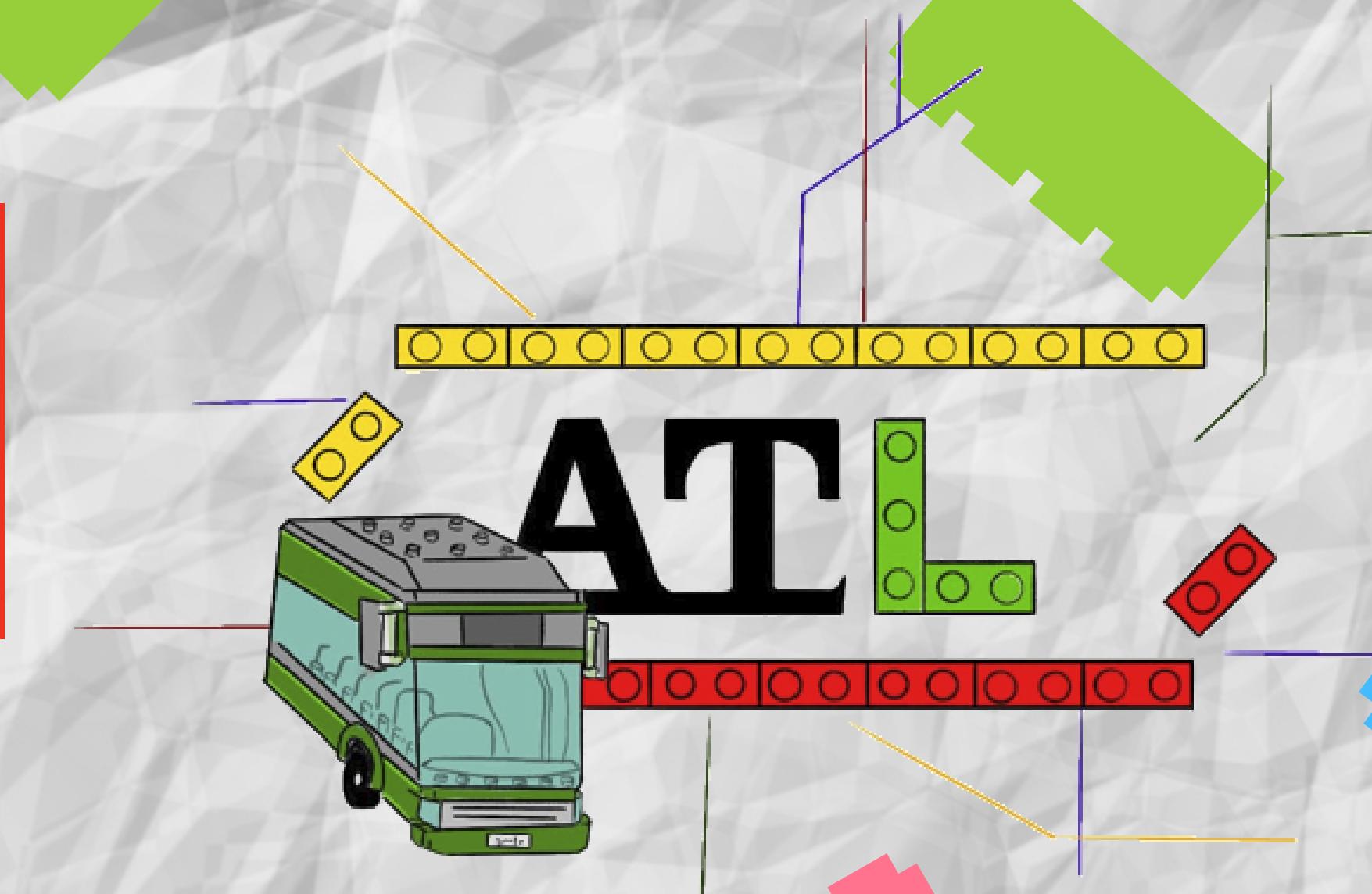


# ATL-AZIENDA TRASPORTI LEGOCITY

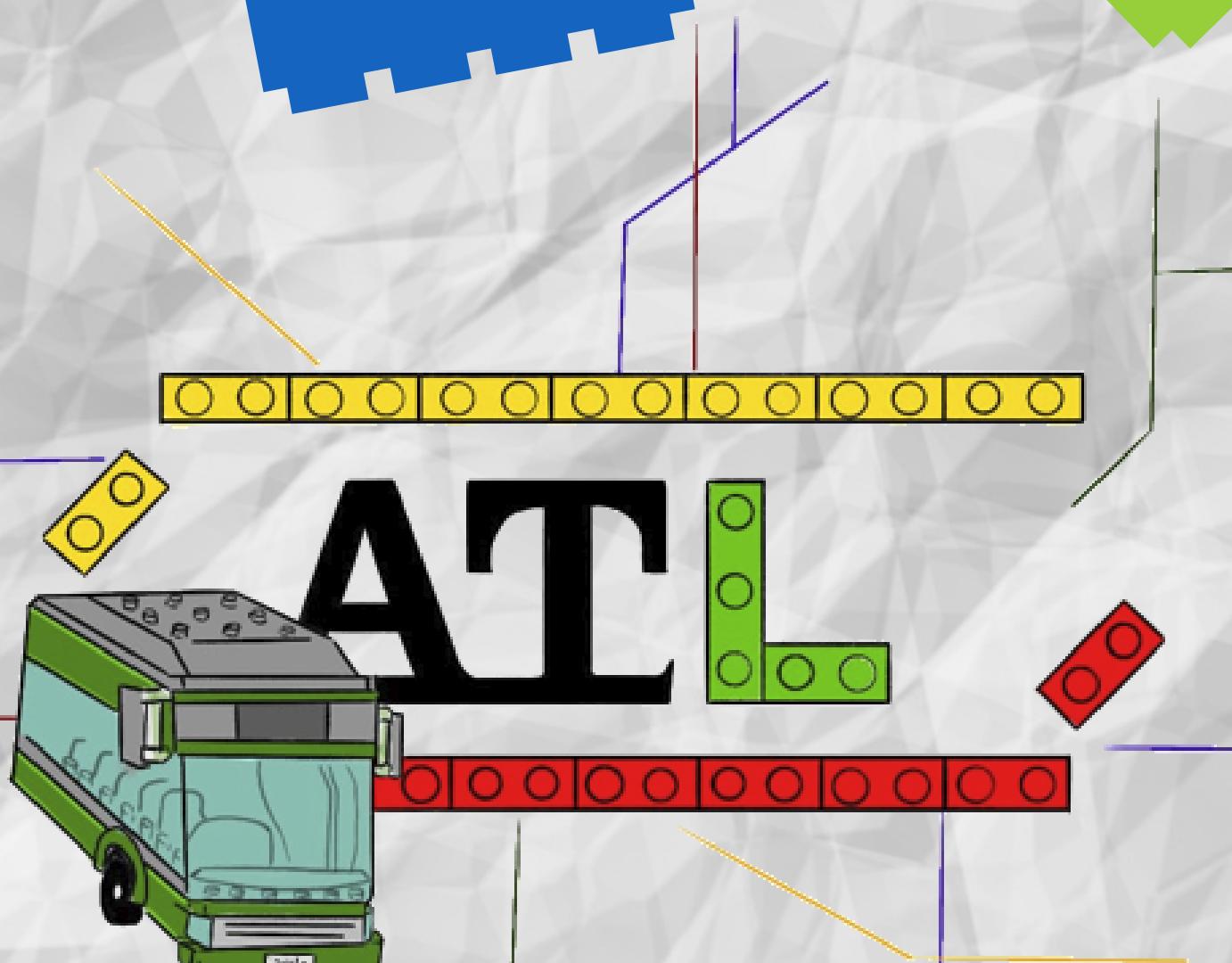
Programmazione Ad Oggetti  
Ingegneria Del Software  
A.A.2022/2023





## Progetto-I23

Sergiu Francisc  
Paolo Lucchese  
Ananda De Freitas  
Federico Spataro  
Dora Konlack

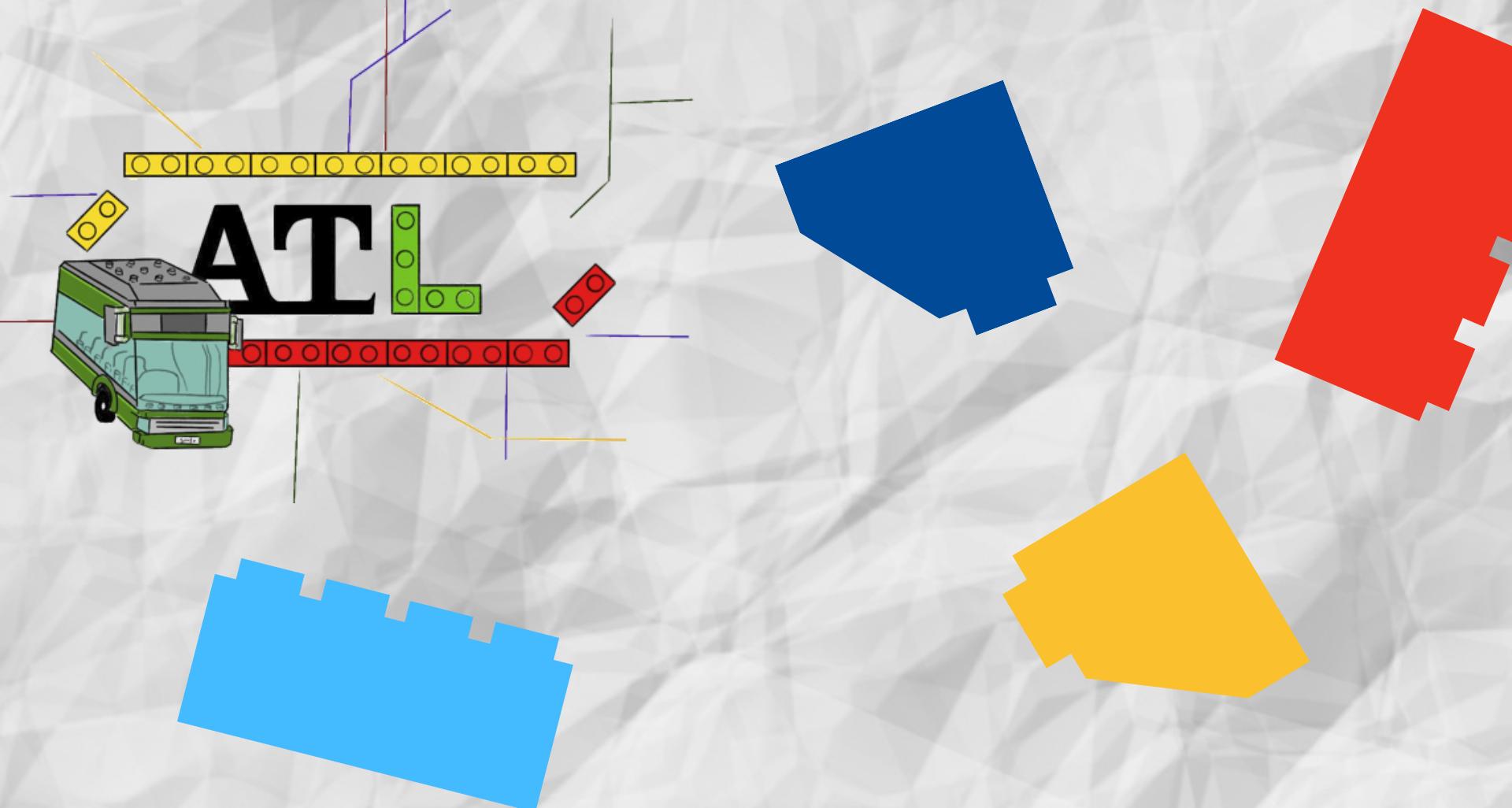


# Introduzione

Sistema informativo che attua la gestione di un'azienda di trasporti.

In particolare, sviluppa le seguenti features:

- Acquisto biglietti/abbonamenti
- Timbratura biglietti
- Post news
- Reclami



## Processo Software

- UP: Unified Process



### Iterazioni

Fase 1: ideazione e elaborazione del piano del progetto

Fase 2: elaborazione della documentazione del progetto e identificazione dei requisiti

Fase 3: Implementazione e sviluppo del software

Fase 4: Collaudo e Testing del sistema

## Requisiti Funzionali

### Dipendente

- Autenticazione al Servizio
- Il Responsabile può effettuare la registrazione degli Impiegati
- Pubblicare News
- Rispondere ai Reclami

### Utente

- Autenticazione al servizio
- Acquistare e visualizzare Biglietti/Abbonamenti
- Timbrare Biglietti
- Visualizzare le News
- Postare Reclami



## Requisiti Non Funzionali

### Tecnici

- Linguaggi : Java 8, SQL
- GUI: Framework SWING

### Organizzativi

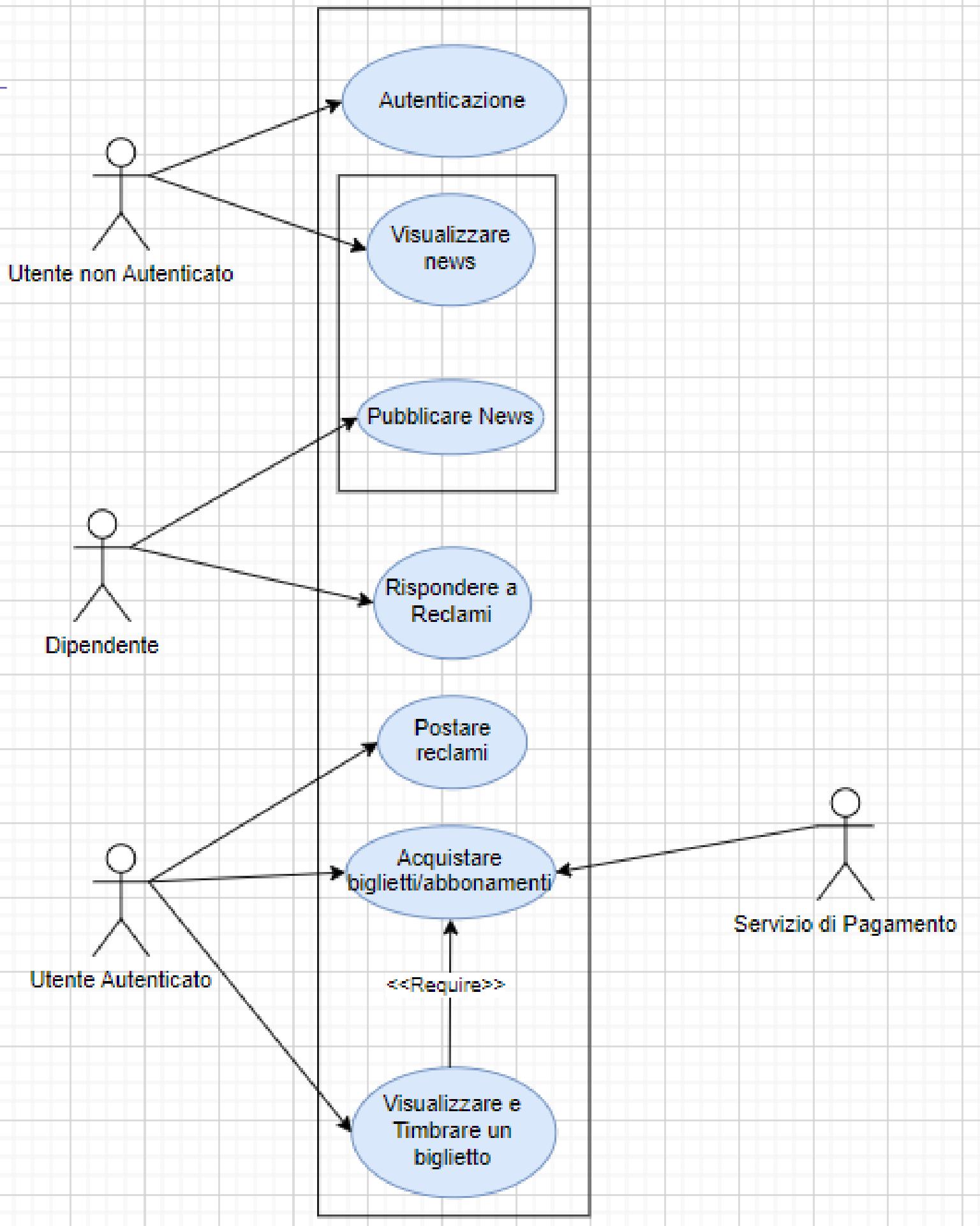
- Termini per la consegna del sistema: 15 febbraio 2023
- Documentazione scritta in lingua italiana ad eccezione del codice

### Esterni

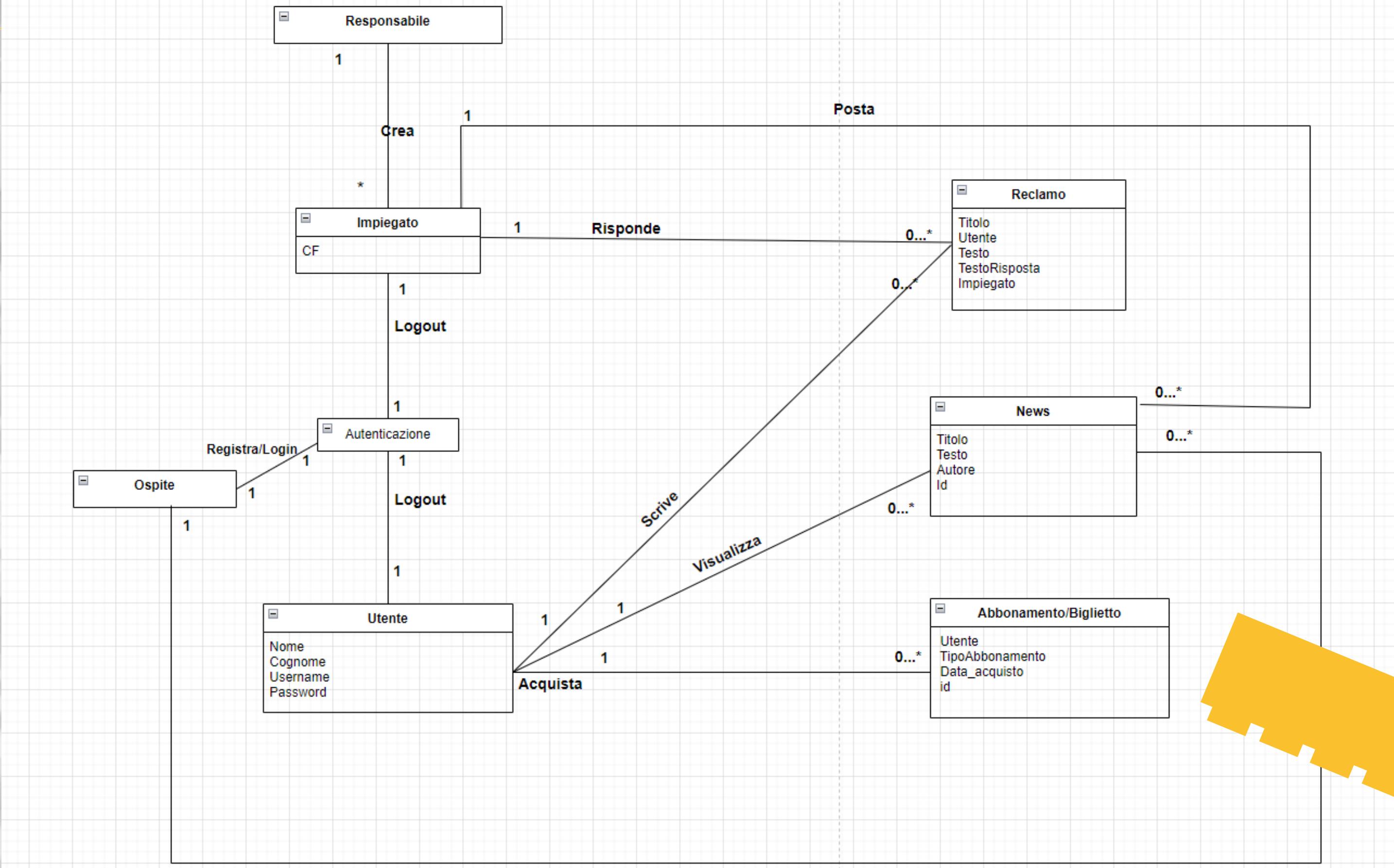
- Autorizzazione al trattamento dei dati personali
- Sistema Bancario Esterno effettua l'addebito

## Casi D'Uso

- UC1: Acquisto Biglietti/Abbonamento
- UC2: Visualizzazione e Timbratura Biglietti
- UC3: Pubblicazione e Visualizzazione News
- UC4: Postare Reclami
- UC5: Rispondere ai Reclami
- UC6: Autenticazione

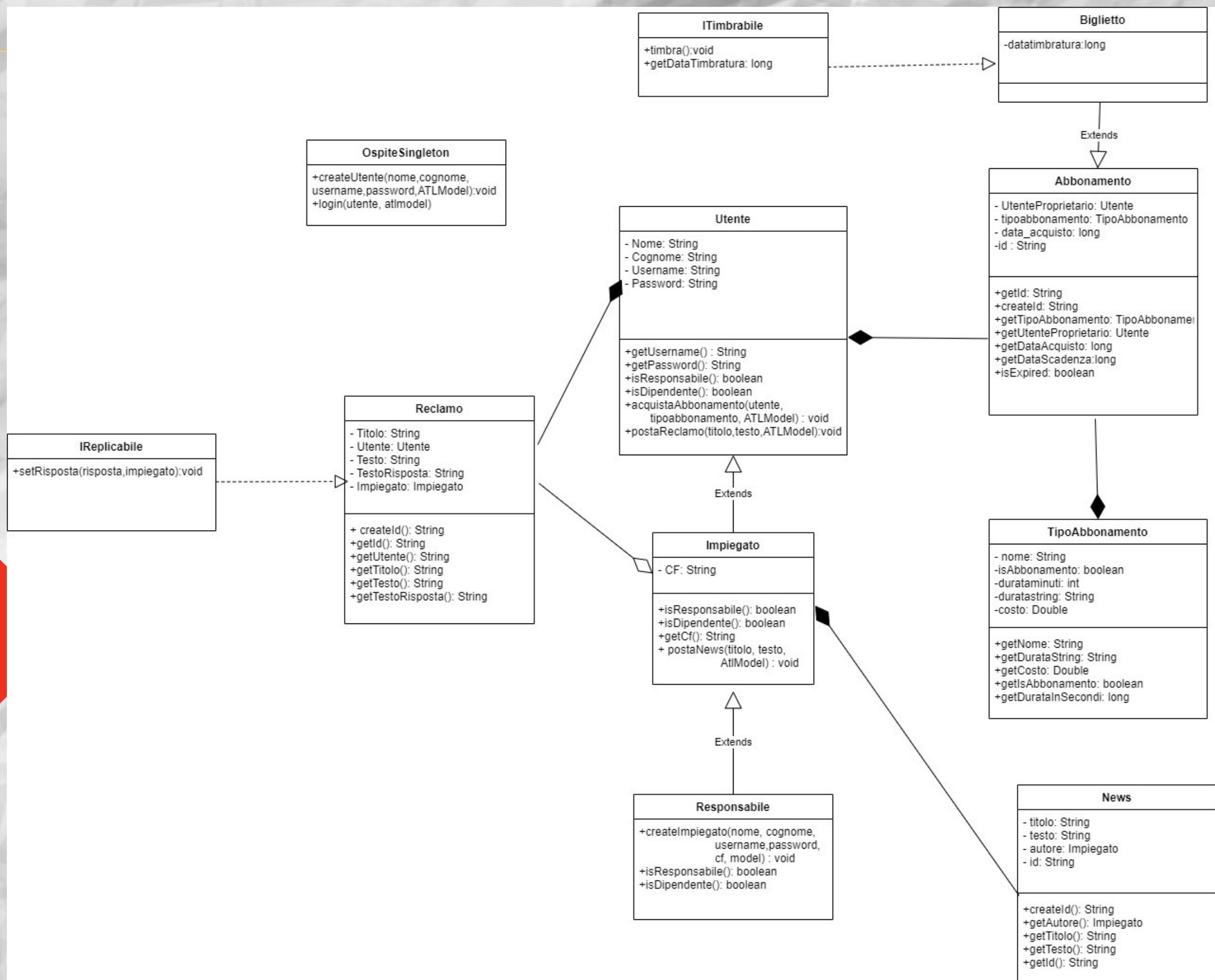


# Modello Di Dominio

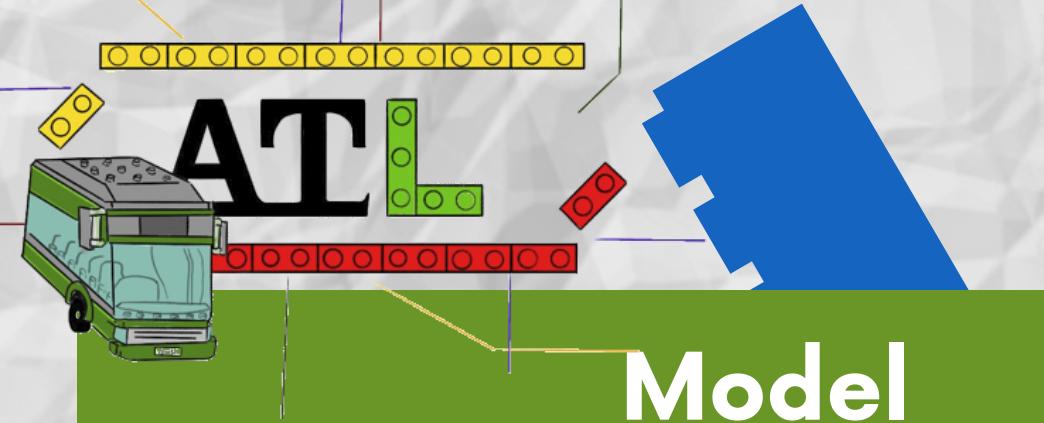


Visualizza

# UML Classi



Flusso Di  
Progettazione



```
▼ it.unipv.ingsfw.ispafd.atl.model
  ▶ ATLModel.java
▼ it.unipv.ingsfw.ispafd.atl.model.abbonamenti
  ▶ Abbonamento.java
  ▶ Biglietto.java
  ▶ TipoAbbonamento.java
▼ it.unipv.ingsfw.ispafd.atl.model.news
  ▶ News.java
▼ it.unipv.ingsfw.ispafd.atl.model.reclami
  ▶ Reclamo.java
▼ it.unipv.ingsfw.ispafd.atl.model.utenti
  ▶ Impiegato.java
  ▶ Ospite.java
  ▶ Responsabile.java
  ▶ Utente.java
```

## View

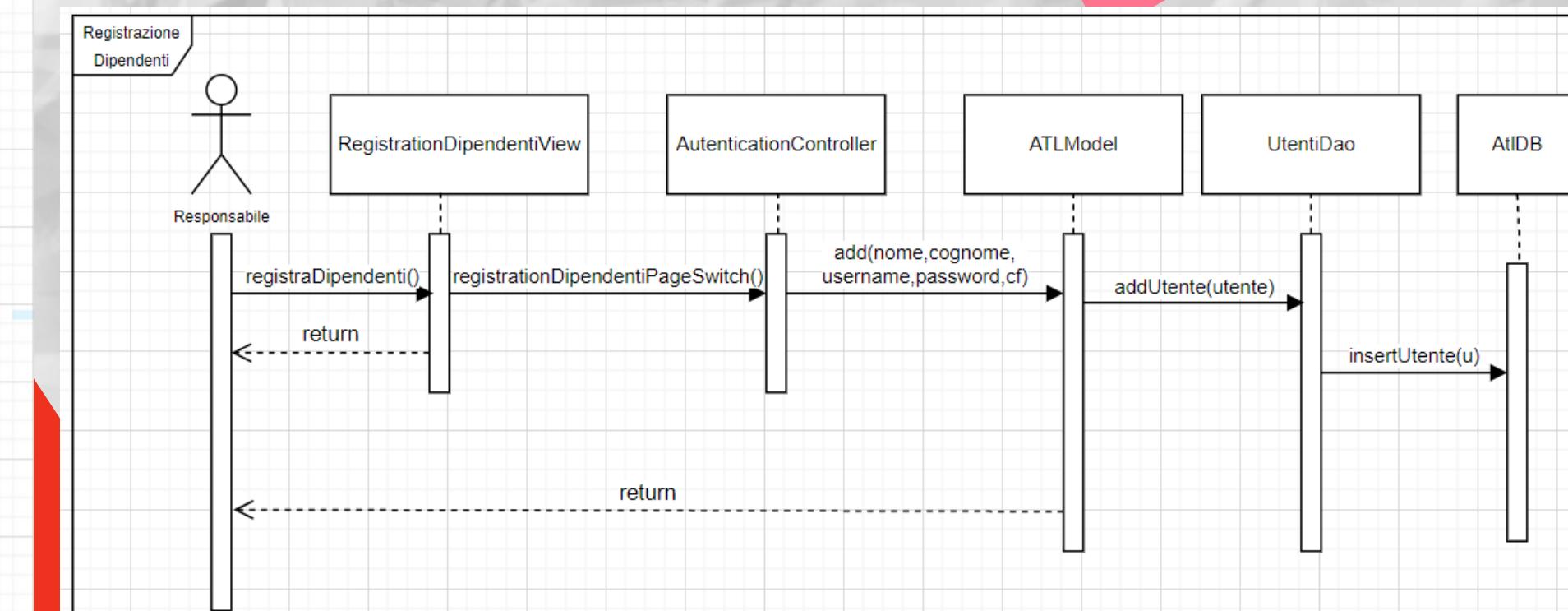
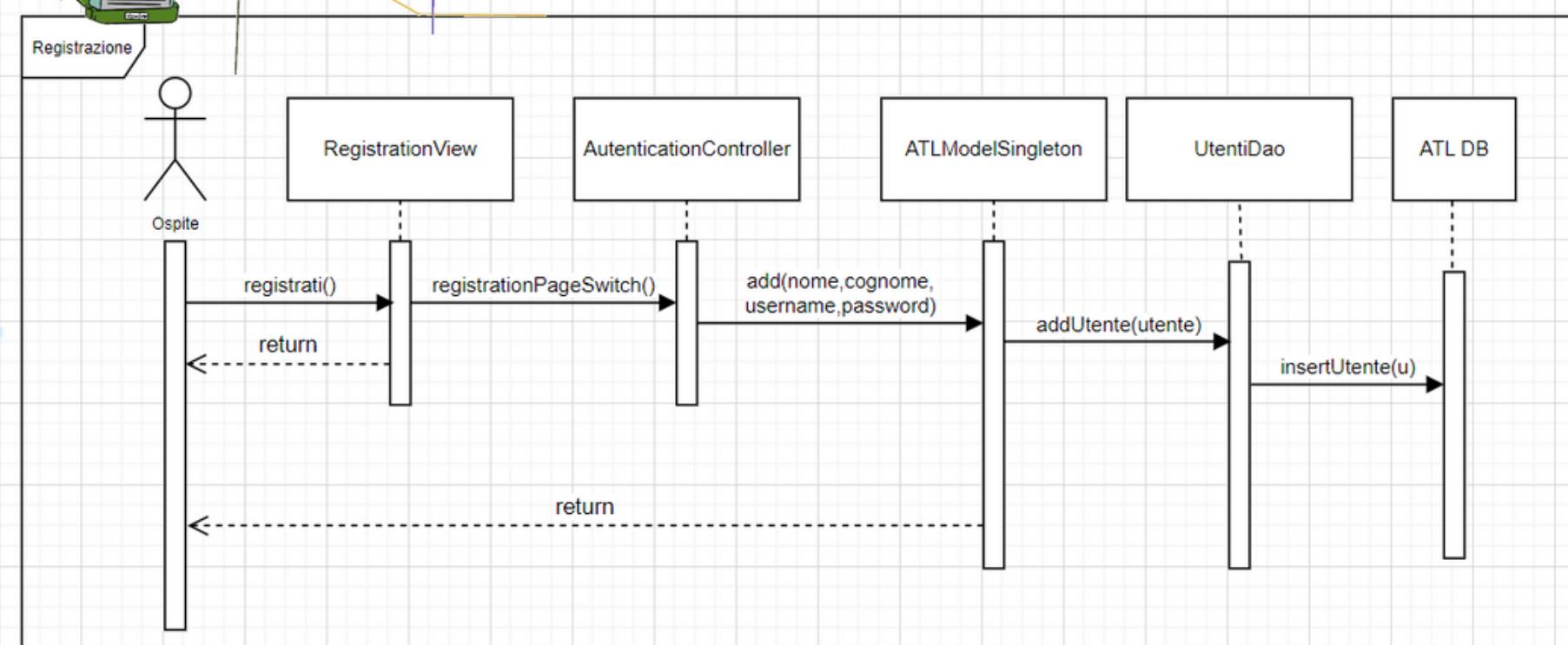
```
▼ it.unipv.ingsfw.ispafd.atl.view
  ▶ AbstractView.java
  ▶ AcquistaBigliettiView.java
  ▶ ButtonCustom.java
  ▶ ListaBigliettiView.java
  ▶ ListaReclamiView.java
  ▶ MainFrame.java
  ▶ MainView.java
  ▶ NewsListView.java
  ▶ Resettabile.java
  ▶ SingoloReclamoView.java
▼ it.unipv.ingsfw.ispafd.atl.view.form
  ▶ AbstractFormView.java
  ▶ FormViewInterface.java
  ▶ LoginView.java
  ▶ PostaNewsView.java
  ▶ PostaReclamiView.java
  ▶ RegistrationDipendentiView.java
  ▶ RegistrationView.java
```

## Controller

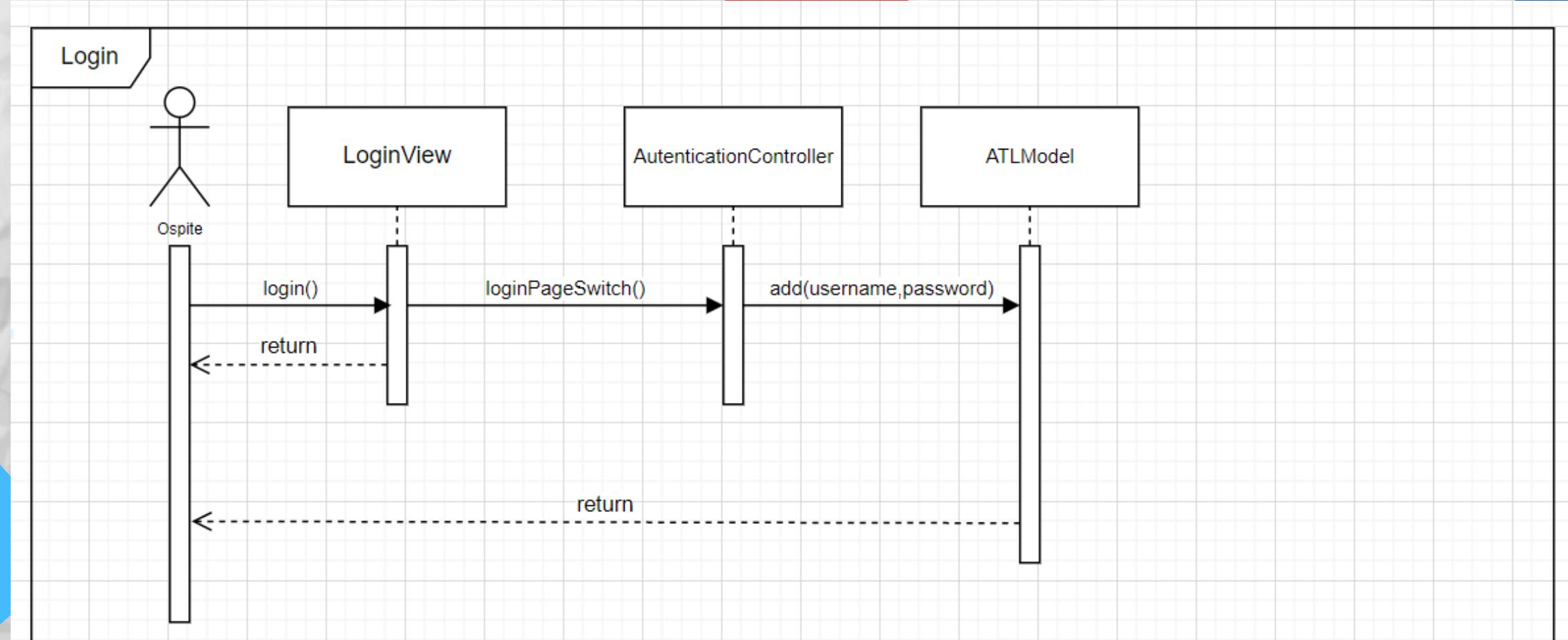
```
it.unipv.ingsfw.ispafd.atl.controller
  ▶ AutenticationController.java
  ▶ BigliettiController.java
  ▶ MVCCController.java
  ▶ NewsController.java
  ▶ ReclamiController.java
```

**Flusso Di  
Implementazione**

# Diagramma di Sequenza Autenticazione

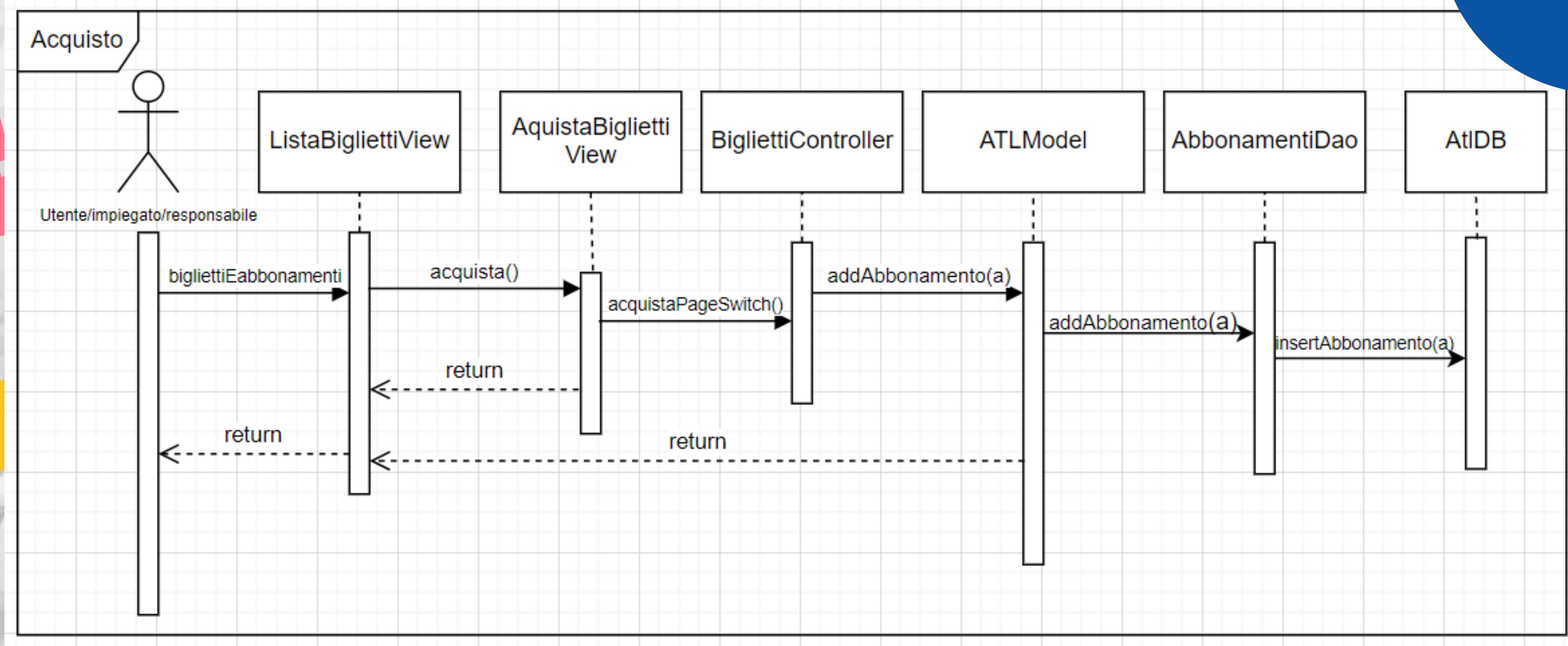


**Flusso Di  
Progettazione**



## Flusso Di Progettazione

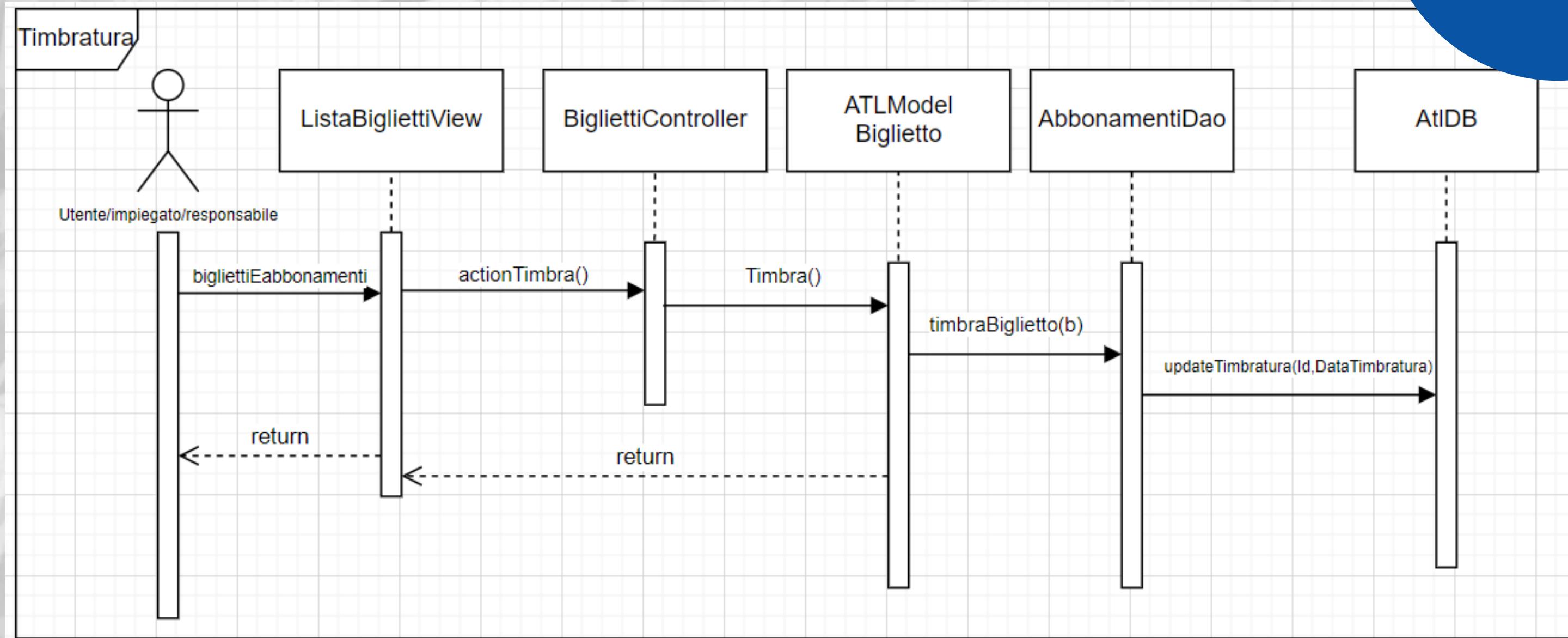
### Diagramma di Sequenza Acquisto Biglietti e Abbonamento



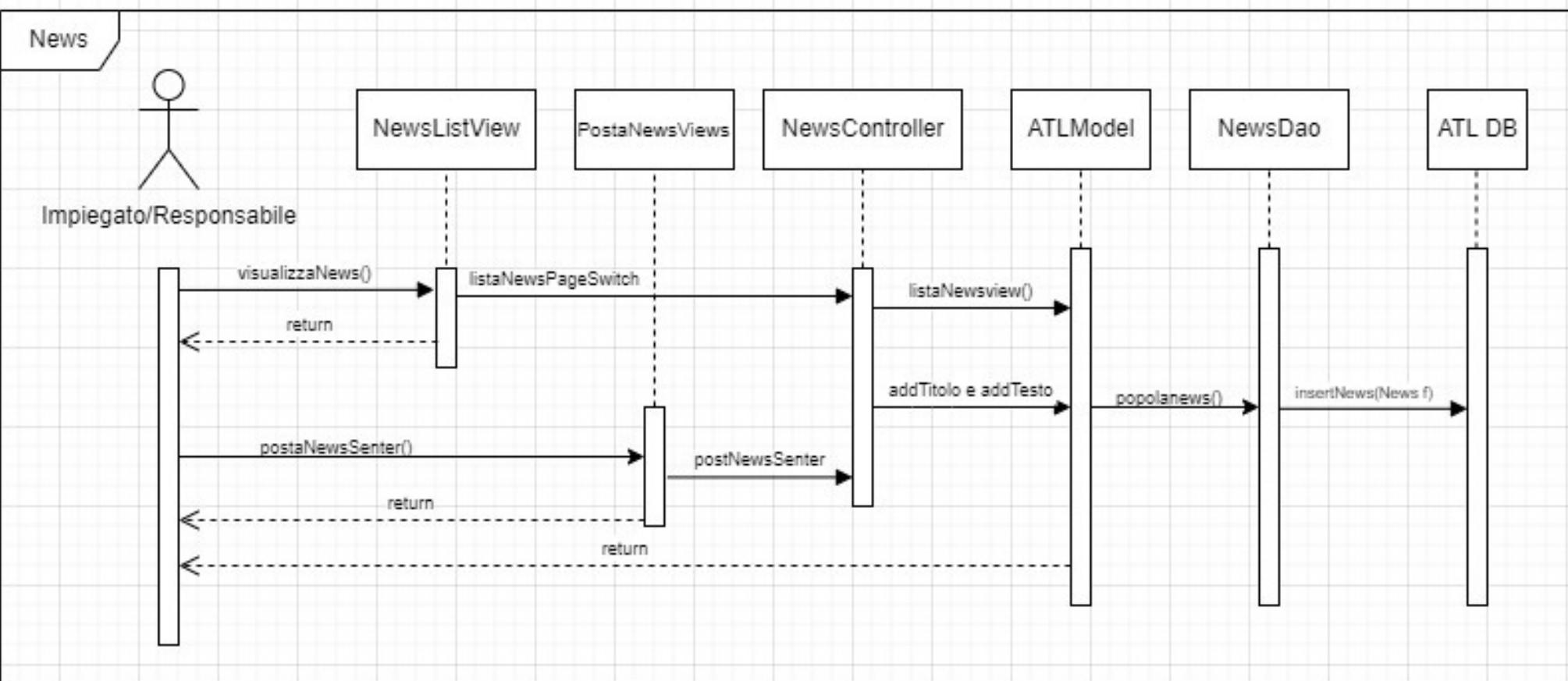
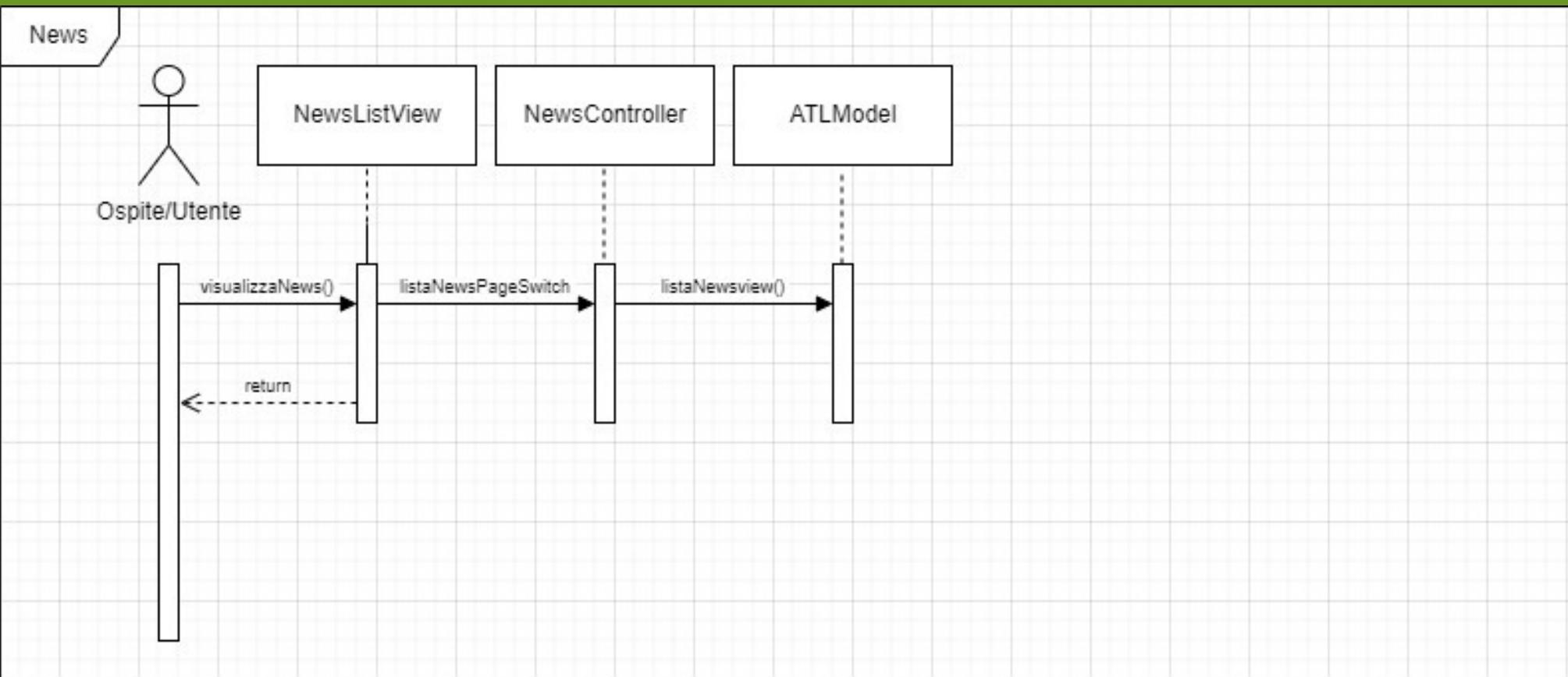
# Diagramma di Sequenza

## Timbratura

Flusso Di  
Progettazione

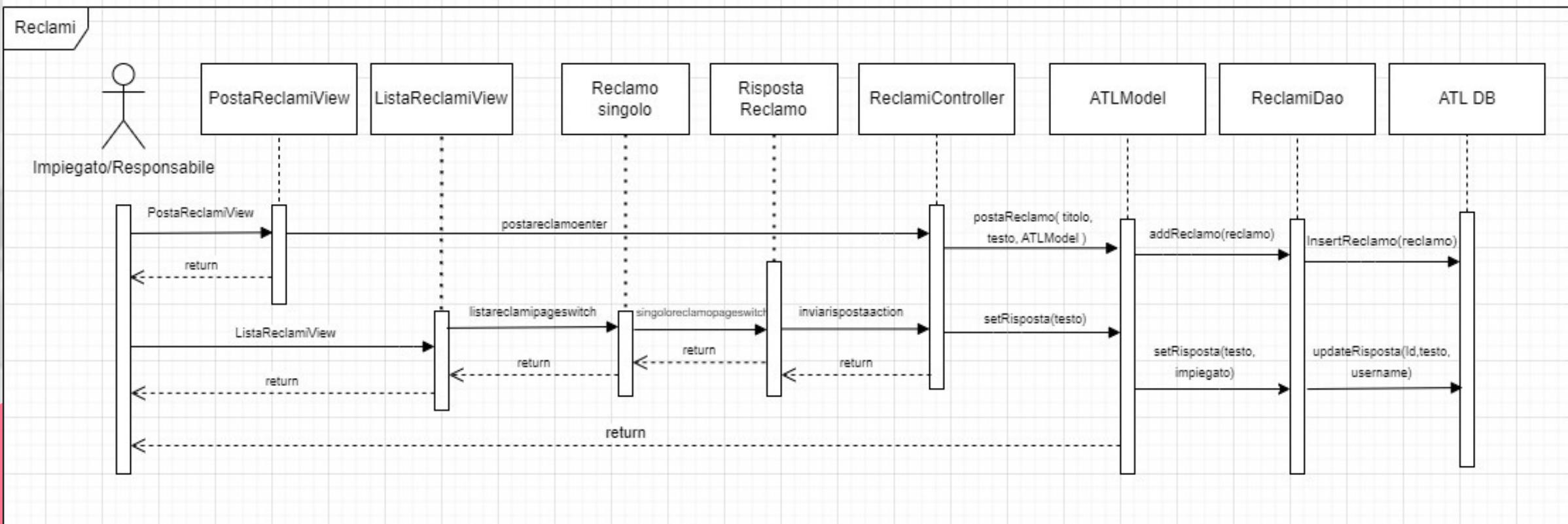
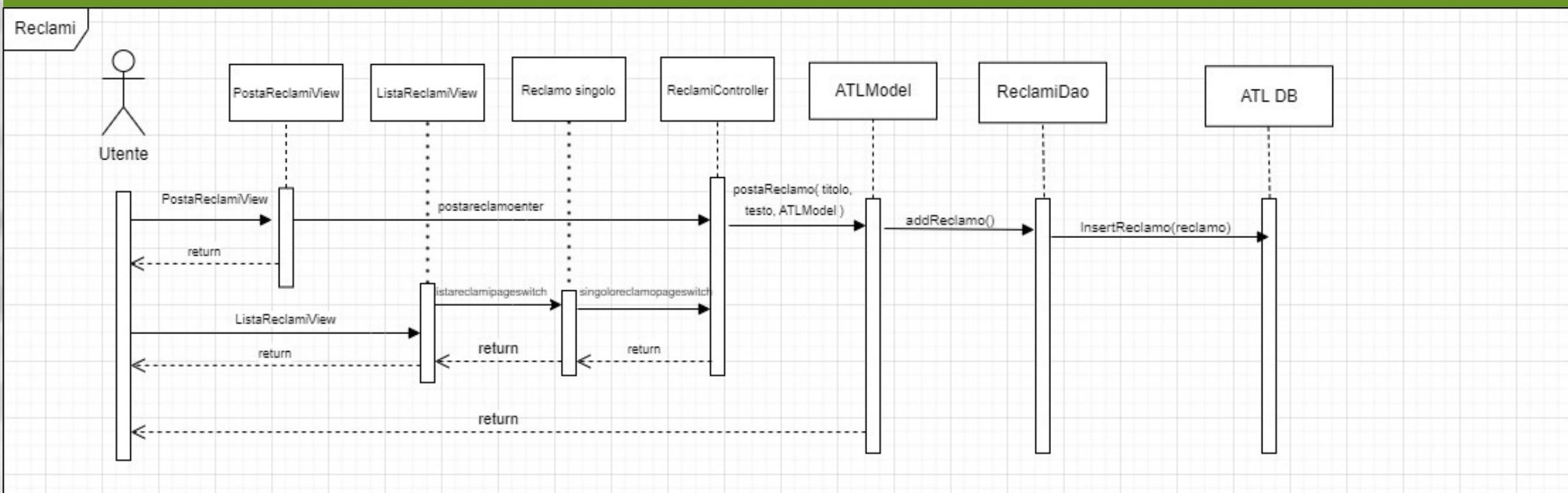


# Diagramma di Sequenza News



Flusso Di  
Progettazione

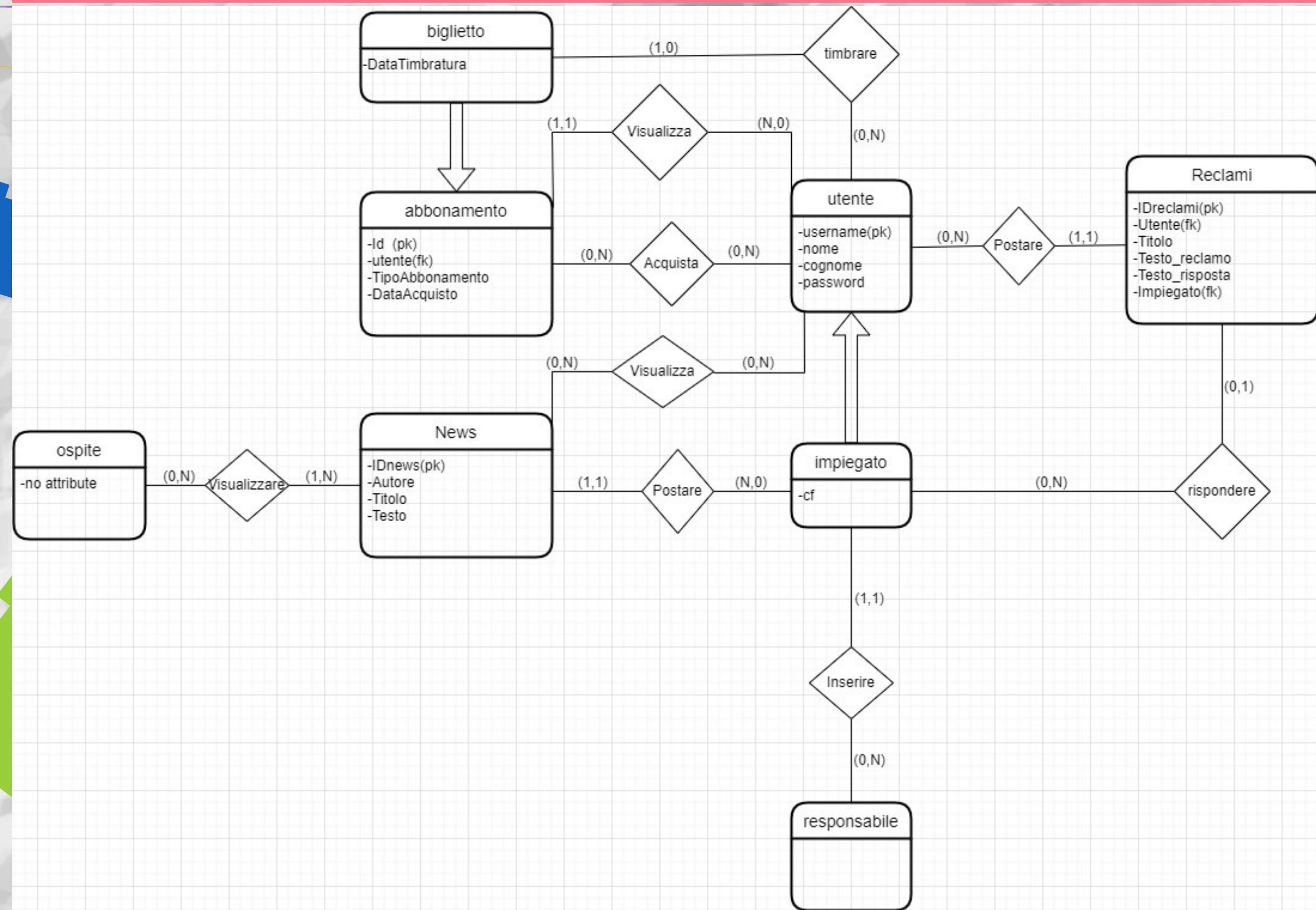
# Diagramma di Sequenza Reclami



Flusso Di  
Progettazione

# Flusso Di Progettazione

## Diagramma ERA





## Flusso Di Implementazione

### Pattern: DAO

- AbbonamentiDAO.java
- DBConnection.java
- NewsDAO.java
- ReclamiDAO.java
- TipoAbbonamentiDAO.java
- UtentiDAO.java

```
public class NewsDAO{  
  
    private String schema;  
    private Connection conn;  
  
    public NewsDAO() {  
        super();  
        this.schema = "atldb";  
        conn=DBConnection.startConnection(conn,schema);  
        //  
    }  
}
```

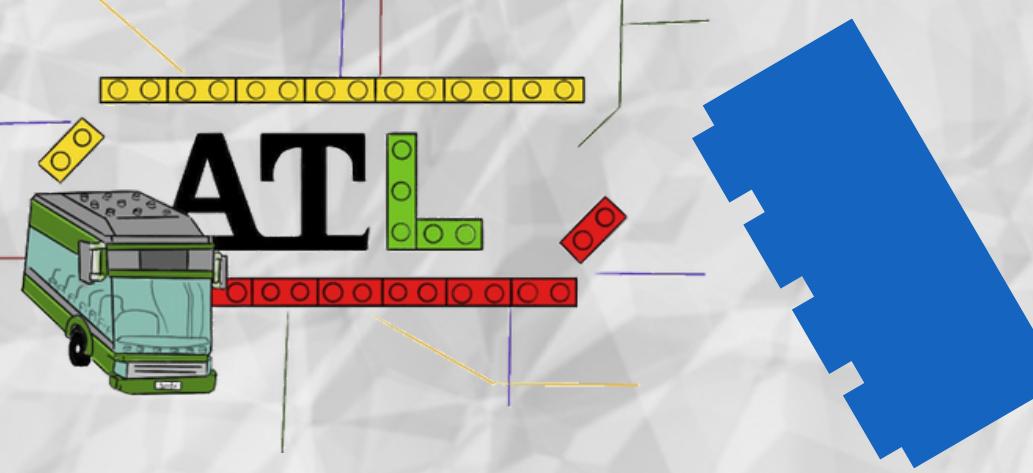
## Pattern: Singleton

- **OspiteSingleton**
- **ATLModelSingleton**

**Flusso Di  
Implementazione**

```
public class Ospite {  
  
    private static Ospite jospite;  
  
    private Ospite() {  
  
    }  
  
    public static Ospite getInstance() {  
  
        if(jospite==null) {  
            jospite = new Ospite();  
        }  
  
        return jospite;  
    }  
}
```

```
public static ATLModelSingleton getInstance() {  
  
    if(jATLModel==null) {  
        jATLModel = new ATLModelSingleton();  
    }  
  
    return jATLModel;  
}
```



## Pattern: Creator

- **OspiteCreator**
- **ResponsabileCreator**

Flusso Di  
Implementazione

```
public void createImpiegato(String nome, String cognome, String username, String password, String cf, ATLModelsingleton m) {  
    Impiegato itemp = new Impiegato(nome,cognome,username,password,cf);  
  
    m.addUtente(itemp);
```

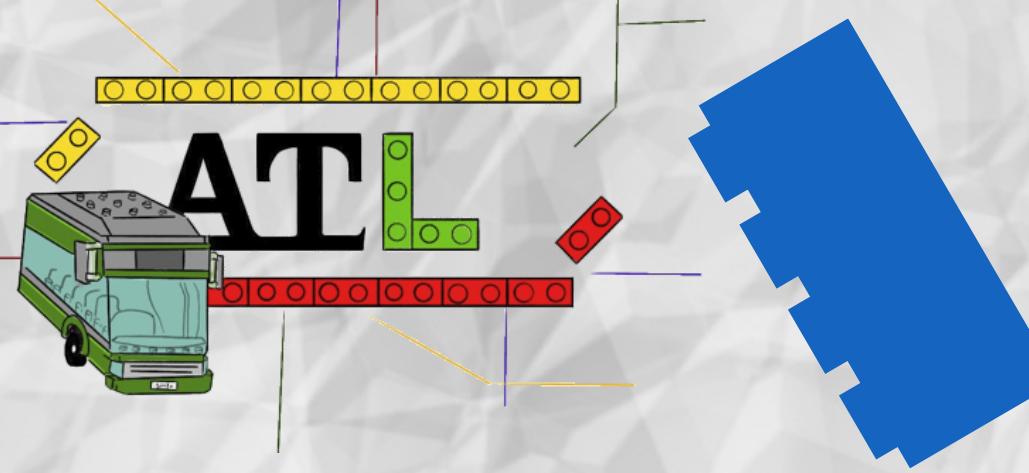
```
public void createUtente(String nome, String cognome, String username, String password, ATLModel m) {  
  
    Utente utemp = new Utente(nome,cognome,username,password);  
  
    m.addUtente(utemp);  
  
    //Pattern Creator: B registra A  
}
```

## Pattern: Facade

- **MVCCControllerFacade**
- **MainFrame**

**Flusso Di  
Implementazione**

```
public MVCCControllerFacade(MainFrame view, ATLModelsSingleton m) {  
  
    this.view = view;  
    this.m = m;  
    this.addListeners();  
}  
  
private void addListeners() {  
  
    AutenticationController.addListener(m, view);  
    NewsController.addListener(m, view);  
    BigliettiController.addListener(m, view);  
    ReclamiController.addListener(m, view);  
}  
}
```



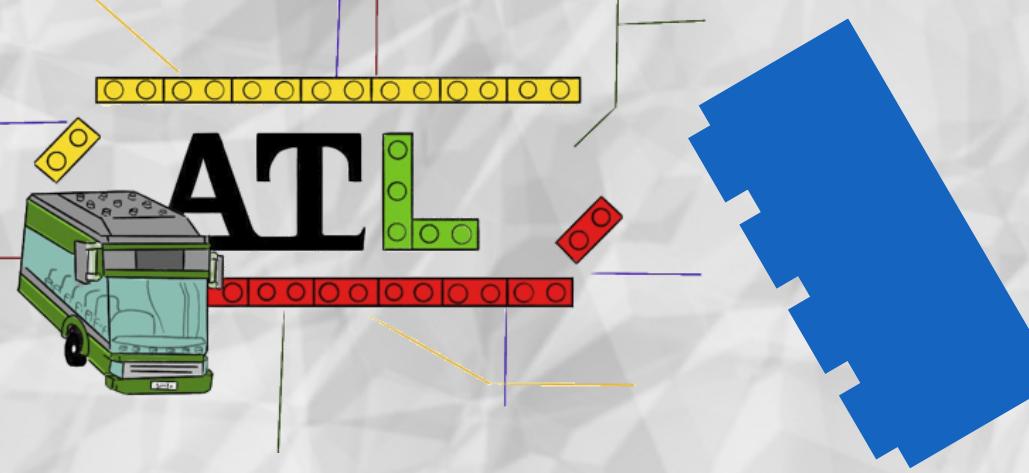
## Verifica e Convalida News

### Test della consistenza del modello

- **Verifica se gli autori  
delle news sono  
dipendenti**

```
@Test  
void AuthorOfNewsShouldBeImpiegato() {  
  
    ATLModelSingleton m = ATLModelSingleton.getInstance();  
  
    ArrayList<News> newslist = m.getNewsArray();  
  
    for(News n: newslist) {  
        assertTrue(n.getAutore().isDipendente());  
    }  
}
```

Flusso Di Test



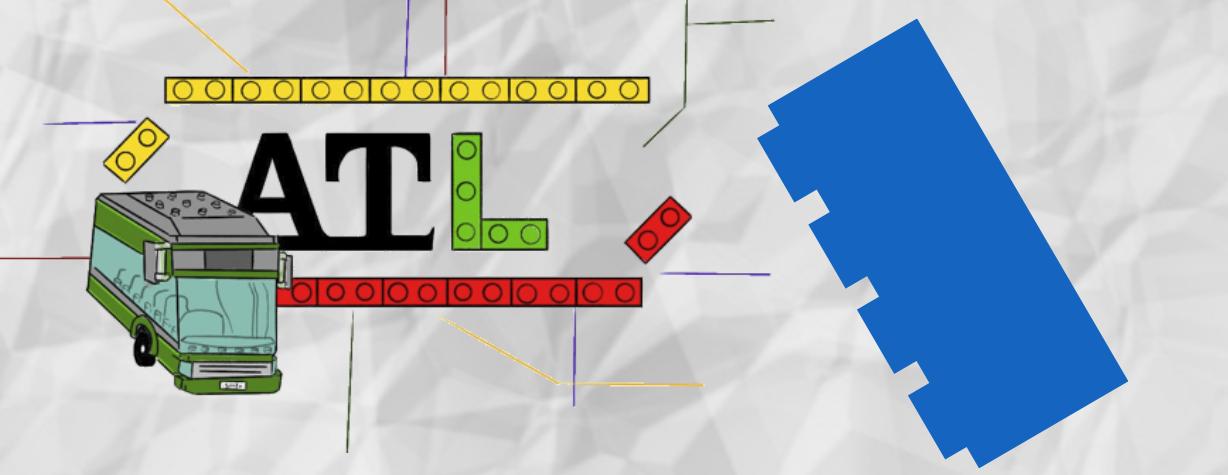
## Verifica e Convalida Reclamo

### Test della consistenza del modello

- **Verifica se chi risponde a un reclamo è un dipendente**

```
@Test  
void AuthorofRispostaReclamoShouldBeImpiegato() {  
  
    ATLModelSingleton m = ATLModelSingleton.getInstance();  
  
    ArrayList<Reclamo> reclamelist = m.getReclamiArray();  
  
    for(Reclamo r: reclamelist) {  
        if(r.getImpiegato()!=null) {  
            assertTrue(r.getImpiegato().isDipendente());  
        }  
    }  
}
```

Flusso Di Test



## Verifica e Convalida Autenticazione

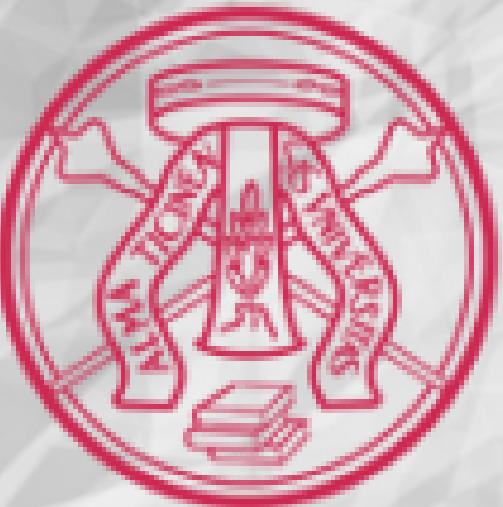
### Test ritorno dei Metodi

- **Verifica se registrando due utenti con lo stesso username, viene lanciata un eccezione**

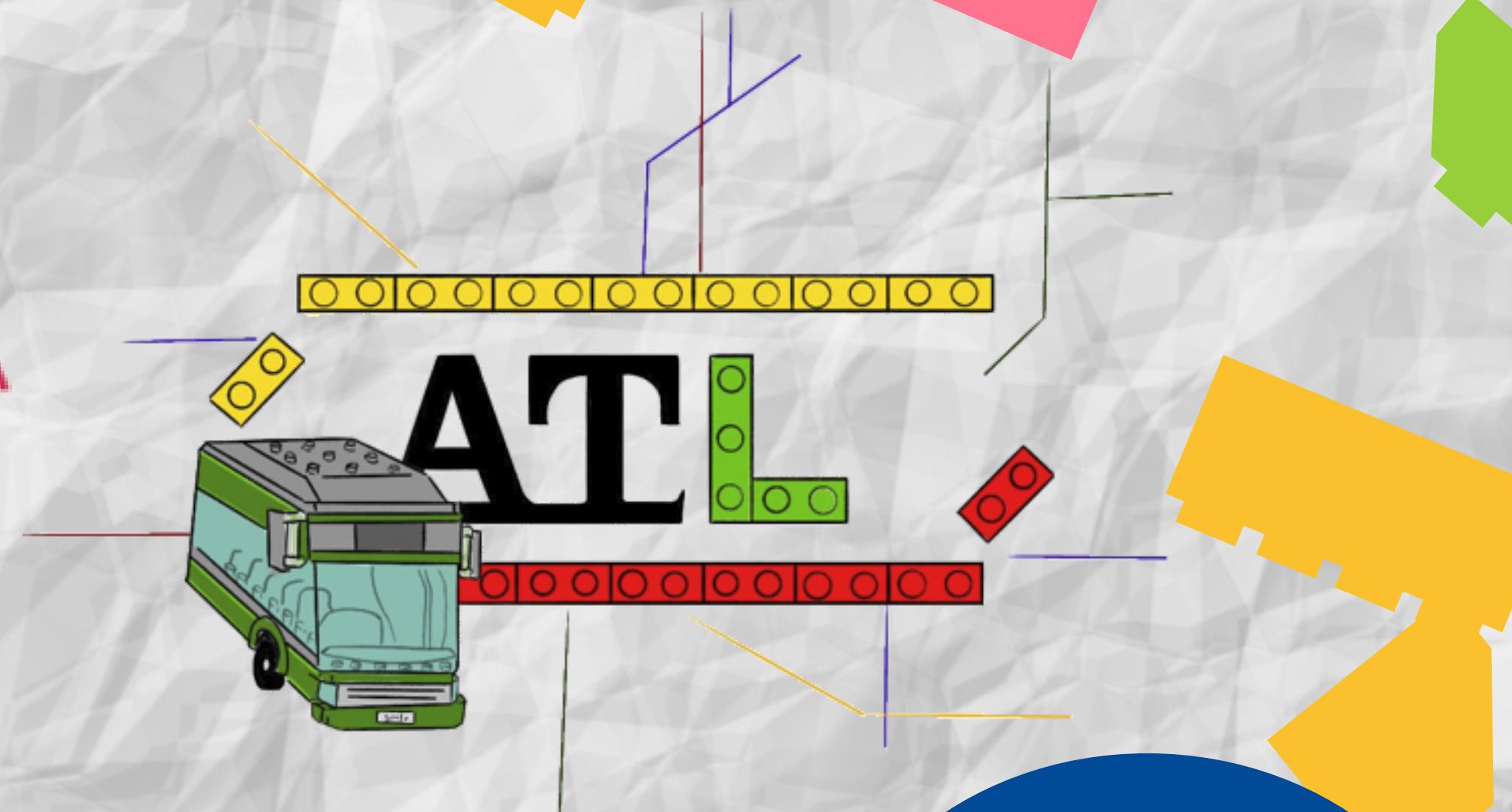
```
@Test  
void NoDuplicatedUsernames() {  
  
    ATLModelsSingleton m = ATLModelsSingleton.getInstance();  
  
    Long timestamp = System.currentTimeMillis();  
  
    String username= timestamp.toString();  
  
    m.addUtente(new Utente("test","test",username,"test")); //forcing a new user without passing from Ospite  
  
    OspiteSingleton o = OspiteSingleton.getInstance();  
  
    ArrayList<String> parameters = new ArrayList<String>();  
    parameters.add("test2"); parameters.add("test2"); parameters.add(username); parameters.add("test2");  
  
    assertThrows(AlreadyExistingUsernameException.class,  
        () -> {  
            o.creaUtente(parameters, m);  
        }  
    );  
}
```



Flusso Di Test



UNIVERSITÀ  
DI PAVIA



**GRAZIE  
DELL'ATTENZIONE !**