Università degli Studi di Salerno

Dipartimento di Informatica

Anno Accademico 2020/2021

GameUp

Object Design Document

**Studenti: Docente:**

*Francesco Foglia Andrea De Lucia*

Cronologia Revisioni

|  |  |  |
| --- | --- | --- |
| Data | Versione | Descrizione |
| N/A | 1.0 | N/A |

1. Introduzione
   1. Compromessi di object design

Il compromesso principale di object design è dato dal poco tempo disponibile per lo sviluppo dell’implementazione core del sistema: per avere una base da cui partire, si sceglierà di usare componenti off-the-shelf laddove possibile, considerando anche l’assenza di un budget dedicato al sistema, sia per la parte relativa al back-end e allo sviluppo della logica di business, sia per lo sviluppo del front-end con le relative interfacce che verranno servite ai vari utenti del sistema.

* + 1. Componenti off-the-shelf

Per la progettazione del back end del sistema useremo *Laravel*, un framework dell’ecosistema del linguaggio di programmazione PHP il quale offre una base potente, fluida ed estremamente comoda nella quale sviluppare una applicazione Web, assieme a svariati componenti per semplificare qualsiasi particolarità si voglia implementare o gestire, come ad esempio *Laravel Sail* per la gestione delle condizioni limite del sistema (startup e terminazione), *Laravel Socialite* per la gestione dell’autenticazione di utenti tramite OAuth e molto altro.

In particolare, per l’implementazione del sistema verrà usata l’ultima attuale versione del framework stabile, ovvero la 8.5.8. I componenti che verranno usati saranno:

* Laravel Sail, per la gestione dello startup e della terminazione del sistema;
* *Eloquent*, come ORM per la comunicazione con il database;
* *Laravel-Admin*, per l’implementazione del pannello di amministrazione;
* *Laravel Blade*, *templating engine* principale di Laravel per l’implementazione del front end, ovvero di tutte le interfacce proposte dal sistema;
* *Laravel Cashier*, per interfacciarsi con il servizio esterno di gestione di pagamenti *Stripe*.

In particolare, Eloquent è stato scelto poiché è già integrato nel framework ed è proposto come ORM principale e per motivi simili sono stati scelt anche Laravel Blade e Laravel Cashier. Invece, Laravel-Admin è stato scelto poiché il componente principale suggerito dagli sviluppatori del framework prevede un costo non irrisorio ($99 per progetto), si è quindi preferito scegliere una alternativa *open-source* e gratis, sulla quale si è già maturata una certa esperienza tramite ulteriori progetti e lavori. Le altre componenti e il framework stesso sono *open-source* e completamente gratis, quindi i requisiti di costo sono soddisfatti.

* + 1. Design Patterns
       1. Service Layer Pattern

Nel framework Laravel, sorge il problema di dove collocare la logica di business, relativa ai servizi individuati durante la fase di System Design. Porre tale logica all’interno dei Controller violerebbe il principio di responsabilità singola: i Controller dovrebbero preoccuparsi solamente di ricevere una richiesta e di fornire una risposta, non di come i dati della risposta vengono ottenuti. Per la risoluzione di tale problematica, verrà applicato il Service Layer pattern, il quale includerà la logica di business del sistema, permettendo ai vari Controller di poter richiedere un particolare tipo di servizio offerto da un sottosistema senza dover comunicare direttamente, ad esempio, con i Model.

Per l’implementazione di tale pattern, verrà creata un package contenente i servizi del nostro sistema e le classi contenuti in tale package potranno essere incluse nei Controller tramite la tecnica della *dependency injection*, così da ridurre quanto più possibile l’accoppiamento tra Controller e servizi, così da permettere eventualmente l’uso di mock di servizi per il testing.

* + 1. Repository Pattern

Come specificato nel documento di System Design, il nostro sistema avrà fonti di dati multiple. La principale consiste nel database relazionale, il quale sarà in comunicazione con il nostro sistema attraverso Eloquent. Ma sono presenti anche due ulteriori fonti di dati:

* I file contenuti nel filesystem, ovvero immagini ed eseguibili dei videogiochi;
* Eventuali servizi esterni futuri, o anche servizi interni per operazioni di caching (es. Redis);

Per tale motivo, si ritiene opportuno astrarre l’accesso alle fonti di dati con un layer composto da implementazioni di una interfaccia Repository, la quale esporrà i metodi per le operazioni principali relative alla persistenza dei dati (creazione, aggiornamento, ricerca e cancellazione). In tal modo, verrà ridotto l’accoppiamento tra il layer dei servizi e quello della persistenza dei dati. Per tale motivo, si ritiene necessaria la creazione di semplici classi PHP per ogni classe del nostro sistema (es. Utente, Videogioco) che verrà usato come tipo di ritorno dei metodi delle nostre Repository e che saranno semplici contenitori di dati con getter, setter e costruttori che permettono l’istanziazione partendo dagli oggetti ottenuti dalle nostre fonti di dati (es. Modelli).

* 1. Linee guida per la documentazione delle interfacce

Agli sviluppatori viene richiesto di seguire le regole di seguito enunciate per mantenere il codice il più consistente possibile, così da mantenere alta la qualità del progetto ed eventualmente favorire l’integrazione di sviluppatori ulteriori in futuro.

* + 1. Regole globali
* Tutto il codice PHP scritto deve rispettare le regole standard di PHP, ovvero le PSR. In particolare, considerato l’uso del framework Laravel di ultima versione, è necessario l’uso delle PSR-12 ([https://www.php-fig.org/psr/psr-12/](https://www.php-fig.org/psr/psr-12/,)), così da ridurre al minimo l’impatto cognitivo del progetto per un eventuale nuovo sviluppatore;
* I nomi dei file contenenti delle sottoclassi dei controller di Laravel (e le sottoclassi stesse) devono essere singolari e devono essere seguiti dalla parola “Controller” (esempio: “VideogameController.php” contenente la classe “VideogameController”). Stessa cosa deve avvenire con i servizi e con le repository: nel primo caso, il termine da usare come suffisso sarà “Service”, nel secondo, “Repository”;
* I nomi dei file contenenti delle sottoclassi dei model di Eloquent (e le sottoclassi stesse) devono essere singolari (esempio: “Videogame.php” contenente la classe “Videogame”);
* Usare sempre oggetti *Carbon*, forniti dal framework Laravel, per rappresentare internamente date e orari;
  + 1. Organizzazione dei file
* Tutti i file devono seguire la gerarchia di cartelle proposta da Laravel, in particolare Controllers, Models e Views.
  1. Definizioni, acronimi ed abbreviazioni
* Off-the-shelf: Componenti oppure framework pronti all’utilizzo per risolvere una determinata problematica;
* Back end: La parte del sistema contenente la logica applicativa;
* Front end: La parte del sistema contenente le interfacce proposte all’utente per interagire con il sistema;
* Laravel: Web Application Framework per la creazione di sistemi sul Web;
* Laravel Sail: Componente di Laravel per la gestione dello startup e della terminazione del sistema;
* ORM: Object relational mapping, una tecnica che permette di interfacciarsi ad un sistema software tramite il paradigma della programmazione orientata agli oggetti;
* Eloquent: ORM principale del framework Laravel;
* Laravel-Admin: Componente di Laravel per l’implementazione di pannelli di amministrazione;
* Laravel Cashier: Componente di Laravel per gestire la comunicazione con servizi di pagamento;
* Stripe: Infrastruttura di pagamenti online;
* PSR: PHP Standard Recommendations;
  1. Riferimenti
* Laravel: <https://laravel.com/>
* Laravel Sail: <https://laravel.com/docs/8.x/sail>
* Eloquent: <https://laravel.com/docs/8.x/eloquent>
* Laravel-Admin: <https://github.com/z-song/laravel-admin>
* Laravel Cashier: <https://laravel.com/docs/8.x/billing>
* Stripe: <https://stripe.com/it>
* PSR-12: <https://www.php-fig.org/psr/psr-12/>

1. Packages

La divisione del sistema in packages verrà realizzata tramite una esatta gerarchia del filesystem del progetto. La struttura di base usata è quella proposta dal framework Laravel, dove in particolare:

* La cartella *app* conterrà tutto il codice per gestire le richieste dei clienti, dall’arrivo della richiesta ad un controller fino all’invio di una risposta;
* La cartella *database* conterrà il codice relativo alla generazione della nostra fonte di dati relazionale, ovvero le migrazioni contenenti gli schemi delle nostre tabelle (così da avere una traccia dei cambiamenti applicati nel tempo al database) ed eventuali classi *seeder* per il riempimento di dati mock per il testing;
* La cartella *resources* conterrà tutta la parte di front end del nostro sistema, categorizzato in tre sottocartelle:
  + *css* per i fogli di stile;
  + *js* per il codice JavaScript client-side;
  + *views* per i documenti rappresentanti le interfacce proposte agli utenti;

In particolare, la cartella *app* sarà quella dove verrà collocata la gran parte del nostro sistema. La sua struttura è come segue:

* Exceptions, contenente le definizioni delle eccezioni lanciate dal nostro sistema;
* Http, contenente una cartella *Controllers* dove verranno realizzati i nostri Controller per ogni pagina del sistema, assieme ad una cartella *Middleware* per eventuali middleware customizzati;
* *Models*, contenente i nostri modelli Eloquent per interfacciarci con il database relazionale;

Inoltre, verranno create due cartelle all’interno della cartella *app* per i layer ulteriori individuati durante la fase di individuazione di design patterns:

* *Services*, contenente i servizi offerti dal nostro sistema, raggruppati in cartelle che rappresentano i sottosistemi ai quali fanno parte;
* *Repositories*, contenente l’interfaccia *Repository* e le repositories per l’accesso ai dati persistenti gestiti dal sistema;

1. Interfacce delle classi

Le classi che comporranno il nostro back end saranno di cinque tipi: Model, Controller, Services, Repositories e le classi PHP rappresentanti gli oggetti trattati dalla logica di business. I model e gli oggetti della logica di business non avranno bisogno di una descrizione della loro interfaccia, poiché non avranno implementazioni relative alla logica di business: il loro obiettivo è di fornire un punto di accesso tramite il paradigma ad oggetti al database relazionale per i primi, mentre i secondi servono come rappresentazione orientata ad oggetti dei dati trattati dal sistema; i servizi useranno questi dati per costruire risposte da fornire ai controller. Il front end sarà composto semplicemente da dei template riempiti con i dati forniti dai controller. Il flusso di esecuzione generale sarà sempre composto da un oggetto Control che chiama un metodo di un oggetto Service per eseguire un servizio, senza mai accedere direttamente agli oggetti Repository; gli oggetti Service accederanno agli oggetti Repository necessari per ottenere i dati necessari; gli oggetti Repository comunicheranno con gli oggetti per l’accesso dei dati, come ad esempio i Model di Laravel.

|  |  |
| --- | --- |
| Nome | UtenteControl |
| Descrizione | Questo control riceve le richieste relative al sottosistema Utenza, invocando i servizi necessari per eseguire operazioni di autenticazione e di gestione del profilo. |
| Attributi | - utenzaService: UtenzaService |
| Firme metodi | + login(Request $request): Response  + logout(): Response  + registrazione(Request $request): Response  + tentaRecuperoPassword(Request $request): Response  + resetPassword(Request $request): Response  + visualizzaProfilo(): Response  + modificaProfilo(): Response  + modificaDatiProfilo(Request $request): Response |
| Pre-condizioni | **context** UtenteControl::login(request)  **pre:** request.has([‘username’, ‘password’]) and !Auth::check()  **context** UtenteControl::logout()  **pre:** Auth::check()  **context** UtenteControl::registrazione(request)  **pre:** request.has([‘username’, ‘email’, ‘password’, ‘confermaPassword’]) and !utenzaService.exists(request.input(‘username’))  **context** UtenteControl::tentaRecuperoPassword(request)  **pre:** !Auth::check() and request.has(‘email’)  **context** UtenteControl::resetPassword(request)  **pre:** !Auth::check() and request.hasValidSignature() and request.has([‘email’, ‘password’, ‘confermaPassword’])  **context** UtenteControl::visualizzaProfilo()  **pre:** Auth::check()  **context** UtenteControl::modificaProfilo()  **pre:** Auth::check()  **context** UtenteControl::modificaDatiProfilo(request)  **pre:** request.has([‘username’, ‘email’, ‘isSviluppatore’, ‘avatar’, ‘password’]) and Auth::check() and utenzaService.checkPassword(request.input(‘password’)) |
| Post-condizioni | **context** UtenteControl::login(request)  **post:** Auth::check()  **context** UtenteControl::logout()  **post:** !Auth::check()  **context** UtenteControl::registrazione(request)  **post:** Auth::check() and utenzaService.exists(request.input(‘username’))  **context** UtenteControl::resetPassword(request)  **post:** Auth::check() and utenzaService.checkPassword(request.input(‘password’))  **context** UtenteControl::modificaDatiProfilo(request)  **post:** utenzaService.getProfilo().username = request.input(‘username’) and utenzaService.getProfilo().email = request.input(‘email’) and utenzaService.getProfilo().isSviluppatore = request.input(‘isSviluppatore’) and utenzaService.getProfilo().avatar = request.input(‘avatar’) |
| Invarianti | **context** UtenteControl  **inv:** utenzaService != null |

|  |  |
| --- | --- |
| Nome | VideogiocoControl |
| Descrizione | Questo control riceve le richieste relative al sottosistema Videogioco, invocando i servizi necessari per eseguire tutte le operazioni relative ai videogiochi contenuti nel sistema. |
| Attributi | - videogiocoService: VideogiocoService  - utenzaService: UtenzaService  - pagamentoService: PagamentoService |
| Firme metodi | + ottieniDatiVideogioco(Request $request): Response  + getListaVideogiochi(): Response  + applicaCriteri(Request $request): JsonResponse  + videogiochiInEvidenza(): Response  + avviaModifica(): Response  + aggiornaDatiVideogioco(Request $request): Response  + visualizzaRichieste(): Response  + visualizzaDettagliRichiesta(Request $request): Response  + risolviRichiesta(Request $request): Response  + richiediModificaVideogioco(): Response()  + modificaDatiVideogioco(Request $request): Response  + richiediPubblicazioneVideogioco(Request $request): Response  + iniziaSponsorizzazione(): Response  + verificaDisponibilitàSettimana(Request $request): JsonResponse  + procediPagamentoSponsorizzazione(): Response  + acquistaVideogioco(Request $request): Response  + downloadVideogioco(Request $request): Response  + avviaProceduraSuggerimentoTags(): Response  + suggerisciTags(Request $request): JsonResponse  + rimuoviSuggerimento(Request $request): JsonResponse  + salvaValutazione(Request $request): JsonResponse  + salvaRecensione(Request $request): JsonResponse  + iniziaReport(): Response  + creaReport(Request $request): Response  + nascondi(Request $request): Response |
| Pre-condizioni | **context** VideogiocoControl::ottieniDatiVideogioco(request)  **pre:** request.has(‘idVideogioco’)  **context** VideogiocoControl::applicaCriteri(request)  **pre:** request.hasAny([‘titolo’, ‘prezzo’, ‘tagsObbligatorie’, ‘tagsOpzionali’, ‘acquistati’, ‘criterioOrdine’])  **context** VideogiocoControl::aggiornaDatiVideogioco(request)  **pre:** Auth::check() and utenzaService.isAdmin() and request.has(‘idVideogioco’) and request.hasAny([‘logo’, ‘titolo’, ‘immagini’, ‘descrizione’, ‘prezzo’])  **context** VideogiocoControl::visualizzaRichieste(request)  **pre:** Auth::check() and utenzaService.isAdmin()  **context** VideogiocoControl::visualizzaDettagliRichiesta(request)  **pre:** Auth::check() and utenzaService.isAdmin() and request.has(‘idRichiesta’)  **context** VideogiocoControl::risolviRichiesta(request)  **pre:** Auth::check() and utenzaService.isAdmin() and request.has([‘idRichiesta’, ‘esito’, ‘commento’])  **context** VideogiocoControl::modificaDatiVideogioco(request)  **pre:** Auth::check() and utenzaService.isSviluppatore() and request.has(‘idVideogioco’) and request.hasAny([‘logo’, ‘titolo’, ‘immagini’, ‘descrizione’, ‘prezzo’])  **context** VideogiocoControl::richiediPubblicazioneVideogioco(request)  **pre:** Auth::check() and utenzaService.isSviluppatore() and request.has([‘logo’, ‘titolo’, ‘immagini’, ‘descrizione’, ‘prezzo’, ‘eseguibile’])  **context** VideogiocoControl::iniziaSponsorizzazione()  **pre:** Auth::check() and utenzaService.isSviluppatore()  **context** VideogiocoControl::verificaDisponibilitàSettimana(request)  **pre:** Auth::check() and utenzaService.isSviluppatore() and request.has(‘settimana’)  **context** VideogiocoControl::procediPagamentoSponsorizzazione(request)  **pre:** Auth::check() and utenzaService.isSviluppatore() and request.has([‘idVideogioco’, ‘settimane’]) and videogiocoService.settimaneDisponibili(request.input(‘settimane’))  **context** VideogiocoControl::acquistaVideogioco(request)  **pre:** Auth::check() and request.has(‘idVideogioco’)  **context** VideogiocoControl::downloadVideogioco(request)  **pre:** Auth::check()  and request.hasAll([‘idVideogioco’, ‘versione’])  and videogiocoService.getVideogiochiAcquistati(auth()->id)->includes(videogiocoService.getVideogioco(request.input(‘idVideogioco’)))  **context** VideogiocoControl::avviaProceduraSuggerimentoTags()  **pre:** Auth::check()  and request.has(‘idVideogioco’)  and videogiocoService.getVideogiochiAcquistati(auth()->id)->includes(videogiocoService.getVideogioco(request.input(‘idVideogioco’)))  **context** VideogiocoControl::suggerisciTags(request)  **pre:** Auth::check()  and request.hasAll([‘idVideogioco’, ‘tags’])  and videogiocoService.getVideogiochiAcquistati(auth()->id)->includes(videogiocoService.getVideogioco(request.input(‘idVideogioco’)))  **context** VideogiocoControl::rimuoviSuggerimento(request)  **pre:** Auth::check()  and request.hasAll([‘idVideogioco’, ‘tags’])  and videogiocoService.getTagsSuggerite(request.input(‘idVideogioco’), auth().id)->intersection(request.input(‘tags’))->size != 0  **context** VideogiocoControl::salvaValutazione(request)  **pre:** Auth::check() and request.has(‘idRecensione’) and videogiocoService.getValutazioneRecensione(request.input(‘idRecensione’), auth()->id) = null  **context** VideogiocoControl::salvaRecensione(request)  **pre:** Auth::check() and request.has(‘idVideogioco’) and videogiocoService.getRecensione(request.input(‘idVideogioco), auth()->id) = null  **context** VideogiocoControl::iniziaReport()  **pre:** Auth::check()  **context** VideogiocoControl::creaReport(request)  **pre:** Auth::check()  and request.hasAll([‘idCommento’, ‘motivo’])  **context** VideogiocoControl::nascondi(request)  **pre:** Auth::check() and utenzaService.isAdmin() and request.has(‘idVideogioco’) and videogiocoService.getVideogioco(request.input(‘idVideogioco’)).nascosto = false |
| Post-condizioni | **context** VideogiocoControl::aggiornaDatiVideogioco(request)  **post:** request.all()->forAll(k: string, v: string | videogiocoService.getVideogioco(request.input(‘id’))[k] = v)  **context** VideogiocoControl::risolviRichiesta(request)  **post:** videogiocoService.getRichiesta(request.input(‘idRichiesta’)).esito = request.input(‘esito’) and videogiocoService.getRichiesta(request.input(‘idRichiesta’)).commento = request.input(‘commento’)  **context** VideogiocoControl::modificaDatiVideogioco(request)  **post:** videogiocoService.  getVideogioco(request.input(‘idVideogioco’)  .getNumRichieste() =  videogiocoService  .getVideogioco(request.input(‘idVideogioco’)  @pre.getNumRichieste() + 1  **context** VideogiocoControl::richiediPubblicazioneVideogioco(request)  **post:** utenzaService  .getSviluppatore()  .getNumRichieste() =  utenzaService  .getSviluppatore()  @pre.getNumRichieste() + 1  **context** VideogiocoControl::procediPagamentoSponsorizzazione(request)  **post:** videogiocoService  .getNumSponsorizzazioni(request.input(‘idVideogioco’)) =  videogiocoService  @pre.getNumSponsorizzazioni(request.input(‘idVideogioco’)) + request.input(‘settimane’).size  **context** VideogiocoControl::acquistaVideogioco(request)  **post:** videogiocoService  .getVideogiochiAcquistati(auth().id) =  videogiocoService  @pre.getVideogiochiAcquistati(auth().id) + 1  and videogiocoService.checkVideogiocoAcquistato(request.input(‘idVideogioco’)) = true  **context** VideogiocoControl::suggerisciTags(request)  **post:**  videogiocoService.getTags(request.input(‘idVideogioco))->intersection(request.input(‘tags’)) = request.input(‘tags’)  **context** VideogiocoControl::rimuoviSuggerimento(request)  **post:** videogiocoService.getTagsSuggerite(request.input(‘idVideogioco’), auth().id)->intersection(request.input(‘tags’))->size = 0  **context** VideogiocoControl::salvaRecensione(request)  **post:** videogiocoService.getRecensione(request.input(‘idVideogioco’), auth()->id) != null  **context** VideogiocoControl::nascondi(request)  **post:** videogiocoService.getVideogioco(request.input(‘idVideogioco’)).nascosto = true |
| Invarianti | **context** VideogiocoControl  **inv:** videogiocoService != null and utenzaService != null and pagamentoService != null |

|  |  |
| --- | --- |
| Nome | ForumControl |
| Descrizione | Questo control riceve le richieste relative al sottosistema Forum, invocando i servizi necessari per eseguire tutte le operazioni relative alle discussioni e ai commenti contenuti nel sistema. |
| Attributi | - forumService: ForumService  - videogiocoService: VideogiocoService  - utenzaService: UtenzaService |
| Firme metodi | + chiudiDiscussione(Request $request): JsonResponse  + creaNuovaDiscussione(Request $request): Response  + creaDiscussione(Request $request): Response  + commenta(Request $request): JsonResponse  + iniziaReport(): Response  + creaReport(Request $request): JsonResponse  + poniInRilievo(Request $request): JsonResponse  + nascondi(Request $request): Response  + visualizzaDettagliReport(Request $request): Response  + risolviReport(Request $request): Response |
| Pre-condizioni | **context** ForumControl::chiudiDiscussione(request)  **pre:** Auth::check()  and request.has(‘idDiscussione’)  and forumService  .checkPermessiDiscussione(request.input(‘idDiscussione’))  and forumService  .getDiscussione(request.input(‘idDiscussione’)).chiusa = false  **context** ForumControl::creaNuovaDiscussione(request)  **pre:** Auth::check()  and request.has(‘idVideogioco’)  and videogiocoService.checkVideogiocoAcquistato(request.input(‘idVideogioco’)) = true  **context** ForumControl::creaDiscussione(request)  **pre:** Auth::check()  and request.hasAll([‘idVideogioco’, ‘titolo’, ‘corpo’])  and videogiocoService.checkVideogiocoAcquistato(request.input(‘idVideogioco’)) = true  **context** ForumControl::commenta(request)  **pre:** Auth::check()  and request.hasAll([‘idDiscussione, ‘corpo’])  and videogiocoService.checkVideogiocoAcquistato(request.input(‘idVideogioco’)) = true  **context** ForumControl::commenta()  **pre:** Auth::check()  **context** ForumControl::creaReport(request)  **pre:** Auth::check()  and request.hasAll([‘idCommento’, ‘motivo’])  **context** ForumControl::poniInRilievo(request)  **pre:** Auth::check() and request.has(‘idDiscussione’) and forumService.checkPermessiDiscussione(request.input(‘idDiscussione’)) and forumService.getDiscussione(request.input(‘idDiscussione’)).in\_rilievo = false  **context** ForumControl::nascondi(request)  **pre:** Auth::check() and utenzaService.isAdmin() and request.has(‘idCommento’) and forumService.getCommento(request.input(‘idCommento’)).nascosto = false  **context** ForumControl::visualizzaDettagliReport(request)  **pre:** Auth::check() and utenzaService.isAdmin() and request.has(‘idReport’)  **context** ForumControl::risolviReport(request)  **pre:** Auth::check() and utenzaService.isAdmin() and request.has(‘idReport’) and forumService.getReport(request.input(‘idReport’)).esito = null |
| Post-condizioni | **context** ForumControl::chiudiDiscussione(request)  **post:** forumService.getDiscussione(request.input(‘idDiscussione’)).chiusa = true  **context** ForumControl::creaDiscussione(request)  **post:** forumService.getNumDiscussioni(request.input(‘idVideogioco’)) = forumService@pre.getNumDiscussioni(request.input(‘idVideogioco’)) + 1  **context** ForumControl::commenta(request)  **post:** forumService.getNumCommenti(request.input(‘idDiscussione’)) = forumService@pre.getNumCommenti(request.input(‘idDiscussione’)) + 1  **context** ForumControl::poniInRilievo(request)  **post:** forumService.getDiscussione(request.input(‘idDiscussione’)).in\_rilievo = true  **context** ForumControl::nascondi(request)  **post:** forumService.getCommento(request.input(‘idCommento’)).nascosto = true  **context** ForumControl::risolviReport(request)  **post** forumService.getReport(request.input(‘idReport’)).esito != null |
| Invarianti | **context** ForumControl  **inv:** forumService != null and videogiocoService != null and utenzaService != null |