Laboratorijska Vježba 9.2
Kreacijski paterni

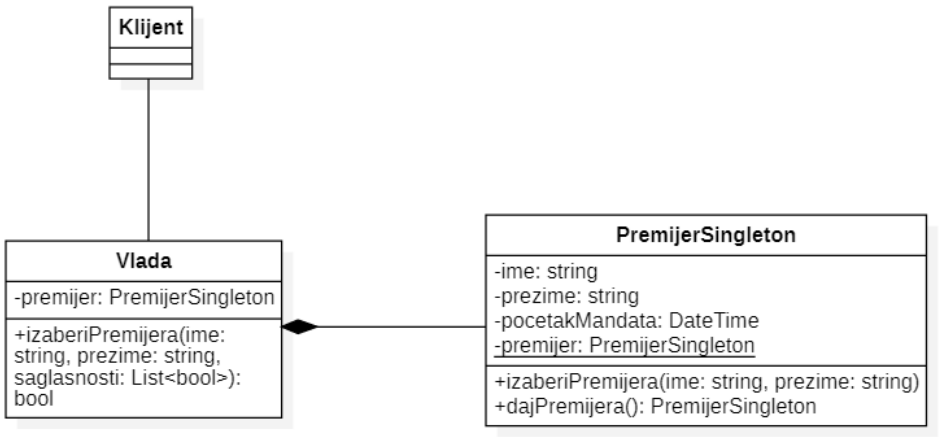
Napomena: Potrebno predznanje za vježbu 9 je gradivo obrađeno u predavanju 10-1.

1. *Singleton* patern

Za sljedeću definiciju sistema potrebno je osmisliti dijagram klasa koji poštuje *singleton* patern.

Klijent posjeduje mogućnost pristupa vladi Federacije Bosne i Hercegovine. Vladu čini više ministarstava i jedan premijer. Izbor premijera vrši se ukoliko više od dvije trećine stranaka daju svoju saglasnost za predloženog kandidata. Nakon izbora premijera, isti se ne može promijeniti do sljedećih izbora za četiri godine.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 1.



Slika 1. Dijagram klasa sa korištenjem singleton paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
    public class PremijerSingleton
    {
        String ime, prezime;
        DateTime pocetakMandata;
        static PremijerSingleton premijer;

        public string Ime { get => ime; set => ime = value; }
        public string Prezime { get => prezime; set => prezime = value; }
        public DateTime PocetakMandata { get => pocetakMandata; set =>
pocetakMandata = value; }

        public void izaberiPremijera(String ime, String prezime)
        {
            if (premijer == null || (DateTime.Now -
premijer.PocetakMandata).TotalDays >= 365 * 4)
            {
```

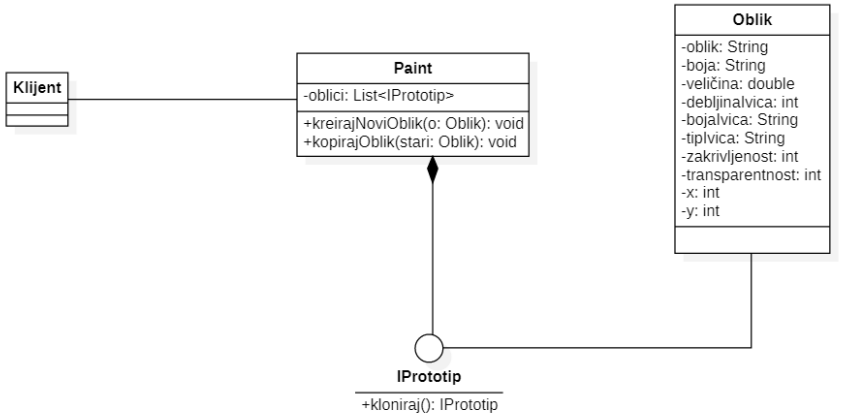
```
                premijer = new PremijerSingleton();
                premijer.Ime = ime;
                premijer.Prezime = prezime;
                premijer.PocetakMandata = DateTime.Now;
            }
        }
        public static PremijerSingleton dajPremijera()
        {
            return premijer;
        }
    }

    public class Vlada
    {
        PremijerSingleton premijer = new PremijerSingleton();
        public bool izaberiPremijera(string ime, string prezime, List<bool>
saglasnosti)
        {
            if (saglasnosti.FindAll(x => x == true).Count > 0.67 *
saglasnosti.Count)
            {
                premijer.izaberiPremijera(ime, prezime);
                return true;
            }
            return false;
        }
    }

        static void Main(string[] args)
        {
            Vlada vlada = new Vlada();
            List<bool> saglasnosti = new List<bool> { true, true };
            vlada.izaberiPremijera("Premijer", "1", saglasnosti);
            Console.Out.WriteLine("Premijer nakon prvog pokušaja postavljanja: " +
PremijerSingleton.dajPremijera().Ime + " " +
PremijerSingleton.dajPremijera().Prezime);

            vlada.izaberiPremijera("Premijer", "2", saglasnosti);
            Console.Out.WriteLine("Premijer nakon drugog pokušaja postavljanja: " +
PremijerSingleton.dajPremijera().Ime + " " +
PremijerSingleton.dajPremijera().Prezime);
        }
    }
```

2. *Prototype* patern

Za sljedeću definiciju sistema potrebno je osmisliti dijagram klasa koji poštuje *prototype* patern.

Klijent crta oblike na ekranu koristeći Paint. Kada odabere željeni oblik, njegovu boju, veličinu, debljinu ivica, boju ivica, tip ivica, zakrivljenost i transparentnost, te nakon što odabere njegovu lokaciju na ekranu, može izvršiti kopiranje objekta. Novi objekat identičan je prethodnome, s time što je njegova lokacija za 10 piksela pomaknuta udesno i udole. Nakon kopiranja objekta, klijent može mijenjati njegove karakteristike bez da to utiče na originalni objekat.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 2.



Slika 2. Dijagram klasa sa korištenjem prototype paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
    public interface IPrototip
    {
        IPrototip kloniraj();
    }

    public class Oblik : IPrototip
    {
        String oblik, boja, bojaIvica, tipIvica;
        double velicina;
        int debljinaIvica, zakrivljenost, transparentnost, x, y;

        public int X { get => x; set => x = value; }
        public int Y { get => y; set => y = value; }

        public Oblik (string o, string b, string b2, string t, double v, int d,
int z, int t2, int x2, int y2)
        {
            oblik = o; boja = b; bojaIvica = b2; tipIvica = t; velicina = v;
debljinaIvica = d; zakrivljenost = z; transparentnost = t2; X = x2; Y = y2;
        }

        public IPrototip kloniraj()
        {
            Oblik klon = new Oblik(oblik, boja, bojaIvica, tipIvica, velicina,
debljinaIvica, zakrivljenost, transparentnost, X + 10, Y + 10);
            return klon;
        }
    }

    public class Paint
    {
        List<IPrototip> oblici = new List<IPrototip>();
        public List<IPrototip> Oblici { get => oblici; set => oblici = value; }
        public void kreirajNoviOblik(Oblik o)
        {
            Oblici.Add(o);
        }
```

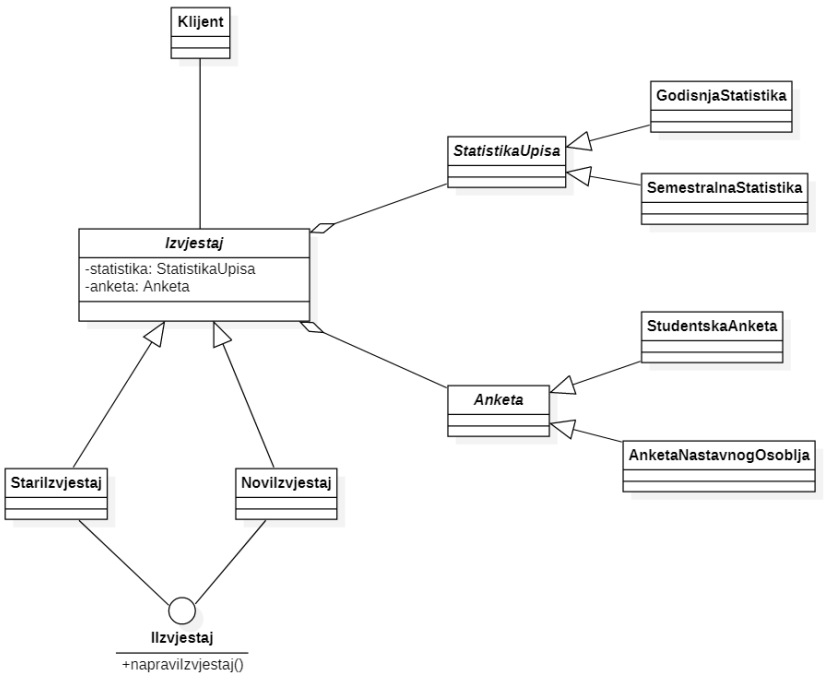
```
        public void kopirajOblik(Oblik o)
        {
            Oblici.Add(o.kloniraj());
        }
    }

        static void Main(string[] args)
        {
            Paint paint = new Paint();
            Oblik crveniTrougao = new Oblik("trougao", "crvena", "crna", "linija",
10, 1, 0, 0, 100, 150);

            paint.kreirajNoviOblik(crveniTrougao);
            paint.kopirajOblik(crveniTrougao);

            Console.Out.WriteLine("Oblici: ");
            foreach (Oblik oblik in paint.Oblici)
                Console.Out.WriteLine("X: " + oblik.X + ", Y: " + oblik.Y);
        }
```

3. *Factory method* patern

Za sljedeću definiciju sistema potrebno je osmisliti dijagram klasa koji poštuje *factory method* patern.

Klijent već godinama koristi sistem za izvještavanje studentske službe. Dosad su se izvještaji kreirali koristeći godišnju statistiku o upisu i rezultate studentskih anketa. Sada je potrebno omogućiti kreiranje izvještaja koristeći semestralnu statistiku o upisu i rezultate anketa nastavnog osoblja.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 3.



Slika 3. Dijagram klasa sa korištenjem factory method paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
    public abstract class StatistikaUpisa { }
    public class GodisnjaStatistika : StatistikaUpisa { }
    public class SemestralnaStatistika : StatistikaUpisa { }

    public abstract class Anketa { }
    public class StudentskaAnketa : Anketa { }
    public class AnketaNastavnogOsoblja : Anketa { }

    public interface IIzvjestaj
    {
        public void napraviIzvjestaj();
    }

    public abstract class Izvjestaj
    {
        StatistikaUpisa statistika;
        Anketa anketa;

        public StatistikaUpisa Statistika { get => statistika; set => statistika =
value; }
        public Anketa Anketa { get => anketa; set => anketa = value; }
    }

    public class StariIzvjestaj : Izvjestaj, IIzvjestaj
    {
        public void napraviIzvjestaj()
        {
            Statistika = new GodisnjaStatistika();
            Anketa = new StudentskaAnketa();
        }
    }

    public class NoviIzvjestaj : Izvjestaj, IIzvjestaj
    {
        public void napraviIzvjestaj()
        {
            Statistika = new SemestralnaStatistika();
            Anketa = new AnketaNastavnogOsoblja();
        }
    }

        static void Main(string[] args)
        {
            List<IIzvjestaj> izvještaji = new List<IIzvjestaj>()
            {
                new StariIzvjestaj(),
                new NoviIzvjestaj()
            };

            izvještaji.ForEach(izvještaj => izvještaj.napraviIzvjestaj());

            foreach (Izvjestaj izvještaj in izvještaji)
            {
                Console.Out.Write("Tip izvještaja: " +
izvještaj.GetType().ToString().Substring(14));
```

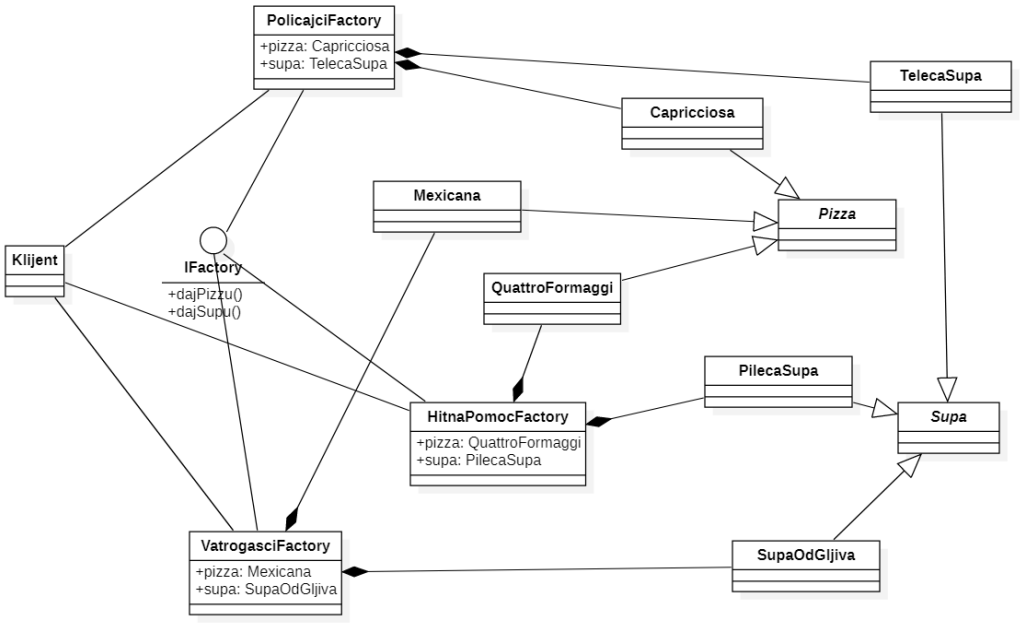
```
                Console.Out.Write(", Tip statistike: " +
izvještaj.Statistika.GetType().ToString().Substring(14));
                Console.Out.Write(", Tip ankete: " +
izvještaj.Anketa.GetType().ToString().Substring(14) + "\n\n");
            }
        }
```

4. *Abstract factory* patern

Za sljedeću definiciju sistema potrebno je osmisliti dijagram klasa koji poštuje *abstract factory* patern.

Klijent želi da omogući naručivanje hrane policajcima, vatrogascima i hitnoj pomoći. Sve tri kategorije mogu naručiti pizzu, pri čemu se za policajce pravi pizza capricciosa, za vatrogasce mexicana, a za hitnu pomoć quattro formaggi. Moguće je naručiti i supe, pri čemu je na menu za policajce teleća supa, za vatrogasce supa od gljiva, a za hitnu pomoć pileća supa.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 4.



Slika 4. Dijagram klasa sa korištenjem abstract factory paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
    public abstract class Pizza { }
    public class Capricciosa : Pizza { }
    public class Mexicana : Pizza { }
    public class QuattroFormaggi : Pizza { }

    public abstract class Supa { }
    public class TelecaSupa : Supa { }
    public class SupaOdGljiva : Supa { }
```

```
public class PilecaSupa : Supa { }

public interface IFactory
{
    public Pizza dajPizzu();
    public Supa dajSupu();
}

public class PolicajciFactory : IFactory
{
    Capricciosa pizza = new Capricciosa();
    TelecaSupa supa = new TelecaSupa();
    public Pizza dajPizzu()
    {
        return pizza;
    }
    public Supa dajSupu()
    {
        return supa;
    }
}

public class VatrogasciFactory : IFactory
{
    Mexicana pizza = new Mexicana();
    SupaOdGljiva supa = new SupaOdGljiva();
    public Pizza dajPizzu()
    {
        return pizza;
    }
    public Supa dajSupu()
    {
        return supa;
    }
}

public class HitnaPomocFactory : IFactory
{
    QuattroFormaggi pizza = new QuattroFormaggi();
    PilecaSupa supa = new PilecaSupa();
    public Pizza dajPizzu()
    {
        return pizza;
    }
    public Supa dajSupu()
    {
        return supa;
    }
}

    static void Main(string[] args)
    {
        List<IFactory> meniji = new List<IFactory>()
        {
            new PolicajciFactory(),
            new VatrogasciFactory(),
            new HitnaPomocFactory()
        };
```

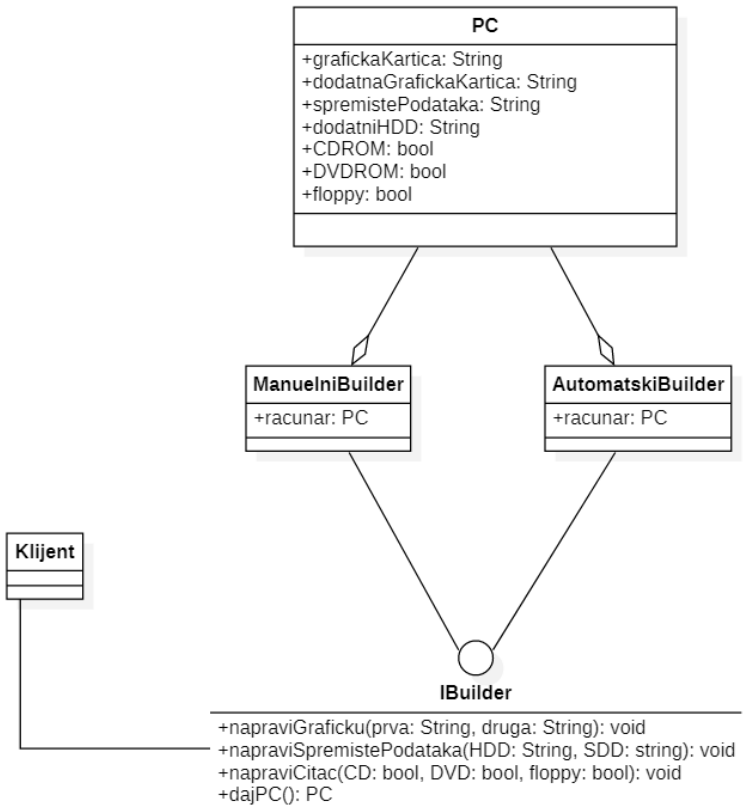
```
            foreach (IFactory menu in meniji)
            {
                Console.Out.Write("Meni: " +
menu.GetType().ToString().Substring(16));
                Console.Out.Write(", Pizza: " +
menu.dajPizzu().GetType().ToString().Substring(16));
                Console.Out.Write(", Supa: " +
menu.dajSupu().GetType().ToString().Substring(16) + "\n\n");
            }
        }
    }
```

5. *Builder* patern

Za sljedeću definiciju sistema potrebno je osmisliti dijagram klasa koji poštuje *builder* patern.

Klijent želi mogućnost da sam sklopi svoj PC koji će mu zatim biti dostavljen na kućnu adresu. Svaki PC mora imati grafičku karticu, a moguće je kupiti i dvije. Za spremanje podataka može se odabrati hard disk ili SSD. Ukoliko se odabere SSD, obavezno se mora odabrati i manji hard disk kao backup. Moguće je odabrati CD-ROM ili DVD-ROM, kao i čitač za floppy disk, ali isto nije neophodno. Ukoliko klijent ne želi sam sklopiti svoj PC, može odabrati neki od standardnih modela i specifikacija komponenti će biti automatski izvršena.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 5.



Slika 5. Dijagram klasa sa korištenjem builder paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
    public class PC
    {
        String grafickaKartica, dodatnaGrafickaKartica, spremistePodataka,
dodatniHDD;
        bool CDROM, DVDROM, floppy;

        public string GrafickaKartica { get => grafickaKartica; set =>
grafickaKartica = value; }
        public string DodatnaGrafickaKartica { get => dodatnaGrafickaKartica; set =>
dodatnaGrafickaKartica = value; }
        public string SpremistePodataka { get => spremistePodataka; set =>
spremistePodataka = value; }
        public string DodatniHDD { get => dodatniHDD; set => dodatniHDD = value; }
        public bool CDROM1 { get => CDROM; set => CDROM = value; }
        public bool DVDROM1 { get => DVDROM; set => DVDROM = value; }
        public bool Floppy { get => floppy; set => floppy = value; }
    }

    public interface IBuilder
    {
        public void napraviGraficku(String prva, String druga);
        public void napraviSpremistePodataka(String HDD, String SSD);
        public void napraviCitac(bool CDROM, bool DVDROM, bool floppy);
        public PC dajPC();
    }

    public class AutomatskiBuilder : IBuilder
    {
        PC racunar = new PC();
        public void napraviGraficku(String prva = "", String druga = "")
        {
            racunar.GrafickaKartica = "Najnovija NVIDIA kartica";
            racunar.DodatnaGrafickaKartica = "Nema";
        }
        public void napraviSpremistePodataka(String HDD = "", String SSD = "")
        {
            racunar.SpremistePodataka = "1 TB";
            racunar.DodatniHDD = "256 GB";
        }
        public void napraviCitac(bool CDROM = false, bool DVDROM = true, bool floppy
= false)
        {
            racunar.CDROM1 = false;
            racunar.DVDROM1 = true;
            racunar.Floppy = false;
        }
        public PC dajPC()
        {
            return racunar;
        }
    }

    public class ManuelniBuilder : IBuilder
    {
        PC racunar = new PC();
        public void napraviGraficku(String prva, String druga)
```

```
        {
            racunar.GrafickaKartica = prva;
            racunar.DodatnaGrafickaKartica = druga;
        }
        public void napraviSpremistePodataka(String HDD, String SSD)
        {
            if (SSD != "")
            {
                racunar.SpremistePodataka = SSD;
                racunar.DodatniHDD = HDD;
            }
            else
            {
                racunar.SpremistePodataka = HDD;
                racunar.DodatniHDD = "Nema";
            }

        }
        public void napraviCitac(bool CDROM, bool DVDROM, bool floppy)
        {
            racunar.CDROM1 = CDROM;
            racunar.DVDROM1 = DVDROM;
            racunar.Floppy = floppy;
        }
        public PC dajPC()
        {
            return racunar;
        }
    }

        static void Main(string[] args)
        {
            List<IBuilder> builderi = new List<IBuilder>()
            {
                new AutomatskiBuilder(),
                new ManuelniBuilder()
            };

            ((AutomatskiBuilder)builderi[0]).napraviGraficku();
            ((AutomatskiBuilder)builderi[0]).napraviSpremistePodataka();
            ((AutomatskiBuilder)builderi[0]).napraviCitac();

            ((ManuelniBuilder)builderi[1]).napraviGraficku("AMD", "NVIDIA");
            ((ManuelniBuilder)builderi[1]).napraviSpremistePodataka("256 GB", "");
            ((ManuelniBuilder)builderi[1]).napraviCitac(false, false, true);

            foreach (IBuilder builder in builderi)
            {
                PC pc = builder.dajPC();

                Console.Out.WriteLine("Vrsta buildera: " +
builder.GetType().ToString().Substring(8));
                Console.Out.WriteLine("Grafička kartica: " + pc.GrafickaKartica +
", Dodatna grafička kartica: " + pc.DodatnaGrafickaKartica);
                Console.Out.WriteLine("Spremište podataka: " + pc.SpremistePodataka
+ ", Dodatni HDD: " + pc.DodatniHDD);
                Console.Out.WriteLine("CDROM: " + pc.CDROM1 + ", DVDROM: " +
pc.DVDROM1 + ", Floppy: " + pc.Floppy + "\n");
            }
        }
```

6. Zadaci za samostalan rad

Programski kod primjera za sve paterne iz vježbe dostupan je u repozitoriju na sljedećem linku: https://github.com/ehlymana/OOADVjezbe, na *branchu* **lv8-2**.

Prepoznati i primijeniti odgovarajuće paterne na sljedeće opise sistema:

1. Klijent želi imati e-restoran. U ovom restoranu prodaju se hamburgeri, hot-dogovi i chicken nuggeti. Za djecu se prodaju mali hamburger i mali hot dog, a chicken nuggeti nisu u meniju. Za odrasle se prodaju veliki hamburger, veliki hot dog i velika porcija chicken nuggeta. Postoji i opcija studentskog menija, gdje se mogu kupiti samo srednji hot dog i mala porcija chicken nuggeta.

2. Klijent ima potrebu za korištenjem platforme za učenje. Ovu platformu mogu koristiti svi učenici i nastavnici škole. Škola ima četiri razreda i po dva odjeljenja za svaki razred. Svaki razred ima svoju Google platformu, koja uključuje Google Chat sobu i Google Meeting link za online nastavu. Budući da platformu za učenje koristi samo ova škola na ovaj način, potrebno je onemogućiti da se napravi još jedna škola i na taj način zloupotrijebe kupljeni resursi.