Università degli Studi di Napoli
Federico II

A.A. 2022-2023

InfoPoint

ID Gruppo: <TODO>

Lucia Brando l.brando@studenti.unina.it N86003382

Dario Morace da.morace@studenti.unina.it N86003778

Valentino Bocchetti vale.bocchetti@studenti.unina.it N86003405

Indice

0 Su InfoPoint 3
0.1 Presentazione 4

1 Guida all'uso 5
1.1 Guida al Server 6
1.1.1 Funzionalità 6
1.1.2 Scelte implementative 6
1.1.3 Tecnologie e strumenti utilizzati 6
1.1.4 Memorizzazione dei dati 6
1.2 Guida al Client 7
1.2.1 Primo avvio 7
1.2.2 Post registrazione 7
1.2.3 Memorizzazione delle informazioni 7
1.2.4 Modelli di dominio 7

2 Protocollo applicativo 8
3 Protocollo applicativo 9
4 Dettagli implementativi 10
4.1 Server 10
4.2 Client 11

5 Codice sorgente sviluppato 12
6 Ringraziamenti 13

Capitolo: Su InfoPoint

Estratto

InfoPoint© è una piattaforma che nasce per offrire un supporto ai visitatori del museo.

L'implementazione è suddivisa in 2 componenti che identificheremo come **server** e **client**

- Un backend scritto in `C` per la gestione dei dati, hostato[1] su una macchina virtuale offerta da Azure[2];
- Una applicazione `Android`, scritta in `Java` che fa da client;

[1] Indica un servizio di rete che consiste nell'allocare su un server web delle pagine web di un sito web o di un'applicazione web, rendendolo così accessibile dalla rete Internet e ai suoi utenti

[2] Per maggiori informazioni visitare il seguente sito

INFOPOINT

Educate Yourself

1 Capitolo: Guida all'uso

Estratto

<TODO>

1.1.1 Funzionalità

Il Sistema, deve offrire, una serie di funzionalità:

- Possibilità di connessione concorrente;
- Possibilità di potersi registrare alla piattaforma[1];
- Possibilità di usufruire dei contenuti in base alla tipologia di utente, in modo da permette un focus diverso in base alle sue caratteristiche[2] ;

1.1.2 Scelte implementative

Seguendo il concetto del *DIVIDE ET IMPERA*[3] si è scelto di spezzare le varie funzionalità che vengono messe a disposizione per rendere il codice facilmente manutenibile ed evitare lo stato di codice monolitico[4].

1.1.3 Tecnologie e strumenti utilizzati

Per una migliore gestione del Sistema, si è fatto uso di una serie di strumenti.

Durante lo sviluppo si è fatto uso dell'utility **cmake**[5] , tool modulare che permette la generazione di un **Makefile**[6] .

Per la fase di deploy invece si è fatto uso di **docker**, tool che permette l'esecuzione di programmi in maniera containerizzata.

Come sperato, la combinazione di questi tool ha permesso un passaggio immediato da una situazione di esecuzione locale (di debug) ad una

Come sperato, avendo adottato entrambe le strategie non si sono riscontrati problemi durante il passaggio da un ambiente locale (di testing) a uno decentralizzato (in produzione).

1.1.4 Memorizzazione dei dati

Per essere sempre in linea con le nuove tendenze e tecnologie si è scelto di abbandonare il classico approccio basato su un collegamento ad una base di dati relazionale, preferendo un approccio di tipo **NOSQL**[7] , che offre una maggiore elasticità e scalabilità nel tempo.

[1] Le credenziali vengono salvate facendo uso di un Database, che risulta molto più affidabile di un semplice file di testo

[2] Ricordiamo che il bacino degli utenti che possono fare uso del sistema può variare da scolaresche, famiglie o esperti

[3] Metodologia per la risoluzione di problemi → Il problema viene diviso in sottoproblemi più semplici e si continua fino a ottenere problemi facilmente risolvibili. Combinando le soluzioni ottenute si risolve il problema originario.

[4] Che risulta notoriamente più difficile da gestire e modificare nel tempo

[5] Per maggiori informazioni visitare il seguente sito

[6] Che contiene tutte le direttive utilizzate dall'utility make, che ne permettono la corretta compilazione

[7] Che a differenza dei classici DBMS relazioni (che offrono un approccio `relazione` ai dati) offre un approccio al documento, rendendo il design più semplice

1.2.1 Primo avvio
1.2.2 Post registrazione
1.2.3 Memorizzazione delle informazioni
1.2.4 Modelli di dominio

Class Diagram

Sequence Diagram

Capitolo: Protocollo applicativo

Come già indicato in precedenza abbiamo preferito il protocollo **TCP** rispetto al protocollo **UDP**, per la presenza di un controllo della congestione e affidabilità in termini di `invio/ricezione` di dati[1] .

Lo sviluppo dell'applicativo è stato inizialmente verticalizzato sulla creazione dello scheletro del Server, per avere un primo approccio nudo e crudo allo scambio di messaggi via **socket**.

Per avere un programma robusto e manutenibile si è fatto largo uso delle **good pratices** che questo tipo di comunicazione richiede. In particolare:

- La connessione viene aperta solo nel momento in cui devono essere `inviati/ricevuti` dati (Si evita in questo modo di tenere aperte connessioni in momenti in cui queste non vengono sfruttate);
- Si effettuano controlli di raggiungibilità del server lato client[2];
- Vengono controllati i dati `inviati/ricevuti` sempre prima di compiere operazioni che possano minare il corretto funzionamento di **Server** e **Client** [3]
- Vengono effettuati controlli e gestione degli stati di tutte le operazioni lato **Server**;

[1] Ricordiamo infatti che UDP non ha garanzie sulla trasmissione dei pacchetti, seguento la logica di best-effort

[2] Non ha senso infatti tenere aperta una connessione se non utilizzata, anzi si rischia anche di causare interruzione di servizio dovuti a timeout improvvisi

[3] Questo avviene anche attraverso un particolare pattern di costruzione dei dati

3 Capitolo: Protocollo applicativo

Come già indicato in precedenza abbiamo preferito il protocollo **TCP** rispetto al protocollo **UDP**, per la presenza di un controllo della congestione e affidabilità in termini di `invio/ricezione` di dati[1] .

Lo sviluppo dell'applicativo è stato inizialmente verticalizzato sulla creazione dello scheletro del Server, per avere un primo approccio nudo e crudo allo scambio di messaggi via **socket**.

Per avere un programma robusto e manutenibile si è fatto largo uso delle **good pratices** che questo tipo di comunicazione richiede. In particolare:

- La connessione viene aperta solo nel momento in cui devono essere `inviati/ricevuti` dati (Si evita in questo modo di tenere aperte connessioni in momenti in cui queste non vengono sfruttate);
- Si effettuano controlli di raggiungibilità del server lato client[2];
- Vengono controllati i dati `inviati/ricevuti` sempre prima di compiere operazioni che possano minare il corretto funzionamento di **Server** e **Client** [3]
- Vengono effettuati controlli e gestione degli stati di tutte le operazioni lato **Server**;

[1] Ricordiamo infatti che UDP non ha garanzie sulla trasmissione dei pacchetti, seguento la logica di best-effort

[2] Non ha senso infatti tenere aperta una connessione se non utilizzata, anzi si rischia anche di causare interruzione di servizio dovuti a timeout improvvisi

[3] Questo avviene anche attraverso un particolare pattern di costruzione dei dati

4 Capitolo: Dettagli implementativi

4.1 Server

5 Capitolo: Codice sorgente sviluppato

Il codice sorgente prodotto durante lo sviluppo di $InfoPoint$ è disponibile sulla piattaforma *GitHub*, che ne ha permesso anche il versionamento.

Di seguito riportiamo un link per il download[1]

[1] Potrebbe essere richiesta l'autenticazione (il repository è per privacy privato)

6 Capitolo: Ringraziamenti

Ringraziamo la professoressa Alessandra Rossi per lo splendido corso, che ci ha permesso di conoscere nuove interessanti tecnologie e del supporto offertoci durante e dopo le lezioni.

Lucia Brando

N86003382

l.brando@studenti.unina.it

Profilo

Valentino Bocchetti

N86003405

vale.bocchetti@studenti.unina.it

Profilo

Profilo

da.morace@studenti.unina.it

N86003778

Dario Morace