

GRIGLIA VALUTAZIONE-AUTOVALUTAZIONE STATO AVANZAMENTO PROGETTI PCTO 4IC – a.s 2020/2021

Short-Streets										
TITOLO PROGETTO:										
PIANO OPERATIVO:										
ORE TOTALI	100	di cui progetto:	15	(15%)	implementazione:	75	(75%)	documentazione:	10	(10%)
FUNZIONALITA' PRINCIPALI		dettaglio funzionalità		costo in ore (*)						
F1: inserimento dati		F1.D1: indicare il numero di pacchi da consegnare e alcune loro caratteristiche come il peso e il volume.		20						
F2: calcolo percorso minimo		F2.D1: calcolare il percorso minimo che il rider deve effettuare in modo da velocizzare e facilitare la consegna.		22						
		F2.D2: implementazione di un algoritmo peso-volume che rappresenta un vincolo in più nel calcolo del percorso minimo.		18						
F3: parte grafica		F3.D1: utilizzare una mappa di un piccolo quartiere sulla quale indicare i punti in cui il corriere deve andare.		10						
		F3.D2: dopo aver individuato il percorso più agevole, evidenziare il percorso che il corriere deve seguire sulla mappa stessa.		15						

(*) con riferimento ai tempi previsti per l'implementazione

(*) con riferimento ai tempi previsti per l'implementazione

STUDENTE	SINTESI COMPITI PERSONALI	LIV. COMPITO (1-2-3) (*)
Bissola Mattia	Testing del software	3
Giuggioli Daniel	Interfaccia grafica, classi per la gestione dei pacchi, utilizzo di una mappa su cui visualizzare i punti presso cui si devono effettuare le consegne e il percorso più breve calcolato dagli algoritmi	3
Majid Zaccaria	Testing del software	3
Salvi Alessandro	Sviluppo dell'algoritmo che consente al rider di trasportare contemporaneamente determinati pacchi secondo determinate caratteristiche, quali il peso e il volume e unione dell'algoritmo peso-volume con quello del percorso minimo	3
Sonzogni Nicolò	Sviluppo dell'algoritmo per il calcolo del percorso minimo che il rider dovrà compiere e unione dell'algoritmo peso-volume con quello del percorso minimo. Utilizzo di una mappa su cui visualizzare i punti presso cui si devono effettuare le consegne e il percorso più breve calcolato dagli algoritmi	3

(*) 3: complesso .. 1:semplice

ORE SVOLTE 35 (35%)

di cui progetto: 10 (67%)

implementazione: 28 (37%)

documentazione: 3 (30%)

(*) percentuali riferite alle rispettive ore del PIANO OPERATIVO

SITUAZIONE PROGETTO

FUNZIONALITA' PRINCIPALI		dettaglio funzionalità	% implementata		bilancio ore		
					previste	svolte	restanti +/- %
F1: inserimento dati		F1.D1: indicare il numero di pacchi da consegnare e alcune loro caratteristiche come il peso, il volume e la destinazione a cui il pacco deve giungere.	70		20	17	3 -
F2: calcolo percorso minimo		F2.D1: calcolare il percorso minimo che il rider deve effettuare in modo da velocizzare e facilitare la consegna.	100		22	18	4 -
F3: parte grafica		F2.D2: implementazione di un algoritmo peso-volume che rappresenta un vincolo in più nel calcolo del percorso minimo.	100		18	15	3 -
		F3.D1: utilizzare una mappa di un piccolo quartiere sulla quale indicare i punti in cui il corriere deve andare.	0		10	0	10 -
		F3.D2: dopo aver individuato il percorso più agevole, evidenziare il percorso che il corriere deve seguire sulla mappa stessa.	0		15	0	15 -

(*) con riferimento ai tempi previsti per l'implementazione

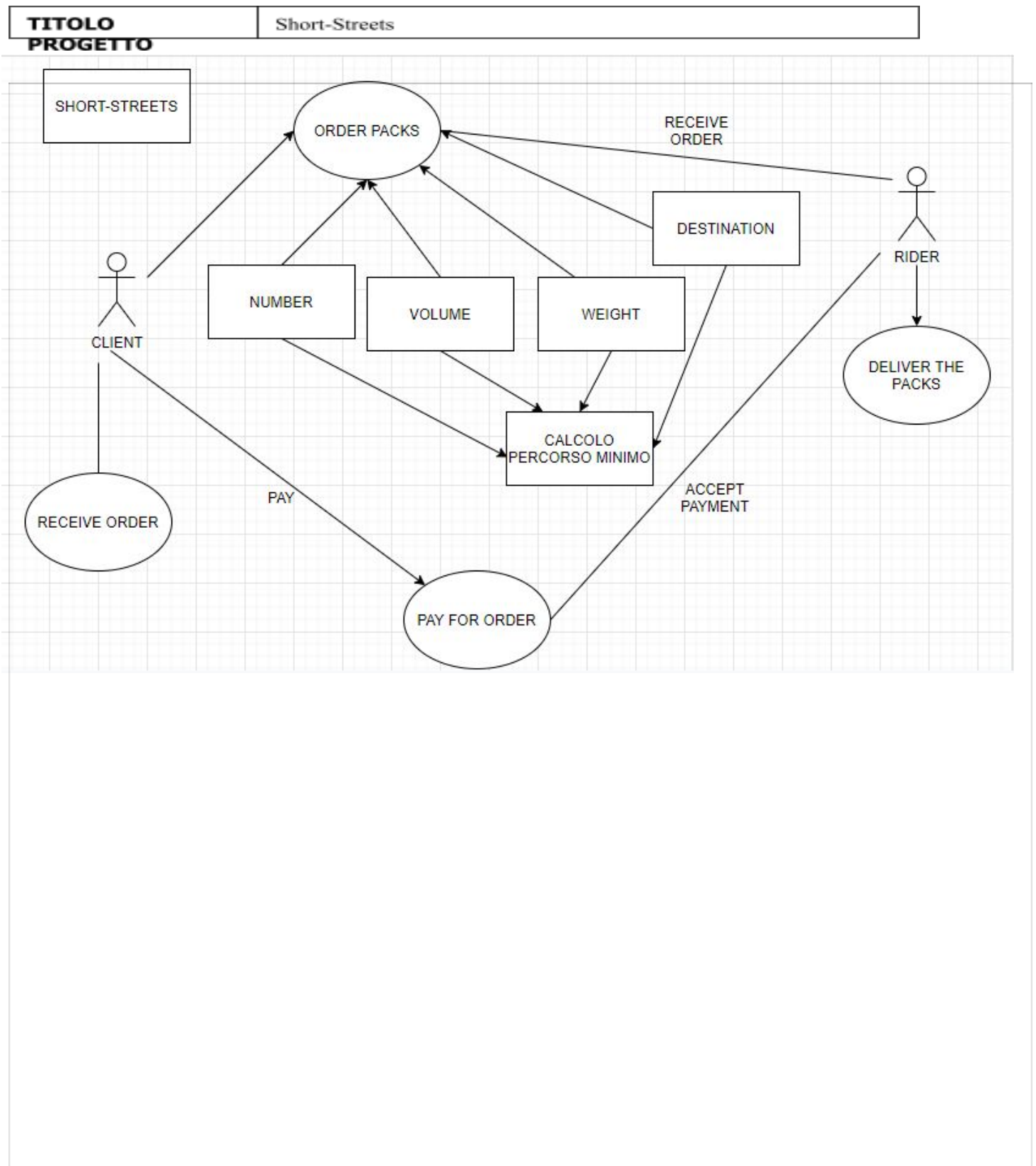
STUDENTE	SINTESI ATTIVITA' SVOLTA		RISULTATI	
	ore svolte	Liv. comp.	% compl.	VALUTAZIONE
Bissola Mattia	12	3	20%	
Giuggioli Daniel	15	3	50%	
Majid Zaccaria	12	3	20%	
Salvi Alessandro	15	3	75%	
Sonzogni Nicolò	18	3	75%	

VALUTAZIONE

POSITIVO
 POSITIVO
 POSITIVO
 BASE
 BASE
 NEGATIVO

	RISULTATO PROGETTO	LIVELLO ATTIVITA'	% COMPITI PERSONALI COMPLETATI
	>= 75%	3 (alto)	>= 50%
	>= 75%	2 (medio)	>= 75%
	>= 75%	1 (base)	> 90%
	>= 75%	1,2,3	MINORE DEL PREVISTO PER IL LIVELLO
	MINORE DEL 75%	1,2,3	MAGGIORI O UGUALI AL PREVISTO
	MINORE DEL 75%	1,2,3	MINORE DEL PREVISTO PER IL LIVELLO

Use Case Diagram



SCREENSHOTS

**TITOLO
PROGETTO**

Short-Streets

F1.D1:



F1.D2:



F2.D1:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace Short_Street
8 {
9     #reference
10     public class vertice
11     {
12         public int Vertice;
13         public List<int> Collegati;
14         public List<int> Costi;
15         #reference
16         public vertice(int vertice, List<int> collegati, List<int> costi)
17         {
18             Vertice = vertice;
19             Collegati = collegati;
20             Costi = costi;
21         }
22     }
23     #reference
24     public class soluzione
25     {
26         public int Prezzo;
27         public List<int> Percorso;
28         #reference
29         public soluzione(int prezzo, List<int> percorso)
30         {
31             Prezzo = prezzo;
32             Percorso = percorso;
33         }
34     }
35     #reference
36     public class Funzioni
37     {
38         #reference
39         public static soluzione CalcoloPercorso(List<vertice> dati)
40         {
41             List<int> minPercorso = new List<int>();
42             List<int> percorso = new List<int>();
43             int minPrezzo = int.MaxValue;
44             soluzione sol = new soluzione(minPrezzo, minPercorso);
45             int prezzo = 0;
46             percorso.Add(dati[0].Vertice);
47             asd(dati[0].Collegati, percorso, dati, prezzo, ref sol);
48             return (sol);
49         }
50     }
51 }
```

```

public static List<int> asd(list<int> coll, list<int> percorso, list<vertici> dati, int prezzo, ref soluzione sol)
{
    foreach (int i in coll)
    {
        if (!percorso.Contains(dati[i - 1].Vertice))
        {
            percorso.Add(i);
            var temp = dati[i - 1].Collegati;
            percorso = asd(temp, percorso, dati, prezzo, ref sol);
        }
    }
    if (percorso.Count == dati.Count)
    {
        prezzo = 0;
        if (dati[0].Collegati.Contains(percorso[percorso.Count - 1]))
        {
            int posizione = 0;
            int x = 0;
            foreach (int i in percorso)
            {
                vertici elemento = dati.Where(y => y.Vertice == i).FirstOrDefault();
                if (i == percorso[percorso.Count - 1])
                {
                    foreach (int a in dati[0].Collegati)
                    {
                        if (a == percorso[percorso.Count - 1])
                        {
                            prezzo = prezzo + elemento.Costi[0];
                            break;
                        }
                    }
                    if (sol.Prezzo > prezzo)
                    {
                        sol.Prezzo = prezzo;
                        sol.Percorso.Clear();
                        foreach (int g in percorso)
                        {
                            sol.Percorso.Add(g);
                        }
                    }
                    break;
                }
                else
                {
                    foreach (int a in elemento.Collegati)
                    {
                        if (a == percorso[x + 1])
                        {
                            prezzo = prezzo + elemento.Costi[posizione];
                            posizione = 0;
                            break;
                        }
                        posizione++;
                    }
                }
                x++;
            }
        }
        percorso.RemoveRange(percorso.Count - 1, 1);
    }
    else
    {
        percorso.RemoveRange(percorso.Count - 1, 1);
    }
    return (percorso);
}
}

```