.find_element(By.CLASS_NAME, value: "submit-button")

    name.send_keys(self.dummy_name)
    surname.send_keys(self.dummy_surname)
    username.send_keys(self.dummy_username)
    email.send_keys(self.dummy_email)
    password.send_keys(self.dummy_password)
    repeat_password.send_keys(self.dummy_password)
    submit.click()

    try:
        result = self.wait.until(EC.presence_of_element_located((By.ID, "dashboard")))
    except NoSuchElementException as e:
        self.fail(f"Register failed: {e}")

    self.assertIsNotNone(result)
    self.assertEqual(first: "Kanboard - Dashboard", driver.title)
```

«test_register»: automatizza la registrazione di un utente, inviando il modulo e verificando il reindirizzamento alla dashboard, la presenza dell'elemento "dashboard" e il titolo della pagina.

«test_logout» registra un nuovo utente, naviga alla dashboard e clicca sul pulsante di logout, verificando che l'utente venga disconnesso correttamente.

```python
def test_logout(self):  1 usage
    try:
        self.test_register()
    except Exception:
        print("[INFO] Could not register")

    driver = self.driver
    driver.get(self.server_url + 'dashboard')

    result = None

    try:
        result = self.wait.until(EC.element_to_be_clickable((By.ID, "logout")))
    except NoSuchElementException as e:
        self.fail(f"Logout failed: {e}")

    result.click()

    try:
        result = self.wait.until(EC.presence_of_element_located((By.CLASS_NAME, "form-title")))
    except NoSuchElementException as e:
        self.fail(f"Logout failed: {e}")

    self.assertIsNotNone(result)
    self.assertEqual(first: "Kanboard - Log In", driver.title)
```

Grazie per l’attenzione

Antonio Masala - 60/61/66133
Diego Mura - 60/61/66117
Francesca Zirone - 60/61/65957
Marco La Civita - 60/61/66150