Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

RAPORT

Proiect de an

Tema: Implementarea design pattern-urilor prin dezvoltarea unei aplicații web de Eventuri.

Student: gr. TI-204, Lungu Ioan

Coordonator: asis.univ., Gaidau Mihai

Chișinău 2023

CUPRINS

INTRODUCERE 3
Context și motivare 3
Obiective 4
1 DESIGN PATTERN-URI 5
1.1 Definiții și concepte 5
1.2 Tipuri de design pattern-uri 5
1.3 Selecția design pattern-uri 7
2 DESCRIEREA APLICAȚIEI 8
2.1 Prezentarea generală 8
2.2 Scopul 8
2.3 Funcționalități 9
2.4 Arhitectura și tehnologii utilizate 10
3 IMPLEMENTAREA DESIGN PATTERN-URILOR 13
3.1 Model de proiectare creational 13
3.2 Model de proiectare structural 15
3.3 Model de proiectare comportamental 17
CONCLUZIE 19
BIBLIOGRAFIE 20

INTRODUCERE

Context și motivare

În ziua de astăzi, aplicațiile web sunt peste tot și fac parte din cotidianul nostru. O aplicație web de tip planificator de evenimente reprezintă o unealtă esențială pentru gestionarea și organizarea activităților personale și profesionale. Aceasta oferă utilizatorilor posibilitatea de a-și organiza evenimentele, de a-și stabili prioritățile și de a planifica eficient activitățile viitoare.

Implementarea design pattern-urilor în dezvoltarea unei aplicații web de tip planificator de evenimente este deosebit de importantă și aduce numeroase beneficii. Design pattern-urile reprezintă soluții recunoscute și testate pentru probleme comune întâlnite în dezvoltarea software. Ele oferă un set de principii și structuri arhitecturale care facilitează dezvoltarea unui cod modular, flexibil și ușor de întreținut.

Prin utilizarea design pattern-urilor într-o aplicație web de tip planificator de evenimente, se obțin următoarele avantaje:

Reutilizarea codului: Design pattern-urile permit extragerea și refolosirea codului, reducând astfel complexitatea și efortul necesar pentru dezvoltarea aplicației. Aceasta conduce la o mai mare eficiență și productivitate în procesul de dezvoltare.

Modularitatea și extensibilitatea: Design pattern-urile încurajează o arhitectură modulară, unde fiecare componentă își are propria responsabilitate și poate fi modificată sau extinsă independent. Acest lucru facilitează adăugarea de noi funcționalități și integrarea cu alte sisteme sau servicii.

Îmbunătățirea calității software-ului: Utilizarea design pattern-urilor contribuie la crearea unui cod mai coerent, mai ușor de înțeles și de întreținut. Ele oferă structuri și convenții standardizate, ceea ce duce la o mai bună organizare a codului și la reducerea erorilor și defectelor.

Flexibilitatea și adaptabilitatea: Implementarea design pattern-urilor permite aplicației web de tip planificator de evenimente să fie mai flexibilă și să se adapteze mai ușor la schimbări și cerințe noi. Aceasta facilitează actualizarea și extinderea aplicației în viitor, fără a afecta funcționalitatea existentă.

Aplicația web de tip planificator de evenimente rezolvă o serie de probleme și aduce numeroase beneficii utilizatorilor. Iată câteva dintre acestea:

Gestionarea eficientă a timpului: O aplicație web de planificare a evenimentelor ajută utilizatorii să-și gestioneze și să-și organizeze activitățile într-un mod structurat și ușor de urmărit. Aceasta contribuie la prioritatea sarcinilor, evitarea suprapunerilor și planificarea eficientă a timpului disponibil.

Notificări și avertizări: Aplicația de planificare a evenimentelor poate trimite notificări și avertizări utilizatorilor pentru a-i reaminti despre evenimentele programate. Acestea pot fi personalizate în funcție de preferințele utilizatorului, contribuind la evitarea uitării și la respectarea termenelor limită sau a

angajamentelor asumate.

Colaborare și partajare: Utilizatorii pot colabora și partaja evenimentele cu alți utilizatori, facilitând coordonarea activităților și creșterea eficienței colaborării.

Accesibilitate și sincronizare între dispozitive: Utilizatorii pot accesa și actualiza evenimentele lor de pe diferite dispozitive, asigurând sincronizarea datelor și disponibilitatea lor în orice moment și de oriunde.

Personalizare și vizualizare intuitivă: Aplicația permite utilizatorilor să personalizeze modul de vizualizare, să adauge etichete sau categorii pentru evenimente și să adapteze interfața la preferințele și nevoile lor individuale.

Backup și recuperare: O aplicație web de gestionare a evenimentelor poate oferi funcționalități de backup și recuperare a datelor, asigurând protejarea informațiilor și evitarea pierderii evenimentelor în caz de probleme tehnice sau înlocuirea dispozitivelor.

Prin implementarea design pattern-urilor în aplicația web de tip planificator de evenimente, se asigură dezvoltarea unei aplicații eficiente, fiabile și ușor de întreținut, care răspunde nevoilor și cerințelor utilizatorilor.

Obiective

Scopul este de a identifica și selecta modelele de proiectare potrivite pentru implementarea unei aplicații web de tip evenimente. Acest lucru implică studierea și înțelegerea diferitelor modele de proiectare disponibile și identificarea celor mai adecvate pentru nevoile aplicației de evenimente. Implementarea modelelor de proiectare selectate în cadrul aplicației web de evenimente este un alt obiectiv important. Acest lucru presupune aplicarea principiilor și structurilor asociate modelelor de proiectare în dezvoltarea aplicației, asigurându-ne că acestea sunt integrate corespunzător și contribuie la îmbunătățirea calității și modularității aplicației. Obiectivul este de a evalua impactul implementării modelelor de proiectare asupra aplicației de evenimente. Acest lucru poate include analiza performanței aplicației, evaluarea modularității și flexibilității acesteia, precum și identificarea avantajelor și dezavantajelor aduse de utilizarea modelelor de proiectare. Un alt obiectiv important este de a analiza beneficiile aduse de implementarea modelelor de proiectare în aplicația de evenimente pentru utilizatori. Acest lucru poate include evaluarea îmbunătățirilor în ceea ce privește experiența utilizatorului, ușurința în utilizarea aplicației, eficiența în gestionarea și planificarea evenimentelor, precum și posibilitatea de colaborare și partajare a calendarului de evenimente. Scopul final este de a documenta rezultatele obținute într-un raport și de a le prezenta într-un mod clar și coerent. Acest lucru implică redactarea unui raport detaliat, care să cuprindă descrierea modelelor de proiectare, procesul de implementare, evaluarea rezultatelor și analiza beneficiilor aduse utilizatorilor.

1 DESIGN PATTERN-URI

1.1 Definiții și concepte

Un design pattern (sau model de proiectare) reprezintă o soluție generală pentru o problemă comună întâlnită în dezvoltarea software. Acesta reprezintă o abstractizare a unei soluții testate și recunoscute, care poate fi aplicată în diverse contexte și în diferite limbaje de programare. Design patterns-urile sunt dezvoltate de experți în domeniul dezvoltării software și sunt documentate în literatura de specialitate, cum ar fi cartea "Design Patterns: Elemente ale Obiectelor Software Reutilizabile" scrisă de Erich Gamma, Richard Helm, Ralph Johnson și John Vlissides (cunoscută și sub numele de "Cartea Design Patterns" sau "GOF"). Importanța design patterns-urilor în dezvoltarea software constă în următoarele aspecte:

reutilizarea soluțiilor testate;

standardizarea și coerența;

modularitatea și extensibilitatea;

performanța și eficiența;

comunitatea și cunoașterea colectivă.

Design patterns-urile reprezintă soluții care au fost testate și validate în diferite contexte și aplicații. Prin utilizarea lor, dezvoltatorii pot evita reinventarea roții și pot beneficia de abordări recunoscute și eficiente pentru problemele comune. Acest lucru conduce la o dezvoltare mai rapidă și mai eficientă a software-ului. Furnizează un vocabular comun și o structură standardizată pentru rezolvarea anumitor probleme. Acest lucru facilitează comunicarea între dezvoltatori și permite o mai bună înțelegere a codului. Utilizarea design patterns-urilor contribuie la crearea unui cod coerent și ușor de înțeles, care este mai ușor de întreținut și de extins în viitor. Design patterns-urile promovează o abordare modulară în dezvoltarea software. Acestea permit împărțirea aplicației în componente independente, fiecare având propria responsabilitate și funcționalitate bine definită. Acest lucru face ca dezvoltarea să fie mai flexibilă și mai ușor de gestionat, facilitând adăugarea de noi funcționalități și modificarea componentelor existente fără a afecta întregul sistem. Sunt adesea optimizate pentru performanță și eficiență. Ele oferă soluții care minimizează utilizarea resurselor și reduc complexitatea algoritmilor. Prin implementarea design patterns-urilor, dezvoltatorii pot obține aplicații mai rapide, mai robuste și mai scalabile. Design patterns-urile sunt parte integrantă a cunoașterii colective a comunității dezvoltatorilor software. Utilizarea lor facilitează colaborarea și schimbul de experiență între dezvoltatori, permițându-le să construiască pe baza cunoștințelor și soluțiilor existente în domeniu.

1.2 Tipuri de design pattern-uri

Există numeroase modele de proiectare utilizate în dezvoltarea aplicațiilor web. Câteva

dintre cele mai relevante modele de proiectare în acest context sunt:

Singleton Pattern este utilizat pentru a se asigura că există o singură instanță a unei clase în întreaga aplicație. Acest pattern este folosit pentru clasele care trebuie să aibă o unică instanță și oferă acces global la acea instanță. În cazul nostru, Singleton Pattern este utilizat pentru a asigura că avem o singură instanță a clasei de gestionare a listei de evenimente.

Factory Method Pattern oferă o modalitate de creare a obiectelor fără a specifica clasa exactă a obiectului care va fi creat. În loc să folosim constructori direcți, acest pattern utilizează o metodă de fabrică pentru a crea obiecte. În aplicația dată, Factory Method Pattern este utilizat pentru a crea obiecte de tip eveniment, indiferent dacă acestea sunt evenimente simple sau evenimente cu termen limită.

Abstract Factory Pattern furnizează o interfață pentru crearea familii de obiecte conexe, fără a specifica clasele concrete ale acestor obiecte. Acest pattern ne permite să creăm obiecte care sunt legate între ele sau depind una de cealaltă, folosind o fabrică abstractă. În cazul nostru, Abstract Factory Pattern este utilizat pentru a crea diferite tipuri de liste de evenimente.

Decorator Pattern este utilizat pentru a adăuga funcționalități suplimentare la un obiect existent, fără a modifica structura sa de bază. Acest pattern permite extinderea comportamentului unui obiect prin încapsularea acestuia într-un decorator, care oferă funcționalități adiționale. În aplicația dată, Decorator Pattern este folosit pentru a adăuga prioritate la evenimente.

Adapter Pattern este utilizat pentru a conecta două clase sau interfețe care nu se potrivesc între ele. Acest pattern permite convertirea interfeței unei clase într-o altă interfață pe care o așteaptă un client. În aplicația dată, Adapter Pattern este utilizat pentru a adapta o clasă existentă de evenimente la o interfață comună.

Bridge Pattern separă abstractizarea de implementare, permițându-le să evolueze independent unul de celălalt. Acest pattern este folosit pentru a gestiona o relație complexă între două clase prin intermediul unei abstracții. În cazul nostru, Bridge Pattern este utilizat pentru a permite gestionarea diferitelor tipuri de evenimente și a detaliilor specifice ale acestora.

Facade Pattern oferă o interfață simplificată pentru a accesa un sistem complex sau un set de clase. Acest pattern ascunde complexitatea din spatele unei interfețe simple și unificate, permițând utilizatorilor să interacționeze cu sistemul fără a cunoaște detaliile interne. În aplicația dată, Facade Pattern este utilizat pentru a oferi o interfață simplificată pentru adăugarea și sortarea evenimentelor.

Observer Pattern este folosit pentru a stabili o relație de tip observator-subiect între obiecte. Acest pattern permite notificarea automată a observatorilor atunci când se produce o schimbare în starea subiectului. În aplicația dată, Observer Pattern este utilizat pentru a notifica observatorii despre modificările din lista de evenimente și pentru a permite reacționarea și actualizarea acestora.

Command Pattern: Command Pattern separă o comandă de executarea sa, permițând manipularea și stocarea acesteia ca un obiect. Acest pattern este folosit pentru a encapsula o operație sau o acțiune într-un obiect, permițând astfel efectuarea acelei acțiuni într-un moment ulterior sau de către un alt obiect. În aplicația dată, Command Pattern este utilizat pentru a stoca și executa comenzi legate de operațiile asupra listei de evenimente.

Strategy Pattern permite schimbarea algoritmului sau strategiei utilizate de un obiect fără a afecta obiectul în sine. Acest pattern este utilizat pentru a separa logica specifică a algoritmului de clasa principală, permițând utilizarea și schimbarea facilă a diferitelor strategii. În aplicația dată, Strategy Pattern este utilizat pentru a oferi diferite strategii de sortare a listei de evenimente, precum sortarea după nume sau sortarea după dată.

1.3 Selecția design pattern-uri

Alegerea design pattern-urilor potrivite pentru implementarea unei aplicații Calendar poate fi influențată de mai mulți factori. Iată câteva criterii comune utilizate în selecția design pattern-urilor pentru acest tip de aplicație. Design pattern-urile trebuie să se potrivească structurii și complexității aplicației Calendar. Dacă aplicația este complexă și implică interacțiuni multiple între obiecte și componente, atunci design pattern-urile cum ar fi Observer, Iterator și Facade ar putea fi utile pentru a gestiona comunicarea și structura aplicației.

Dacă aplicația Calendar trebuie să fie ușor de extins și de întreținut în viitor, design pattern-urile cum ar fi Builder și Strategy pot fi folosite pentru a permite adăugarea ușoară de noi funcționalități și pentru a schimba comportamentul aplicației fără a afecta codul existent. acă există componente sau servicii care trebuie să fie unice și accesibile global în întreaga aplicație Calendar, design pattern-ul Singleton poate fi util pentru a asigura că o singură instanță a acestor componente este creată și utilizată în cadrul aplicației.

Dacă aplicația Calendar implică accesul la resurse costisitoare sau la servicii externe, design pattern-ul Proxy poate fi utilizat pentru a controla și gestiona accesul la aceste resurse sau servicii, oferind o abstracție intermediară și adăugând funcționalități suplimentare, cum ar fi caching-ul sau securitatea. Dacă aplicația Calendar trebuie să reacționeze la evenimente sau modificări în starea unor obiecte sau componente, design pattern-ul Observer poate fi folosit pentru a notifica și sincroniza automat alte obiecte interesate de aceste evenimente.

Acestea sunt doar câteva exemple de criterii care pot fi luate în considerare în alegerea design pattern-urilor potrivite pentru implementarea unei aplicații Calendar. În general, se recomandă să se analizeze în mod atent cerințele și caracteristicile specifice ale aplicației, pentru a identifica design pattern-urile care corespund cel mai bine nevoilor și scopului proiectului.

2 DESCRIEREA APLICAȚIEI

2.1 Prezentarea generală

Într-o lume în continuă mișcare și cu un ritm alert, organizarea eficientă a timpului devine din ce în ce mai importantă pentru fiecare individ. Aplicația web Calendar, creată cu grijă și atenție de către echipa noastră, a fost concepută cu un scop precis în minte: să ofere utilizatorilor o platformă extrem de eficientă și ușor de utilizat pentru gestionarea evenimentelor, programărilor și întâlnirilor lor.

În zilele noastre, tehnologia digitală a devenit o parte indispensabilă a vieții noastre cotidiene. Cu toate acestea, cu un volum uriaș de informații și distrageri constante la îndemână, este ușor să ne pierdem într-un labirint de programări și evenimente. De aceea, un calendar bine organizat este crucial pentru a ne păstra pe drumul cel bun și pentru a nu pierde evenimente importante sau întâlniri cheie.

Calendarul web reprezintă soluția ideală pentru această provocare. Cu o interfață intuitivă și prietenoasă, utilizatorii pot adăuga cu ușurință evenimente, programări sau întâlniri în calendarul lor personal. Fie că este vorba de întâlniri de afaceri, evenimente sociale sau simpla organizare a programului zilnic, aplicația noastră le oferă utilizatorilor un instrument de gestionare centralizat și ușor de utilizat.

Unul dintre principalele avantaje ale Calendarului web este funcția de notificare personalizată. Utilizatorii pot seta alerte și notificări pentru evenimentele lor, astfel încât să fie avertizați în avans cu privire la întâlniri sau evenimente importante. Această caracteristică esențială asigură că niciun eveniment crucial nu va fi uitat sau trecut cu vederea.

În plus, Calendarul oferă și alte funcționalități utile pentru o experiență completă și eficientă de gestionare a timpului. Utilizatorii pot crea liste de sarcini, adăuga note sau observații la evenimente și pot vizualiza programul lor zilnic, săptămânal sau lunar. Aceste caracteristici adiționale îmbunătățesc eficiența și organizarea generală a utilizatorilor, oferindu-le un instrument puternic pentru gestionarea timpului.

Aplicația reprezintă soluția ideală pentru toți cei care doresc să-și organizeze eficient timpul și să nu piardă evenimente sau întâlniri importante. Cu o interfață intuitivă, notificări personalizate și o gamă largă de funcționalități, această aplicație oferă utilizatorilor o platformă eficientă și ușor de utilizat pentru gestionarea evenimentelor lor. Indiferent dacă este vorba de o întâlnire de afaceri crucială sau de o simplă întâlnire cu prietenii, Calendarul vă ajută să fiți la curent și să vă organizați în mod corespunzător viața.

2.2 Scopul

Aplicația a fost dezvoltată cu un scop precis în minte: să ofere utilizatorilor o modalitate intuitivă și ușor de utilizat pentru gestionarea și organizarea evenimentelor lor. În lumea aglomerată și plină de activități în care trăim, este crucial să putem programa și planifica în mod eficient pentru a ne optimiza timpul și pentru a fi mai organizați.

Prin intermediul aplicației noastre, utilizatorii au posibilitatea de a crea și gestiona evenimente

într-un mod simplu și intuitiv. Aceasta înseamnă că pot adăuga rapid și ușor informații despre întâlniri, termene limită, activități sau orice alt eveniment important în calendarul lor personal. Cu doar câteva clicuri, utilizatorii pot stabili data, ora, durata și detalii suplimentare pentru fiecare eveniment în parte.

Unul dintre avantajele majore ale aplicației Calendar este posibilitatea de a programa notificări și alerte. Utilizatorii pot seta reminder-e personalizate pentru fiecare eveniment, astfel încât să primească notificări înainte de momentul programat. Aceste notificări sunt extrem de utile pentru a nu uita evenimente importante și pentru a fi mereu la curent cu programul personal. Pe lângă gestionarea evenimentelor individuale, aplicația Calendar oferă și funcționalități avansate pentru organizarea și planificarea generală. Utilizatorii pot vizualiza calendarul lor în diferite moduri, cum ar fi modul zilnic, săptămânal sau lunar, pentru a avea o perspectivă de ansamblu asupra programului lor. Această funcționalitate permite utilizatorilor să-și distribuie în mod eficient timpul și să evite suprapunerea sau programarea conflictuală a evenimentelor.

De asemenea, aplicația oferă posibilitatea de a partaja evenimentele cu alți utilizatori. Aceasta înseamnă că puteți invita prietenii, colegii sau membrii familiei la evenimentele dvs. și puteți sincroniza calendarul cu alte persoane pentru o mai bună coordonare și comunicare.

Aplicația Calendar are un scop clar și anume de a facilita gestionarea și organizarea evenimentelor utilizatorilor. Prin intermediul funcționalităților intuitive și ușor de utilizat, utilizatorii pot programa și planifica în mod eficient, optimizându-și timpul și fiind mai organizați. Indiferent dacă este vorba despre întâlniri de afaceri, activități personale sau orice alt tip de eveniment, aplicația Calendar este instrumentul ideal pentru a vă ajuta să fiți mereu la curent și bine organizat.

2.3 Funcționalități

În cadrul aplicației, utilizatorii au la dispoziție o serie de funcționalități avansate pentru a-și gestiona și organiza evenimentele într-un mod eficient:

- gestionarea setărilor utilizatorilor;
- gestionarea tagurilor;
- gestionarea evenimentelor;
- editarea setărilor utilizatorului din profil;
- trimiterea notificărilor despre eveniment;
- afișarea calendarului.

Utilizatorii pot personaliza aplicația conform preferințelor lor individuale. Aceasta include configurarea formatului și limbii calendarului, preferințele de notificare, setările de afișare etc. Prin adaptarea aplicației la nevoile și preferințele utilizatorilor, aceștia pot avea o experiență personalizată și plăcută în utilizarea calendarului. Aplicația permite utilizatorilor să atribuie etichete sau taguri fiecărui

eveniment în parte. Această funcționalitate permite organizarea și clasificarea evenimentelor în funcție de categorii relevante, cum ar fi "muncă", "personal", "important" etc.

Utilizatorii pot găsi rapid evenimentele dintr-o anumită categorie prin filtrarea după taguri. Utilizatorii au posibilitatea de a adăuga, edita și șterge evenimente în calendar. Aceasta implică specificarea detaliilor evenimentului, cum ar fi data, ora, locația și descrierea.

Utilizatorii pot programa întâlniri de afaceri, evenimente sociale, activități personale și multe altele, având control total asupra programului lor. Pot modifica setările lor individuale direct din profilul lor. Aceasta include actualizarea informațiilor personale, parola, preferințele de notificare și alte setări specifice utilizatorului. Prin această funcționalitate, utilizatorii pot ajusta și adapta aplicația pentru a se potrivi nevoilor lor în cursul timpului.

Aplicația poate trimite notificări utilizatorilor pentru a-i aminti de evenimentele programate. Utilizatorii pot alege să primească notificări prin e-mail sau prin notificări push direct pe dispozitivul lor mobil. Această funcționalitate asigură că utilizatorii vor fi la curent și nu vor uita evenimentele importante din calendarul lor. Utilizatorii pot vizualiza calendarul în diferite moduri, dar și într-un mod personalizat în funcție de preferințele lor. Aceasta include afișarea lunară, săptămânală și zilnică a evenimentelor. Utilizatorii pot selecta intervalul de timp dorit și pot vizualiza programul lor într-un mod clar și concis.

Prin intermediul acestor funcționalități avansate, aplicația oferă utilizatorilor posibilitatea de a gestiona și organiza evenimentele într-un mod eficient. Utilizatorii pot fi mai organizați, mai productivi și mai bine pregătiți pentru fiecare eveniment.

2.4 Arhitectura și tehnologii utilizate

JavaScript (JS) este un limbaj de programare de nivel înalt, interpretat și orientat pe obiecte, care este utilizat în principal pentru dezvoltarea aplicațiilor web. Este un limbaj de scripting flexibil și puternic, care permite programatorilor să creeze interactivitate și să manipuleze elemente dinamic în paginile web.

Aplicația prezentată în codul dat demonstrează utilizarea JavaScript în dezvoltarea unei aplicații de gestionare a evenimentelor într-un calendar. Aceasta utilizează mai multe concepte și modele de design, precum strategii de sortare, clase, observatori, comenzi și multe altele.Principalele funcționalități ale aplicației sunt următoarele:

- Adăugarea și ștergerea de evenimente: Utilizatorii pot adăuga noi evenimente în calendar și le pot șterge din lista existentă.
- Sortarea evenimentelor: Aplicația permite sortarea evenimentelor după nume sau dată, în ordine crescătoare sau descrescătoare. Utilizează strategii de sortare pentru a aplica algoritmul corespunzător.

- Marcare evenimente ca finalizate sau nefinalizate: Utilizatorii pot marca evenimentele ca finalizate sau nefinalizate prin simpla lor selectare.
- Randarea și afișarea evenimentelor: Lista evenimentelor este randată și afișată în interfața utilizatorului, folosind elemente HTML create și modificate dinamic.

În aplicația dată, avem diverse entități care gestionează diferite aspecte ale funcționalității. Aceste entități sunt:

- SortByNameStrategy: Această clasă gestionează sortarea evenimentelor după nume. Metoda sort primește lista de evenimente și un indicator boolean care specifică dacă se dorește sortarea în ordine crescătoare sau descrescătoare.
- SortByDateStrategy: Această clasă gestionează sortarea evenimentelor după dată. Metoda sort primește lista de evenimente și un indicator boolean care specifică dacă se dorește sortarea în ordine crescătoare sau descrescătoare.
- Event: Această clasă reprezintă un singur eveniment din calendar. Are o proprietate description care reține descrierea evenimentului și o proprietate completed care indică dacă evenimentul este finalizat sau nu. Clasa are și o metodă toggleCompletion care inversează starea de finalizare a evenimentului.
- EventList: Această clasă reprezintă lista de evenimente din calendar. Are o proprietate events care este un array ce conține evenimentele, o proprietate sortStrategy care reține strategia de sortare utilizată și o proprietate asc care indică dacă sortarea se face în ordine crescătoare sau descrescătoare. Clasa are metode pentru adăugarea și ștergerea de evenimente, precum și pentru setarea și aplicarea strategiei de sortare.
- DueDateEvent: Această clasă reprezintă un eveniment cu o dată limită. Mosteneste clasa Event și adaugă o proprietate dueDate care reține data limită a evenimentului. Clasa are și o metodă isOverdue care verifică dacă evenimentul este depășit și nu este finalizat.
- EventRenderer: Această clasă este responsabilă de randarea listei de evenimente în interfața utilizatorului (UI). Primeste ca parametri o instanță a clasei EventList și un ID de element HTML care reprezintă containerul în care vor fi afișate evenimentele. Metoda render sortează evenimentele și le afișează în containerul specificat în interfața utilizatorului. De asemenea, adaugă evenimente pentru acțiuni cum ar fi ștergerea unui eveniment sau comutarea stării de finalizare a unui eveniment.
- SingletonEventList: Această clasă implementează Singleton Pattern și gestionează lista de evenimente pentru aplicație. Asigură existența unei singure instanțe a listei de evenimente și oferă metode pentru adăugarea și ștergerea de evenimente.
- EventFactory: Această clasă implementează Factory Method Pattern și furnizează o metodă createEvent pentru crearea de obiecte de tip Event sau DueDateEvent în funcție de prezența unei date limită.
- AbstractEventListFactory și EventListFactory: Aceste clase implementează Abstract Factory Pattern și

definesc o metodă createEventList pentru crearea de instanțe ale clasei EventList.

- SingletonEventListFactory: Această clasă extinde AbstractEventListFactory și implementează metoda createEventList pentru crearea unei instanțe a clasei SingletonEventList.
- EventWithPriority: Această clasă extinde clasa Event și implementează Decorator Pattern. Adaugă proprietatea priority la un eveniment existent și redirectează apelurile metodei toggleCompletion către evenimentul decorat.
- LegacyEvent și LegacyEventAdapter: Aceste clase implementează Adapter Pattern pentru a adapta obiectele de tip LegacyEvent la interfața evenimentelor din aplicație.
- EventListRenderer, HtmlEventListRenderer și TextEventListRenderer: Aceste clase implementează Bridge Pattern pentru a separa implementarea randării evenimentelor de interfața de randare specifică (HTML sau text).
- EventListObserver și EventListView: Aceste clase implementează Observer Pattern pentru a permite observarea și notificarea modificărilor din lista de evenimente.
- EventListCommand: Această clasă implementează Command Pattern și gestionează lista de comenzi pentru adăugarea și ștergerea de evenimente din lista de evenimente.
- EventFacade: Această clasă servește ca o interfață simplificată pentru a interacționa cu lista de evenimente și funcționalitățile asociate. Folosește clasele și metodele definite mai sus pentru a adăuga evenimente, a sorta evenimentele și a le afișa în interfața utilizatorului.

Aceste entități colaborează împreună pentru a gestiona funcționalitatea aplicației de gestionare a evenimentelor.

3 IMPLEMENTAREA DESIGN PATTERN-URILOR

3.1 Model de proiectare creational

Modelele de design creationale sunt un subset al modelelor de design utilizate în proiectarea software-ului. Acestea se concentrează pe procesul de creare și inițializare a obiectelor, oferind o abordare flexibilă și extensibilă pentru gestionarea creării acestora.Principala preocupare a modelelor de design creationale este de a reduce cuplajul dintre clasele care creează obiecte și clasele care le utilizează, oferind un nivel ridicat de abstractizare și modularitate. Acestea oferă o metodă standardizată și elegantă de creare a obiectelor, astfel încât să se poată evita dependența strânsă de clasele concrete și să se poată extinde și modifica procesul de creare în mod ușor.Prin utilizarea modelelor creationale, putem îmbunătăți reutilizarea codului și putem facilita testarea și întreținerea aplicațiilor. Aceste modele sunt utile în situații în care trebuie să creăm obiecte complexe, cu configurații diferite sau atunci când avem nevoie să izolăm procesul de creare într-o singură clasă sau metodă.

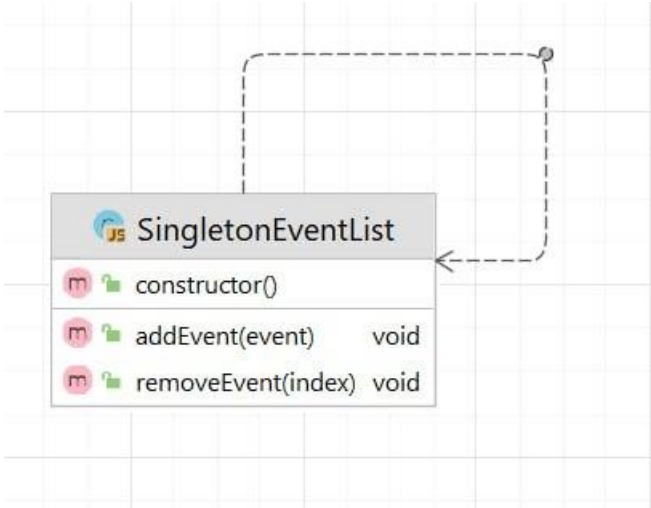

Figura 3.1 - Structura Singleton

Clasa SingletonEventList este implementată ca un singleton pentru a asigura că există o singură instanță a listei de evenimente în întreaga aplicație.Constructorul clasei este privat, astfel încât să nu se poată crea instanțe directe ale clasei.Clasa conține o variabilă statică instance, care stochează instanța unică a clasei.Metoda constructorului constructor() verifică dacă există deja o instanță și returnează acea instanță, în loc să creeze una nouă.

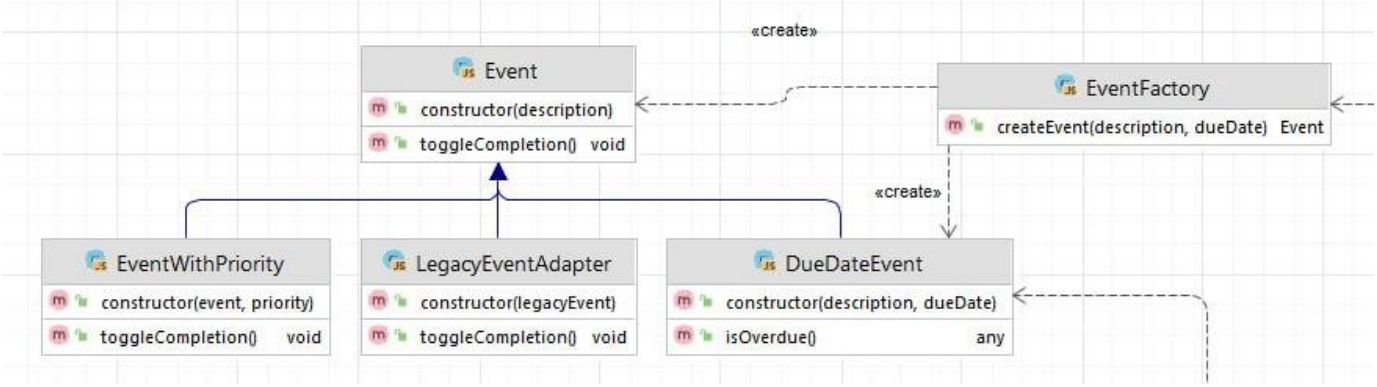

Figura 3.2 - Structura Factory Method

Clasa EventFactory oferă o metodă createEvent pentru a crea obiecte de tip Event sau DueDateEvent în funcție de prezența unei date limită.Metoda createEvent primește descrierea evenimentului și opțional, o dată limită.În funcție de prezența datei limită, metoda creează și returnează un obiect de tipul corespunzător (Event sau DueDateEvent).

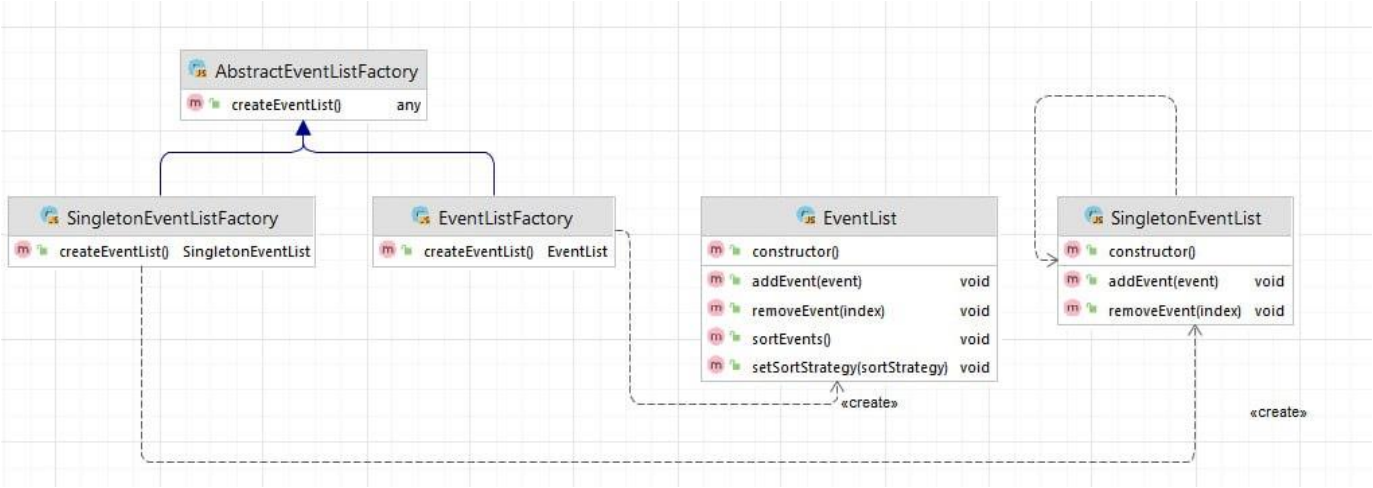

Figura 3.3 - Structura Abstract Factory

Clasa abstractă AbstractEventListFactory definește metoda createEventList pentru crearea de instanțe ale clasei EventList.Clasele EventListFactory și SingletonEventListFactory extind clasa abstractă și implementează metoda createEventList.Astfel, aceste clase furnizează o interfață pentru crearea de instanțe ale clasei EventList sau SingletonEventList, în funcție de necesități.Utilizarea Abstract Factory Pattern permite schimbul facil între diferite implementări ale listei de evenimente, fără a modifica codul existent care folosește aceste instanțe.

Respectiv ca o concluzie la cele evidențiate mai sus Singleton Pattern asigură că există o singură instanță a clasei SingletonEventList, facilitând accesul global la lista de evenimente. Factory Method Pattern permite crearea de obiecte de tip Event sau DueDateEvent în funcție de nevoile aplicației, prin intermediul clasei EventFactory. Abstract Factory Pattern furnizează o interfață pentru crearea de instanțe ale clasei EventList sau SingletonEventList, permițând schimbarea ușoară a implementării listei de evenimente fără a afecta codul existent. Aceste design patterns îmbunătățesc modularitatea, flexibilitatea și reutilizabilitatea codului, oferind soluții eficiente pentru problemele specifice întâlnite în proiectarea de software.

3.2 Model de proiectare structural

Modelele structurale de design sunt un set de modele utilizate în proiectarea software-ului pentru a organiza și relaționa diferitele componente ale unui sistem. Aceste modele se concentrează pe structura și compoziția obiectelor și claselelor, oferind modalități flexibile de organizare a acestora pentru a îndeplini cerințele sistemului.Scopul principal al modelelor structurale este de a îmbunătăți modularitatea, flexibilitatea și reutilizarea componentelor software. Acestea permit obținerea unor designuri eficiente și scalabile, în care componentele pot fi adăugate, eliminate sau modificate fără a afecta întregul sistem.

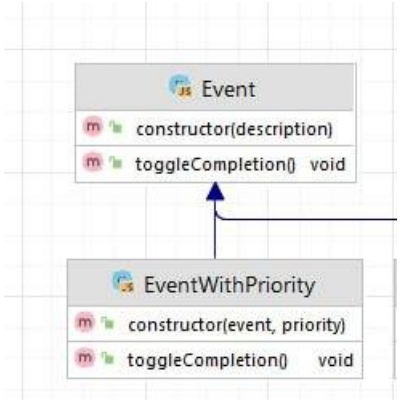

Figura 3.4 – Structura Decorator

Clasa EventWithPriority extinde clasa Event și adaugă o proprietate priority la un eveniment existent.Aceasta permite decorarea unui obiect Event cu informații suplimentare despre prioritate, menținând în continuare funcționalitatea de bază a evenimentului..

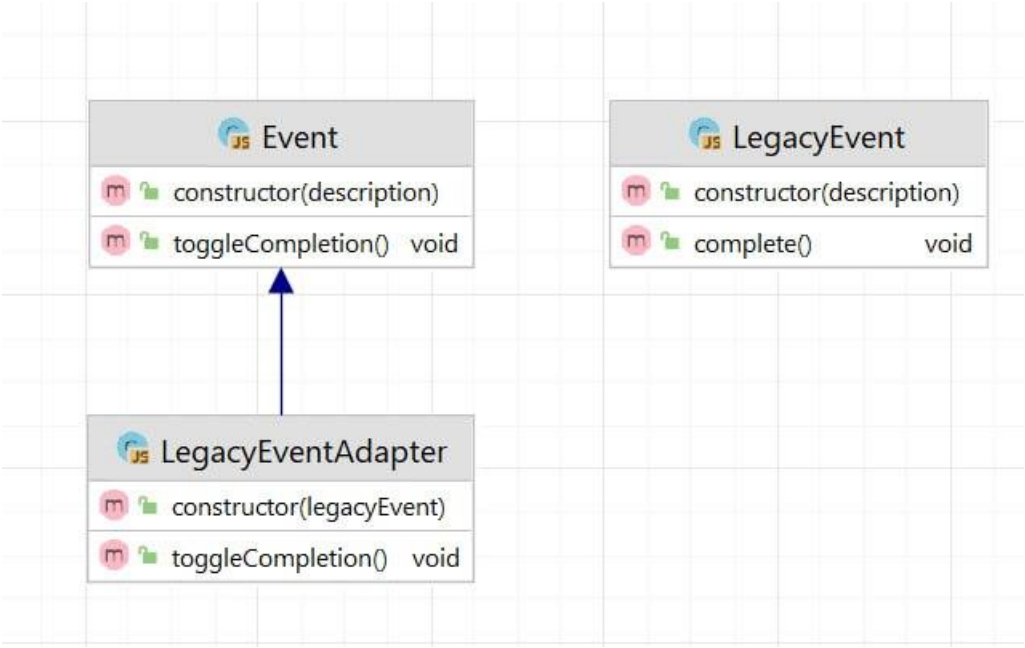

Figura 3.5 – Structura Adapter

Clasa LegacyEventAdapter adaptează obiectele de tip LegacyEvent (un format mai vechi de eveniment) la interfața evenimentelor din aplicație.Aceasta permite integrarea obiectelor de tip LegacyEvent în codul existent, prin intermediul unei clase adapter care traduce și adaptează interfețele și comportamentul acestora la cel al evenimentelor utilizate în aplicație.

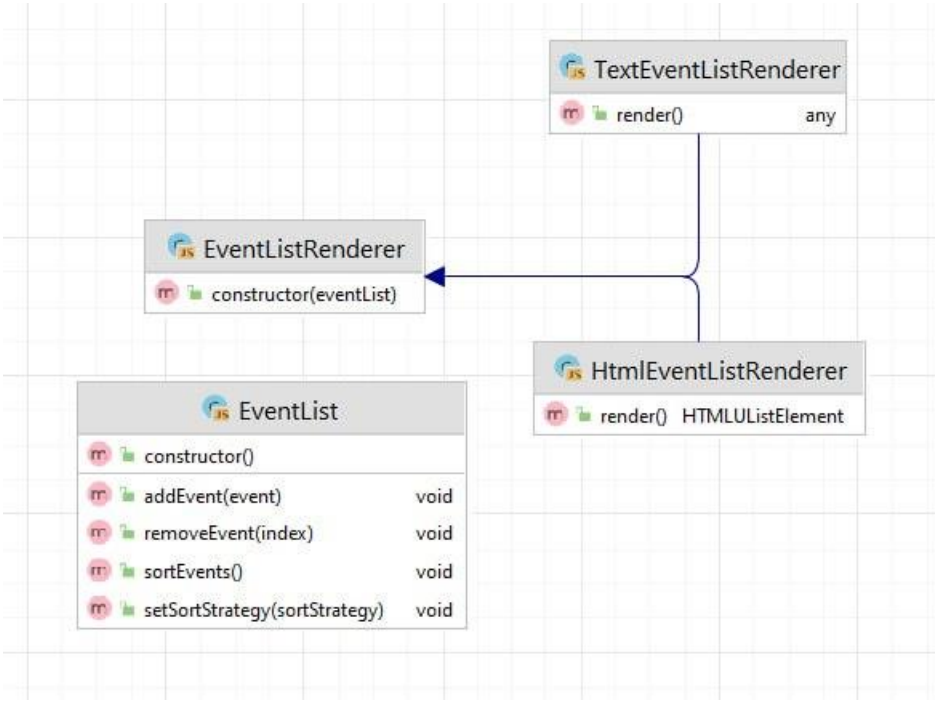

Figura 3.5 – Structura Bridge

Clasele EventListRenderer, HtmlEventListRenderer și TextEventListRenderer implementează Bridge Pattern pentru a separa implementarea randării evenimentelor de interfața de randare specifică (HTML sau text).Clasa abstractă EventListRenderer acționează ca punte între EventList și diferitele metode de randare (HTML sau text), permițând o flexibilitate mai mare în implementarea și schimbul de implementări de randare.

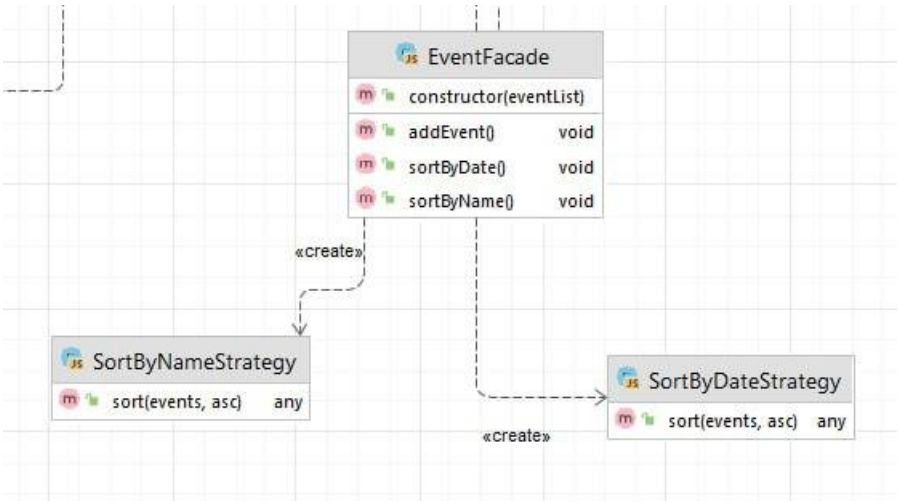

Figura 3.5 – Structura Facade

Clasa EventFacade oferă o interfață simplificată pentru a accesa și utiliza funcționalitățile complexe ale claselor asociate, cum ar fi EventList și EventRenderer.Aceasta abstractizează logica complexă și oferă metode mai simple și mai intuitive pentru adăugarea, sortarea și afișarea evenimentelor în interfața utilizatorului.Facade Pattern simplifică utilizarea și integrarea funcționalității complexe, oferind o interfață coezivă și ușor de utilizat pentru utilizatorii săi.

În Concluzie putem spune că Decorator Pattern permite adăugarea de funcționalități suplimentare la un obiect existent, Adapter Pattern facilitează integrarea obiectelor incompatibile în codul existent, Bridge Pattern separă implementarea de interfața de randare specifică, iar Facade Pattern simplifică utilizarea și interacțiunea cu funcționalitățile complexe prin intermediul unei interfețe mai simple și mai intuitive. Toate aceste design patterns contribuie la dezvoltarea unui cod mai modular, mai flexibil și mai ușor de întreținut.

3.3 Model de proiectare comportamental

Modelele comportamentale în designul software reprezintă un set de șabloane și abordări utilizate pentru gestionarea comportamentului și interacțiunii între diferitele obiecte și clase ale unui sistem. Aceste modele se concentrează pe definirea și gestionarea comportamentului dinamic al aplicației, permițând flexibilitate și extensibilitate în timpul rulării.Scopul principal al modelelor comportamentale este de a separa și organiza responsabilitățile diferitelor obiecte, de a permite schimbul și reutilizarea ușoară a comportamentului și de a permite adaptabilitatea la cerințele în schimbare ale sistemului.

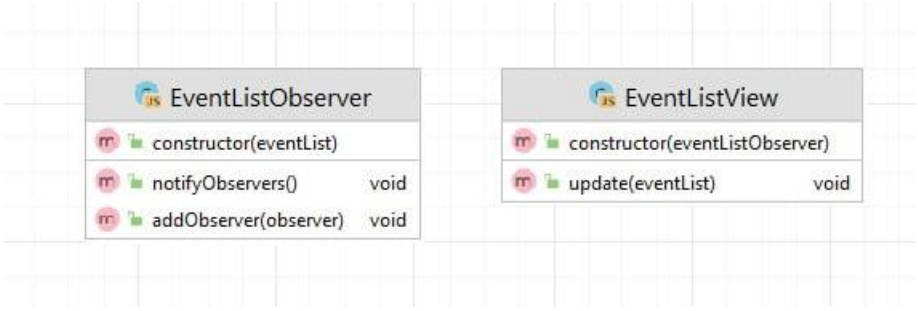

Figura 3.6 - Structura Observer

Clasa EventListObserver și EventListView implementează Observer Pattern. Acest pattern permite observarea și notificarea modificărilor din lista de evenimente. EventListObserver se înregistrează ca observator al clasei EventList și primește notificări atunci când lista de evenimente suferă modificări. EventListView poate reacționa și se poate actualiza în consecință, afișând lista actualizată de evenimente. Astfel, Observer Pattern permite o comunicare eficientă între componente și sincronizarea datelor.

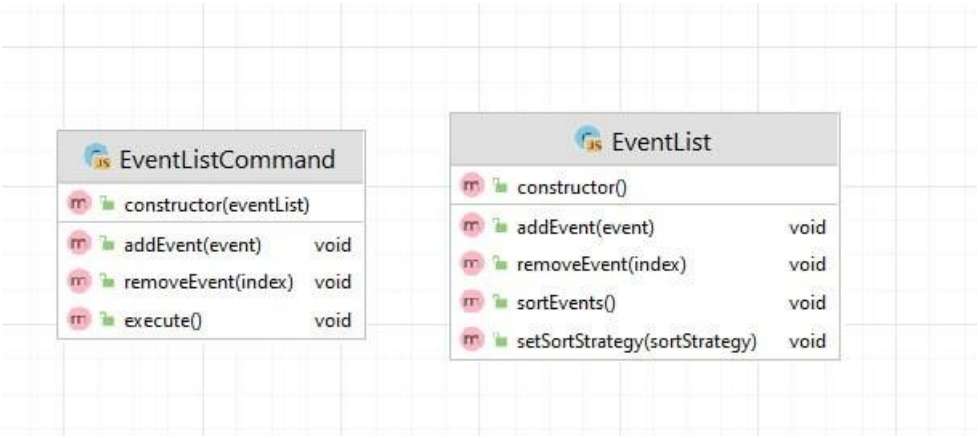

Figura 3.7 - Structura Command

Clasa EventListCommand implementează Command Pattern și gestionează lista de comenzi pentru adăugarea și ștergerea de evenimente din lista de evenimente. Acest pattern separă comenzile de operațiunile specifice, permițând gestionarea lor independentă. EventListCommand definește interfața comună pentru toate comenzile și poate executa comenzile într-un mod abstractizat. Astfel, se obține o modularitate sporită și o flexibilitate crescută în gestionarea operațiunilor pe lista de evenimente.

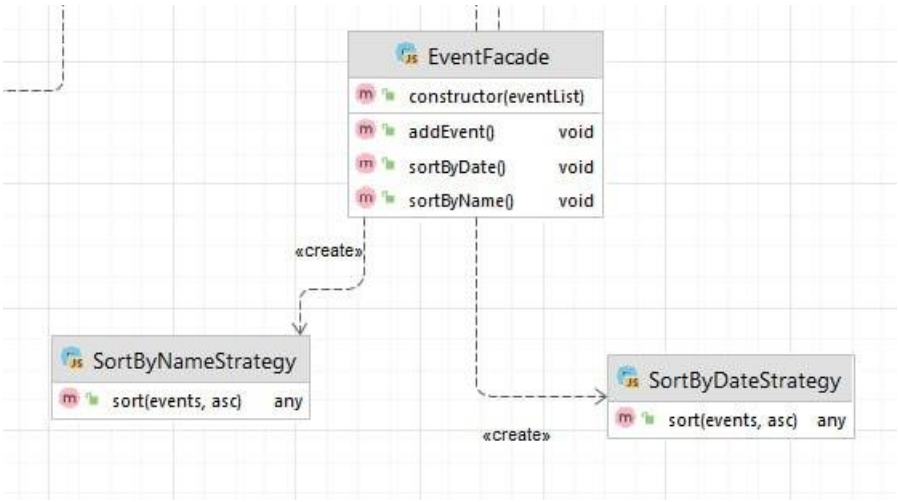

Figura 3.5 – Structura Strategy

Clasele SortByNameStrategy și SortByDateStrategy implementează Strategy Pattern pentru sortarea evenimentelor în funcție de nume sau dată. Acest pattern permite definirea mai multor strategii de sortare și selectarea și aplicarea uneia dintre ele în funcție de setările dorite. Clasa EventList utilizează o strategie de sortare specifică pentru a ordona lista de evenimente. Astfel, Strategy Pattern oferă o flexibilitate ridicată în schimbul și adaptarea comportamentului unei componente.

Observer Pattern, Command Pattern și Strategy Pattern sunt instrumente puternice în proiectarea software-ului, care rezolvă probleme recurente și permit dezvoltarea unei arhitecturi robuste și flexibile. Prin separarea responsabilităților, comunicarea eficientă între componente și adaptarea comportamentului, aceste pattern-uri îmbunătățesc calitatea codului și permit adăugarea ușoară de noi funcționalități.

CONCLUZIE

Implementarea design pattern-urilor în dezvoltarea unei aplicații web de tip event reprezintă o abordare benefică și practică în rezolvarea problemelor comune întâlnite în acest domeniu. Design pattern-urile oferă un set de soluții recurente și testate în timp, care au fost dezvoltate pentru a aborda anumite cerințe și provocări specifice.

Aplicarea design pattern-urilor în proiectarea și implementarea aplicației web de tip event contribuie la obținerea unei structuri coerente și bine organizate a codului. Prin separarea funcționalităților în componente distincte și prin definirea unor relații clare între acestea, design pattern-urile facilitează dezvoltarea unui cod modular și ușor de întreținut. Aceasta permite dezvoltatorilor să lucreze eficient asupra unor părți specifice ale aplicației și să aducă modificări fără a afecta întreaga arhitectură.

Un alt beneficiu important al implementării design pattern-urilor este extensibilitatea aplicației. Prin utilizarea design pattern-urilor, dezvoltatorii pot proiecta aplicația într-un mod flexibil, astfel încât să fie ușor de adăugat sau modificat funcționalități noi în viitor. Design pattern-urile oferă un cadru clar și bine definit pentru extinderea aplicației, asigurând că modificările sunt integrate în mod coerent și fără a afecta funcționalitatea existentă.

Pe lângă eficiența dezvoltării și extensibilitatea aplicației, design pattern-urile facilitează și gestionarea interacțiunilor între componente. Prin definirea unor interfețe standardizate și utilizarea design pattern-urilor precum Observer sau Command, dezvoltatorii pot stabili o comunicare eficientă și clară între diferitele părți ale aplicației. Aceasta permite schimbul de informații și acțiuni între componente într-un mod flexibil și decuplat, ceea ce conduce la un sistem mai modular și mai ușor de înțeles și de întreținut.

În concluzie, implementarea design pattern-urilor în dezvoltarea unei aplicații web de tip event aduce numeroase beneficii și soluții practice. Acestea contribuie la obținerea unei structuri coerente și modulare a aplicației, facilitează extensibilitatea și gestionarea interacțiunilor între componente. Utilizarea design pattern-urilor în dezvoltarea software-ului este o practică recomandată, care sporește eficiența și calitatea aplicațiilor dezvoltate.

BIBLIOGRAFIE

1. Design Patterns" pe site-ul oficial al Gang of Four (GoF):Link: https://refactoring.guru/design-patterns
2. "Head First Design Patterns" de Eric Freeman, Elisabeth Robson, Bert Bates și Kathy Sierra:Link: https://www.oreilly.com/library/view/head-first-design/0596007124/
3. "Design Patterns Explained: A New Perspective on Object-Oriented Design" de Alan Shalloway și James R. Trott:Link: https://www.oreilly.com/library/view/design-patterns-explained/9780321630049/
4. "Agile Software Development, Principles, Patterns, and Practices" de Robert C. Martin:Link: https://www.pearson.com/us/higher-education/program/Martin-Agile-Software-Development-Principles-Patterns-and-Practices/PGM255604.html
5. "Design Patterns in Dynamic Programming" de Erich Gamma:Link: https://dl.acm.org/doi/abs/10.1145/21619.21636
6. "Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL" de Dean Allemang și James Hendler:Link: https://www.elsevier.com/books/semantic-web-for-the-working-ontologist/allemang/978-0-12-385965-5