Object Design Document (ODD)

MeerKat

Object Design Document

Versione 0.1.4

Coordinatore del progetto:

Nome	Matricola

Revision History

Data	Versione	Descrizione	Autore
28/11/2024	0.1	Scrittura indice.	Stefano Nicolò Zito , Francesco Giuseppe Trotta, Giuseppe Ballacchino, Gabriel Tabasco
2/12/2024	0.1.1	Introduzione	Gabriel Tabasco
10/12/2024	0.1.2	Sezione Pacchetti	Giuseppe Ballacchino,Gabriel Tabasco
11/12/2024	0.1.3	Interfacce delle classi	Francesco Giuseppe Trotta, Stefano Nicolò Zito
14/02/2025	0.1.4	Final revision	Stefano Nicolò , Francesco Giuseppe Trotta, Giuseppe Ballacchino, Gabriel Tabasco

Indice

1. Introduzione

1.1 Trade-off nel design degli oggetti

1.2 Linee guida per la documentazione delle interfacce

1.3 Definizioni e abbreviazioni

1.4 Riferimenti

2. Pacchetti

2.1 Descrizione generale

2.2 Dipendenze tra pacchetti

2.3 Organizzazione dei file

3. Interfacce delle classi

4. Design Pattern Autorizzati

---

1. Introduzione

1.1 Trade-off nel design degli oggetti

- Descrizione delle decisioni principali prese nel design:
  - Memoria vs Tempo di risposta: Si punta a metodi di accesso che prioritizzino velocità di risposta rispetto alla quantità di memoria utilizzata.
  - Costruire vs Comprare: *Utilizzo di strutture dati e framework esistenti dove possibile.*

1.2 Linee guida per la documentazione delle interfacce

- Convenzioni:
  - **Nomi delle classi:** Singolare (es. `User`, `ChatRoom`).
  - **Nomi dei metodi:** Frasi verbali (es. `getNome`, `inviaMessaggio`).
  - **Eccezioni:** Restituite tramite meccanismi dedicati.

1.3 Definizioni e abbreviazioni

- **UUID:** Identificatore univoco universale.
- **Byte array:** Rappresentazione binaria di dati (es. immagini).
- **DateTime:** Tipo di dato per rappresentare date e orari.

1.4 Riferimenti

- Specifiche dei requisiti (RAD).
- Diagrammi UML correlati.

---

2. Pacchetti

2.1 Descrizione generale

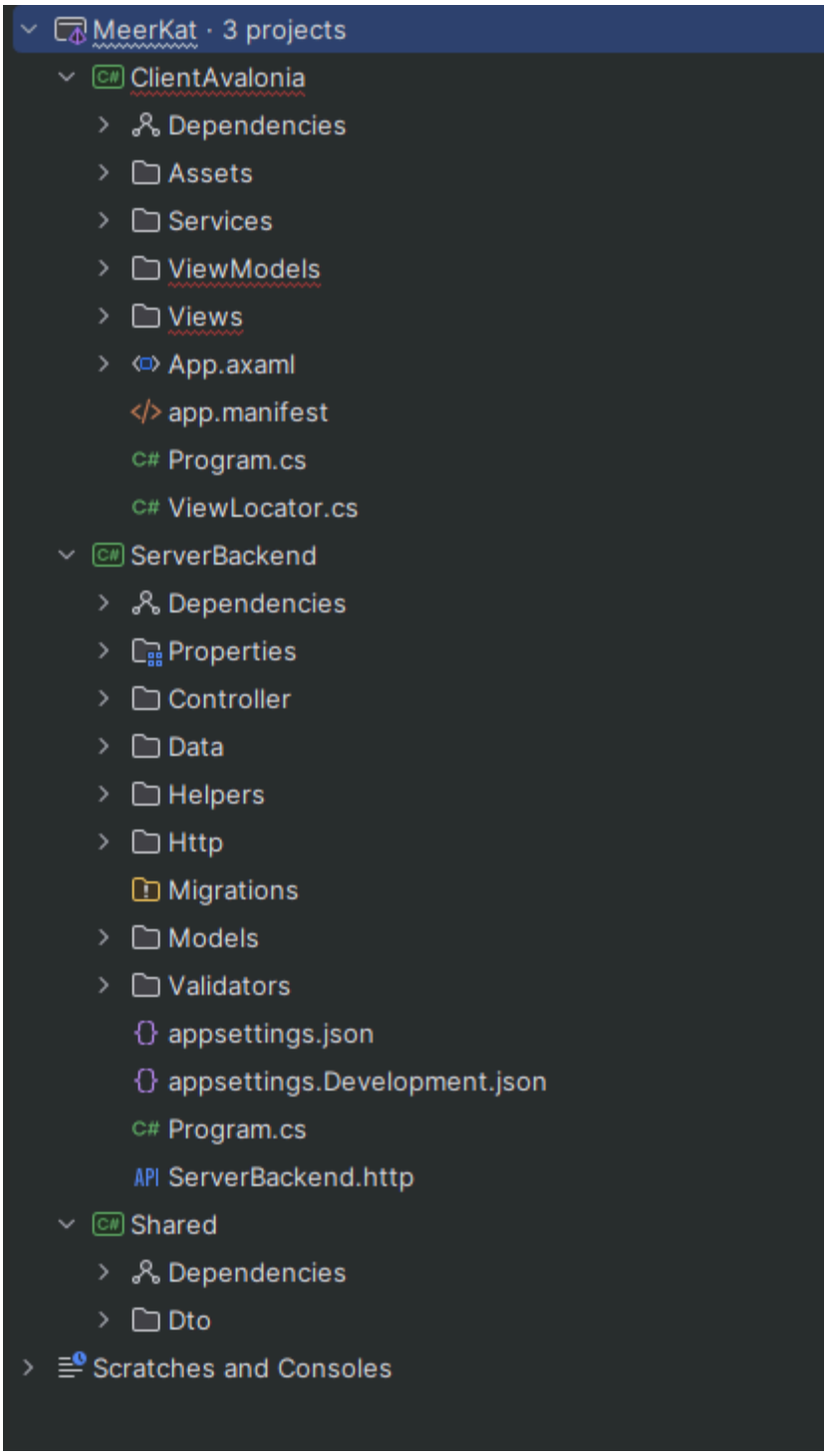
Nel progetto sono presenti tre pacchetti inerenti al pattern MVVM utilizzato per la creazione del Sistema Software. I pacchetti sono:

- **Models:** Models è l'implementazione del modello di dominio, che comprende la logica di business e di validazione (e.g. DTO).
- **ViewModels:** Intermerdiario tra **Views** e **Models**, è responsabile della gestione della logica delle interfacce grafiche, effettuando il binding dei dati gli eventi della View e i metodi del Model.
- **Views:** Responsabile della definizione della struttura, il layout e l'aspetto di ciò che l'utente vede sullo schermo.

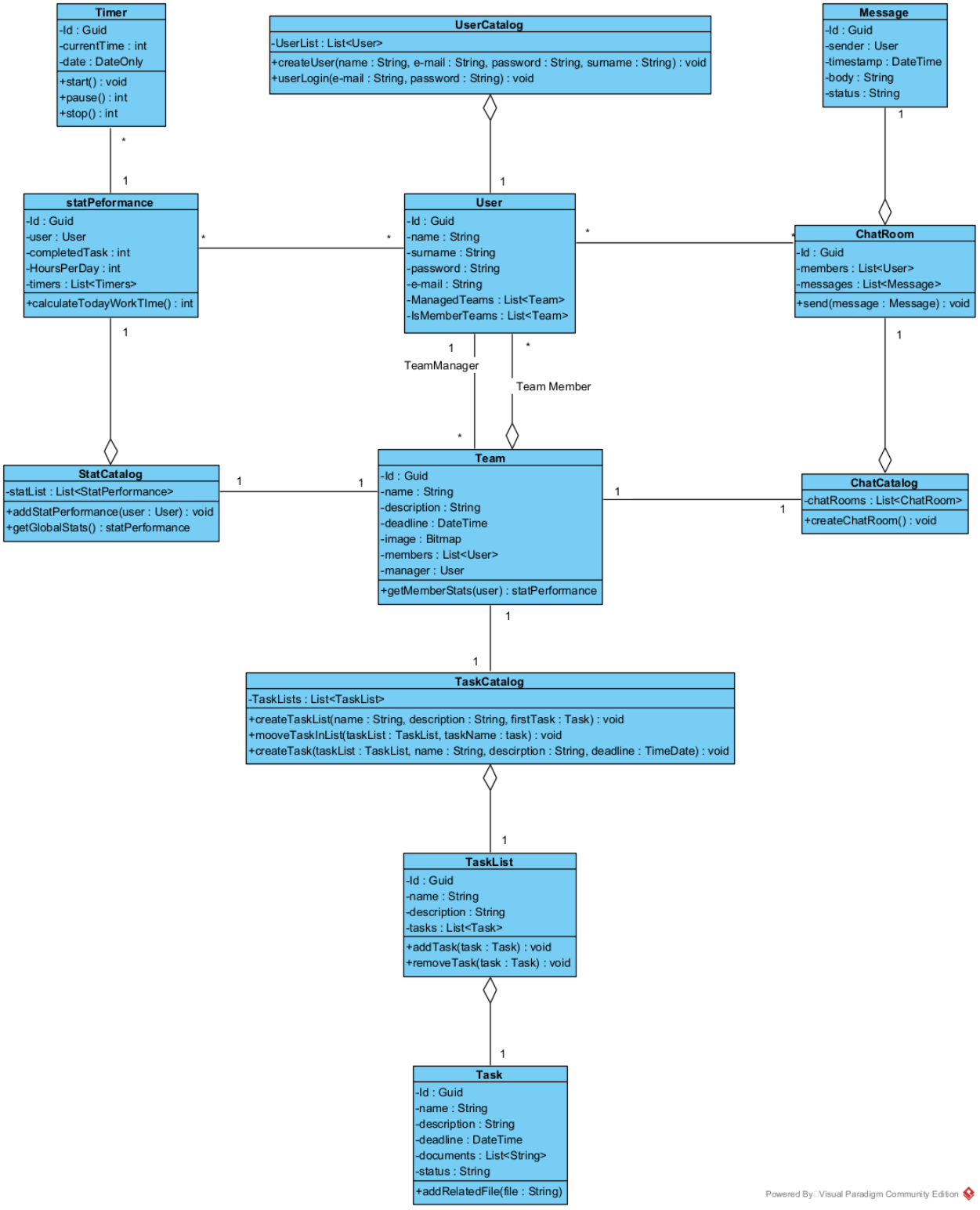
Invece i package generali del progetto sono:

- **ClientAvalonia**
  - Componenti dell'interfaccia grafica, UI e gestione della UI
- **ServerBackend**
  - Componenti della gestione del backend del progetto
- **Shared**
  - Componenti di comunicazione con il database

2.3 Struttura dei pacchetti



3. Diagramma delle classi aggiornato



4. Interfacce delle classi

4.1 Class: `User`

- **Attributi:**
  - `+id: Guid`
  - `+name: String`
  - `+surname: String`
  - `+password: String`
  - `+email: String`
  - `+managedTeams: List(Team)`
  - `+isMemberTeams: List(Team)`

4.2 Class: `Team`

- **Attributi:**
  - `+id: Guid`
  - `+name: String`
  - `+description: String`
  - `+deadline: DateTime`
  - `+image: Bitmap`
  - `+members: Lst(User)`
  - `+manager: User`
- **Metodi:**
  - `+getMemberStats(user: User): StatPerformance`

- `context Team::getMemberStats(user: User) : StatPerformance`
  - pre : `user <> null and user in self.members`
  - post : `result <> null and result.user = user`

4.3 Class: `TaskCatalog`

- **Attributi:**
  - `+taskLists: List(TaskList)`
- **Metodi:**
  - `+createTaskList(name: String, description: String, firstTask: Task): void`
  - `+moveTaskInList(taskList: TaskList, taskName: String): void`
  - `+createTask(taskList: TaskList, name: String, description: String, deadline: DateTime): void`

- `context TaskCatalog::createTaskList(name: String, description: String, firstTask: Task) : void`
  - pre : `name <> null and description <> null and firstTask <> null`
  - post : `self.taskLists->exists(tl | tl.name = name and tl.description = description and tl.tasks->includes(firstTask))`
- `context TaskCatalog::moveTaskInList(taskList: TaskList, taskName: String) : void`
  - pre : `taskList <> null and taskName <> null and taskList in self.taskLists`
  - post : `taskList.tasks->exists(t | t.name = taskName)`

- `context TaskCatalog::createTask(taskList: TaskList, name: String, description: String, deadline: DateTime) : void`
  - pre : `taskList <> null and name <> null and description <> null and deadline <> null`
  - post : `taskList.tasks->exists(t | t.name = name and t.description = description and t.deadline = deadline)`

4.4 Class: `TaskList`

- Attributi:
  - `+id: Guid`
  - `+name: String`
  - `+description: String`
  - `+tasks: List(Task)`

- Metodi:
  - `+addTask(task: Task): void`
  - `+removeTask(task: Task): void`

- `context TaskList::addTask(task: Task) : void`
  - pre : `task <> null`
  - post : `self.tasks→includes(task)`

- `context TaskList::removeTask(task: Task) : void`
  - pre : `task <> null and task in self.tasks`
  - post : `not self.tasks→includes(task)`

4.5 Class: `Task`

- **Attributi:**
  - `+id: Guid`
  - `+name: String`
  - `+description: String`
  - `+deadline: DateTime`
  - `+documents: List(String)`
  - `+status: String`
- **Metodi:**
  - `+addRelatedFile(file: String): void`

- `context Task::addRelatedFile(file: String) : void`
  - pre : `file <> null`
  - post : `self.documents→includes(file)`

4.6 Class: `Timer`

- Attributi:
  - `+id: Guid`
  - `+currentTime: DateTime`
  - `+currentDate: DateTime`
- Metodi:
  - `+start(): void`
  - `+pause(): void`
  - `+stop(): void`

- `context Timer::start() : void`
  - pre : `self.currentTime = null and self.currentDate = null`
  - post : `self.currentTime <> null and self.currentDate <> null`

- `context Timer::pause() : void`
  - pre : `self.currentTime <> null and self.currentDate <> null`
  - post : `self.currentTime <> null and self.currentDate <> null`

- `context Timer::stop() : void`
  - pre : `self.currentTime <> null and self.currentDate <> null`
  - post : `self.currentTime = null and self.currentDate = null`

4.7 Class: `UserCatalog`

- Attributi:

- `+userList: List(User)`

- Metodi:
  - `+createUser(name: String, email: String, password: String, surname: String): User`
  - `+userLogin(email: String, password: String): Boolean`

- `context UserCatalog::createUser(name: String, email: String, password: String, surname: String) : User`
  - pre : `name <> null and email <> null and password <> null and surname <> null`
  - post : `self.userList→includes(result)`

- `context UserCatalog::userLogin(email: String, password: String) : Boolean`
  - pre : `email <> null and password <> null`
  - post : `result = self.userList→exists(u | u.email = email and u.password = password)`

4.8 Class: `ChatRoom`

- Attributi:
  - `+id: Guid`
  - `+members: List(User)`
  - `+messages: List(Message)`
- Metodi:
  - `+send(message: Message): void`

- `context ChatRoom::send(message: Message) : void`
  - pre : `message <> null and message.sender in self.members`
  - post : `self.messages→includes(message)`

12. Class: `ChatCatalog`

- Attributi:
  - `+chatRooms: List(ChatRoom)`
- Metodi:
  - `+createChatRoom(name: String): ChatRoom`

    - `context ChatCatalog::createChatRoom(name: String) : ChatRoom`
      - **pre :** **`name <> null and name <> ""`**
      - **post :** **`self.chatRooms→includes(result) and result.name = name`**

4.9 Class: `Message`

- Attributi:
  - `+id: Guid`
  - `+sender: User`
  - `+timestamp: DateTime`
  - `+body: String`
  - `+status: String`

5.0 Class: `StatPerformance`

- Attributi:
  - `+id: Guid`
  - `+user: User`
  - `+completedTask: Integer`
  - `+hoursPerDay: Integer`
  - `+timers: List(Timer)`
- Metodi:

- `+calculateTodayWorkTime(): Integer`

- `context StatPerformance::calculateTodayWorkTime() : Integer`
  - pre : `self.timers <> null`
  - post : `result >= 0`

5.1 Class: `StatCatalog`

- Attributi:
  - `+statList: List(StatPerformance)`
- Metodi:
  - `+addStatPerformance(user: User): void`
  - `+getGlobalStats(): StatPerformance`

- `context StatCatalog::addStatPerformance(user: User) : void`
  - pre : `user <> null`
  - post : `self.statList→exists(s | s.user = user)`

- `context StatCatalog::getGlobalStats() : StatPerformance`
  - pre : `self.statList <> null`
  - post : `result <> null`

---

4. Design Pattern Utilizzati

Singleton

Il pattern singleton consente di creare una classe che esiste come unica istanza e a cui e possibili accedere a livello globale. Nel progetto Meerkat e stato applicato in classi come ad esempio i Catalog che richiedono un unica istanza, inquanto devono solo raccogliere istanze delle altre classi.

Façade

Il Façade pattern consiste nel creare un oggetto che fornisca un interfaccia per nascondere un interfaccia di codice più complessa. Questo pattern e stato utilizzato nel progetto inquanto dovendo gestire un sistema distribuito con client e server e stato necessario andare a creare delle classi al interno della parte client che facciano da interfaccia nascondendo l'implementazione che invece e avvenuta lato server.