Instituto Tecnológico de Costa Rica
Campus Tecnológico Central Cartago
Escuela de Ingeniería en Computación

Fase 3 Documentación GuiaTEC

Diseño de Software IC6821 GR 2

Prof. Ing. Ericka Solano Fernández

Equipo 2

Jose Pablo Barquero Díaz c.2022119938

Dominic Casares Aguirre c.2022085016

Jose David Fernandez Salas c.2022045079

Mariana Viquez Monge c.2022029468

17 de junio del 2024
IS 2024

Patrón Estructural

Adapter

Justificación

Se escogió el patrón adapter para la integración del tipo de usuario por su gran flexibilidad el cual permite agregar la nueva clase de estudiante sin tener que modificar las clases de usuario y estudiante existentes. Al cumplir con el principio de responsabilidad única permite crear la clase por aparte sin interferir en la estructura original. Esto es una gran ventaja porque al usar el adaptador no existe la preocupación de cambiar una gran cantidad de cosas al incorporar adiciones como en el caso de estudiantes donde al estar separado permite funcionalidades separadas y más ordenadas.

Otra razón de usar el patrón adapter es su encapsulación y abstracción porque permite que el adapter tenga métodos especiales para el estudiante. Además de esto permite la incorporación del buzón al estudiante sin la necesidad de modificar el mismo, permitiendo así la posibilidad de añadir nuevas funcionalidades sin tocar ninguna de las dos clases originales.

La implementación de este patrón no genera mayores cambios en las clases del modelo, los únicos cambios que debieron realizarse fueron del lado de la persistencia de datos y el sistema del buzón. Esto debido a que al agregar nuevos datos a la entidad del estudiante fue necesario poder almacenarlos y modificarlos.

Diagrama oficial patrón estructural Adapter

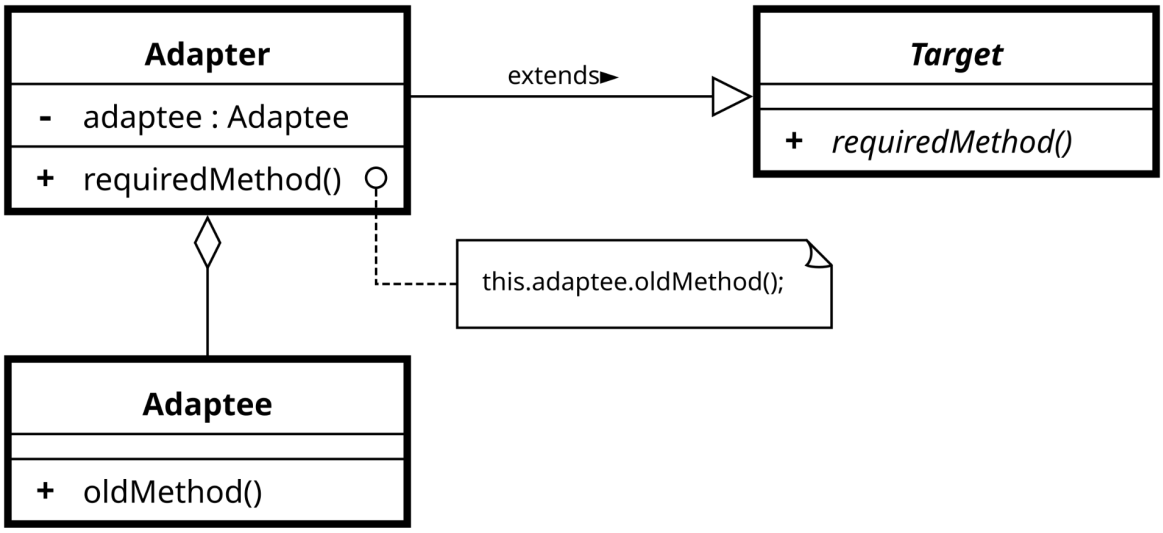


Diagrama incorporación patrón estructural Adapter

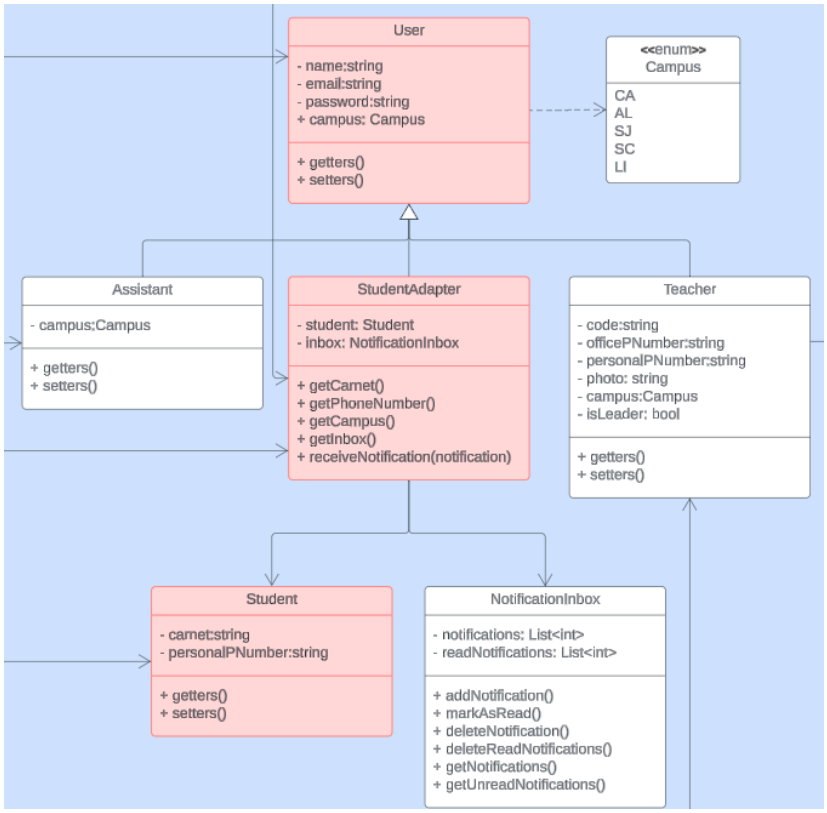


Imagen de prueba

```ts
import User from "./User";
import Student from "./Student";
import CampusENUM from "./campusENUM";
import StudentDTO from "../DTOs/student";
import NotificationInbox from "./NotificationInbox";

export default class StudentAdapter extends User {
  private student: Student;
  private inbox: NotificationInbox;

  constructor(student: StudentDTO) {
    super(
      student.name,
      student.email,
      student.password || student.carnet.toString(),
      student.campus,
      "student",
      undefined,
      undefined,
      student.carnet.toString(),
      student._id
    );
    this.student = new Student(student);
    if (!student.inbox) {
      student.inbox = { notifications: [], readNotifications: [] };
    }
    this.inbox = new NotificationInbox(student.inbox);
  }

  public getCarnet(): number {
    return this.student.getCarnet();
  }

  public getPhoneNumber(): number {
    return this.student.getPhoneNumber();
  }

  public getCampus(): CampusENUM {
    return this.student.getCampus();
  }

  public getInbox(): NotificationInbox {
    return this.inbox;
  }

  public receiveNotification(notificationID: number): void {
    console.log(`Notification received: ${notificationID}`);
    this.inbox.addNotification(notificationID);
  }
}
```

Diagrama oficial patrón comportamiento Observer

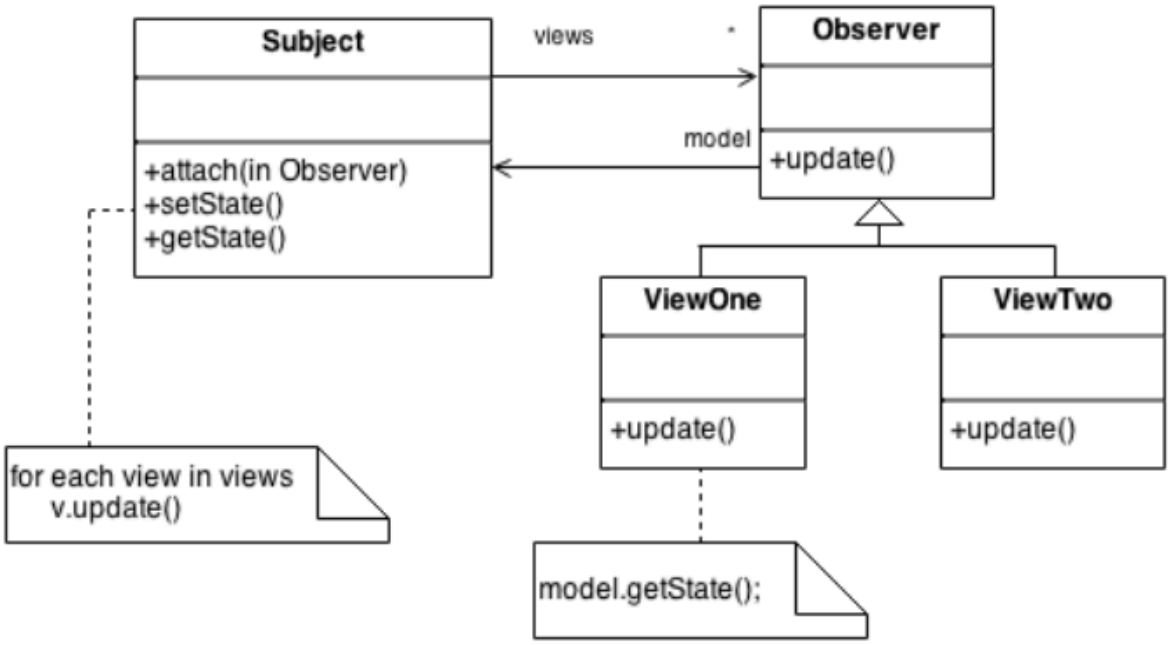


Diagrama incorporación del patrón comportamiento Observer

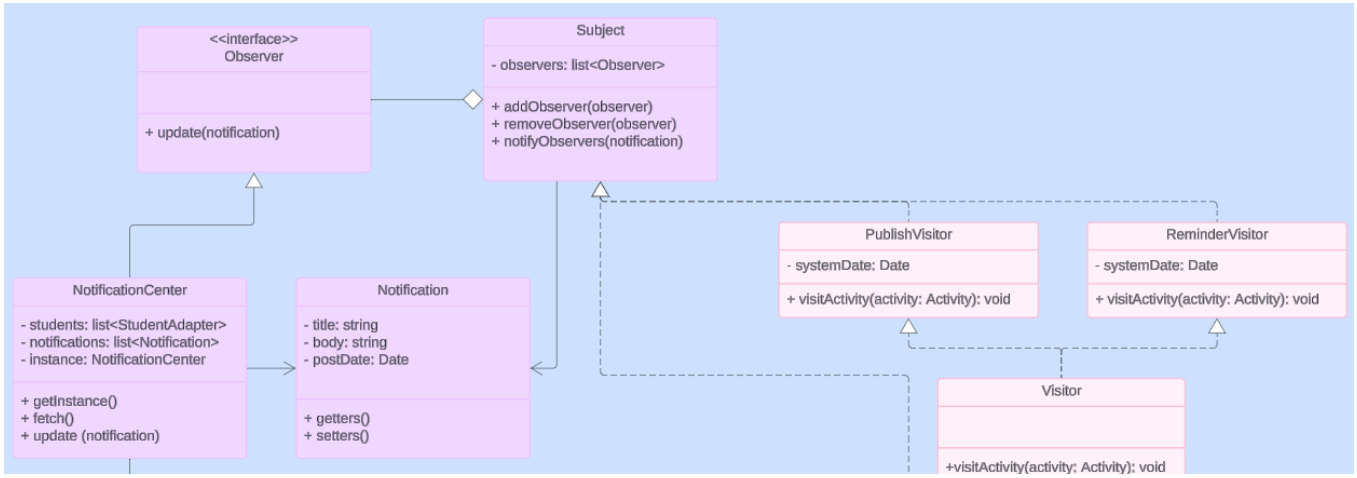


Imagen de prueba

```typescript
import Notification from "./Notification";

export interface Observer {
  update(notification: Notification): void;
}
```

```ts
import Student from "./Student";
import Notification from "./Notification";
import StudentDAO from "../DAOs/student";
import { Observer } from "./Observer";
import StudentAdapter from "./StudentAdapter";
import AlertDAO from "../DAOs/alert";

export default class NotificationCenter implements Observer {
  private static instance: NotificationCenter;
  private notifications: Notification[] = [];
  private students: StudentAdapter[] = [];

  private constructor() {
    this.fetch();
  }

  public static getInstance(): NotificationCenter {
    if (!NotificationCenter.instance) {
      NotificationCenter.instance = new NotificationCenter();
    }
    return NotificationCenter.instance;
  }

  public fetch() {
    // Fetch data from database

    // Fetch students from database
    StudentDAO.getAllStudentsAdapted().then((students) => {
      this.students = students;
    });
    AlertDAO.getAllAlerts().then((notifications) => {
      this.notifications = notifications;
    });
  }

  public update(notification: Notification): void {
    this.fetch();
    this.notifications.push(notification);
    const notificationID = this.notifications.length - 1;

    AlertDAO.saveAlert(notification);


    console.log("sendingAlert:" + notificationID);

    this.students.forEach((student) => {
      student.receiveNotification(notificationID);
      StudentDAO.updateStudentInbox(
        student.getCarnet().toString(),
        student.getInbox()
      );
    });
  }
}
```

```typescript
import Notification from "./Notification";
import NotificationCenter from "./NotificationCenter";
import { Observer } from "./Observer";

export default class Subject {
  private observers: Observer[] = [];

  public constructor() {
    this.observers.push(NotificationCenter.getInstance());
  }

  public addObserver(observer: Observer): void {
    if (!this.observers.includes(observer)) {
      this.observers.push(observer);
    }
  }

  public removeObserver(observer: Observer): void {
    const index = this.observers.indexOf(observer);
    if (index !== -1) {
      this.observers.splice(index, 1);
    }
  }

  public notifyObservers(notification: Notification): void {
    this.observers.forEach((observer) => {
      observer.update(notification);
    });
  }
}
```

Diagrama oficial patrón de comportamiento Visitor

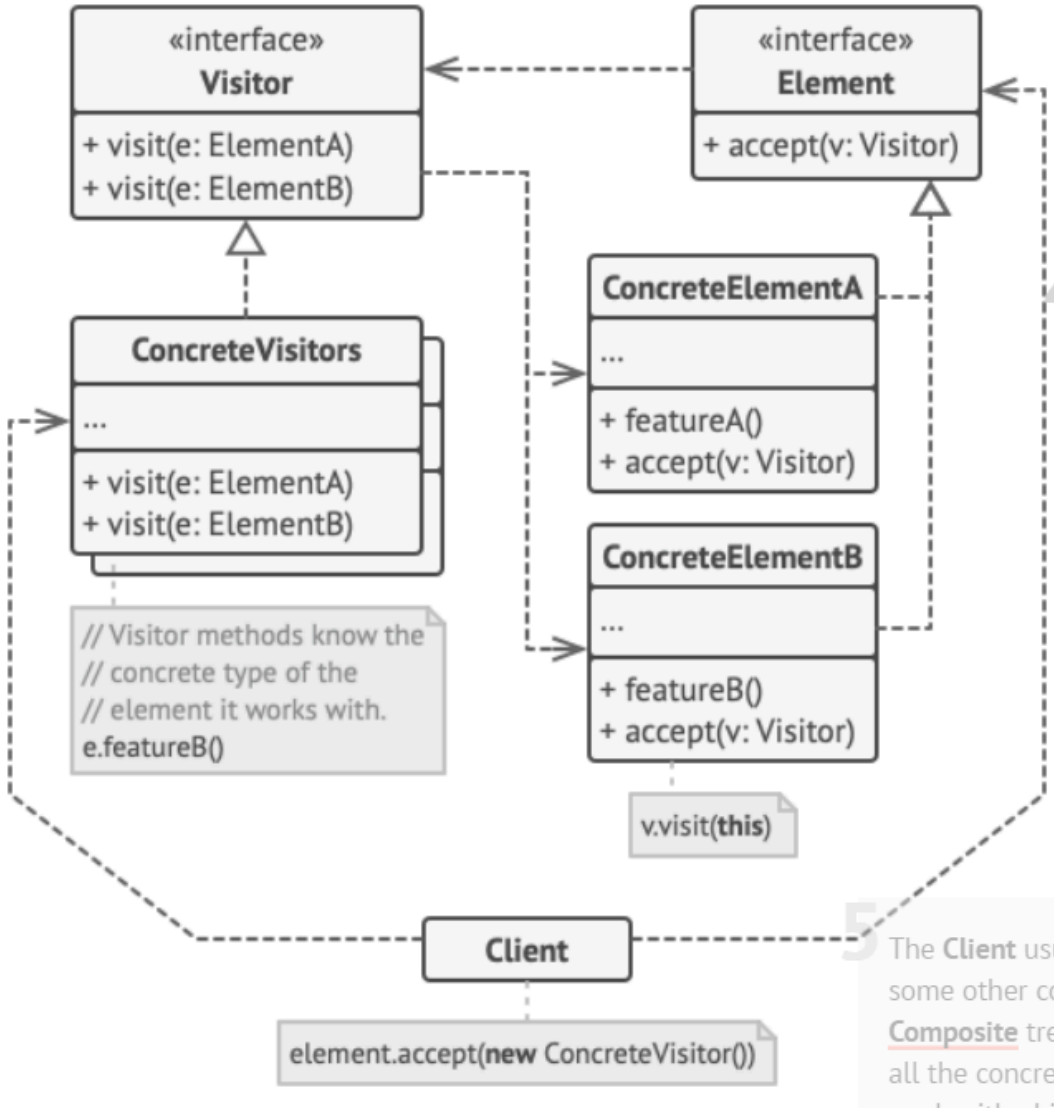


Diagrama incorporación del patrón de comportamiento Visitor

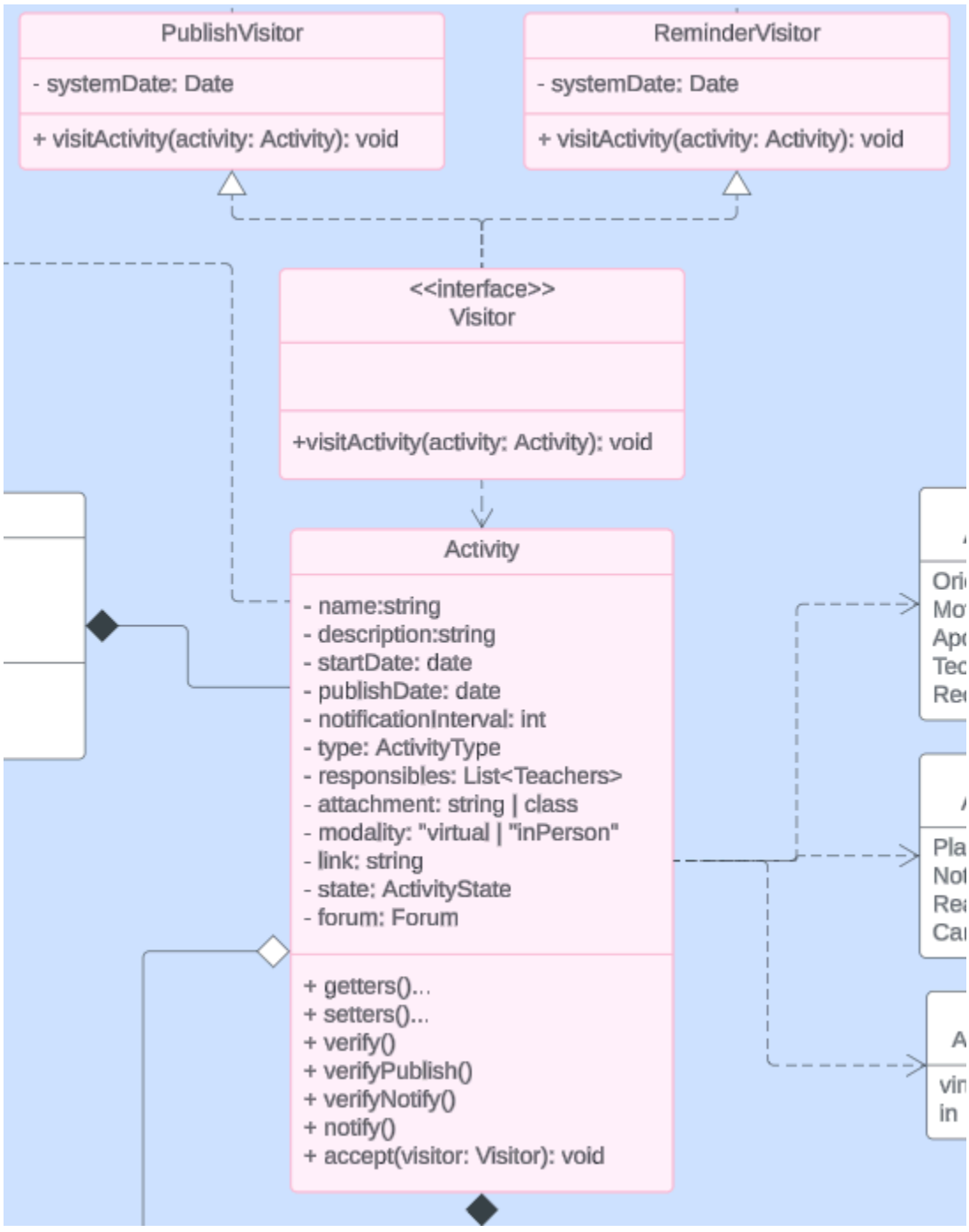
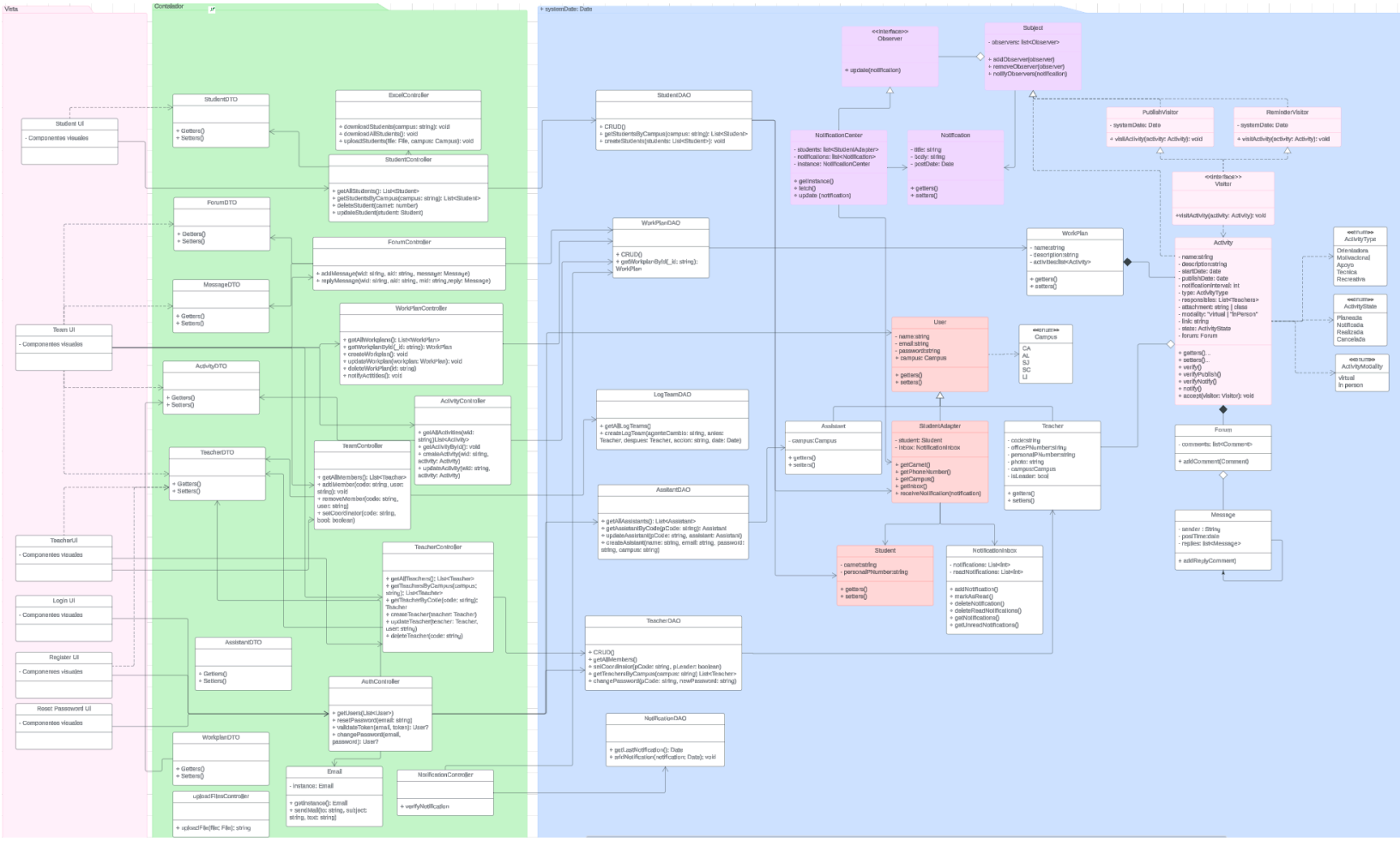


Diagrama de clases



[Enlace al Diagrama de Clases en Lucid]

Cuadro de análisis

Criterio de evaluación para estudiantes

Objetivo	Descripción	Porcentaje completitud	Observaciones
Implementación del patrón estructural	Se implementa el patrón estructural para incorporar estudiante sin modificar el trabajo previo	100%	Se desarrolla sin complicaciones

Implementación del usuario Estudiante	Se restringen ciertas acciones para los estudiantes ya sea para modificar información o lo que pueden ver.	100%	
Implementación de nuevas pantallas para estudiantes	El estudiante cuenta con una pantalla para visualizar su información personal	100%	
Funcionalidades dentro del perfil de Estudiante	Los estudiantes solo pueden modificar su número de teléfono y fotografía como máximo.	100%	
Funcionalidades dentro del buzón del Estudiante	Permite ingreso automático de mensajes en orden de llegada. Además desplegar mensajes leídos y no leídos con las opciones de eliminar y filtrar las notificaciones.	100%	

Criterio de evaluación para notificaciones

Objetivo	Descripción	Porcentaje completitud	Observaciones
Implementación del patrón Visitor	Se implementa para lograr añadir las nuevas funcionalidades de revisión de cambio de estado en las actividades de manera encapsulada. Además de conecta con el observer		Se desarrolla sin complicaciones
Implementación de patrón Observer	Se implementa para los buzones de estudiantes donde	100%	Se desarrolla sin complicaciones

	existe un centro de notificaciones que interactúa con cada uno de ellos.		
Sistema de notificación automática	Un sistema que genere notificaciones automáticas para revisar todos los días el estado de las actividades	100%	
Persistencia de datos en la notificaciones y buzones	Se implementa el flujo de persistencia en base de datos	100%	

Lecciones aprendidas

Jose Pablo Barquero: Además de lo aprendido en las primeras entregas, esta etapa me hizo entender la importancia de una buena planificación y estructuración. Gracias a la buena arquitectura que teníamos fue sencillo añadir nuevas funcionalidades, demostrando así el impacto que una buena organización a futuro puede tener. También mejoré mis habilidades en cuestión del uso de patrones para mejorar la arquitectura, siento que contar con la capacidad de detectar posibles implementaciones de patrones es muy beneficiosa al momento de desarrollar software.

Dominic Casares: En mi caso, haber participado en el desarrollo de este proyecto fue una experiencia provechosa, programar web es divertido e implementar los patrones aprendidos en clase para esta tercera fase fue un reto que disfruté, desde la segunda fase del proyecto puse a prueba mis habilidades y aprendí de nueva tecnologías pero con esta nueva etapa pude culminar mi desarrollo de habilidades técnicas, por lo que me siento satisfecho de haber concluido el proyecto con mis compañeros, sin muchos inconvenientes.

David Fernandez: Este proyecto fue de mucho provecho al permitirnos crear una página web desde cero con las tecnologías que nosotros decidimos. Todas las etapas tenían un enfoque especial, en este caso el enfoque de la tercera etapa era poner a prueba las habilidades de diseño de los patrones vistos en el curso. Al crear estos patrones la perspectiva, percepción y comprensión de los patrones mejora porque encajar dichos patrones para solucionar situaciones en el código ayuda mucho a la comprensión de estos. Algo que aprendí fue la

importancia de tener un código ordenado y modulado, porque al incluir fue fácil que encajaran bien y además no se tocó de gran manera la estructura original.

Mariana Viquez: Tuve la experiencia en esta etapa de probar diferentes tecnologías y lograr desarrollar nuevas habilidades, como por ejemplo mejorar mi comprensión de la programación web y también aprender sobre la aplicación de patrones de diseño para resolver casos específicos. Además de estas habilidades, aprendí la importancia de la comunicación efectiva y asertiva, ya que al utilizarla en el equipo pudimos lograr una organización óptima y por lo tanto, un resultado exitoso en nuestro proyecto.