UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

IzyVote: Tu Voto Seguro, con Tecnología y Simplicidad

Integrantes:

Rondan Ttito, Fredy	20230174E
Morillo Zavaleta, Cristhian Samuel	20240156J
Porras Cajahuaman, Margaret Anacristina	20232707K
Flores Rivas, Abdel Jose	20232661K
Solís Córdova, Valeria Mercedes	20231245C

Docente:

Tello Canchapoma Yury

Curso:

Programación Orientada a Objetos

Código de curso:

BMA15Q

2025-1

INDICE

I. INTRODUCCIÓN

IzyVote es una plataforma web orientada a modernizar el proceso de votación estudiantil mediante el uso de tecnología blockchain, que permite registrar cada voto de forma segura, transparente y descentralizada. El sistema está dirigido exclusivamente a estudiantes de la Universidad Nacional de Ingeniería (UNI), quienes acceden a la plataforma utilizando su correo institucional, garantizando así la autenticidad del votante y evitando accesos no autorizados.

Desde el punto de vista técnico, IzyVote fue desarrollado integrando una robusta pila tecnológica. En el lado del cliente (frontend), se utilizó HTML para la estructura del contenido, CSS para el diseño visual y JavaScript para aportar interactividad, logrando una interfaz amigable e intuitiva para el usuario. En el lado del servidor (backend), se empleó Python junto con el microframework Flask para implementar la lógica del sistema y gestionar las comunicaciones del servidor. La gestión de datos se realizó mediante SQL, y se integró el protocolo SMTP para la validación de usuarios a través del correo institucional. Además, se incorporó tecnología blockchain para garantizar que cada voto sea único, verificable e inalterable, reforzando así la transparencia y la seguridad del sistema.

IzyVote representa una solución integral que combina facilidad de uso con herramientas avanzadas de protección y verificación, respondiendo a los desafíos actuales de confiabilidad y accesibilidad en los procesos electorales digitales.

II. ANTECEDENTES

A **continuación,** se presentan tres proyectos open-source que implementan votación electrónica segura mediante blockchain:

1. (Blockchain-Based-voting-platform)
   https://github.com/tarun7r/Blockchain-Based-voting-platform/
   Es una plataforma de votación electrónica basada en Python y Flask, que registra cada voto como una transacción inmutable en una red privada Ethereum simulada con Ganache.
   Autenticación de votantes generando un vote ID único cuyo hash se almacena en la blockchain para garantizar anonimato y trazabilidad.

Ofrece transparencia, inmutabilidad y experiencia de usuario accesible, al permitir votar desde cualquier lugar con un recibo de transacción verificable.

2. (Voting-System)
   https://github.com/Shashwat1729/Voting-System
   Es un prototipo en Python de votación blockchain que usa un esquema ZKP de claves privadas, públicas y compartidas para autenticar votos sin revelar secretos. Solo votantes legítimos pueden generar una prueba criptográfica válida, y la coincidencia de esa prueba en el verificador confirma la autenticidad e integridad de cada voto, garantizando al mismo tiempo la privacidad del emisor y la inmutabilidad en la cadena de bloques en el verificador confirma la autenticidad e integridad del voto.
3. (Abhiramborige/Online-Voting-Using-Blockchain)
   https://github.com/ngocbh/voting-blockchain
   Es una aplicación didáctica en Flask que permite emitir votos mediante un formulario web y los almacena en una cadena de bloques en memoria, implementando una prueba de trabajo básica para demostrar cómo el nonce y el encadenamiento de hashes garantizan la inmutabilidad de los votos durante la ejecución del servidor. Es una versión simplificada y minimalista, diseñada para ilustrar los conceptos clave de blockchain sin la complejidad de sistemas distribuidos o contratos inteligentes.

III. OBJETIVOS

- Crear un sistema de votación digital seguro, usando blockchain dándonos **seguridad y** confianza sin precedentes.
- Hacer que la votación electrónica sea segura, transparente y accesible para los estudiantes UNI.
- Proteger el sistema de votación frente a manipulaciones externas o internas mediante el uso de tecnología blockchain, garantizando la integridad de cada voto emitido. Además, permite que cada voto pueda ser verificado individualmente sin comprometer el anonimato del votante, asegurando así confianza y trazabilidad en todo el proceso.
- Asegurar que los resultados del proceso electoral sean públicos, transparentes y auditables por cualquier participante interesado. Al mismo tiempo, garantizar que solo los estudiantes debidamente autorizados —verificados a través de su correo institucional de la UNI— puedan acceder al sistema y emitir su voto.

IV. DIAGRAMA UML

Un diagrama UML (Lenguaje Unificado de Modelado) es una herramienta visual utilizada para representar la estructura y comportamiento de un sistema de software. Permite organizar clases, atributos, métodos y las relaciones entre distintos componentes, facilitando así la planificación y comprensión del sistema antes de su programación.

En este caso, el diagrama UML representa el diseño estructural de un sistema de votación electrónica basado en blockchain (ver figura 1). Se muestran clases clave como Votante, Candidato, BlockchainGestor e IzyVOTE, junto con sus atributos, métodos y las relaciones entre ellas. La clase Cédula se encarga de validar la identidad del votante, mientras que ResultadosVotación permite procesar y mostrar los resultados. Además, la clase "facultad" se encarga de tareas administrativas como auditorías y generación de reportes. Todo el sistema se apoya en la clase BlockchainGestor, que permite la interacción con la red blockchain para garantizar la integridad y seguridad del proceso electoral.

Este diagrama proporciona una visión clara de cómo se estructura e interconecta cada parte del sistema, sirviendo como base para su posterior desarrollo.

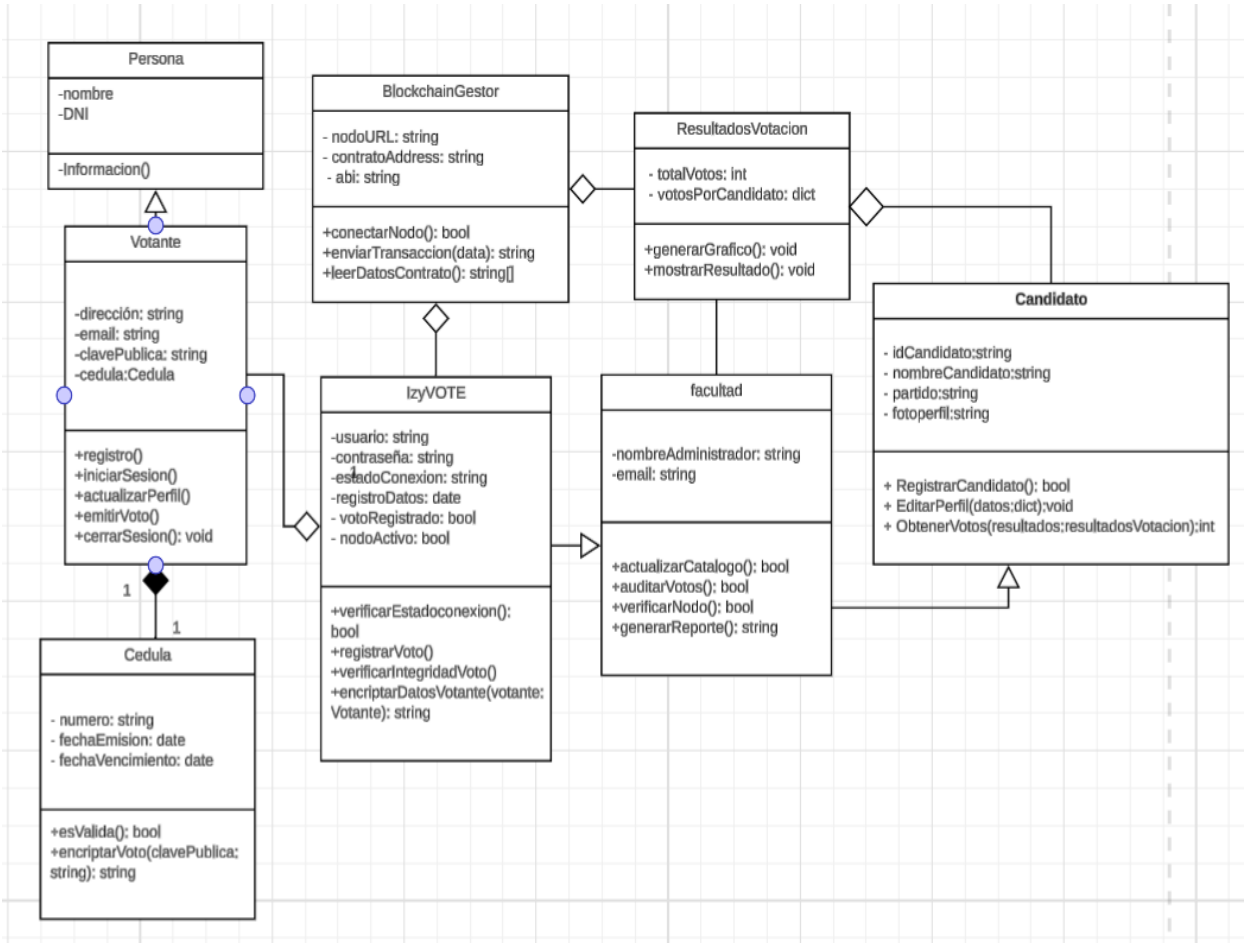

figura 1

El sistema está compuesto por ocho clases principales, cada una con atributos y métodos que interactúan entre sí para implementar un sistema de votación electrónica basado en blockchain (ver figura 1):

1. Persona: Clase base que representa a una persona con atributos como nombre y DNI. Contiene el método Informacion() para acceder a sus datos personales.
2. Votante: Hereda de la clase Persona y representa al usuario que emite el voto. Incluye atributos como dirección, email, clave pública y un objeto de tipo Cédula. Sus métodos permiten registrarse, iniciar y cerrar sesión, actualizar perfil y emitir un voto.
3. Cédula: Contiene los datos de identidad del votante, como número, fechas de emisión y vencimiento. Sus métodos permiten validar la cédula y encriptar con clave pública.
4. IzyVOTE: Gestiona el acceso y la integridad del sistema de votación. Incluye atributos como usuario, contraseña, estado de conexión y si el voto fue registrado. Sus métodos permiten verificar la conexión, registrar votos, validar la integridad del voto y encriptar los datos del votante.
5. BlockchainGestor: Encargada de la interacción con la red blockchain. Posee los atributos del nodo, dirección del contrato y ABI. Sus métodos permiten conectar con el nodo, enviar transacciones y leer datos del contrato.
6. ResultadosVotación: Clase que gestiona el conteo de votos. Tiene atributos como el total de votos y un diccionario de votos por candidato. Sus métodos permiten generar gráficos y mostrar resultados.
7. Candidato: Representa a un participante en el proceso electoral. Incluye atributos como ID, nombre, partido y foto de perfil. Sus métodos permiten registrar, editar el perfil y obtener votos.
8. Facultad: Representa la entidad organizadora del proceso electoral. Tiene atributos administrativos y métodos para actualizar el catálogo, auditar votos, verificar nodos y generar reportes del proceso.

Relaciones entre clases:

1. Relación entre Votante y Persona
    - Tipo: Herencia
    - Descripción: La clase Votante hereda de Persona, lo que significa que un votante es una persona y comparte atributos comunes como nombre y DNI. Esta

relación permite extender las capacidades de Persona para incluir funcionalidades específicas de un votante, como emitir votos.

2. Relación entre Votante y Cédula
    - Tipo: Asociación de agregación
    - Descripción: Cada Votante tiene una instancia de Cédula. La relación indica que la cédula pertenece al votante, pero puede existir de manera independiente. La cédula permite validar la identidad del votante mediante sus datos.
3. Relación entre Votante e IzyVOTE
    - Tipo: Asociación
    - Descripción: El Votante se comunica con IzyVOTE para acceder al sistema de votación. Esta clase sirve como puente para verificar conexiones, registrar votos y garantizar la seguridad del proceso.
4. Relación entre IzyVOTE y BlockchainGestor
    - Tipo: Asociación
    - Descripción: IzyVOTE interactúa con BlockchainGestor para enviar transacciones y validar información en la red blockchain. Esta relación permite que el sistema almacene los votos de forma segura y trazable.
5. Relación entre BlockchainGestor y ResultadosVotación
    - Tipo: Asociación
    - Descripción: BlockchainGestor proporciona los datos necesarios para que ResultadosVotación pueda contabilizar y mostrar los resultados electorales. Esta relación refleja cómo los datos de la blockchain alimentan el sistema de resultados.
6. Relación entre ResultadosVotación y Candidato
    - Tipo: Asociación
    - Descripción: ResultadosVotación mantiene un registro de los votos obtenidos por cada Candidato, utilizando un diccionario. Esta relación es necesaria para construir los resultados finales de la elección.
7. Relación entre Facultad y Candidato
    - Tipo: Asociación de agregación
    - Descripción: La Facultad administra o supervisa a los Candidatos. Puede actualizarlos o gestionarlos, pero los candidatos pueden existir de forma autónoma. La relación representa una función administrativa.
8. Relación entre Facultad y ResultadosVotación

- Tipo: Asociación
- Descripción: La Facultad accede a los resultados para verificar o auditar el proceso. Esta relación refleja el rol fiscalizador o de control que cumple esta clase dentro del sistema.

Estas relaciones reflejan una arquitectura orientada a objetos bien organizada, donde cada clase cumple una función específica y colabora con otras para permitir un proceso de votación electrónico seguro, transparente y verificable.

Cabe señalar que este es el diagrama inicial del sistema, correspondiente al código base actual. A lo largo del desarrollo, algunas estructuras han sido modificadas o ampliadas, pero el diseño conserva su esencia y propósito funcional.

V. CLASES Y DISEÑO DEL CÓDIGO

A continuación, se describen las principales clases (más importantes), atributos, bibliotecas y métodos, que se usaron para el proyecto IzyVote.

Bibliotecas y herramientas principales

hashlib: Cálculo de hashes SHA-256 para proteger el anonimato del votante y garantizar la integridad de datos.
json: Serialización y deserialización de objetos Python a JSON, especialmente útil en la API REST y en la construcción de bloques de votos.
datetime: Gestión de marcas de tiempo en transacciones y códigos de verificación.
Flask: Framework web para exponer rutas HTML y API JSON.
Flask-SQLAlchemy: ORM para definir modelos de datos y simplificar operaciones con la base de datos.
random & string: Generación de códigos de verificación aleatorios.

Descripción de las clases:

1. Election(ubicada en models.py):

   Representa el permiso de una elección disponible en la plataforma.

```python
class Election(db.Model):
    """Administrar elecciones"""
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(200), nullable=False)
    description = db.Column(db.Text)
    options = db.Column(db.Text, nullable=False)  # JSON string para opciones de votación
    is_active = db.Column(db.Boolean, default=True)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
```

2. VerificationCode(ubicada en models.py):

   Es la clase encargada de verificar los codigos que se envian por correo para validar el derecho al voto.

```python
class VerificationCode(db.Model):
    """Conecta códigos de verificación con elecciones específicas"""
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), nullable=False)
    code = db.Column(db.String(6), nullable=False)
    election_id = db.Column(db.Integer, db.ForeignKey('election.id'), nullable=False)
    used = db.Column(db.Boolean, default=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    election = db.relationship('Election', backref='verification_codes')
```

3. VoterHistory(ubicada en models.py):

Registra qué correos (en forma de hash) ya han emitido voto en cada elección.

```python
class VoterHistory(db.Model):
    """Garantiza que un email solo vote UNA vez por elección"""
    id = db.Column(db.Integer, primary_key=True)
    email_hash = db.Column(db.String(64), nullable=False)  # Hashed email for privacy
    election_id = db.Column(db.Integer, db.ForeignKey('election.id'), nullable=False)
    voted_at = db.Column(db.DateTime, default=datetime.utcnow)

    election = db.relationship('Election', backref='voter_history')

    __table_args__ = (db.UniqueConstraint('email_hash', 'election_id', name='unique_voter_election'),)
```

4. EmailService(ubicada en email_service.py):

   Encapsula la generación y envío de correos de verificación y confirmación (es el encargado de mandar los correos de verificación).

```python
class EmailService:
    """Servicio de correo electrónico para enviar códigos de verificación y confirmaciones de voto"""

    def __init__(self):
        self.smtp_server = os.environ.get("SMTP_SERVER", "smtp.gmail.com")
        self.smtp_port = int(os.environ.get("SMTP_PORT", "587"))
        self.email_username = os.environ.get("EMAIL_USERNAME", "sistemadevotacionblockchain@gmail.com")
        self.email_password = os.environ.get("EMAIL_PASSWORD")

    def generate_verification_code(self) -> str:
        """Genera un código de verificación aleatorio de 6 dígitos"""
        return ''.join(random.choices(string.digits, k=6))

    def send_verification_email(self, to_email: str, verification_code: str, election_title: str) -> bool:
        """Envía un correo de verificación"""
        try:
            msg = MIMEMultipart()
            msg['From'] = self.email_username
            msg['To'] = to_email
            msg['Subject'] = f"Código de Verificación - {election_title}"

            body = f"""
            <html>
            <body>
                <h2>Verificación de Votación</h2>
                <p>Hola,</p>
                <p>Tu código de verificación para la elección "<strong>{election_title}</strong>" es:</p>
                <h3 style="color: #007bff; font-size: 24px; letter-spacing: 3px;">{verification_code}</h3>
                <p>Este código es válido por 30 minutos.</p>
                <p>Si no solicitaste este código, puedes ignorar este correo.</p>
                <br>
                <p>Saludos,<br>Sistema de Votación Blockchain</p>
            </body>
            </html>
            """

            msg.attach(MIMEText(body, 'html'))

            # Envía correo real si las credenciales están proporcionadas
            if self.email_password:
                server = smtplib.SMTP(self.smtp_server, self.smtp_port)
                server.starttls()
                server.login(self.email_username, self.email_password)
                text = msg.as_string()
                server.sendmail(self.email_username, to_email, text)
                server.quit()
                logging.info(f"Verification email sent to {to_email}")
            else:
                # Para desarrollo, solo registra el contenido del correo
                logging.info(f"Development mode - Verification email to {to_email}: Code {verification_code}")
                print(f"VERIFICATION EMAIL - To: {to_email}, Code: {verification_code}")

            return True

        except Exception as e:
            logging.error(f"Error sending verification email: {str(e)}")
            return False

    def send_vote_confirmation(self, to_email: str, election_title: str) -> bool:
        """Send vote confirmation email"""
        try:
            msg = MIMEMultipart()
            msg['From'] = self.email_username
            msg['To'] = to_email
            msg['Subject'] = f"Voto Confirmado - {election_title}"

            body = f"""
            <html>
            <body>
                <h2>¡Voto Registrado Exitosamente!</h2>
                <p>Hola,</p>
                <p>Tu voto para la elección "<strong>{election_title}</strong>" ha sido registrado exitosamente en la blockchain.</p>
                <p>Características de tu voto:</p>
                <ul>
                    <li>✅ Verificado y válido</li>
                    <li>🔒 Completamente anónimo</li>
                    <li>⛓️ Almacenado en blockchain</li>
                    <li>🛡️ Inmutable y seguro</li>
                </ul>
                <p>Gracias por participar en el proceso democrático.</p>
                <br>
                <p>Saludos,<br>Sistema de Votación Blockchain</p>
            </body>
            </html>
            """

            msg.attach(MIMEText(body, 'html'))

            # Envía correo real si las credenciales están proporcionadas
            if self.email_password:
                server = smtplib.SMTP(self.smtp_server, self.smtp_port)
                server.starttls()
                server.login(self.email_username, self.email_password)
                text = msg.as_string()
                server.sendmail(self.email_username, to_email, text)
                server.quit()
                logging.info(f"Vote confirmation sent to {to_email}")
            else:
                # Para desarrollo, solo registra el contenido del correo
                logging.info(f"Development mode - Vote confirmation to {to_email}")
                print(f"VOTE CONFIRMATION EMAIL - To: {to_email}")

            return True

        except Exception as e:
            logging.error(f"Error sending vote confirmation: {str(e)}")
            return False
```

5. VotingBlockchain (ubicada en blockchain.py):

Es la cadena de bloques inmutables donde cada voto es registrado como una transacción:

```python
class VotingBlockchain:
    """Implementación de la cadena de bloques para el sistema de votación"""

    def __init__(self):
        self.chain: List[Block] = []
        self.pending_votes: List[Dict] = []
        self.blockchain_file = "instance/blockchain.json"
        self.load_blockchain()
        if not self.chain:
            self.create_genesis_block()

    def create_genesis_block(self):
        """Create the first block in the chain"""
        genesis_block = Block(0, [], "0")
        genesis_block.mine_block()
        self.chain.append(genesis_block)

    def get_latest_block(self) -> Block:
        """Get the most recent block"""
        return self.chain[-1]

    def add_vote(self, election_id: int, option: str):
        """Add a vote to pending votes"""
        vote = {
            "election_id": election_id,
            "option": option,
            "timestamp": datetime.utcnow().isoformat(),
            "vote_id": hashlib.sha256(f"{election_id}{option}{datetime.utcnow()}".encode()).hexdigest()
        }
        self.pending_votes.append(vote)

    def mine_pending_votes(self):
        """Mina todos los votos pendientes en un nuevo bloque"""
        if not self.pending_votes:
            return

        block = Block(
            len(self.chain),
            self.pending_votes.copy(),
            self.get_latest_block().hash
        )
        block.mine_block()
        self.chain.append(block)
        self.pending_votes = []
        self.save_blockchain()  # Auto-save after mining

    def get_votes_for_election(self, election_id: int) -> List[Dict]:
        """Obtiene todos los votos para una elección específica"""
        votes = []
        for block in self.chain:
            for vote in block.votes:
                if vote["election_id"] == election_id:
                    votes.append(vote)
        return votes

    def get_vote_count(self, election_id: int) -> Dict[str, int]:
        """Obtiene el conteo de votos para una elección específica"""
        votes = self.get_votes_for_election(election_id)
        vote_count = {}
        for vote in votes:
            option = vote["option"]
            vote_count[option] = vote_count.get(option, 0) + 1
        return vote_count

    def is_chain_valid(self) -> bool:
        """Valida la cadena de bloques"""
        for i in range(1, len(self.chain)):
            current_block = self.chain[i]
            previous_block = self.chain[i - 1]

            if current_block.hash != current_block.calculate_hash():
                return False

            if current_block.previous_hash != previous_block.hash:
                return False

        return True

    def save_blockchain(self):
        """Guarda la cadena de bloques en un archivo"""
        try:
            os.makedirs("instance", exist_ok=True)
            with open(self.blockchain_file, 'w') as f:
                json.dump({
                    "chain": [block.to_dict() for block in self.chain],
                    "pending_votes": self.pending_votes
                }, f, indent=2)
        except Exception as e:
            print(f"Error saving blockchain: {e}")

    def load_blockchain(self):
        """Carga la cadena de bloques desde un archivo"""
        try:
            if os.path.exists(self.blockchain_file):
                with open(self.blockchain_file, 'r') as f:
                    data = json.load(f)

                    # Reconstruct blocks
                    for block_data in data.get("chain", []):
                        block = Block(
                            block_data["index"],
                            block_data["votes"],
                            block_data["previous_hash"]
                        )
                        block.timestamp = block_data["timestamp"]
                        block.nonce = block_data["nonce"]
                        block.hash = block_data["hash"]
                        self.chain.append(block)

                    self.pending_votes = data.get("pending_votes", [])
        except Exception as e:
            print(f"Error loading blockchain: {e}")
```

VI. CONCLUSIONES

- La implementación de blockchain en procesos de votación permite garantizar la seguridad e integridad de cada voto, eliminando la posibilidad de alteraciones, duplicaciones o manipulaciones, y brindando mayor confianza al votante.
- IzyVote ha demostrado que es posible combinar tecnología avanzada con una experiencia de usuario sencilla y accesible, permitiendo que cualquier estudiante pueda participar en el proceso electoral sin dificultades técnicas.
- El uso del correo institucional como método de autenticación asegura que únicamente estudiantes autorizados puedan registrarse y votar, fortaleciendo la validez del proceso y evitando fraudes.
- La transparencia del sistema se ve reforzada por la naturaleza pública y auditable del blockchain, lo cual permite que los resultados puedan ser verificados por cualquier participante sin comprometer la privacidad de los votantes.
- La arquitectura web empleada, basada en tecnologías como HTML, CSS, JavaScript, Python, Flask y SQL, demuestra la viabilidad de construir sistemas democráticos eficientes utilizando herramientas de desarrollo web modernas.
- IzyVote representa un paso hacia la transformación digital en contextos universitarios, abriendo la puerta a procesos más confiables, eficientes y accesibles para toda la comunidad estudiantil.