

Specifiche per il Progetto di Laboratorio di Sistemi Operativi

Tutor:

Matteo Trentin - matteo.trentin2@unibo.it

Gejsi Vjerdha - gejsi.vjerdha2@unibo.it

Anno Accademico 2023 - 2024

Questo documento contiene le specifiche del progetto del corso di Sistemi Operativi, per la Laurea in Informatica per il Management dell'Università di Bologna, anno accademico 2023-2024. Questa è la versione 1.0 di questo documento; questo numero verrà aggiornato in caso di future modifiche e correzioni.

Eventuali modifiche o note sulle specifiche verranno comunicate tramite la bacheca ufficiale del corso su Virtuale.

In qualsiasi momento è possibile prenotare un ricevimento con i tutor in presenza o su Teams; il ricevimento va prenotato via email, contattando uno dei tutor, o all'indirizzo matteo.trentin2@unibo.it, o all'indirizzo gejsi.vjerdha2@unibo.it.

1 Gruppi

I gruppi devono essere costituiti da tre (3) o quattro (4) persone. Gruppi da cinque (5) persone sono considerati un caso limite, e in fase di orale verranno esaminati ognuno su tutto il progetto.

Gli studenti intenzionati a sostenere l'esame devono comunicare entro il **31 maggio 2024** la composizione del gruppo via email. Questa deve essere inviata dall'indirizzo istituzionale (@studio.unibo.it).

Gli studenti intenzionati a sostenere l'esame al **primo appello** devono comunicare il gruppo entro il **13 maggio 2024** (data di scadenza per la

consegna del progetto).

La mail deve avere come oggetto **[LABSO] FORMAZIONE GRUPPO** e contenere:

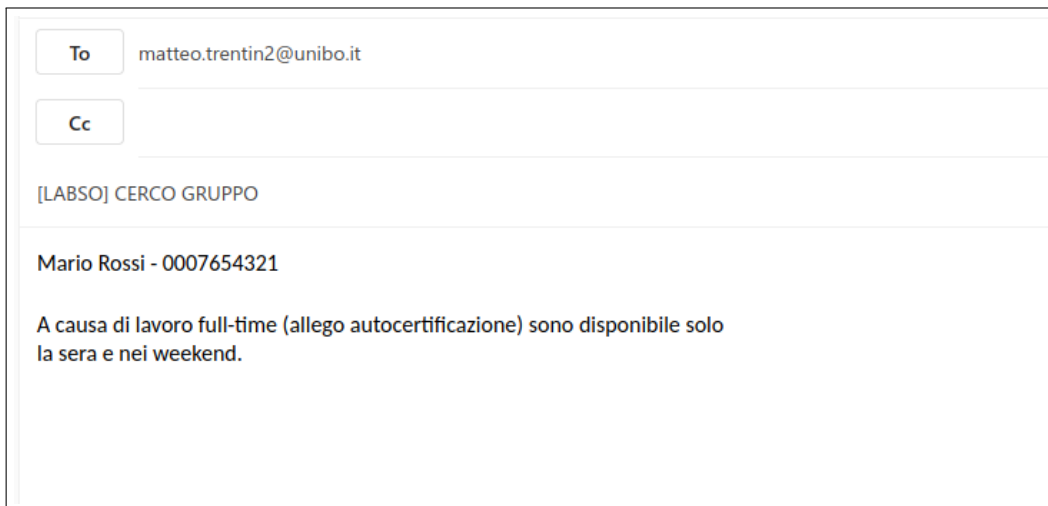
- Il nome del gruppo
- Per ogni componente: nome, cognome, numero di matricola
- Un indirizzo email di riferimento a cui inviare notifiche. È responsabilità del referente trasmettere le comunicazioni al resto del gruppo.

To	matteo.trentin2@unibo.it
Cc	
[LABSO] FORMAZIONE GRUPPO	
NOME GRUPPO	
Mario Rossi - 0007654321	
John Doe - 0001234567	
Jimi Hendrix - 0007898223	
mario.rossi@studio.unibo.it	

Esempio di email di formazione gruppo

Chi non riuscisse a trovare un gruppo può inviare una mail con oggetto **[LABSO] CERCO GRUPPO**, specificando:

- Nome, cognome, numero di matricola
- Eventuali preferenze legate a tempi di lavoro. Si cercherà di costituire gruppi di persone con tempi di lavoro compatibili, nel limite delle possibilità.



Esempio di email di ricerca gruppo

Le persone senza un gruppo vengono assegnate il prima possibile **senza possibilità di ulteriori modifiche**. Per tale motivo è caldamente consigliato rivolgersi al tutor per la ricerca di un gruppo come *ultima* soluzione.

1.1 Repository

Il codice sorgente dei progetti dovrà essere caricato come repository su GitHub¹ o GitLab².

Il repository deve essere **privato** e deve avere il nome LABSO_<NOME GRUPPO>.

Sul repository va caricato **il codice del progetto**; non sono accettati repository contenenti, ad esempio, un file .zip con al suo interno il progetto. Sul repository può essere caricata anche la documentazione del progetto, ma non è obbligatorio.

1.1.1 GitHub

1. Ogni membro del gruppo crea un account su GitHub (a meno che non ne abbia già uno).

¹<https://github.com/>



²<https://gitlab.com>

2. Il referente del gruppo crea un nuovo progetto cliccando su “+” → “**New repository**” nella barra superiore della schermata principale di GitHub. Inserisce LABSO_<NOME_GRUPPO> come nome del progetto, imposta il repository come privato e clicca su “**Create repository**”.
3. Il referente aggiunge ogni membro del gruppo al repository. Per fare ciò, dal menu del repository seleziona “**Settings**” → “**Collaborators**” e in seguito clicca su “**Add people**”. Nella schermata di invito membri, il referente cerca ciascun membro col nome utente con cui quest’ultimo è iscritto a GitHub e clicca su “**Add <username> to this repository**”.

Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).


Owner * **Repository name ***

 mattrent / LABSO_GruppoBello 

Great repository names are short and memorable. Need inspiration? How about [bookish-disco?](#)

Description (optional)

 **Public**
Anyone on the internet can see this repository. You choose who can commit.


 **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

Add a README file
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

 You are creating a private repository in your personal account.

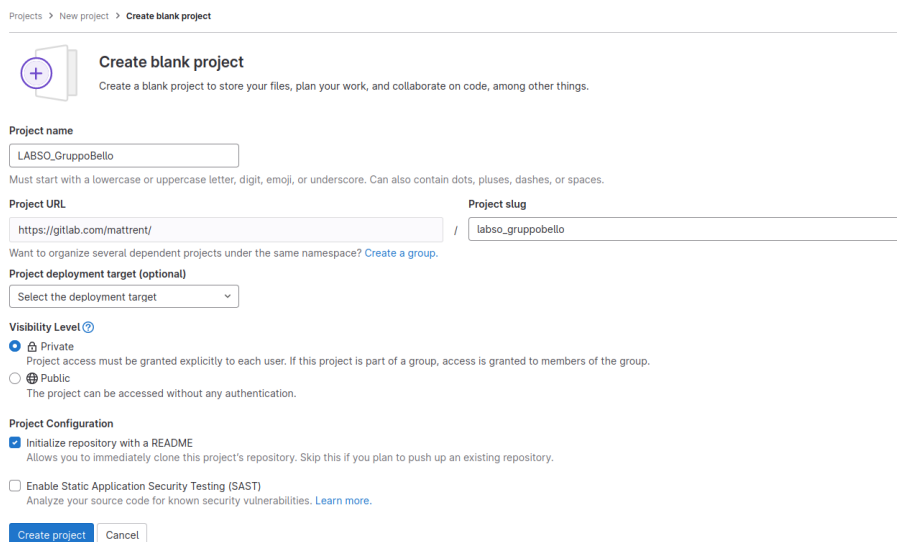
[Create repository](#)

Figure 1: Schermata di creazione progetto su GitHub

1.1.2 GitLab

1. Ogni membro del gruppo crea un account su GitLab (a meno che non ne abbia già uno).

2. Il referente del gruppo crea un nuovo progetto cliccando su “+” → “**New project/repository**” nella barra superiore della schermata principale di GitLab e selezionando “**Create blank project**”. Inserisce LABSO_<NOME_GRUPPO> come nome del progetto, imposta il repository come privato e clicca su “**Create project**”.
3. Il referente aggiunge ogni membro del gruppo al repository. Per fare ciò, dal menu del repository seleziona “**Project information**” → “**Members**” e in seguito clicca su “**Invite members**”. Nella schermata di invito membri, il referente cerca ciascun membro col nome utente con cui quest’ultimo è iscritto a GitLab, seleziona come ruolo **Developer** e clicca su “**Invite**”.



Projects > New project > Create blank project

Create blank project
Create a blank project to store your files, plan your work, and collaborate on code, among other things.

Project name
LABSO_GruppoBello
Must start with a lowercase or uppercase letter, digit, emoji, or underscore. Can also contain dots, pluses, dashes, or spaces.

Project URL / **Project slug**
Want to organize several dependent projects under the same namespace? [Create a group](#).

Project deployment target (optional)
Select the deployment target

Visibility Level ⓘ
 Private
Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.
 Public
The project can be accessed without any authentication.

Project Configuration
 Initialize repository with a README
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.
 Enable Static Application Security Testing (SAST)
Analyze your source code for known security vulnerabilities. [Learn more](#).

Figure 2: Schermata di creazione progetto su GitLab

2 Progetto

2.1 Funzionalità

Il progetto richiede di implementare un servizio publish/subscribe, dove multipli client possono “isciversi” a determinati topic, e ricevere messaggi inviati su di essi.

Nello specifico, il servizio permette a più client di connettersi a un singolo server, e permette a ognuno di questi client di assumere il ruolo di *publisher* o di *subscriber* su un determinato topic.

I **publisher** potranno:

- Inviare messaggi sul loro topic
- Richiedere la lista di tutti i messaggi da loro inviati fino a quel momento
- Richiedere la lista di tutti i messaggi inviati sul loro topic (quindi anche da altri publisher) fino a quel momento

I **subscriber** potranno:

- Ricevere messaggi sul topic da loro scelto
- Richiedere la lista di tutti i messaggi inviati fino a quel momento sul loro topic

Infine, il **server** potrà:

- Ispezionare in modo interattivo i vari topic, eliminando messaggi a scelta

Ogni topic ha un nome univoco. I topic non vengono creati con un'istruzione esplicita, ma vengono automaticamente aggiunti quando un client li richiede (ad esempio, se un publisher comunica di voler inviare messaggi su `topic-1`, questo verrà creato dalla piattaforma, se non è già esistente). Per semplicità durante lo sviluppo, si possono pre-allocare dei topic esemplificativi.

Un topic che stia venendo ispezionato dal server non sarà utilizzabile fino al termine della fase di ispezione. I comandi relativi a quel topic **non dovranno terminare con errori**, e rimarranno invece in attesa.

2.2 Requisiti implementativi

2.2.1 Linguaggio

Il progetto deve essere implementato in Java (ultime versioni LTS: Java SE 17, Java SE 21). La comunicazione di rete è implementata attraverso i socket. Il progetto deve essere diviso in un'applicazione Client e un'applicazione Server, che implementino le funzionalità descritte nella sezione [2.1](#).

2.2.2 Client

Il client viene avviato da linea di comando e richiede come parametri l'indirizzo IP e la porta del server a cui connettersi, ad esempio:

```
java Client 127.0.0.1 9000
```

Se il server non è raggiungibile all'indirizzo e alla porta specificati, il comando restituisce un messaggio di errore. Se la connessione va a buon fine, il client rimane in attesa delle istruzioni dell'utente.

Una volta connesso al server, ogni client ha a disposizione tre comandi:

- Il comando `publish` registra il client come publisher. Il comando richiede un solo argomento `<topic>`, che rappresenta per l'appunto il topic su cui il client pubblicherà. Ad esempio:

```
publish ExampleTopic
```

- Il comando `subscribe` registra il client come subscriber. Il funzionamento è analogo al comando `publish`, ad esempio:

```
subscribe ExampleTopic
```

Quando un client effettua la *subscribe* ad un topic, **riceve solo i messaggi pubblicati da quel momento in poi**. Di conseguenza non riceverà tutti i messaggi inviati sul topic in precedenza.

- Il comando `show` mostra una lista di tutti i topic creati dai publisher.

```
> show
Topics:
  - tech
  - cinema
  - sport
```

- Il comando `quit` arresta il client (senza registrarlo come publisher o subscriber).

I comandi richiesti per i **publisher** sono i seguenti:

- Il comando **send** invia un messaggio al server, sul topic specificato in precedenza. Il comando richiede un singolo parametro `<message>`, il testo del messaggio. Ad esempio:

```
send LoremIpsum
```

- Il comando **list** restituisce un elenco di tutti i messaggi mandati dal publisher corrente sul suo topic. Ogni voce dell'elenco deve contenere:
 - Un identificativo univoco per il messaggio (ad esempio un indice)
 - Il testo del messaggio
 - La data e l'ora in cui è stato inviato (si considera “inviato” un messaggio che sia stato ricevuto dal server)

Ad esempio:

```
> list
Messaggi:
- ID: 2
  Testo: LoremIpsum
  Data: 01/01/1970 - 12:34:56
- ID: 6
  Testo: Dolor sit amet
  Data: 02/01/1970 - 13:42:53
```

Il formato dell'output è a piacere, ma deve contenere le informazioni sopra elencate.

- Il comando **listall** funziona in modo analogo al comando **list**, elencando però tutti i messaggi inviati sul topic (non solo quelli inviati dal publisher corrente).
- Il comando **quit** arresta il client.

I comandi richiesti per i **subscriber** sono i seguenti:

- Il comando `listall` elenca tutti i messaggi inviati sul proprio topic, in modo analogo al comando `listall` dei publisher.
- Il comando `quit` arresta il client.

I subscriber non usano un comando esplicito per ricevere messaggi, ma **ricevono in automatico i messaggi del topic quando vengono inviati.**

2.2.3 Server

Il server viene avviato da linea di comando, e accetta come unico parametro la porta su cui restare in ascolto, ad esempio:

```
java Server 9000
```

I comandi richiesti per il server sono i seguenti:

- Il comando `quit` disconnette tutti i client ancora connessi e successivamente termina il server.
- Il comando `show` funziona in maniera del tutto analoga al `show` dei client, quindi, mostra la lista di tutti i topic presenti sulla piattaforma.
- Il comando `inspect` apre una sessione interattiva in cui è possibile analizzare un topic. Richiede un solo argomento `<topic>`. Se il topic non esiste, il comando restituisce un errore. Se il topic è esistente, la sessione interattiva risultante accetta tre comandi:
 - Il comando `:listall` elenca tutti i messaggi inviati sul topic, in modo analogo al comando descritto per i client.
 - Il comando `:delete` elimina un messaggio selezionato. Richiede un argomento `<id>`, che identifica il messaggio da eliminare. Questo corrisponde all'identificativo univoco del messaggio, mostrato in `listall`. Se l'identificativo non corrisponde a nessun messaggio **nel topic selezionato**, il comando restituisce un errore.
 - Il comando `:end` termina la sessione interattiva.

I comandi standard (i.e. `quit` e `inspect`) sono disabilitati finché è aperta una sessione interattiva. I comandi dei client `send`, `list` e `listall` relativi al topic selezionato rimarranno in attesa, finché la sessione non sarà terminata.

Un esempio di questo comando può essere:

```

> inspect ExampleTopic
ExampleTopic:
  > :listall
  - ID: 1
    Testo: ....
    Data: 01/01/1970 - 10:10:10
  - ID: 2
    Testo: LoremIpsum
    Data: 01/01/1970 - 12:34:56
  - ID: 3
    Testo: Donec eu nunc turpis
    Data: 01/01/1970 - 14:43:34
  > :delete 3
  > :listall
  - ID: 1
    Testo: ....
    Data: 01/01/1970 - 10:10:10
  - ID: 2
    Testo: LoremIpsum
    Data: 01/01/1970 - 12:34:56
  > :end
> ...

```

Si consiglia di essere anche molto verbosi nei log di tutti i comandi (sia server che client). Prendendo `listall` come esempio:

```

> listall
Sono stati inviati 0 messaggi in questo topic.
> listall
Sono stati inviati 2 messaggi in questo topic.
- ID: 1
  Testo: Lorem
  Data: 01/01/1970 - 10:10:10
- ID: 2
  Testo: Ipsum
  Data: 01/01/1970 - 12:34:56

```

2.3 Documentazione

La documentazione è parte integrante del progetto. Non vi sono vincoli sugli strumenti utilizzati per redigere la documentazione (e.g. LaTeX, MS Word, Google Docs, etc.), l'importante è che al termine della stesura venga consegnato un file **PDF** e che tale file rispetti i requisiti descritti in questa sezione. La documentazione deve avere una lunghezza di **almeno 10 pagine** (intese come facciate), compresa l'intestazione, e deve essere scritta con **font di grandezza 12pt**. Il limite di pagine è un *lower bound*: non esiste un *upper bound* per la lunghezza della documentazione, che può quindi essere lunga a piacimento.

2.3.1 Struttura della documentazione

L'intestazione della documentazione deve avere titolo “Laboratorio di Sistemi Operativi A.A. 2023-24” e deve contenere

- Il nome del gruppo
- L'indirizzo email del referente del gruppo
- Per ogni componente del gruppo:
 - Nome, cognome, matricola

Il corpo della documentazione deve coprire almeno i seguenti argomenti:

1. Descrizione del progetto consegnato:
 - (a) Architettura generale: visione **di alto livello** di quali sono le componenti in gioco, di come interagiscono tra loro e delle informazioni che si scambiano per far funzionare il progetto. In questa sezione sono particolarmente utili degli **schemi**.
 - (b) Descrizione dettagliata delle singole componenti:
 - Client, server e relativa suddivisione dei compiti
 - Sotto-componenti di client e server: thread, unità logiche, etc.
 - Se necessaria, descrizione delle classi fondamentali e dei loro metodi principali
 - (c) Suddivisione del lavoro tra i membri del gruppo

2. Descrizione e discussione del processo di implementazione:
 - (a) **Descrizione dei problemi** e degli ostacoli incontrati durante l'implementazione, con discussione e giustificazione delle **soluzioni adottate** e di eventuali **soluzioni alternative**. In particolare:
 - Problemi legati alla concorrenza: quali sono le risorse condivise, quando e perché si rende necessaria la mutua esclusione, etc.
 - Problemi legati al modello client-server: come vengono instaurate, mantenute e chiuse le connessioni, cosa succede in caso di interruzioni anomale del client o del server, etc.
 - (b) Descrizione degli strumenti utilizzati per l'organizzazione. In particolare, applicazioni, piattaforme, servizi utilizzati per:
 - Sviluppare il progetto (e.g. Eclipse, IntelliJ IDEA, Visual Studio Code, etc.)
 - Comunicare tra i membri del gruppo (e.g. Teams, Skype, Discord, etc.)
 - Condividere il codice prodotto (e.g. come avete usato Git-Lab).
 - Tenere traccia del lavoro svolto, del lavoro rimasto da svolgere, delle decisioni ad alto livello prese dal gruppo, etc. (e.g. Google Docs, Trello, etc.)
3. Requisiti e istruzioni passo-passo per compilare e usare le applicazioni consegnate
 - Sono graditi esempi degli output attesi per ogni comando
 - Se presenti, descrizione delle estensioni implementate e di come usarle

L'organizzazione delle sezioni non deve per forza rispecchiare esattamente quella appena riportata. Per esempio, è possibile trattare un argomento in più sezioni, o trattare più argomenti nella stessa sezione. È anche possibile aggiungere informazioni non espressamente richieste nell'elenco qualora fossero utili. L'importante è che *almeno* i contenuti elencati siano *facilmente* rintracciabili nel corpo della documentazione.

2.3.2 Scopo della documentazione

La documentazione deve puntare a dare al lettore una visione chiara di come funziona il progetto e di come sono stati affrontati gli ostacoli di implementazione, *senza* che il lettore debba conoscere il codice sorgente. I rimandi al codice (e.g. “Vedi `Server.java`, righe 150-155”) sono apprezzati, ma il codice sorgente *non* deve sostituire la documentazione. Dall’altro lato, la documentazione non deve essere una semplice ripetizione delle specifiche. Per la natura del progetto è pressoché scontato che ci sia (ad esempio) una classe del server in cui vengono ricevute le richieste del client, o una classe in cui vengono gestiti i topic, etc. Quello su cui la documentazione deve concentrarsi è *cosa* accade quando arriva una richiesta, *come* sono gestiti i topic, etc. nella *vostra particolare implementazione* delle specifiche.

3 Consegna del progetto

Al momento della consegna, occorre creare un tag di nome **Consegna** all’interno del proprio repository. Le modalità di creazione del tag sono definite ai seguenti indirizzi:

- Per progetti caricati su **GitHub**: <https://docs.github.com/en/repositories/releasing-projects-on-github/managing-releases-in-a-repository>, nella sezione “**Creating a release**”. La release sarà anch’essa chiamata **Consegna**.
- Per progetti caricati su **GitLab**: <https://docs.gitlab.com/ee/user/project/repository/tags/>, nella sezione “**Create a tag**”. È essenziale aggiungere un messaggio qualsiasi *non vuoto* per annotare il tag con la data e l’ora della creazione.

I tutor dovranno essere aggiunti come Collaborator (se il progetto è su GitHub) o Reporter (se il progetto è su GitLab) del repository, seguendo la stessa modalità descritta in sezione 1.1. Questa operazione può anche essere svolta subito prima della consegna, ma è necessaria per rendere accessibile il codice. Gli account da aggiungere sono:

- **mattrent** sia su GitHub (<https://github.com/mattrent>), sia su GitLab (<https://gitlab.com/mattrent>).

- Gejsi sia su GitHub (<https://github.com/Gejsi>), sia su GitLab (<https://gitlab.com/Gejsi>).

Una volta fatto ciò, notificare la consegna al tutor inviando un'email a `matteo.trentin2@unibo.it` con oggetto **[LABSO] CONSEGNA < NOME GRUPPO >**. La mail dovrà includere:

- L'indirizzo del repository
- In allegato, un file PDF di nome `DOCUMENTAZIONE.pdf`, contenente la documentazione del progetto

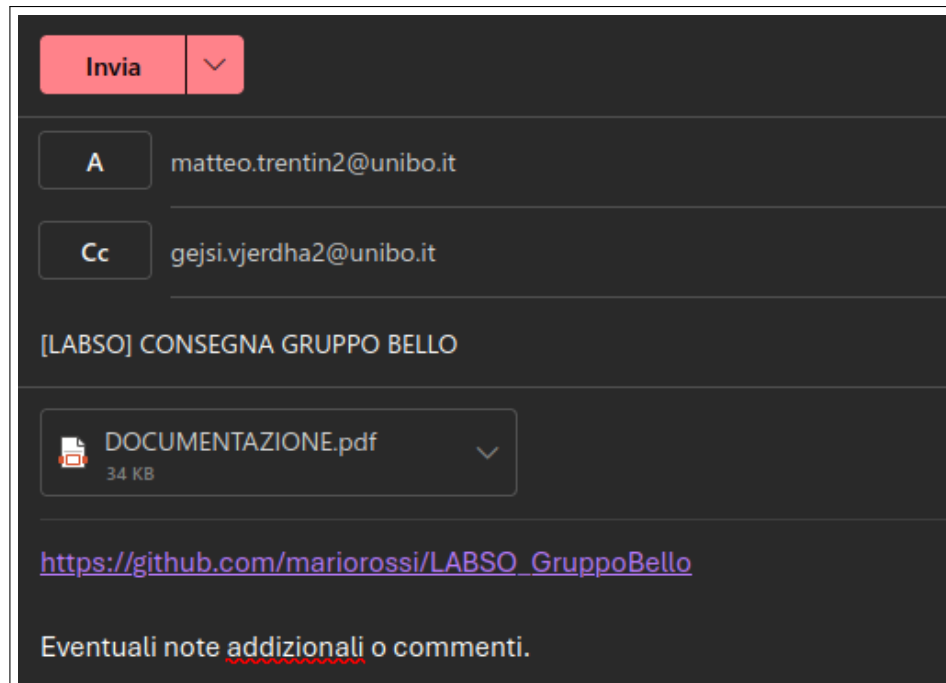


Figure 3: Esempio di email di notifica di consegna

Sono presenti quattro appelli per la consegna del progetto, con le rispettive scadenze:

- Lunedì **13 Maggio 2024** (13/05/2024), ore 23:59.
- Lunedì **24 Giugno 2024** (24/06/2024), ore 23:59.

- Lunedì **30 Settembre 2024** (30/09/2024), ore 23:59.
- Lunedì **25 Novembre 2024** (25/11/2024), ore 23:59.

Su AlmaEsami è presente un appello per ciascuna di queste scadenze. Il voto finale è individuale, per cui *tutti* i membri del gruppo sono tenuti ad iscriversi su AlmaEsami all'appello in cui il gruppo intende discutere il progetto.

N.B. La data dell'appello su AlmaEsami corrisponde alla data di scadenza, non alla data di discussione.

4 Discussione

In seguito alla consegna, il luogo, la data e l'ora della discussione verranno fissati e comunicati al referente del gruppo. Solitamente, la discussione avviene entro due settimane dalla scadenza di consegna. La discussione consiste in:

- Una breve demo del progetto implementato, che può essere effettuata indifferentemente su rete locale (client e server eseguono sulla stessa rete e/o macchina) o su internet (ad esempio avviando da remoto il server su una macchina di laboratorio).
- Alcune domande ai membri del gruppo sull'implementazione del progetto, l'organizzazione del lavoro e i contributi personali di ciascuno. Durante questa fase verrà richiesto di mostrare e spiegare frammenti di codice sorgente.

Siccome la discussione prevede di dimostrare il proprio progetto e spiegare il proprio codice, è consigliabile (benché non obbligatorio) che il gruppo si presenti in sede di discussione con un proprio portatile.

Al termine della discussione, ad ogni membro del gruppo viene assegnato un punteggio da 0 a 8 in base all'effettivo contributo alla realizzazione del progetto dimostrato in sede di discussione. Questo punteggio si somma alla valutazione del progetto (vedi Sezione 5.4) per determinare il voto finale in trentesimi di ogni singolo membro del gruppo.

5 Griglie di Valutazione

5.1 Implementazione

La valutazione dell'implementazione del progetto si basa sull'analisi del codice Java, sull'implementazione corretta delle specifiche e sull'uso dei costrutti del linguaggio per la creazione di soluzioni efficienti e tolleranti ai guasti. Da questo punto di vista, l'obiettivo fondamentale del progetto è quello di dimostrare che i membri del gruppo sono in grado di:

- Utilizzare il multithreading per gestire situazioni in cui molti processi (e.g. più letture contemporanee) devono poter eseguire simultaneamente.
- Riconoscere quando una struttura dati è una risorsa condivisa, ovvero quando viene acceduta concorrentemente da più thread, e individuare i problemi di concorrenza associati.
- Adottare di conseguenza i costrutti di mutua esclusione e sincronizzazione adeguati al caso.

Un progetto in cui non viene fatto uso di alcun costrutto di sincronizzazione è pertanto automaticamente insufficiente, così come un progetto che evita i problemi di concorrenza eseguendo tutta la logica applicativa su un solo thread (nonostante nella vita reale questa sia una strada assolutamente percorribile).

La valutazione tiene conto anche di aspetti esterni alla programmazione intesa in senso stretto, come la ripartizione e l'organizzazione del lavoro all'interno del gruppo, la qualità del processo di implementazione e la tracciabilità degli artefatti di sviluppo. In particolare, una ripartizione precisa dei compiti all'interno del gruppo è importante. Questo non significa che più membri del gruppo non possono collaborare o non devono sapere nulla l'uno del codice degli altri, ma significa che ciascun componente è l'esperto di una (o più) parti circoscritte del progetto (e.g. gestione della connessione client-server, gestione dei comandi dell'utente, etc.) e se ne assume la responsabilità. Per ciascuno degli aspetti riportati nella seguente tabella viene assegnato un punteggio da 0 (insufficiente) a 8 (ottimo).

CRITERIO	DESCRIZIONE
Rispetto delle specifiche	Il progetto implementa correttamente le funzionalità descritte in Sezione 2.1 e rispetta i requisiti di Sezione 2.2.
Qualità del codice	Il gruppo usa correttamente i costrutti e le strutture dati offerti da Java per gestire concorrenza e distribuzione. Gestione adeguata di eccezioni e casi limite. Il codice è leggibile e ben commentato.

5.2 Documentazione

La valutazione della documentazione verte sull'analisi dello scritto e sulla sua capacità di descrivere con chiarezza il prodotto consegnato, i problemi riscontrati durante l'implementazione e le soluzioni adottate, **soprattutto grazie all'uso di esempi**. Per ciascuno degli aspetti riportati nella seguente tabella viene assegnato un punteggio da 0 (insufficiente) a 4 (ottimo).

CRITERIO	DESCRIZIONE
Qualità dell'informazione	La documentazione fornisce una descrizione chiara e completa del progetto implementato e del processo di implementazione.
Uso di esempi	Presenza di esempi (narrativi, grafici, etc.) utili alla comprensione delle scelte implementative del gruppo o di scenari d'uso specifici.
Analisi delle scelte implementative	Individuazione e descrizione dei problemi incontrati durante l'implementazione, con particolare enfasi sui problemi legati alla concorrenza e alla distribuzione. Discussione delle soluzioni adottate e di soluzioni alternative valide.

5.3 Organizzazione del lavoro nel gruppo

Una voce aggiuntiva riguarda la distribuzione del lavoro all'interno del gruppo, nello specifico:

- La divisione dei compiti è ben delineata ed omogenea
- Ciascun membro del gruppo è capace di indicare e spiegare i propri contributi
- Il processo di implementazione è ben delineato e tracciabile.

A questa voce viene assegnato un punteggio da 0 (insufficiente) a 6 (ottimo).

5.4 Voto di progetto e voto finale

I punteggi relativi a ciascuno degli aspetti descritti in questa sezione vengono sommati per ottenere un punteggio di base che va da 0 a 34. Terminata la discussione, a questo punteggio si somma il punteggio individuale ottenuto

da ciascun membro del gruppo. Il risultato è un punteggio da 0 a 42 per ogni membro, che corrisponde al suo voto individuale di progetto. Un punteggio pari o superiore a 31/42 corrisponde a 30 e Lode.

Il voto finale del corso viene calcolato a partire dal voto di progetto e dal voto dello scritto. Per maggiori dettagli sulla modalità di calcolo e di verbalizzazione del voto finale, fare riferimento alla pagina web del Prof. Sangiorgi: <http://www.cs.unibo.it/~sangio/>.