

Università degli Studi di Napoli Federico II

**Corso di Laurea in Ingegneria Informatica**

**Corso di Ingegneria del Software**

**Prof. S. Russo - A.A. 2021 - 22**

***Progetto***

Gestione Pescheria

Francesco Rosucci N46004970 f.rosucci@studenti.unina.it

**INDICE**

1. Specifiche informali 1

2. Analisi e specifica dei requisiti 2

2.1 Analisi nomi-verbi 2

2.2 Revisione dei requisiti 2

2.3 Glossario dei termini 3

2.4 Classificazione dei requisiti 3

2.4.1 Requisiti funzionali 4

2.4.2 Requisiti sui dati 4

2.4.3 Vincoli / Altri requisiti 4

2.5 Modellazione dei casi d’uso 5

2.5.1 Attori e casi d’uso 5

2.5.2 Diagramma dei casi d’uso 5

2.5.3 Scenari 5

2.6 Modellazione dei dati 2

2.6.1Progettazione concettuale 2

2.7 Diagramma delle classi 2

2.8 Diagrammi di sequenza 2

2.9 Verifica della completezza dei requisiti 2

3. Stima dei costi 3

4. Piano di test funzionale 4

5. Progettazione 6

5.1 Progettazione della base di dati 6

5.1.1 Progettazione logica 6

5.2 Diagramma delle classi 6

5.3 Diagrammi di sequenza 6

6. Implementazione 7

7. Testing 8

7.1 Test strutturale 8

7.1.1 Complessità ciclomatica 8

7.1.2 Test di unità 9

7.2 Test funzionale 9

# Specifiche informali

La pescheria “o’ sole mio” ha commissionato la realizzazione di un sistema software per la vendita di pescato online. Il gestore della pescheria può inserire nel proprio catalogo diversi tipi di alimenti, ognuno identificato da un codice, una tipologia (e.g., pesce di acqua salata, pesce di acqua dolce, crostacei, molluschi etc.), la quantità disponibile (in Kg) e il prezzo (al Kg). Inoltre, è necessario indicare per ogni pesce se è di allevamento oppure selvatico; in entrambi i casi è presente un codice numerico che identifica il paese di provenienza, ma solo nel caso in cui è di allevamento è necessario specificare anche la tipologia di allevamento (intensivo, estensivo, in vasca, in mare aperto). Il gestore può mettere in vendita dei mix di crudi, composti da diversi alimenti in diverse quantità. Il prezzo è quello del pescato presente nel mix, con un supplemento del 10%. Gli utenti che intendono acquistare dal sito della pescheria devono registrarsi al sistema. Gli utenti registrati sono caratterizzati da un nome utente ed una password. Una volta effettuato l'accesso l'utente può selezionare il pescato e/o i mix crudi da acquistare specificando le rispettive quantità. Il sistema verifica la disponibilità dei prodotti acquistati in pescheria comunicando all'utente l'esito dell'operazione. In caso affermativo, l'utente inserisce i dati di pagamento. Di ogni acquisto si vuole tenere traccia della data di acquisto e del prezzo complessivo. Ogni 15 giorni il sistema deve generare in maniera automatica un report contenente la graduatoria degli alimenti e dei mix più venduti. Il gestore della pescheria può consultare tale graduatoria per valutare l’andamento delle vendite, generando opzionalmente un report.

# Analisi e specifica dei requisiti

## Analisi nomi-verbi

*La pescheria “o’ sole mio” ha commissionato la realizzazione di un sistema software per la vendita di pescato online.*

*Il gestore della pescheria può inserire nel proprio catalogo diversi tipi di alimenti, ognuno identificato da un codice, una tipologia (e.g., pesce di acqua salata, pesce di acqua dolce, crostacei, molluschi etc.), la quantità disponibile (in Kg) e il prezzo (al Kg). Inoltre, è necessario indicare per ogni pesce se è di allevamento oppure selvatico; in entrambi i casi è presente un codice numerico che identifica il paese di provenienza, ma solo nel caso in cui è di allevamento è necessario specificare anche la tipologia di allevamento (intensivo, estensivo, in vasca, in mare aperto). Il gestore può mettere in vendita dei mix di crudi, composti da diversi alimenti in diverse quantità. Il prezzo è quello del pescato presente nel mix, con un supplemento del 10%. Gli utenti che intendono acquistare dal sito della pescheria devono registrarsi al sistema. Gli utenti registrati sono caratterizzati da un nome utente ed una password. Una volta effettuato l'accesso l'utente può selezionare il pescato e/o i mix crudi da acquistare specificando le rispettive quantità. Il sistema verifica la disponibilità dei prodotti acquistati in pescheria comunicando all'utente l'esito dell'operazione. In caso affermativo, l'utente inserisce i dati di pagamento. Di ogni acquisto si vuole tenere traccia della data di acquisto e del prezzo complessivo. Ogni 15 giorni il sistema deve generare in maniera automatica un report contenente la graduatoria degli alimenti e dei mix più venduti. Il gestore della pescheria può consultare tale graduatoria per valutare l’andamento delle vendite, generando opzionalmente un report.*

*LEGENDA:  
Classe  
Attributo  
Funzionalità  
Attore*

*Classe-Attore*

## Revisione dei requisiti

1. Il sistema deve consentire al gestore della pescheria di inserire diversi tipi di alimenti nel catalogo.
2. Di ogni alimento si vuole memorizzare codice identificativo, tipologia, quantità, prezzo e codice identificativo del paese di provenienza.
3. La quantià di ogni alimento memorizzato deve essere espressa in Kilogrammi.
4. Il prezzo di ogni alimento memorizzato deve essere espresso in euro al kilogrammo.
5. Per ogni pesce si deve indicare se quest’ultimo è di allevamento o selvatico.
6. Nel caso un pesce sia di allevamento è necessario indicare la tipologia di allevamento.
7. Il sistema deve consentire al gestore di mettere in vendita dei mix di crudi.
8. I mix di crudi sono composti da diversi alimenti in diverse quantità.
9. Il prezzo di un mix di crudi corrisponde a quello del pescato presente nel mix con un supplemento del 10%,
10. Il sistema deve consentire agli utenti di registrarsi.
11. Il sistema deve consentire agli utenti registrati di effettuare il login.
12. Gli utenti registrati sono caratterizzati da un nome utente ed una password.
13. Il sistema deve consentire all’utente registrato di acquistare i prodotti del catalogo.
14. Il sistema deve consentire all’utente registrato di selezionare il pescato o i mix di crudi da acquistare specificando le rispettive quantità.
15. Il sistema deve verificare la disponibilità dei prodotti che l’ utente intende acquistare.
16. Il sistema deve richiedere i dati di pagamento dopo aver verificato la disponibilità dei prodotti che l’utente vuole acquistare.
17. Di ogni acquisto si vuole memorizzare data d’acquisto e prezzo complessivo.
18. Il sistema deve generare in maniera automatica un report contenente la graduatoria degli alimenti e dei mixdi crudi più venduti ogni 15 giorni.
19. Il sistema deve consentire al gestore della pescheria di consultare la graduatoria degli alimenti e dei mix di crudi più venduti.
20. Il sistema deve consentire al gestore della pescheria di generare opzionalmente un report dopo aver consultato la graduatoria.
21. Gli utenti che intendono acquistare dal sito della pescheria devono registrarsi al sistema.
22. Il sistema deve consentire al cliente registrato di acquistare solo dopo aver effettuato l’accesso.

## Glossario dei termini

|  |  |  |
| --- | --- | --- |
| Termine | Descrizione | Sinonimi |
| Alimenti | Prodotto del pescato tra cui pesce d’acqua dolce o salata,molluschi,crostacei etc. | Pesce,Pescato |
| Mix di crudi | Un mix composto da diversi alimenti crudi in diverse quantità |  |
| Utente | Cliente della pescheria | Cliente |
| Utente registrato | Cliente della pescheria registrato al sito | Cliente registrato |
| Graduatoria | Graduatoria degli alimenti e dei mix di crudi più venduti |  |
| Report | Stampa della graduatoria |  |
| Acquisto | Un acquisto effettuato da un cliente sul sito | Ordine |
| Prodotto | Un prodotto della pescheria che può essere un alimento o un mix di crudi |  |

## Classificazione dei requisiti

### Requisiti funzionali

|  |  |  |
| --- | --- | --- |
| ID | Requisito | Origine (n. frase dei requisiti revisionati) |
| RF01 | Il sistema deve consentire al gestore della pescheria di inserire diversi tipi di alimenti nel catalogo | 1 |
| RF02 | Il sistema deve consentire al gestore di mettere in vendita dei mix di crudi. | 7 |
| RF03 | Il sistema deve consentire agli utenti di registrarsi. | 10 |
| RF04 | Il sistema deve consentire agli utenti registrati di effettuare il login. | 11 |
| RF05 | Il sistema deve consentire all’utente registrato di acquistare i prodotti del catalogo | 13 |
| RF06 | Il sistema deve consentire all’utente di selezionare il pescato o i mix di crudi da acquistare specificando le rispettive quantità. | 14 |
| RF07 | Il sistema deve verificare la disponibilità dei prodotti che l’ utente intende acquistare | 15 |
| RF08 | Il sistema deve richiedere i dati di pagamento dopo aver verificato la disponibilità dei prodotti che l’utente vuole acquistare | 16 |
| RF09 | Il sistema deve generare in maniera automatica un report contenente la graduatoria degli alimenti e dei mixdi crudi più venduti ogni 15 giorni. | 18 |
| RF10 | Il sistema deve consentire al gestore della pescheria di consultare la graduatoria degli alimenti e dei mix di crudi più venduti. | 19 |
| RF11 | Il sistema deve consentire al gestore della pescheria di generare opzionalmente un report dopo aver consultato la graduatoria. | 20 |

### Requisiti sui dati

|  |  |  |
| --- | --- | --- |
| ID | Requisito | Origine (n. frase dei requisiti revisionati) |
| RD01 | Di ogni alimento si vuole memorizzare codice identificativo, tipologia, quantità, prezzo e codice identificativo del paese di provenienza | 2 |
| RD02 | La quantià di ogni alimento memorizzato deve essere espressa in  Kilogrammi. ? | 3 |
| RD03 | Nel caso un pesce sia di allevamento è necessario indicare la tipologia di allevamento. | 6 |
| RD04 | I mix di crudi sono composti da diversi alimenti in diverse quantità. | 8 |
| RD05 | Il prezzo di un mix di crudi corrisponde a quello del pescato presente nel mix con un supplemento del 10%. | 9 |
| RD06 | Gli utenti registrati sono caratterizzati da un nome utente ed una password. | 12 |
| RD07 | Di ogni acquisto si vuole memorizzare data d’acquisto e prezzo complessivo. | 17 |

### Vincoli / Altri requisiti

*V01: Gli utenti che intendono acquistare dal sito della pescheria devono registrarsi al sistema*

*V02: Il sistema deve consentire al cliente registrato di acquistare solo dopo aver effettuato l’accesso.*

*RNF01:* *La quantià di ogni alimento memorizzato deve essere espressa in Kilogrammi*

*RNF02: Il prezzo di ogni alimento memorizzato deve essere espresso in euro al kilogrammo.*

## Modellazione dei casi d’uso

### Attori e casi d’uso

***Attori Primari:***

* Utente
* Utente registrato
* Gestore
* Tempo

***Attori Secondari:***

|  |  |
| --- | --- |
| ***Casi d’uso:***   * UC1:InserisciAlimento * UC2:InserisciMix * UC3:Registrazione * UC4:Login * UC5:AcquistoProdotti * UC6:ConsultaGraduatoria * UC7:GeneraReportGraduatoria | ***Casi d’ uso di inclusione:***   * UC8:VerificaDisponibilità * UC9:VerificaDisponibilitàMix |

***Casi d’ uso di estensione:***

* UC11:GeneraReport
* UC10:VerificaDisponibilitàAlimento

|  |  |  |  |
| --- | --- | --- | --- |
| Caso d’uso | Attori Primari | Attori Secondari | Incl. / Ext. |
| InserisciAlimento | Gestore | - |  |
| InserisciMix | Gestore | - |  |
| Registrazione | Utente | - |  |
| Login | UtenteRegistrato | - |  |
| AcquistoProdotti | UtenteRegistrato | - | Include VerificaDisponibilità |
| ConsultaGraduatoria | Gestore | - | Include GeneraReport |
| GeneraReportGraduatoria | Tempo | - |  |
| VerificaDisponibilità | - | - | Incluso in AcquistaProdotti  Generalizzazione di VerificaDisponibilitàMix,  VerificaDisponibilitàAlimento |
| VerificaDisponibilitàMix | - | - |  |
| VerificaDisponibilitàAlimento | - | - |  |
| GeneraReport |  | - | Estensione di ConsultaGraduatoria |

### Diagramma dei casi d’uso

Diagram

Description automatically generated

### Scenari

|  |  |
| --- | --- |
| Caso d’uso: | AcquistaProdotti |
| Attore primario | UtenteRegistrato |
| Attore secondario | - |
| Descrizione | L’utente seleziona ed acquista dei prodotti dal catalogo |
| Pre-Condizioni | L’utente ha effettuato il login ed ha visualizzato il catalogo della pescheria |
| Sequenza di eventi  principale | 1. Il caso d’uso inizia quando l’utente richiede l’acquisto di prodotti 2. Finchè l’utente non indica di voler terminare la selezione    1. L’utente inserisce il codice e la quantità del prodotto che vuole acquistare    2. Se il codice corrisponde a quello di un mix       1. Il sistema verifica la disponibilità della quantità di mix richiesto    3. Se il codice corrisponde a quello di un alimento       1. Il sistema verifica la disponibilità della quantità di alimento richiesto    4. Se la disponibilità non è sufficiente       1. Il sistema restituisce all’utente un messaggio di errore indicando che la quantità disponibile per il prodotto selezionato non è sufficiente    5. Altrimenti       1. Il sistema aggiorna la quantità disponibile del prodotto 3. Se l’utente ha selezionato prodotti validi    1. Il sistema conferma il corretto esito della selezione di prodotti mostrando all’utente il prezzo totale    2. L’utente inserisce i dati di pagamento ed effettua l’acquisto.    3. Il sistema registra l’acquisto con la data corrente. |
| Post-Condizioni | Le quantità disponibili dei prodotti vengono aggiornate e l’acquisto è stato memorizzato nel sistema. |
| Casi d’uso correlati | *nessuno* |
| Sequenza di eventi  alternativi | *nessuna* |

## Modellazione dei dati

### 2.6.1Progettazione concettuale

Diagram

Description automatically generated

## Diagramma delle classi

A screenshot of a computer

Description automatically generated with medium confidence

Diagramma delle classi raffinato (con classi Boundary e Control)

A screenshot of a computer

Description automatically generated with medium confidence

## Diagrammi di sequenza

Diagram

Description automatically generated

## Verifica della completezza dei requisiti

Legenda: UCD = Use Case Diagram, CD = Class Diagram.

- RF01 è modellato nell’UCD con l’attore “Gestore” e con il caso d’uso UC1

- RF02 è modellato nell’UCD con l’attore “Gestore” e con il caso d’uso UC2

- RF03 è modellato nell’UCD con l’attore “Utente” e con il caso d’uso UC3

- RF04 è modellato nell’UCD con l’attore “UtenteRegistrato” e con il caso d’uso UC4

- RF05 è modellato nell’UCD con l’attore “UtenteRegistrato” e con il caso d’uso UC5

- RF06 è modellato nell’UCD con il caso d’uso UC8,UC9,UC10

- RF07 è modellato nell’UCD con il caso d’uso UC5

- RF08 è modellato nell’UCD con l’attore “Tempo” con il caso d’uso UC7

- RF09 è modellato nell’UCD con l’attore “Gestore” con il caso d’uso UC6

- RF10 è modellato nell’UCD con l’attore “Gestore” con il caso d’uso UC11

- RD01 è modellato nel CD con la classe “Alimento”

- RD02 è modellato nel CD con le classi AlimentoAllevato e AlimentoSelvatico

- RD03 è modellato nel CD con l’attributo “tipoAllevamento” della classe AlimentoAllevato

- RD04 è modellato nel CD con l’aggregazione tra la classe MixCrudi e Alimento e dalla classe

associativa composizione

- RD05 è modellato nel CD con l’attributo derivato /Prezzo della classe MixCrudi

- RD06 è modellato nel CD con la classe UtenteRegistrato

- RD07 è modellato nel CD con la classe Acquisto

# Stima dei costi

|  |  |  |  |
| --- | --- | --- | --- |
|  | SEMPLICE | MEDIO | COMPLESSO |
| NILF | **7** | **10** | **15** |
| NEIF | **5** | **7** | **10** |
| NEI | **3** | **4** | **6** |
| NEO | **4** | **5** | **7** |
| NEQ | **4** | **4** | **6** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | VALORE | SEMPLICE | MEDIO | COMPLESSO | TOT |
| NILF | 1 |  | 10 |  | 10 |
| NEIF | 0 |  |  |  |  |
| NEI | 3 | 3 |  |  | 9 |
| NEO | 0 |  |  |  |  |
| NEQ | 0 |  |  |  |  |

NILF: L’acquisto viene creato dal sistema, lo identifichiamo come ILF[1 medio]

NEI: Prodotto,Quantità,Carta di credito. [3 semplici]

|  |  |  |  |
| --- | --- | --- | --- |
| FATTORI CORRETTIVI | | | |
| COMUNICAZIONE DATI | | | 1 |
| DISTRIBUZIONE ELABORAZIONE | | | 0 |
| PRESTAZIONI | | | 3 |
| UTILIZZO INTENSIVO CONFIGURAZIONE | | | 2 |
| FREQUENZA DELLE TRANSAZIONI | | | 3 |
| INSERIMENTO DATI INTERATTIVO | | | 3 |
| EFFICIENZA PER L’UTENTE FINALE | | | 4 |
| AGGIORNAMENTO INTERATTIVO | | | 2 |
| COMPLESSITA’ ELABORATIVA | | | 0 |
| RIUSABILITA’ | | | 3 |
| FACILITA’ INSTALLAZIONE | | | 2 |
| FACILITA’ GESTIONE OPERATIVA |  |  | 2 |
| MOLTEPLICITA’ DI SITI | | | 0 |
| FACILITA’ DI MODIFICA | | | 3 |
|  | | | **28** |

|  |
| --- |
| **UFP = 19** |
| LLOC/FP = 53 |

|  |
| --- |
| FP=17.67 |
| JAVA=936.51 |

# Piano di test funzionale

**PIANO DI TEST UTILIZZANDO IL METODO DEL *CATEGORY-PARTITION TESTING* PER LA FUNZIONALITÀ “*AcquistaProdotto*”.**

|  |  |  |
| --- | --- | --- |
| **Prodotto** | **Quantità** | **Elemento di sistema database** |
| * Stringa con formato corretto per un alimento(carattere maiuscolo A seguito da 4 interi) [ALIMENTO] * Stringa con formato corretto per un mix(carattere maiuscolo M seguito da 4 inter) [MIX] * Stringa con formato errato [ERROR] | * Numero >=0 * Numero <0 [ERROR] | * Disponibilità alimento>= Quantità[IF ALIMENTO] * Disponibilità Mix>=Quantità [IF MIX] * Disponibilità alimento< Quantità [ERROR] [IF ALIMENTO] * Disponibilità Mix < Quantità [ERROR] [IF MIX] |

Il numero di test da effettuare senza particolari vincoli è 3\*2\*4 = 24.

Non consideriamo i vincoli [ERROR] e consideriamo le combinazioni con i vincoli [PROPERTY]: [ALIMENTO] 1\*1\*1=1; [MIX] 1\*1\*1= 1;

I vincoli [ERROR] sono 4;

I casi di test senza nessun vincolo sono 0;

In totale abbiamo 1+1+4 = 6 casi di test.

**TEST SUITE**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Descrizione** | **Classi di equivalenza coperte** | **Pre-condizioni** | **Input** | **Output Attesi** | **Post-condizioni Attese** |
| 1 | Tutti gli input sono Validi, prodotto = alimento | Prodotto valido  Quantità valido  Elemento di sistema DB valido | Il database è inizializzato correttamente.  C’è disponibilità per l’alimento selezionato | {Prodotto:“A0032”,  Quantità:“0.3”} | Inserire un ulteriore prodotto | Le quantità disponibili vengono aggiornate. |
| 2 | Tutti gli input sono Validi, prodotto = mix | Prodotto valido  Quantità valido  Elemento di sistema DB valido | Il database è inizializzato correttamente.  C’è disponibilità per il mix selezionato | { Prodotto:”M0001”,  Quantità:”1”} | Inserire un ulteriore prodotto | Le quantità disponibili vengono aggiornate. |
| 3 | Prodotto formato non valido | Prodotto formato non valido [ERROR]  Quantità valido  Elemento di sistema DB valido | Il cliente ha selezionato la funzionalità di acquisto. Il cliente ha inserito un codice prodotto non valido | { Prodotto: “A002”,  Quantità:“1.00”} | Codice prodotto non valido! | Il sistema chiede nuovamente di inserire il codice dell prodotto. |
| 4 | Quantità <0 | Prodotto valido  Quantità <0 [ERROR]  Elemento di sistema DB valido | Il Cliente ha selezionato la funzionalità di acquisto. Il cliente ha inserito il codice di un prodoto che vuole acquistare. Il sistema richiede all’utente l’inserimento della quantità | { Prodotto: “A0002”,  Quantità:“-2”} | Quantità Alimento richiesta non valida! | Il sistema chiede nuovamente di inserire la quantità |
| 5 | Disponibilità Mix < Quantità | Prodotto valido  Quantità valido  Elemento di sistema DB non valido [ERROR] | Il Cliente ha selezionato la funzionalità di acquisto. Il cliente ha inserito il codice di un mix. L’utente ha inserito la quantità desiderata.  La quantità disponibile nel database del mix richiesto è inferiore a quella inserita dall’utente. | { Prodotto:“M0001”,  Quantità:“200”} | Quantità richiesta non disponibile | La quantità disponibile nel database del mix richiesto è inferiore a quella inserita dall’utente. |
| 6 | Disponibilità alimento< Quantità | Alimento valido  Quantità valido  Elemento di sistema DB non valido [ERROR] | Il Cliente ha selezionato la funzionalità di acquisto. Il cliente ha inserito il codice di un alimento. L’utente ha inserito la quantità desiderata.  La quantità disponibile nel database dell’alimento richiesto è inferiore a quella inserita dall’utente. | { Prodotto:“A0032”,  Quantità:“300”} | Quantità richiesta non disponibile | La quantità disponibile nel database dell’alimento richiesto è inferiore a quella inserita dall’utente. |
|  |  |  |  |  |  |  |

# Progettazione

## Progettazione della base di dati

### Progettazione logica

|  |
| --- |
| **Alimenti**(**Codice**,Quantita,Tipologia,Prezzo,CodiceProvenienza,TipoAllevamento)  **MixCrudi**(**Codice**,Prezzo)  **Acquisti**(**Id**,PrezzoComplessivo,Data,Utente:UTENTEREGISTRATO)  **UtentiRegistrati**(NomeUtente,Password)  **Composizioni**(**Alimento**:ALIMENTO,**MixCrudi**:MIXCRUDI,Quantita)  **Selezioni**(**Acquisto**:ACQUISTO,**Alimento**:ALIMENTO,Quantita)  **Scelte**(**Acquisto**:ACQUISTO,**MixCrudi**:MIXCRUDI,Quantita) |

create table Alimenti

(

Codice char(5) NOT NULL,

Quantita number CHECK (Quantita>=0) NOT NULL,

Tipologia varchar2(30) NOT NULL,

Prezzo number CHECK(Prezzo>0) NOT NULL,

CodiceProvenienza number(4) NOT NULL,

TipoAllevamento varchar2(30),

CONSTRAINT pk\_alimenti PRIMARY KEY(Codice)

);

create table MIXCRUDI

(

Codice char (5) NOT NULL,

Prezzo number CHECK (Prezzo > 0) NOT NULL,

CONSTRAINT pk\_mix PRIMARY KEY (Codice)

);

create table UTENTIREGISTRATI

(

NomeUtente varchar2(10) NOT NULL,

Password varchar2(20) NOT NULL,

CONSTRAINT pk\_utente PRIMARY KEY (Nome\_Utente)

);

create table ACQUISTI

(

ID number NOT NULL,

PrezzoComplessivo number CHECK (PrezzoComplessivo > 0) NOT NULL,

Data date NOT NULL,

Utente varchar2(10) NOT NULL,

CONSTRAINT fk\_acquisti FOREIGN KEY (Utente) REFERENCES UtentiRegistrati (NomeUtente),

CONSTRAINT pk\_acquisti PRIMARY KEY (ID)

);

create table COMPOSIZIONI

(

Quantità number CHECK (Quantita>0) NOT NULL,

Alimento char(5) NOT NULL,

MixCrudi char(5) NOT NULL,

CONSTRAINT fk\_composizioni\_a FOREIGN KEY(Alimento) REFERENCES ALIMENTI(Codice),

CONSTRAINT fk\_composizioni\_m FOREIGN KEY(MixCrudi) REFERENCES MIXCRUDI(Codice),

CONSTRAINT pk\_composizioni PRIMARY KEY (Alimento, MixCrudi)

);

create table SELEZIONI

(

Quantità number CHECK (Quantita>0) NOT NULL,

Acquisto number NOT NULL,

Alimento char(5) NOT NULL,

CONSTRAINT fk\_selezioni\_ali FOREIGN KEY (Alimento) REFERENCES ALIMENTI (Codice),

CONSTRAINT fk\_selezioni\_acq FOREIGN KEY (Acquisto) REFERENCES ACQUISTI (ID),

CONSTRAINT pk\_selezioni PRIMARY KEY (Acquisto, Alimento)

);

create table SCELTE

(

Quantità number CHECK (Quantita>0) NOT NULL,

Acquisto number NOT NULL,

MixCrudi char(5) NOT NULL,

CONSTRAINT fk\_scelte\_acq FOREIGN KEY (Acquisto) REFERENCES ACQUISTI(ID),

CONSTRAINT fk\_Scelte\_mix FOREIGN KEY (MixCrudi) REFERENCES MIXCRUDI(codice),

CONSTRAINT pl\_scelte PRIMARY KEY(Acquisto,MixCrudi)

);

A picture containing graphical user interface

Description automatically generated

## Diagramma delle classi

A screenshot of a computer

Description automatically generated with low confidence

## Diagrammi di sequenza

Diagram

Description automatically generated

# Implementazione

-Package definiti: boundary, control, database, entity, exception e pescheria;

1. Boundary: all’interno del package sono riportate le classi boundary degli attori primari della nostra applicazione. Nello specifico, per la funzionalità scelta, ho implementato la Boundary BUtenteRegistrato;
2. Control: all’interno del package è riportata la classe control
3. Database: all’interno del package sono riportate le classi DAO, usate per l’accesso al database: per la funzionalità “AcquistaProdotto” si è fatto riferimento a AlimentoDAO [readQuantita, readPrezzo, updateQuantita], AcquistoDAO[create, readLastID], SceltaDAO [create], SelezioneDAO [create], ComposizioneDAO [readComposizione]. All’interno del package troviamo anche la classe DBManager per aprire e chiudere la connessione al database.
4. Entity: all’interno del package sono riportate le classi del dominio applicativo, con i metodi ‘get’ e ‘set’ dei propri attributi. All’interno delle entity Alimento e Mixcrudi sono riportati due metodi statici che permettono di verificare se un codice è corretto per le rispettive classi.
5. Exception: all’interno del package sono riportate le classi usate per la gestione delle eccezioni.
6. Pescheria: all’interno del package è riportata la classe ‘Pescheria’ che presenta il main, punto d’inizio del flusso di esecuzione.

Per l’esecuzione del programma, si necessita l’installazione del database h2, di facile utilizzo e soprattutto facilmente condivisibile. La versione di java utilizzata è la 17.0.2, file.jar usato per h2 è “h2-2.1.214.jar”

# Testing

## Test strutturale

### Complessità ciclomatica

**public** **void** acquistaProdotto(String codiceProdotto,**float** quantitaSelezionata,ArrayList<EntityAlimento> alimentiAcquistati,EntityAcquisto acquisto, ArrayList<EntitySelezione> alimentiSelezionati, ArrayList<EntityScelta> mixScelti) **throws** InputException, DAOException, DBConnectionException, OperationException

{

**float** quantita;

**int** quantitaMix;

**float** prezzo;

ArrayList<EntityComposizione> composizioneMix = **new** ArrayList<EntityComposizione>();

**if**(EntityAlimento.*checkCodice*(codiceProdotto))

{

//Se il prodotto è un alimento verifica la sua disponibilità

**try** {

quantita = verificaDisponibilitaAlimento(codiceProdotto,quantitaSelezionata);

} **catch** (DAOException e) {

**throw** **new** DAOException(e.getMessage());

} **catch** (DBConnectionException e) {

**throw** **new** DBConnectionException(e.getMessage());

} **catch**(OperationException e)

{

**throw** **new** OperationException(e.getMessage());

}

//aggiorna quantita disponibile nel database

**try** {

AlimentoDAO.*updateQuantita*(codiceProdotto,quantita-quantitaSelezionata);

} **catch** (DAOException e) {

**throw** **new** DAOException(e.getMessage());

} **catch** (DBConnectionException e) {

**throw** **new** DBConnectionException(e.getMessage());

}

//aggiungi prezzo all'acquisto

**try** {

prezzo = AlimentoDAO.*readPrezzo*(codiceProdotto);

acquisto.addToPrezzoC(prezzo\*quantitaSelezionata);

} **catch** (DAOException e) {

**throw** **new** DAOException("Impossibile accedere agli alimenti nel database");

} **catch** (DBConnectionException e) {

**throw** **new** DBConnectionException("Impossibile connettersi al database");

}

//aggiungi l'alimento alla lista degli alimenti acquistati (per eventuale ripristino db)

EntityAlimento al = **new** EntityAlimento(codiceProdotto,quantitaSelezionata,prezzo);

alimentiAcquistati.add(al);

//aggiungi l'alimento alla lista degli alimenti selezionati per registrare l'acquisto al termine dell'operazione

EntitySelezione sel = **new** EntitySelezione(quantitaSelezionata,acquisto,al);

alimentiSelezionati.add(sel);

}

**else**

{

**if**(EntityMixCrudi.*checkCodice*(codiceProdotto))

{

//ottieni gli alimenti che compongono il mix e le loro quantita

**try** {

composizioneMix = verificaDisponibilitaMix(codiceProdotto,(**int**)quantitaSelezionata);

} **catch** (DAOException e) {

**throw** **new** DAOException(e.getMessage());

} **catch** (DBConnectionException e) {

**throw** **new** DBConnectionException(e.getMessage());

} **catch**(OperationException e)

{

**throw** **new** OperationException(e.getMessage());

}

//aggiorna le quantita disponibili nel database

quantitaMix = (**int**)quantitaSelezionata;

**for**(Iterator<EntityComposizione> i = composizioneMix.iterator(); i.hasNext();)

{

EntityComposizione temp = i.next();

String codiceAlimentoMix = **new** String(temp.getAlimento().getCodice());

**float** quantitaComposizione = temp.getQuantita();

**try** {

quantita = AlimentoDAO.*readQuantita*(codiceAlimentoMix);

AlimentoDAO.*updateQuantita*(codiceAlimentoMix, quantita-(quantitaMix\*quantitaComposizione ));

} **catch** (DAOException e) {

**throw** **new** DAOException(e.getMessage());

} **catch** (DBConnectionException e) {

**throw** **new** DBConnectionException(e.getMessage());

}

//aggiungi l'alimento alla lista degli alimenti acquistati (per eventuale ripristino db)

EntityAlimento al = **new** EntityAlimento(codiceAlimentoMix,quantitaMix\*quantitaComposizione );

alimentiAcquistati.add(al);

}

//aggiungi prezzo all'acquisto

**try** {

prezzo = MixCrudiDAO.*readPrezzo*(codiceProdotto);

acquisto.addToPrezzoC(prezzo\*quantitaMix);

} **catch** (DAOException e) {

**throw** **new** DAOException("Impossibile accedere agli alimenti nel database");

} **catch** (DBConnectionException e) {

**throw** **new** DBConnectionException("Impossibile connettersi al database");

}

//aggiungi il mix alla lista dei mix scelti per registrare l'acquisto al termine dell'operazione

EntityMixCrudi mixTemp = **new** EntityMixCrudi(prezzo,codiceProdotto,composizioneMix);

EntityScelta scel = **new** EntityScelta((**int**)quantitaSelezionata,acquisto,mixTemp);

mixScelti.add(scel);

}

**else**

{

**throw** **new** InputException("Codice prodotto non valido");

}

}

}

Background pattern

Description automatically generatedControl Flow Graph

NUMERO CICLOMATICO:

numero di regioni del grafo (3 regioni chiuse + 1 reg. convessa)= 4

numero di nodi predicati (0,2,5) +1 = 4

# archi – # nodi + 2 = (11 - 9) + 2 = 4

CAMMINI:

1. 0-1
2. 0-2-4
3. 0-2-3-5-7
4. 0-2-3-5-6-5-7

### Test di unità

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Descrizione** | **Pre-condizioni** | **Input** | **Output Attesi** | **Post-condizioni Attese** |
| T1 | Caso di test della funzionalità in corrispondenza del cammino   1. 0-1 | Il cliente ha selezionato la funzionalità di acquisto. Il cliente ha inserito un codice prodotto corrispondente ad un alimento. Il database è inizializzato correttamente. | {Prodotto:“A0032”,  Quantità:“0.3”} | Inserire un ulteriore prodotto | Le quantità disponibili vengono aggiornate. |
| T2 | Caso di test della funzionalità in corrispondenza del cammino  2)0-2-4 | Il cliente ha selezionato la funzionalità di acquisto. Il cliente ha inserito un codice prodotto non valido | { Prodotto: “A002”,  Quantità:“1.00”} | Codice prodotto non valido! | Il sistema chiede nuovamente di inserire il codice dell prodotto. |
| T3 | Caso di test della funzionalità in corrispondenza del cammino  3)0-2-3-5-7 | - | - | - | - |
| T4 | Caso di test della funzionalità in corrispondenza del cammino  4)0-2-3-5-6-5-7 | Il cliente ha selezionato la funzionalità di acquisto. Il cliente ha inserito un codice prodotto corrispondente ad un mix. Il database è inizializzato correttamente. | { Prodotto:”M0001”,  Quantità:”1”} | Inserire un ulteriore prodotto | Le quantità disponibili vengono aggiornate. |

## Test funzionale

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Descrizione** | **Classi di equivalenza coperte** | **Pre-condizioni** | **Input** | **Output Attesi** | **Post-condizioni Attese** | **Output Ottenuti** | **Post-condizioni Ottenute** | **Esito**  **(*FAIL*, *PASS*)** |
| 1 | Tutti gli input sono Validi, prodotto = alimento | Prodotto valido  Quantità valido  Elemento di sistema DB valido | Il database è inizializzato correttamente.  C’è disponibilità per l’alimento selezionato | {Prodotto:“A0032”,  Quantità:“0.3”} | Inserire un ulteriore prodotto | Le quantità disponibili vengono aggiornate. | Inserire un ulteriore prodotto | Le quantità disponibili vengono aggiornate | PASS |
| 2 | Tutti gli input sono Validi, prodotto = mix | Prodotto valido  Quantità valido  Elemento di sistema DB valido | Il database è inizializzato correttamente.  C’è disponibilità per il mix selezionato | { Prodotto:”M0001”,  Quantità:”1”} | Inserire un ulteriore prodotto | Le quantità disponibili vengono aggiornate. | Inserire un ulteriore prodotto | Le quantità disponibili vengono aggiornate. | **PASS** |
| 3 | Prodotto formato non valido | Prodotto formato non valido [ERROR]  Quantità valido  Elemento di sistema DB valido | Il cliente ha selezionato la funzionalità di acquisto. Il cliente ha inserito un codice prodotto non valido | { Prodotto: “A002”,  Quantità:“1.00”} | Codice prodotto non valido! | Il sistema chiede nuovamente di inserire il codice dell prodotto. | Codice prodotto non valido! | Il sistema chiede nuovamente di inserire il codice dell prodotto. | **PASS** |
| 4 | Quantità <0 | Prodotto valido  Quantità <0 [ERROR]  Elemento di sistema DB valido | Il Cliente ha selezionato la funzionalità di acquisto. Il cliente ha inserito il codice di un prodoto che vuole acquistare. Il sistema richiede all’utente l’inserimento della quantità | { Prodotto: “A0012”,  Quantità:“-2”} | Quantità Alimento richiesta non valida! | Il sistema chiede nuovamente di inserire la quantità | - | Il sistema chiede di inserire un ulteriore prodotto o uscire | **FAIL** |
| 5 | Disponibilità Mix < Quantità | Prodotto valido  Quantità valido  Elemento di sistema DB non valido [ERROR] | Il Cliente ha selezionato la funzionalità di acquisto. Il cliente ha inserito il codice di un mix. L’utente ha inserito la quantità desiderata.  La quantità disponibile nel database del mix richiesto è inferiore a quella inserita dall’utente. | { Prodotto:“M0001”,  Quantità:“200”} | Quantità richiesta non disponibile | La quantità disponibile nel database del mix richiesto è inferiore a quella inserita dall’utente. | Quantità richiesta non disponibile | La quantità disponibile nel database del mix richiesto è inferiore a quella inserita dall’utente. | **PASS** |
| 6 | Disponibilità alimento< Quantità | Alimento valido  Quantità valido  Elemento di sistema DB non valido [ERROR] | Il Cliente ha selezionato la funzionalità di acquisto. Il cliente ha inserito il codice di un alimento. L’utente ha inserito la quantità desiderata.  La quantità disponibile nel database dell’alimento richiesto è inferiore a quella inserita dall’utente. | { Prodotto:“A0032”,  Quantità:“300”} | Quantità richiesta non disponibile | La quantità disponibile nel database dell’alimento richiesto è inferiore a quella inserita dall’utente. | Quantità richiesta non disponibile | La quantità disponibile nel database dell’alimento richiesto è inferiore a quella inserita dall’utente. | PASS |

Il TestCase 4 non restituiva l’output atteso. Per eliminare il difetto che causava questo malfunzionamento è stato necessario inserire un controllo sulla quantità successivo all’inserimento da parte dell’utente. Un eventuale test strutturale non avrebbe potuto rilevare il difetto poiché esso era dovuto ad una parte di codice mancante e che quindi non poteva essere analizzata tramite test costruito sul codice.