

**UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO**

**FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA,  
INFORMÁTICA Y MECÁNICA**

**ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE  
SISTEMAS**



**Informe N°1 - Tercer Parcial:**

---

**“Patrones de diseño”**

---

**DOCENTE:** ROXANA LISETTE QUINTANILLA PORTUGAL

**ALUMNOS:**

- |                                  |        |
|----------------------------------|--------|
| ● Cordova Castro, Marko Leugim   | 160890 |
| ● Guevara Ferro, Cristian Luis   | 171061 |
| ● Luna Ccasani, Charlie Joel     | 161368 |
| ● Mallqui Apaza, Nadiabeth Diana | 171063 |
| ● Quispe Leon, Widmar Raul       | 171259 |
| ● Rojas Cahuana Etson Ronaldo    | 124821 |
| ● Sullca Peralta, Melanie Indira | 171070 |
| ● Rodriguez Hanco, Rudy Rodrigo  | 171068 |

**CUSCO – PERÚ  
2021**

## PATRONES DE DISEÑO

### Concepto

Pues ni más ni menos son formas “estandarizadas” de resolver problemas comunes de diseño en el desarrollo de software.

- Las ventajas del uso de patrones son evidentes:
- Conforman un amplio catálogo de problemas y soluciones
- Estandarizan la resolución de determinados problemas
- Condensan y simplifican el aprendizaje de las buenas prácticas
- Proporcionan un vocabulario común entre desarrolladores
- Evitan “reinventar la rueda”

### Tipos de patrones

Según la finalidad del patrón, estos se clasifican en tres tipos:

#### - Patrones Creacionales

Estos patrones vienen a solucionar o facilitar las tareas de creación o instanciación de objetos, hacen hincapié en la encapsulación de la lógica de la instanciación, ocultando detalles concretos de cada objeto y permitiéndonos trabajar con abstracciones, proporcionan varios mecanismos de creación de objetos que incrementan la flexibilidad y la reutilización del código existente.









#### - Patrones estructurales

Los patrones estructurales explican cómo ensamblar objetos y clases en estructuras más grandes, a la vez que se mantiene la flexibilidad y eficiencia de estas estructuras.



## - Patrones comportamentales

Los patrones de comportamiento se tratan con algoritmos y la asignación de responsabilidades entre objetos.

 <p><b>Chain of Responsibility</b></p> <p>Permite pasar solicitudes a lo largo de una cadena de manejadores. Al recibir una solicitud, cada manejador decide si la procesa o si la pasa al siguiente manejador de la cadena.</p>	 <p><b>Command</b></p> <p>Convierte una solicitud en un objeto independiente que contiene toda la información sobre la solicitud. Esta transformación te permite parametrizar los métodos con diferentes solicitudes, retrasar o poner en cola la ejecución de una solicitud y soportar operaciones que no se pueden realizar.</p>	 <p><b>Iterator</b></p> <p>Permite recorrer elementos de una colección sin exponer su representación subyacente (lista, pila, árbol, etc.).</p>
 <p><b>State</b></p> <p>Permite a un objeto alterar su comportamiento cuando su estado interno cambia. Parece como si el objeto cambiara su clase.</p>	 <p><b>Strategy</b></p> <p>Permite definir una familia de algoritmos, colocar cada uno de ellos en una clase separada y hacer sus objetos intercambiables.</p>	 <p><b>Template Method</b></p> <p>Define el esqueleto de un algoritmo en la superclase pero permite que las subclases sobrescriban pasos del algoritmo sin cambiar su estructura.</p>

## BACKEND

### Patrones creacionales

#### - Factory Method:

Se pueden instanciar objetos directamente en los Resources con constructores de clases creados para cada modelo, por ejemplo para la clase Teacher.

```

resources
> __pycache__
  __init__.py
  appointment.py
  authenticate.py
  coordinator.py
  create_student_accounts.py
  create_tutor_accounts.py
  distribute_student.py
  documentation.py
  filter_teachers_for_tutors.py
  new.py
  principal.py
  student_helper.py
  student.py
  teacher.py
  tutor_student.py
  tutor.py
  tutoring_program.py
  update_credentials_coordinator.py
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
if claims['role'] != 'coordinator':
    return {'message': 'You are not allowed to do this'}, 401

# Verify if all arguments are correct
ans, data = TeacherList.parser.parse_args(dict(request.json))
if not ans:
    return data

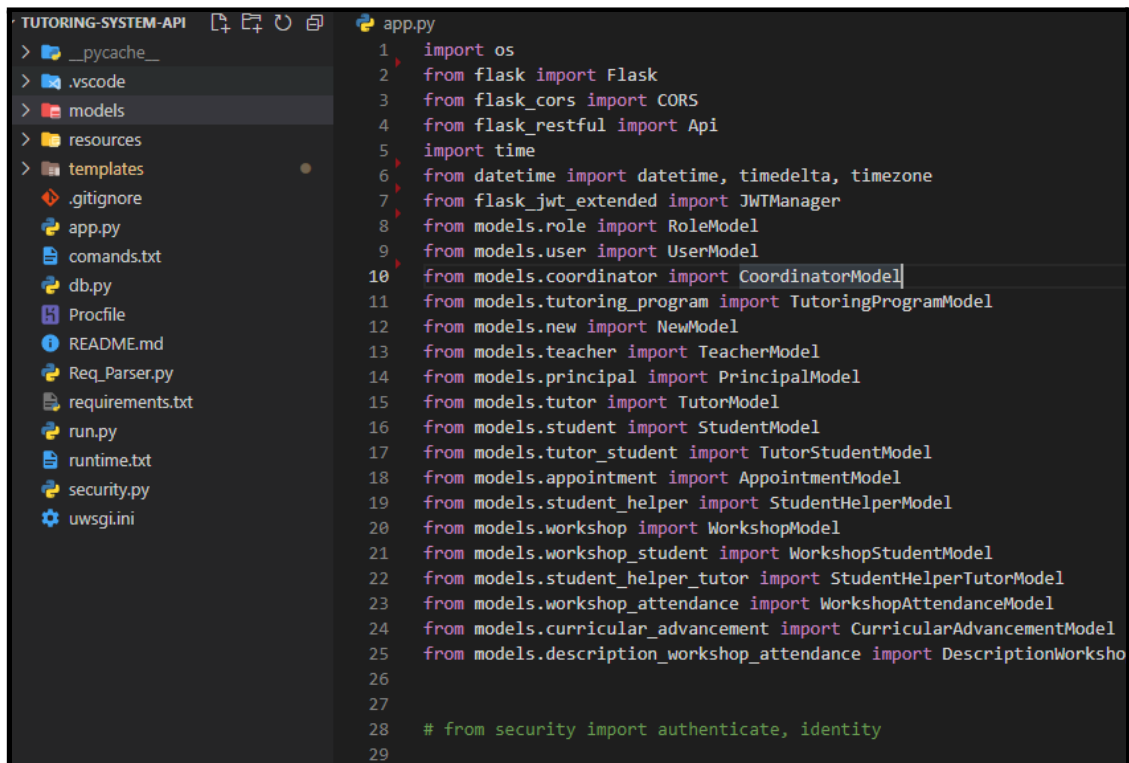
# Get the tutoring program active
tutoring_program_active = TutoringProgramModel.find_tutoring_program_active()
data['cod_tutoring_program'] = tutoring_program_active.cod_tutoring_program

# Verify if teacher exists in database
teacher = TeacherModel.find_by_cod_teacher(cod_teacher)
if teacher:
    try:
        teacher.update_data(**data)
        teacher.save_to_db()
        return teacher.json(), 200
    except:
        return {'message': 'An error occurred while updating the teacher on DB'}, 500
return {'message': 'Teacher not found.'}, 404

```

### - Singleton:

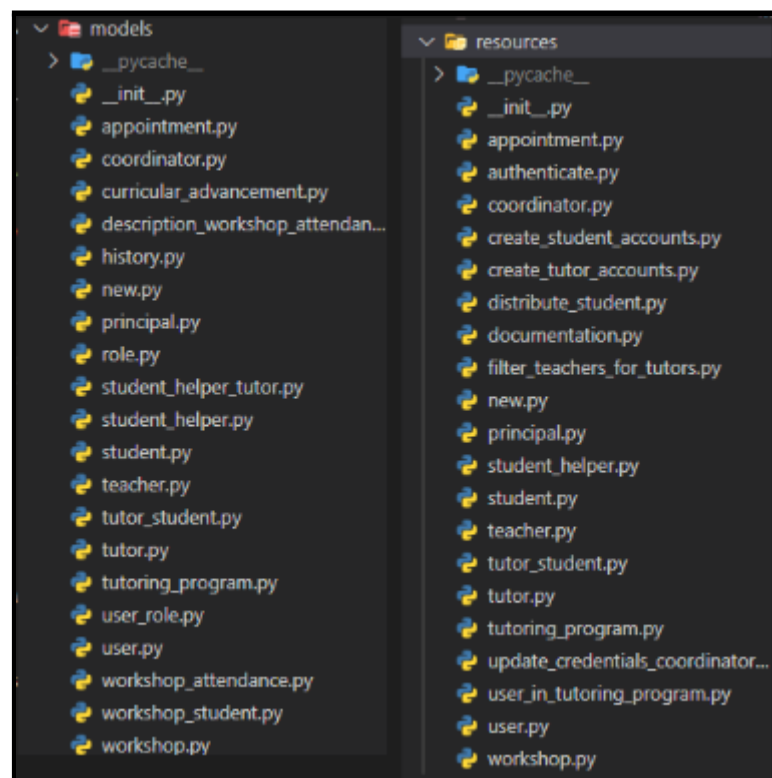
Se tiene un punto donde convergen todas las instancias necesarias para el funcionamiento del backend.



```
1 import os
2 from flask import Flask
3 from flask_cors import CORS
4 from flask_restful import Api
5 import time
6 from datetime import datetime, timedelta, timezone
7 from flask_jwt_extended import JWTManager
8 from models.role import RoleModel
9 from models.user import UserModel
10 from models.coordinator import CoordinatorModel
11 from models.tutoring_program import TutoringProgramModel
12 from models.new import NewModel
13 from models.teacher import TeacherModel
14 from models.principal import PrincipalModel
15 from models.tutor import TutorModel
16 from models.student import StudentModel
17 from models.tutor_student import TutorStudentModel
18 from models.appointment import AppointmentModel
19 from models.student_helper import StudentHelperModel
20 from models.workshop import WorkshopModel
21 from models.workshop_student import WorkshopStudentModel
22 from models.student_helper_tutor import StudentHelperTutorModel
23 from models.workshop_attendance import WorkshopAttendanceModel
24 from models.curricular_advancement import CurricularAdvancementModel
25 from models.description_workshop_attendance import DescriptionWorksho
26
27
28 # from security import authenticate, identity
29
```

### - Abstract Factory

En el backend se tienen 2 grandes familias de objetos que son: models y resources



## Patrones estructurales

### - Bridge

Para los objetos que tenemos se dividió en 2 jerarquías, que en nuestro caso son models y resources.

Ejemplo: appointment.py

```
models > appointment.py
1  from db import db
2  from datetime import date
3
4  class AppointmentModel(db.Model):
5      __tablename__ = 'appointments'
6
7      # -- Attributes --
8      cod_appointment = db.Column(db.String(6), primary_key=True)
9      cod_tutor = db.Column(db.String(6), db.ForeignKey('tutors.cod_tutor'), primary_key=True)
10     cod_student = db.Column(db.String(6), db.ForeignKey('students.cod_student'), primary_key=True)
11     date_time = db.Column(db.Date, nullable=False, default=date.ctime)
12     general_description = db.Column(db.String(300))
13     private_description = db.Column(db.String(300))
14     diagnosis = db.Column(db.String(300))
15     cod_tutoring_program = db.Column(db.String(6), db.ForeignKey('tutoring_programs.cod_tutoring_program'))
16
17     # -- Relations --
18
19     def __init__(self, cod_appointment, cod_tutor, cod_student, date_time, general_description, private_description):
20         self.cod_appointment = cod_appointment
21         self.cod_tutor = cod_tutor
22         self.cod_student = cod_student
23         self.date_time = date_time
24         self.general_description = general_description
25         self.private_description = private_description
26         self.diagnosis = diagnosis
27         self.cod_tutoring_program = cod_tutoring_program
28
29     def json(self):
30         return {'cod_appointment': self.cod_appointment,
31                 'cod_tutor': self.cod_tutor,
32                 'cod_student': self.cod_student,
33                 'date_time': self.date_time,
34                 'general_description': self.general_description,
35                 'private_description': self.private_description,
36                 'diagnosis': self.diagnosis,
37                 'cod_tutoring_program': self.cod_tutoring_program}
```

```
resources > appointment.py
1  from datetime import datetime
2  from flask_restful import Resource
3  from flask import request
4  from models.appointment import AppointmentModel
5  from models.tutoring_program import TutoringProgramModel
6  from models.teacher import TeacherModel
7  from models.tutor import TutorModel
8  from models.student import StudentModel
9  from Req_Parser import Req_Parser
10 from datetime import date, datetime
11 from flask_jwt_extended import jwt_required, get_jwt
12
13 class Appointment(Resource):
14     parser = Req_Parser()
15     parser.add_argument('cod_appointment', str, True)
16     parser.add_argument('cod_tutor', str, True)
17     parser.add_argument('cod_student', str, True)
18     parser.add_argument('date_time', str, True)
19     parser.add_argument('general_description', str, True)
20     parser.add_argument('private_description', str, True)
21     parser.add_argument('diagnosis', str, True)
22     parser.add_argument('cod_tutoring_program', str, True)
23
24     @jwt_required()
25     def put(self, cod_appointment):
26         claims = get_jwt()
27
28         if claims['role'] != 'tutor':
29             return {'message': 'You are not allowed to do this'}, 401
30         # Verify if all attributes are in request and are of correct type
31         ans, data = AppointmentList.parser.parse_args(dict(request.json))
32         if not ans:
```

## - Decorator

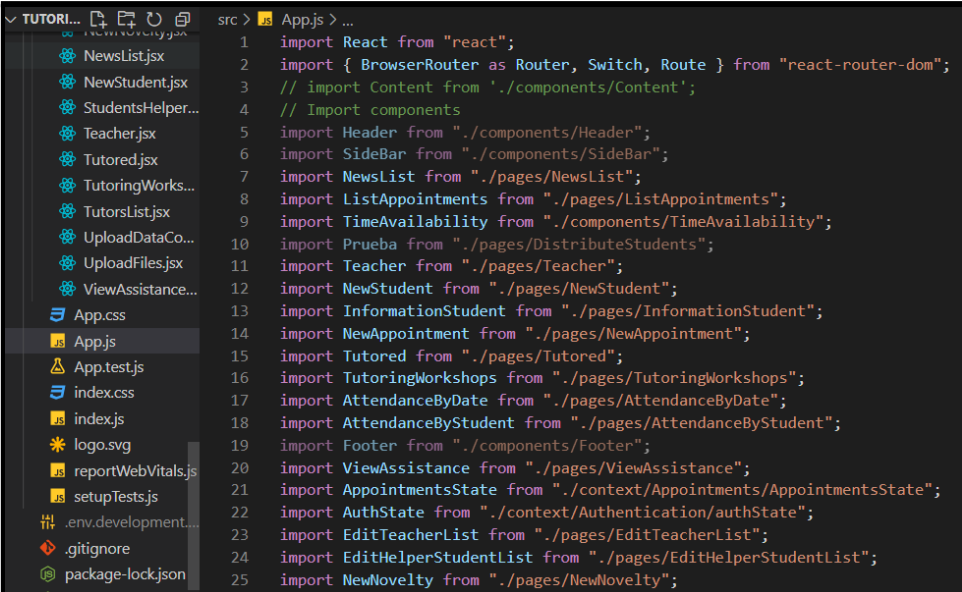
Este patrón de diseño se puede observar en los objetos de la familia resources en donde cada objeto posee funcionalidades como: put, get, delete, post entre otros.

```
resources > appointment.py
23
24 @jwt_required()
25 def put(self, cod_appointment):
26     claims = get_jwt()
27
28     if claims['role'] != 'tutor':
29         return {'message': 'You are not allowed to do this'}, 401
30     # Verify if all attributes are in request and are of correct type
31     ans, data = AppointmentList.parser.parse_args(dict(request.json))
32     if not ans:
33         return data
34     # Create a instance of TutoringProgramModel with the data provided
35     appointment = AppointmentModel.find_by_cod_appointment(cod_appointment)
36     if appointment:
37         appointment.update_data(**data)
38         appointment.save_to_db()
39         return appointment.json(), 200
40     return {'message': 'Appointment not found.'}, 404
41
42 @jwt_required()
43 def get(self, cod_appointment):
44     claims = get_jwt()
45     if claims['role'] != 'tutor':
46         return {'message': 'You are not allowed to do this'}, 401
47     appointment = AppointmentModel.find_by_cod_appointment(cod_appointment)
48     if appointment:
49         return appointment.json(), 200
50     return {'message': 'Appointment not found.'}, 404
51
52 @jwt_required()
53 def delete(self, cod_appointment):
54     claims = get_jwt()
55     if claims['role'] != 'tutor':
```

## FRONTEND

### Patrones creacionales

- **Singleton** : Nos permite asegurarnos que una clase tenga una única instancia



```
src > App.js > ...
1 import React from "react";
2 import { BrowserRouter as Router, Switch, Route } from "react-router-dom";
3 // import Content from './components/Content';
4 // Import components
5 import Header from "./components/Header";
6 import SideBar from "./components/SideBar";
7 import NewsList from "./pages/NewsList";
8 import ListAppointments from "./pages/ListAppointments";
9 import TimeAvailability from "./components/TimeAvailability";
10 import Prueba from "./pages/DistributeStudents";
11 import Teacher from "./pages/Teacher";
12 import NewStudent from "./pages/NewStudent";
13 import InformationStudent from "./pages/InformationStudent";
14 import NewAppointment from "./pages/NewAppointment";
15 import Tutored from "./pages/Tutored";
16 import TutoringWorkshops from "./pages/TutoringWorkshops";
17 import AttendanceByDate from "./pages/AttendanceByDate";
18 import AttendanceByStudent from "./pages/AttendanceByStudent";
19 import Footer from "./components/Footer";
20 import ViewAssistance from "./pages/ViewAssistance";
21 import AppointmentsState from "./context/Appointments/AppointmentsState";
22 import AuthState from "./context/Authentication/authState";
23 import EditTeacherList from "./pages/EditTeacherList";
24 import EditHelperStudentList from "./pages/EditHelperStudentList";
25 import NewNovelty from "./pages/NewNovelty";
```

- **Factory Method:** Nos proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.

```

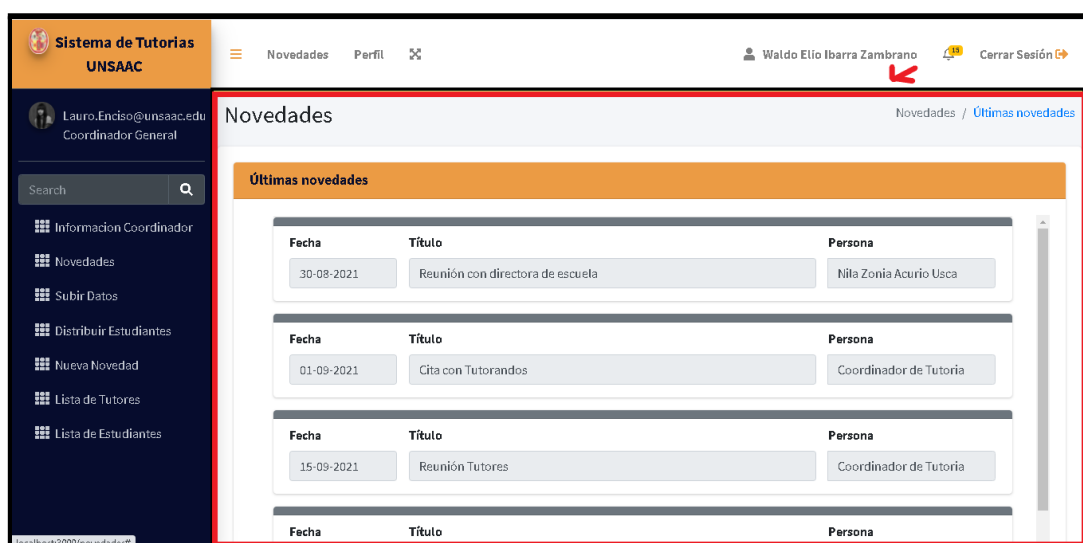
Teacher.jsx
src > pages > Teacher.jsx > Teacher
32 return (
33   <div className="content-wrapper">
34     { /* Content Header (Page header) */ }
35     <section className="content-header">
36       <div className="container-fluid">
37         <div className="row mb-2">
38           <div className="col-sm-6">
39             <h1>Registro de Docentes</h1>
40           </div>
41           <div className="col-sm-6">
42             <ol className="breadcrumb float-sm-right">
43               <li className="breadcrumb-item active">Registro de Docentes</li>
44               <li className="breadcrumb-item">
45                 { /* <a href="#">Home</a> */ }
46                 { /* <Link to="Detalle_Servicio"> Detalle Servicio</Link> */ }
47               </li>
48             </ol>
49           </div>
50         </div>
51       </div>
52     </section>

```

```

Tutored.jsx
src > pages > Tutored.jsx > Tutored
33 return (
34   <Fragment>
35     <Header />
36     <SideBar />
37
38     <div className="content-wrapper">
39       <section className="content-header">
40         <div className="container-fluid">
41           <div className="row mb-2">
42             <div className="col-sm-6">
43               <h1>Lista de Tutorados</h1>
44             </div>
45             <div className="col-sm-6">
46               <ol className="breadcrumb float-sm-right">
47                 <li className="breadcrumb-item active">Tutorados</li>
48                 <li className="breadcrumb-item">
49                   { /* <a href="#">Home</a> */ }
50                   <Link to="Nuevo_Estudiante"> Nuevo Tutorado</Link>
51                 </li>
52               </ol>
53             </div>
54           </div>
55         </div>
56       { /* /.container-fluid */ }
57     </section>

```



- **Builder** : Patrón de diseño que nos permite construir objetos complejos paso a paso.

```

    /* <div className="col-md-12"> */
    <div className="col-md-12">
      <div className="card card-primary">
        /* <div className="card-header">
          <h3 className="card-title">
            Datos Personales
          </h3>
        </div> */
        <div className="card-body">
          Datos estudiante
          <hr />
          <div className="form-group row">
            <label
              className="col-sm-3 col-form-label"
            >
              Codigo:
            </label>
            <div className="col-sm-9">
              <input
                type="text"
                className="form-control"
                placeholder="Codigo"
                defaultValue={171259}
                disabled
              />
            </div>
          </div>
          <div className="form-group row">
            <label
              className="col-sm-3 col-form-label"
            >
              Nombres:
            </label>
            <div className="col-sm-9">
              <input
                type="text"
                className="form-control"
                placeholder="Nombres"
                defaultValue={"Widmar Raul"}
                disabled
              />
            </div>
          </div>
          <div className="form-group row">
            <label
              className="col-sm-3 col-form-label"
            >
              Apellidos:
            </label>
            <div className="col-sm-9">
              <input
                type="text"

```

## Patrones estructurales

- **Decorator**: Permite agregar funcionalidades a objetos colocando estos objetos dentro de otros objetos.  
Se tiene el objeto newappointment el cual contiene un botón que al presionar coloca a otro objeto



