Security development report: B2D Venture

Prepared For

Assistant Professor Usa Sammapun

Prepared By

Wissarut Kanasub	651054572
Chaiyawut Thengket	651054549
Sukprachoke Leelapisuth	6510545748

This Project is collected and evaluated in
01219325 Software Development Security

Software and knowledge Engineers

B2DVenture
web application Security report

`

In today's rapidly evolving world, innovations and new technologies are being developed at an unprecedented pace, sparking creativity and inspiration among people. However, turning these ideas into reality often requires a crucial resource: funding.`

This online investment platform is designed to bridge the gap between startup entrepreneurs with innovative ideas but limited financial resources, and investors who are eager to support them. This platform provides a space where entrepreneurs can showcase their ideas, while investors can easily find opportunities that align with their interests.`

Our goal is to create an environment that fosters connections between entrepreneurs and investors, enabling them to collaborate and bring new innovations to life that can positively impact the world.

Breif Use cases:

Investor:

- **Register/Login:** Register an account or log into the system.
- **View & Search Fundraising campaign:** Browse or search for available fundraising opportunities.
- **View Company Details:** Read details about specific companies.
- **Invest in Companies:** Select and invest in a company.
- **Track Investments:** Track the history of investments(Statements).
- **View Portfolio:** View overall investment portfolio.

Admin:

- **Access Admin Control Panel:** Access the control panel to manage fundraising and users.
- **Manage Fundraising Campaign**: Create, update, delete, and view fundraising campaigns.
- **View Users:** View user's account details.
- **View Fundraising Statements:** View detailed statements for each fundraising campaign, including user investments, timestamps, and amounts.

Company:

- **Contact Admin:** Contact the Admin to provide information and initiate a fundraising campaign.

Team member:

- Wissarut Kanasub 6510545721 Sec450
- Chaiyawut Thengket 651054549 Sec450
- Sukprachoke Leelapisuth 6510545748 Sec450

OWASP ASVS Security Requirements

- **Authentication and Session Management**
  - **Register/Login (Auth0)**
    - **V2.1 Authentication**: Ensures secure login using Auth0's Universal Login, aligned with OAuth standards.
    - **V3.1 Session Management**: Manages session tokens with short expiration times for enhanced security.

- **Investor Use Cases**
  - **View & Search Fundraising Campaigns / View Company Details**
    - **V4.3 Access Control**: Implements Role-Based Access Control (RBAC) using Auth0 and middleware to restrict access.
  - **Invest in Companies / Track Investments / View Portfolio**
    - **V7.1 Business Logic Security**: Enforces secure business logic and authorization processes for investments.
    - **V9.2 Data Protection**: Encrypts transaction data using HTTPS to secure sensitive information.

- **Admin Use Cases**
  - **Access Admin Control Panel / Manage Fundraising Campaigns / View Users / View Fundraising Statements**
    - **V4.2 Authentication and Authorization Verification**: Limits access to admin-only features through middleware.
    - **V4.4 Administrative Interface Security**: Protects admin interfaces and sensitive features using strict access controls

    .

- **Common Security Features Across the Application**
  - **Secure Payment Processing**
    - **V9.1 Data Protection**: Utilizes Stripe to securely manage payment data.
  - **GDPR/PDPA Compliance**
    - **V14.2 Privacy Verification**: Encrypts sensitive data and controls data access to comply with privacy regulations.
  - **Credential Recovery**
    - **V2.2 Credential Recovery Verification**: Ensures secure account recovery processes to prevent unauthorized access.

- **Machine-to-Machine API Protection**
    - **V13.2 RESTful Web Services**: Secures backend API endpoints using Auth0's M2M authentication to ensure only authorized machines with valid Bearer tokens can access APIs. This includes:
        - Preventing unauthorized access by separating user tokens and M2M tokens.
        - Limiting access to trusted machines or services.
        - Protecting endpoints such as `/api/accesstoken` with secret keys for secure token requests.

Threat Modeling Using STRIDE

Identify Core Components and Data Flows

1. **Front-end (Next.js)**: User interface where Investors, Admins, and Companies interact with the application.
2. **Backend (Next.js dynamic routing)**: Handles API requests, data processing, and communication between the front end and database.
3. **Auth0**: Manages user authentication and authorization.
4. **Stripe**: Handles payment processing.
5. **Database (MongoDB)**: Stores user and investment data securely.
6. **Machine-to-Machine API Protection**: Ensures secure communication between trusted services and backend APIs using Bearer tokens.

1. Spoofing

Definition: An attacker impersonates another user.
Relevant Components: Auth0, Frontend Login
Potential Threats:

- An attacker tries to log in as an Admin or Investor without proper credentials.
- A machine impersonates another trusted machine to access APIs.

Mitigations:

- Enforce strong password policies and monitor unusual login patterns in Auth0.
- Use **Machine-to-Machine API Protection** to validate Bearer tokens for API requests, ensuring only authorized machines can access backend services.

2. Tampering

Definition: Unauthorized modification of data.
Relevant Components: Backend API, Database, Machine-to-Machine API Protection
Potential Threats:

- An attacker tampers with investment data or alters user details by modifying API requests.
- An unauthorized machine attempts to modify sensitive data via backend APIs.

Mitigations:

- Validate and sanitize all incoming API requests to prevent malicious payloads.
- Use HTTPS to secure data in transit and prevent tampering during transmission.
- Implement strict **Machine-to-Machine API Protection** to authenticate and authorize requests, ensuring only trusted machines can modify data.

3. Repudiation

Definition: Denying an action or transaction.
Relevant Components: Backend API, Stripe, MongoDB
Potential Threats:

- A user denies making a specific investment transaction.
- A machine denies submitting API requests that caused a data change.

Mitigations:

- Maintain detailed logging of all actions, with timestamps and user identifiers.

4. Information Disclosure

Definition: Unauthorized access to sensitive data.
Relevant Components: Database, APIs, Frontend, Machine-to-Machine API Protection
Potential Threats:

- Sensitive data (e.g., investment history, user details) is exposed to unauthorized users or machines.
- An unauthorized machine accesses sensitive API endpoints.

Mitigations:

- Apply Role-Based Access Control (RBAC) to ensure users can only access authorized information.
- Use data encryption for sensitive data stored in MongoDB and enforce HTTPS for all communications.
- Restrict sensitive API endpoints to authorized machines using Machine-to-Machine API Protection.

5. Denial of Service (DoS)

Definition: Preventing legitimate users from accessing the service.
Relevant Components: Backend API, Frontend, Machine-to-Machine API Protection

6. Elevation of Privilege

Definition: Gaining unauthorized permissions.
Relevant Components: Auth0, Backend APIs, Machine-to-Machine API Protection
Potential Threats:

- An Investor or Company user tries to access Admin functionalities by exploiting authorization flaws.
- A machine with insufficient privileges attempts to access admin-level APIs.

Mitigations:

- Implement strict RBAC policies with Auth0 to restrict access based on user roles.
- Use middleware to enforce role checks on all sensitive routes.
- For machine-to-machine communication, assign specific roles and scopes to tokens, ensuring machines cannot escalate privileges.

security principles

1. Least Privilege

- **Description:** Users and services should have the minimum level of access required to perform their functions.
- **Implementation in Project:**
  - **Role-Based Access Control (RBAC):** Different roles (Investor, Admin, Company) have distinct access levels. For example, only Admins can access the admin control panel and manage fundraising campaigns, while Investors have read-only access to public information about campaigns.
  - **Middleware:** Enforces access control for routes and API endpoints, ensuring that each user role only accesses relevant data and actions.

2. Defense in Depth

- **Description:** Multiple layers of security controls are implemented to protect against different types of attacks.
- **Implementation in Project:**
  - **HTTPS and Data Encryption:** All data in transit is encrypted using HTTPS, and sensitive information (such as payment data via Stripe) is further protected.

3. Fail-Secure (Fail-Safe)

- **Description:** Systems should fail in a secure way to prevent exposure of sensitive data or unauthorized access.
- **Implementation in Project:**
  - **Access Denied by Default:** By default, routes and resources are protected, and only authorized users are allowed access through Auth0 RBAC. Any unauthorized request is rejected with a secure error message.
  - **Stripe Error Handling:** If a payment fails or an error occurs in processing, users are notified securely without exposing internal details of the system.

4. Separation of Duties

- **Description:** Important tasks are divided among multiple roles or users to prevent a single point of failure or abuse.
- **Implementation in Project:**
  - **Role Separation (Admin, Investor):** Each role has distinct permissions, ensuring that no single role can perform all actions. For example, only

Admins can manage fundraising campaigns.
    - **Financial Transactions (Stripe):** Only Investors can initiate investments, and the transaction processing is handled by Stripe, separating financial functions from other app functionalities.

5. Secure by Default

- **Description:** The application is configured to be secure out of the box, with sensible defaults that prioritize security.
- **Implementation in Project:**
    - **Default Auth0 Security Settings:** Auth0 enforces secure defaults such as password complexity requirements and token expiration settings.

6. Minimize Attack Surface

- **Description:** Reducing the number of entry points and exposed features that attackers could exploit.
- **Implementation in Project:**
    - **Limited Access to Sensitive Routes:** Only allowed machines can access api endpoints.

7. Complete Mediation

- **Description:** Every access to resources should be verified to ensure the user has the correct permissions.
- **Implementation in Your Project:**
    - **Auth0 RBAC:** Every API call and page access is checked to ensure the user has the appropriate role and permissions.
    - **Database Access Control:** Each data request is checked to ensure that users cannot view or modify information they do not have permission for, such as preventing Investors from viewing other users' portfolios.

8. Privacy by Design

- **Description:** The system is designed to protect user privacy, with a focus on minimizing data collection and ensuring compliance with privacy regulations.
- **Implementation in Project:**
    - **GDPR/PDPA Compliance:** Data handling and access controls align with

privacy regulations, ensuring data is collected, stored, and used responsibly.

- **Data Minimization and Encryption:** Only necessary user data is collected, and sensitive data is encrypted both in transit and at rest, protecting user privacy.

Task Mapping & Status Table

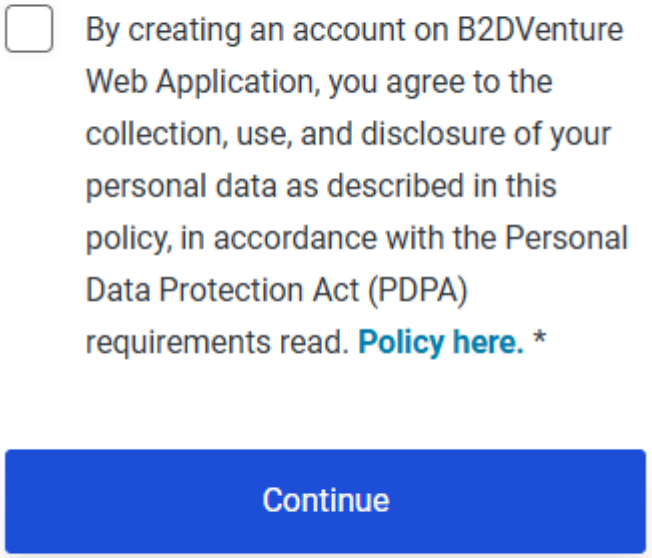
Task	Threat Model (STRIDE)	Security Design Principle(s)	Status
Privacy Notification and Consent Form (GDPR/PDPA)	Information Disclosure	Privacy by Design, Least Privilege	Success
Authentication (Login/Logout) and User Management	Spoofing	Authentication, Defense in Depth	Success
Password Masking with Option to Display Plain Text	Information Disclosure	Secure by Default	Success
Password Confirmation	Information Disclosure	Fail-Secure	No develop…
Password Requirements (Length, Characters, etc.)	Spoofing	Least Privilege, Secure by Default	Success
Password Hashing	Information Disclosure	Data Protection, Defense in Depth	Success
Authorization Process with Roles	Elevation of Privilege	Least Privilege, Separation of Duties	Success
Users Can't Access Unauthorized URLs	Elevation of Privilege	Least Privilege, Complete Mediation	Success
Session Management	Spoofing	Session Management, Defense in Depth	Success
CSRF Token	Tampering, Elevation of Privilege	Complete Mediation, Defense in Depth	Unnecessary **reason to support it.**
Secure HTTP Header Settings	Information Disclosure, Spoofing	Secure by Default, Defense in Depth	Success
Random Unique ID Generation	Tampering	Minimize Attack Surface	Success

Input Validation and SQL Injection Protection	Tampering, Information Disclosure	Defense in Depth, Input Validation	Success
Error Handling	Information Disclosure	Fail-Secure, Defense in Depth	Success
Logging and Monitoring	Repudiation	Complete Mediation, Fail-Secure	Success
Environment Variables	Information Disclosure	Privacy by Design, Secure by Default	Success
Database Encryption	Information Disclosure	Data Protection, Privacy by Design	Success
Security Code Analysis	-	Defense in Depth, Minimize Attack Surface	Success
DAST(Dynamic Application Security Testing) or Pen Test	-	Defense in Depth, Complete Mediation	Success

4 advanced security features Task

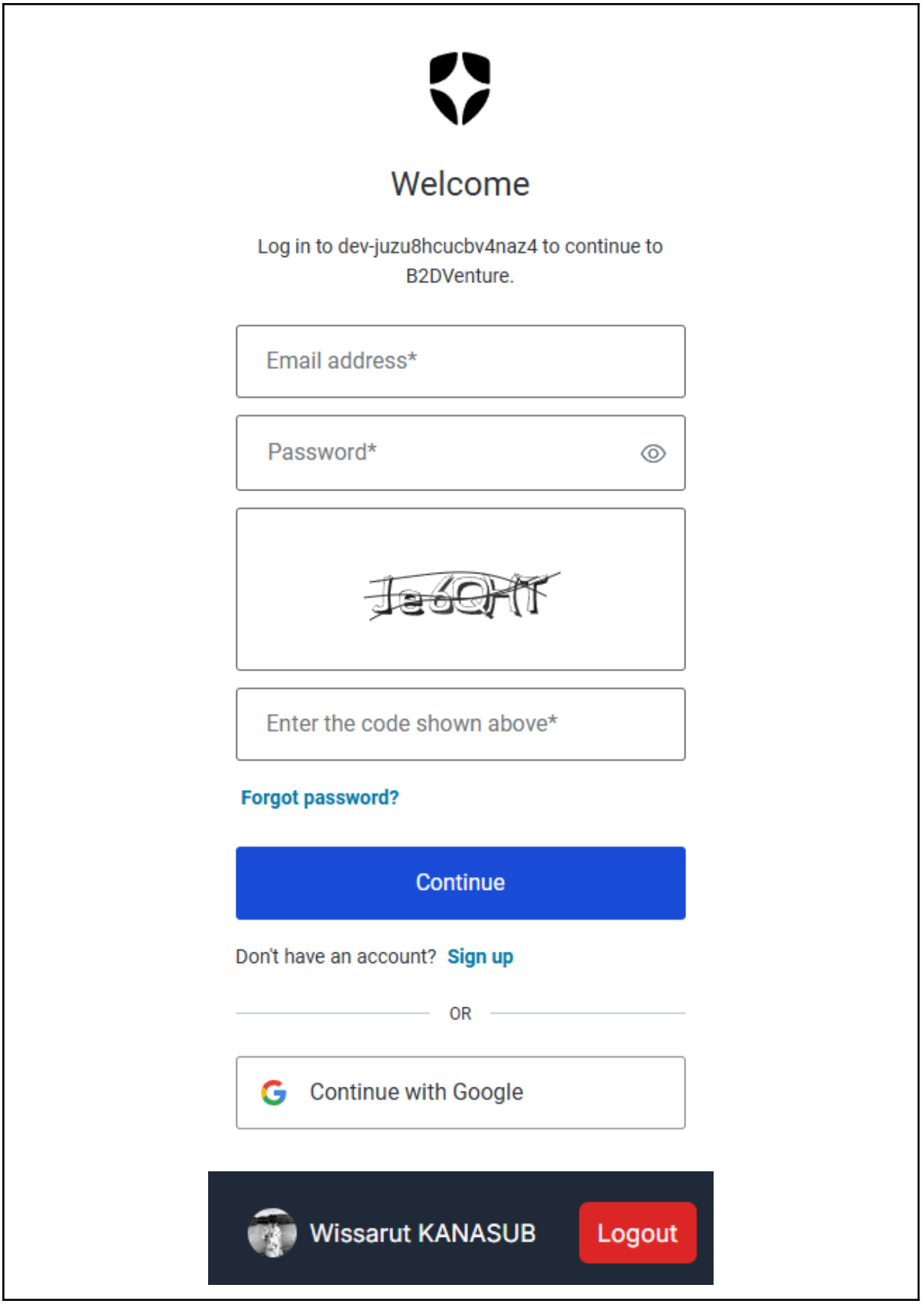
OAuth/OpenID Social Login	Spoofing, Information Disclosure	Authentication, Privacy by Design	Success
Captcha	Denial of Service, Spoofing	Minimize Attack Surface, Defense in Depth	Success
Login Timeout	Spoofing, Denial of Service	Defense in Depth, Session Management	Success
Forgot Password Feature	Spoofing	Fail-Secure, Authentication	Success

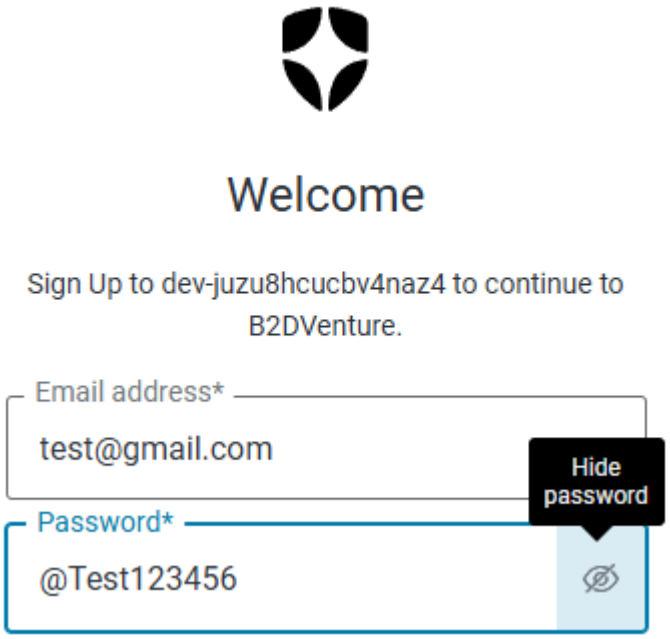
Task Description

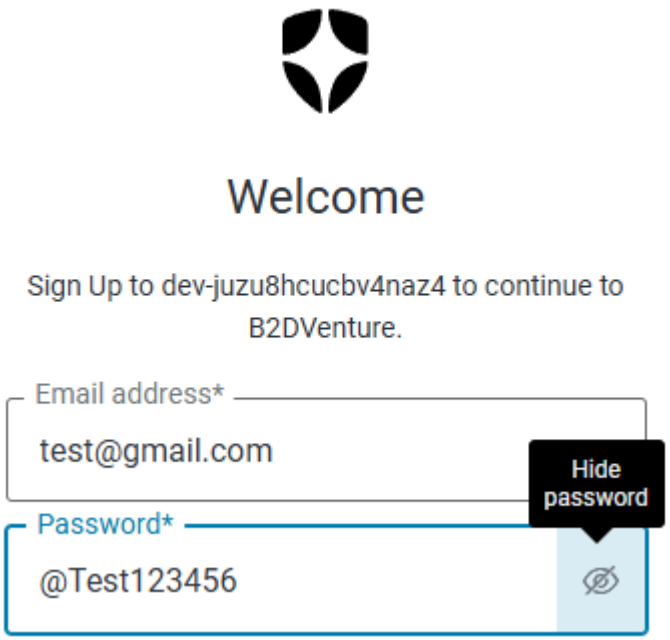
Task: Privacy Notification and Consent Form (GDPR/PDPA)
Description: We write privacy read here [Privacy Policy and Consent for B2DVenture Web Application · Qosanglesz/B2D Wiki]


Task: Authentication (Login/Logout) and User Management
Description: Auth0 provide login/ logout page by create/calling an API that Auth0 provideUs **Login:** “/auth/login” **logout:** “auth/logout”

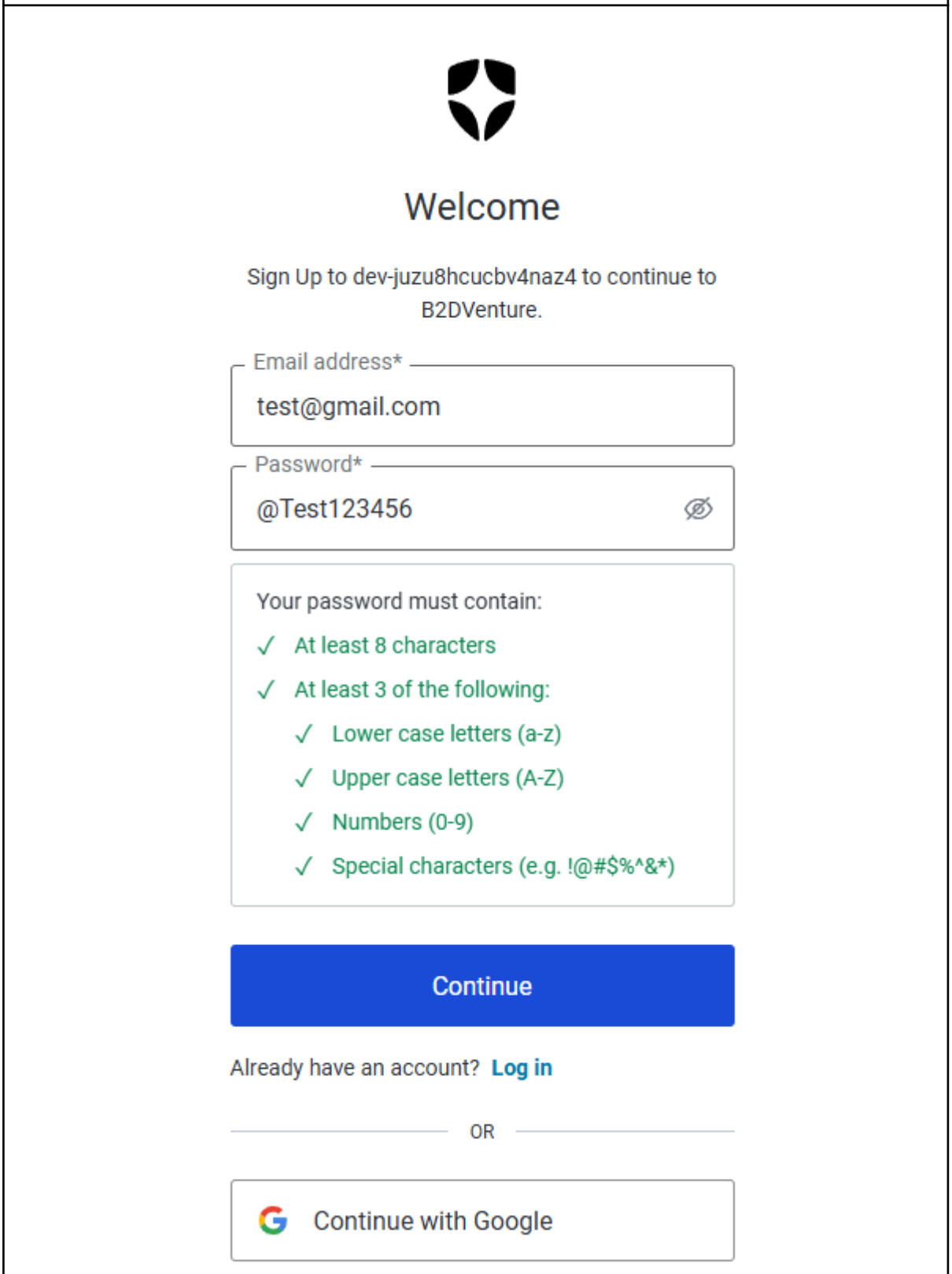
```
import { handleAuth, handleLogin} from '@auth0/nextjs-auth0';

export const GET = handleAuth({
    login : handleLogin (
        {
            authorizationParams : {
                audience : process.env.AUTH0_AUDIENCE,
                scope : 'openid profile email read:b2d-system'
            }
            ,returnTo: '/home'
        }
    )
});
```


Welcome
Log in to dev-juzu8hcucbv4naz4 to continue to B2DVenture.
Email address*
Password*
Enter the code shown above*
Forgot password?
Continue
Don't have an account? Sign up
OR
Continue with Google
Wissarut KANASUB
Logout

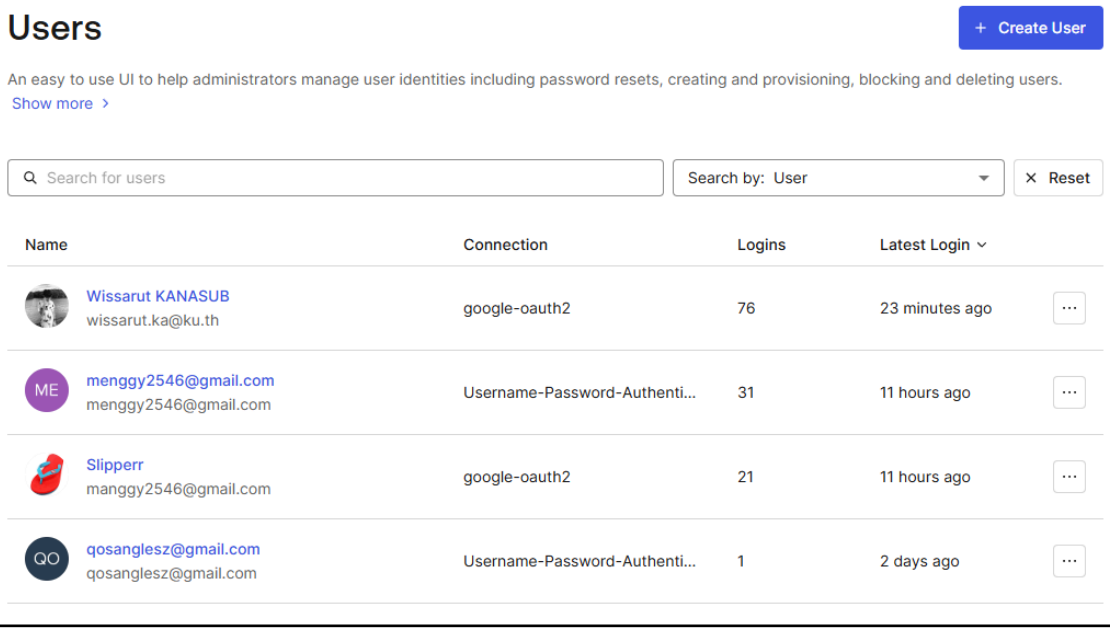
Task: Password Masking with Option to Display Plain Text
Description: D


Task: Password Confirmation
Description: Auth0 doesn't not provide this feature in their Universal login


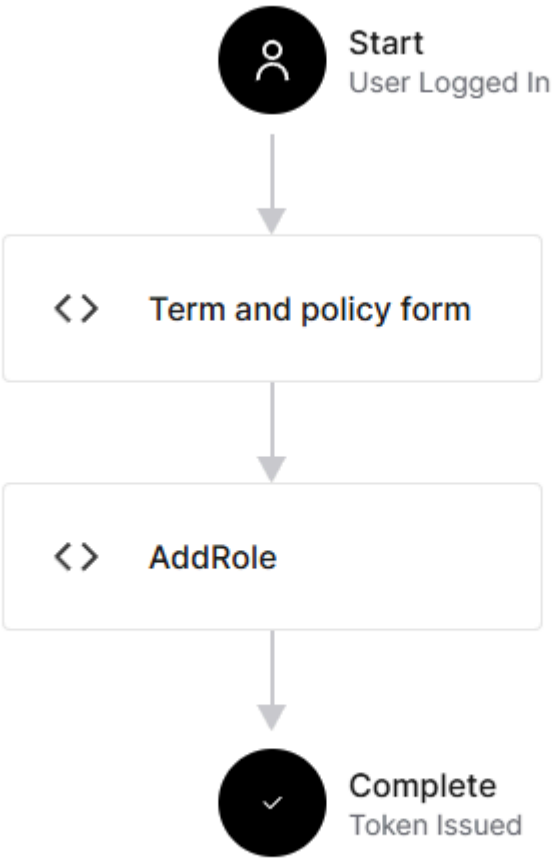
Task: Password Requirements (Length, Characters, etc.)
Description: Auth0 provide us this services in Universal login



Task: Password Hashing
Description: Auth0 Stores users by them self on their database and hashing users information using high secure algorithm such bcrypt, Argon2
None

Task: Authorization Process with Roles
Description: This authorization process defines two roles: • Common User – The default role assigned to all users upon registration. • Admin Role – A special role with additional permissions, managed through the Auth0 Dashboard. • The setup involves the following steps: • Role Assignment: All users are automatically assigned the "Common User" role by default. • Admin Role Management: The "Admin" role is manually assigned to specific users via the Auth0 Dashboard to allow access to admin-specific areas and actions. • Adding Role to User Session: In the Post Login Flow, the user's role is added to their session, enabling role-based access control within the application.



Assign Roles
Select roles to assign to this user. You may assign up to 50 roles per user.
Select Roles
Admin B2D
Admin can do everything
User
user can read
wissarut.dev@gmail.com


Start
User Logged In
Term and policy form
AddRole
Complete
Token Issued

Task: Users Can’t Access Unauthorized URLs

Description: This setup uses the `middleware.ts` file in Next.js, along with the Auth0 `get session` function, to enforce authentication and authorization checks. Unauthorized users are prevented from accessing restricted pages, such as admin pages. Here’s how the middleware achieves this:

- **Authentication Check:** Ensures users are logged in by redirecting unauthenticated users to the login page.
- **Role-Based Access Control:** Checks if a user has the “Admin” role to access admin-specific paths . “/admin”
- **User-Specific Data Access Control:** Restricts regular users from accessing or modifying other users' data by verifying that the user ID in the URL path matches the authenticated user’s ID.

Middleware.ts

```
// Redirect unauthenticated users to login page for other paths
if (!session) {
  console.log(`[${timestamp}] Unauthenticated access attempt on: ${pathname}`);
  return NextResponse.redirect(new URL(`${process.env.NEXT_PUBLIC_BASE_URL}/api/auth/login`));
}
```

```
// Restrict access to "/admin" paths to users with the "Admin B2D" role
if (pathname.startsWith("/admin") && !roles.includes("Admin B2D")) {
  console.log(`[${timestamp}] Access denied for user with sub: ${session.user.sub} on admin page: ${pathname}`);
  return NextResponse.redirect(new URL(`${process.env.NEXT_PUBLIC_BASE_URL}/home`));
}
```

Task: Random Unique ID Generation

Description:

- MongoDB automatically assigns a unique `_id` field to each document, which serves as a random UUID.
- In certain scenarios, custom unique ID generation may be required.
- The `uuidv4` library can be used to generate unique IDs separately, providing more flexibility and control over the ID generation process.


```
_id: ObjectId('670f87825a2b86ab234cab83')
id : "423546ce-e71a-4145-aa10-b81a1f31ad5d"
name : "Innovative Tech Ltd."
status : true
description : "Raising funds to revolutionize the AI and automation industry with gro…"
▸ pictureFiles : Array (5)
companyName : "Innovative Tech Ltd."
website : "https://innotech.com"
founderName : "Jane Doe"
email : "jane.doe@innotech.com"
linkedInProfile : "https://www.linkedin.com/in/janedoe"
companyStage : "AI and Automation"
industry : "Technology"
sector : "Artificial Intelligence and Machine Learning"
amountRaised : 103637
targetAmount : 1000000
teamSize : "100"
headquartersLocation : "San Francisco, CA"
productAvailable : true
location : "San Francisco, CA"
incorporationDate : "2024-10-16"
▸ investors : Array (7)
companyNumber : "1234567891"
companyVision : "To become the leading provider of AI-driven automation solutions acros…"
endInDate : "2024-12-31"
```

Task: Environment Variables

Description:

- Environment variables are managed using an `env.local` file.
- This approach ensures sensitive data, such as API keys and configuration settings, is securely stored and easily configurable.
- The `env.local` file is excluded from version control to prevent exposing sensitive information.

```
# AUTH0 variables
AUTH0_SECRET=''
AUTH0_BASE_URL=''
AUTH0_ISSUER_BASE_URL=''
AUTH0_CLIENT_ID=''
AUTH0_CLIENT_SECRET=''

AUTH0_AUDIENCE=''
AUTH0_SESSION_AUTO_SAVE=false

AUTH0_MANAGEMENT_CLIENT_ID=''
AUTH0_MANAGEMENT_CLIENT_SECRET=''

NEXT_PUBLIC_ACCESS_TOKEN_API_KEY=''


# MONGODB variables
MONGODB_URL=''

# STRIPE variables
STRIPE_SECRET_KEY=''
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY=''
STRIPE_WEBHOOK_SECRET=''

#NEXT
NEXT_PUBLIC_BASE_URL=''

UPLOADTHING_TOKEN=''

#CoinBase
COINBASE_COMMERCE_API_KEY=''
COINBASE_COMMERCE_WEBHOOK_SECRET=''
```

Task: Input Validation and SQL Injection Protection

Description:

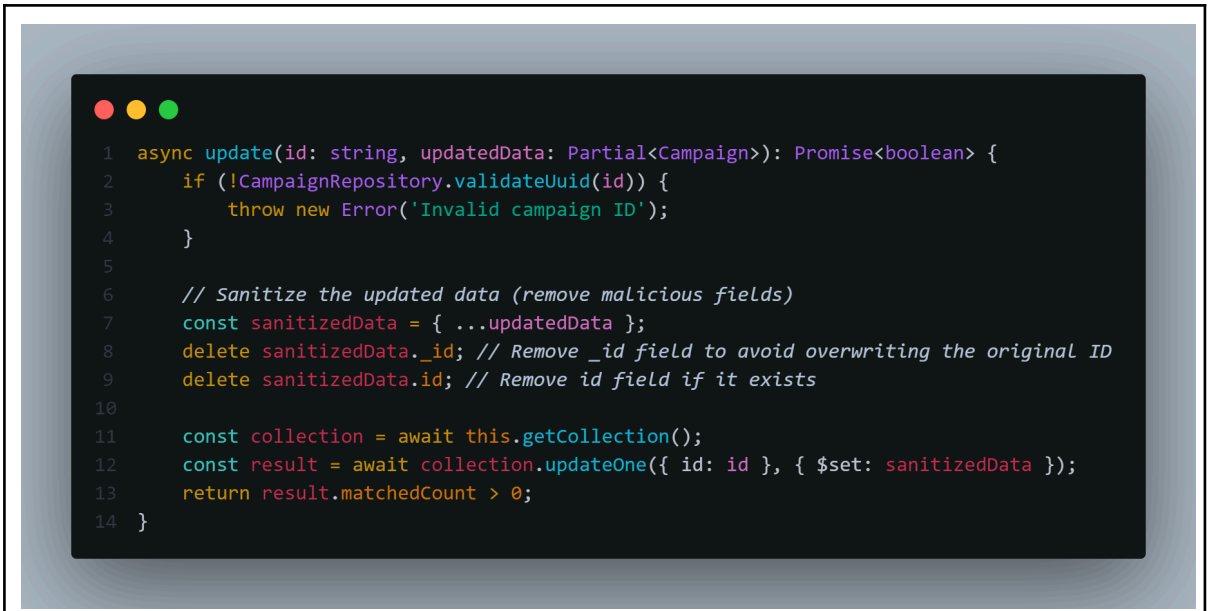
The web application, built with Next.js App Router and MongoDB, is inherently resistant to SQL injection due to MongoDB being a NoSQL database. However, to further secure the application, the following measures were implemented:

1. **UUIDv4 Validation**: Campaign IDs are validated to ensure they follow the UUIDv4 format, protecting against malformed IDs and NoSQL injection.
2. **Input Sanitization**: Before database operations, sensitive fields like `id` and `_id` are removed to prevent accidental overwriting of primary keys.
3. **Required Fields Validation**: Essential fields (e.g., `companyName`, `website`) are checked to ensure completeness before inserting or updating data.
4. **NoSQL Injection Protection**: Input validation and sanitization prevent malicious data from being executed in MongoDB queries, further securing the application.

In CampaignRepository.ts

```
// Validate if the ID is a valid UUIDv4
private static validateUuid(id: string): boolean {
    const uuidPattern = /^[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-4[0-9a-fA-F]{3}-[89aAbB][0-9a-fA-F]{3}-[0-9a-fA-F]{12}$/;
    return uuidPattern.test(id);
}
```

```
async delete(id: string): Promise<boolean> {
        if (!CampaignRepository.validateUuid(id)) {
            throw new Error('Invalid campaign ID');
        }

        const collection = await this.getCollection();
        const result = await collection.deleteOne({ id: id });
        return result.deletedCount > 0;
    }
```



```
async update(id: string, updatedData: Partial<Campaign>): Promise<boolean> {
    if (!CampaignRepository.validateUuid(id)) {
        throw new Error('Invalid campaign ID');
    }

    // Sanitize the updated data (remove malicious fields)
    const sanitizedData = { ...updatedData };
    delete sanitizedData._id; // Remove _id field to avoid overwriting the original ID
    delete sanitizedData.id; // Remove id field if it exists

    const collection = await this.getCollection();
    const result = await collection.updateOne({ id: id }, { $set: sanitizedData });
    return result.matchedCount > 0;
}
```

Task: Secure HTTP header setting

Description:
Enhance security by adding HTTP headers to protect the Next.js application from common web vulnerabilities.

Actions:

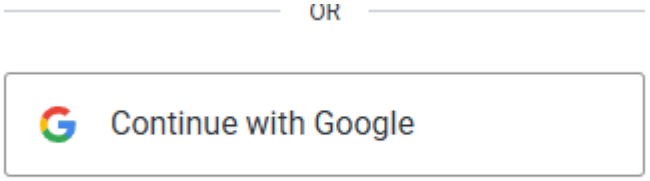
1. **Strict-Transport-Security (HSTS):** Enforces HTTPS communication, preventing MITM attacks.
2. **Content-Security-Policy (CSP):** Restricts content sources to prevent XSS attacks.
3. **X-Content-Type-Options:** Prevents MIME type sniffing by browsers.
4. **X-Frame-Options:** Blocks clickjacking by preventing the app from being embedded in iframes.
5. **X-XSS-Protection:** Activates the browser's XSS filter to block attacks.
6. **Referrer-Policy:** Limits the exposure of referrer information during navigation.

In Middleware.ts

```
response.headers.set('Strict-Transport-Security',
'max-age=31536000; includeSubDomains; preload');
  response.headers.set('Content-Security-Policy', "default-src
'self'; script-src 'self'; style-src 'self'; img-src 'self';
font-src 'self'; connect-src 'self';");
  response.headers.set('X-Content-Type-Options', 'nosniff');
  response.headers.set('X-Frame-Options', 'DENY');
  response.headers.set('X-XSS-Protection', '1; mode=block');
  response.headers.set('Referrer-Policy',
'no-referrer-when-downgrade');
```

Task: OAuth/OpenID Social Login

Description: Auth0 Provide many social media login but in this project we use google authentication



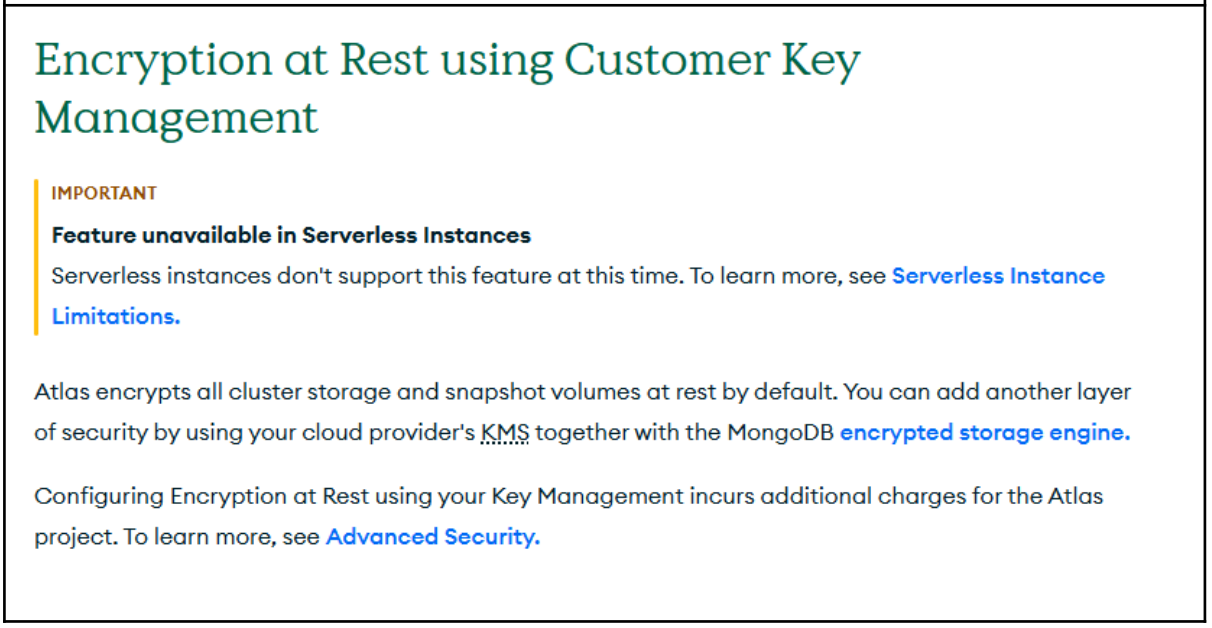
Task: Database Encryption

Description:
For database encryption, the tool being used is **MongoDB Atlas**, which automatically provides **Encryption at Rest** by default. This means that all data stored in the MongoDB Atlas clusters is encrypted at rest using **AES-256** encryption without requiring any manual configuration or implementation by the user. The encryption is handled by MongoDB Atlas, ensuring that sensitive data is secure without the need for custom coding or manual intervention.

As a result, there is no additional development work required to implement database encryption, as MongoDB Atlas takes care of it by default.

Reference:

- MongoDB Atlas Encryption at Rest: https://www.mongodb.com/docs/atlas/security/encryption-at-rest/



Encryption at Rest using Customer Key Management

IMPORTANT

Feature unavailable in Serverless Instances

Serverless instances don't support this feature at this time. To learn more, see Serverless Instance Limitations.

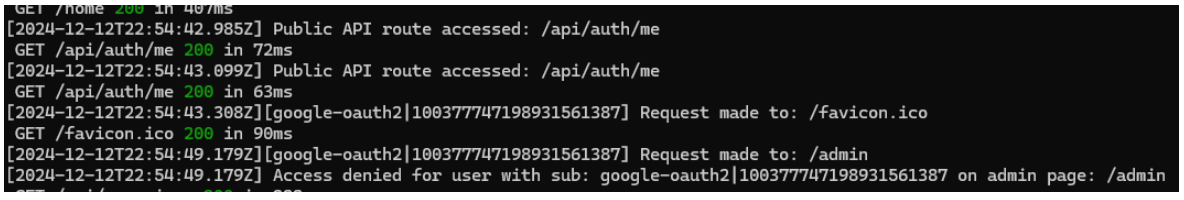
Atlas encrypts all cluster storage and snapshot volumes at rest by default. You can add another layer of security by using your cloud provider's KMS together with the MongoDB encrypted storage engine.

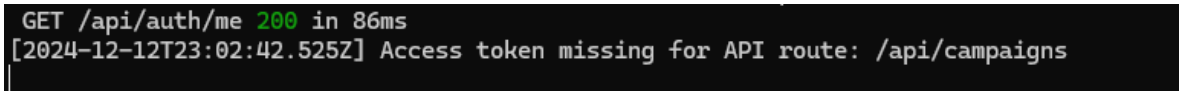
Configuring Encryption at Rest using your Key Management incurs additional charges for the Atlas project. To learn more, see Advanced Security.

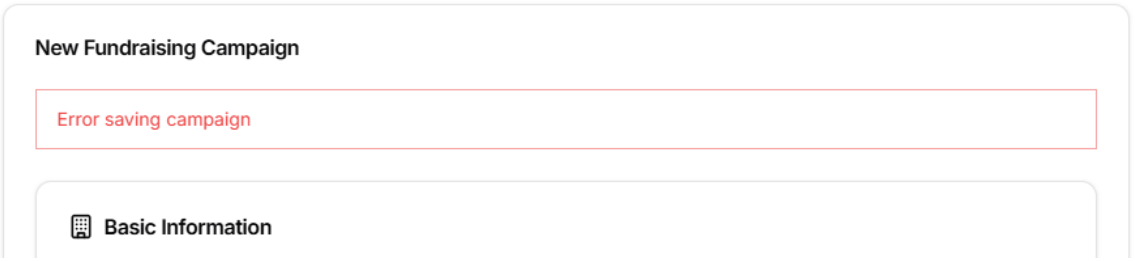
Task: Logging Monitoring & Error Handling

Description: In this project, we implement Logging and Error Handling strategies both for the server-side (SSR) and client-side components.

- **Server-Side (SSR) Logging:**
  - In the `middleware.ts`, we use `console.log()` for logging access attempts and activities. This helps monitor server-side events such as user authentication and access to various API routes.
  - For error handling in SSR, any potential issues (such as invalid tokens) are logged with detailed error messages and relevant context (including user `sub` and timestamp).
- **Client-Side Error Handling:**
  - On the client-side, we wrap potentially error-prone code in `try-catch` blocks to catch exceptions.
  - If an error occurs, we notify the user by displaying error messages directly on the screen, ensuring the user experience is not disrupted by unhandled errors.



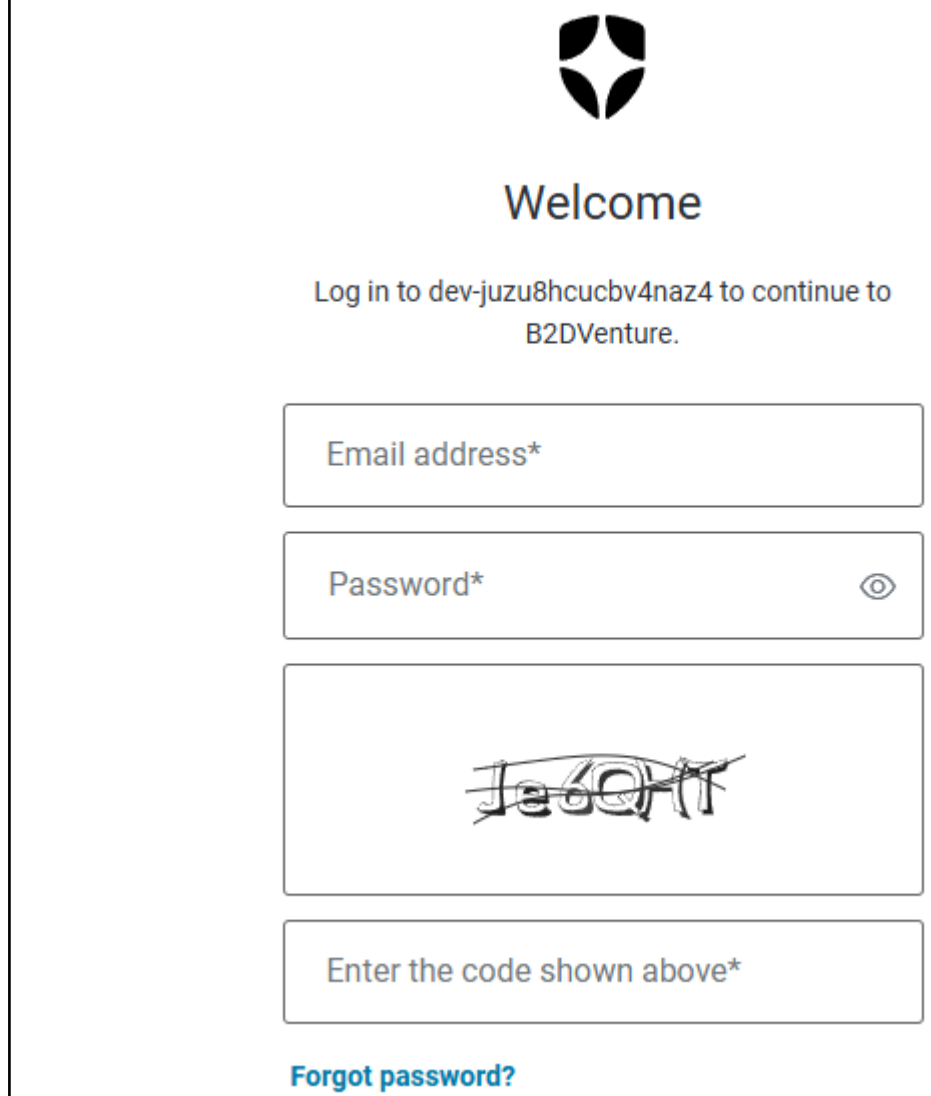




Task: Captcha

Description: In this project, we integrate **Captcha** functionality using **Auth0's** built-in service. This Captcha is triggered during key authentication processes to prevent automated bot activity.

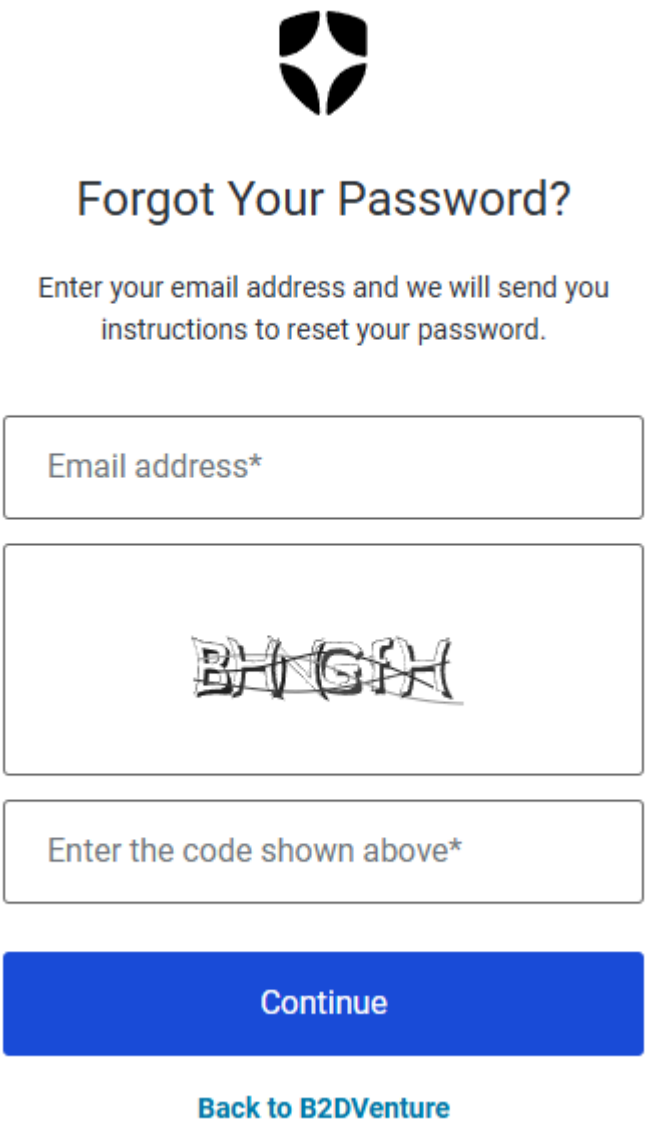
- **Triggering Events**:
  - **Login**: Captcha will appear during the login process to ensure the user is human.
  - **Sign Up**: Captcha will also be required during user sign-up to prevent automated account creation.
  - **Forgot Password**: If a user requests a password reset, Captcha will be triggered to avoid automated password reset requests.



Task: Forgot Password Feature

Description: The **Forgot Password** feature is implemented using **Auth0's** built-in service, which simplifies the process of password recovery and ensures secure handling of user credentials.

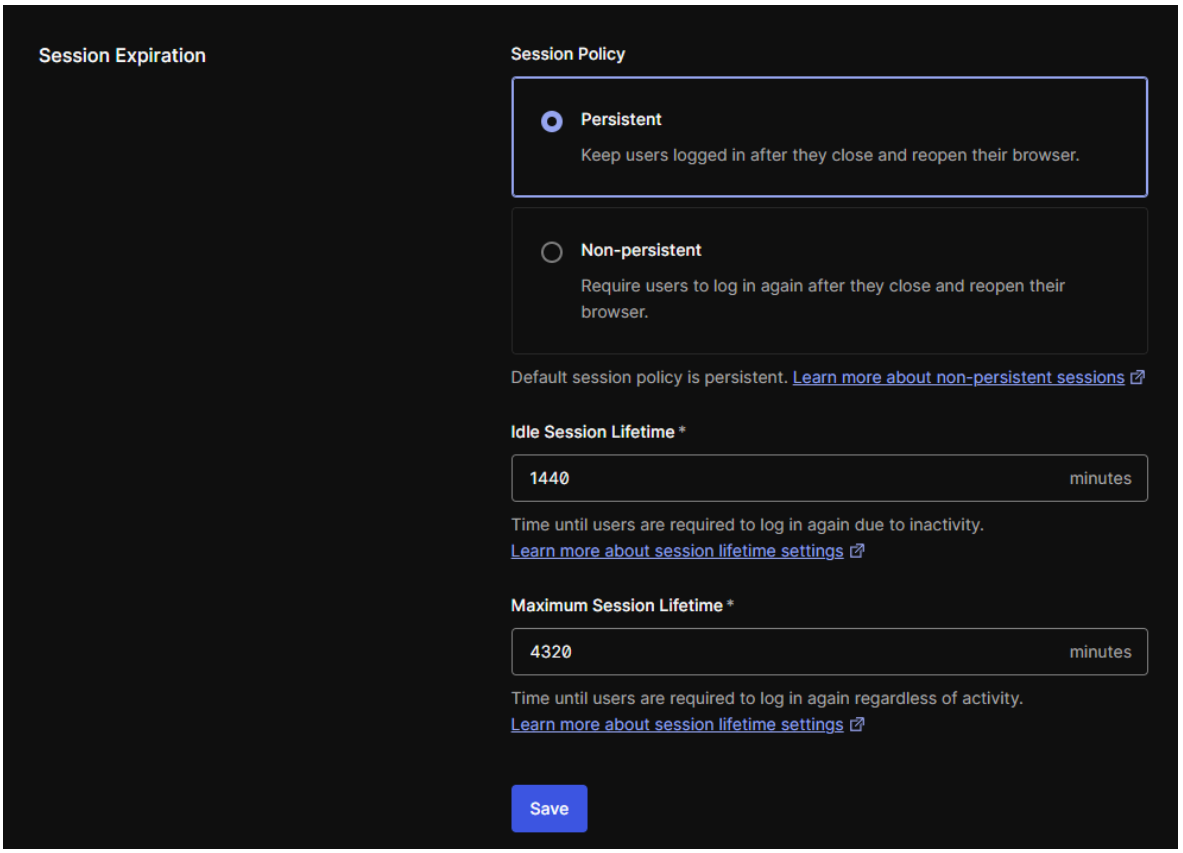
- **Functionality**:
  - When users forget their password, they can request a password reset by entering their registered email.
  - Auth0 will send a password reset email with a secure link for the user to reset their password.



Task: Login Timeout

Description: The **Login Timeout** feature is managed through **Auth0** by configuring session expiration settings.

- **Session Expiration**:
  - The session timeout is set to automatically expire after a period of inactivity.
  - **Idle Timeout**: The session will expire after 1 day of inactivity.
  - **Maximum Session Duration**: The session will not last longer than 3 days, ensuring users must re-authenticate after this period, enhancing security.



Task: Backend API Authentication with Tokens

Description: For securing the **backend API endpoints**, we are using **Auth0's Machine-to-Machine (M2M) API protection**. This ensures that only authorized systems or machines (like the frontend application) can access the backend services, using a valid **Bearer token**.

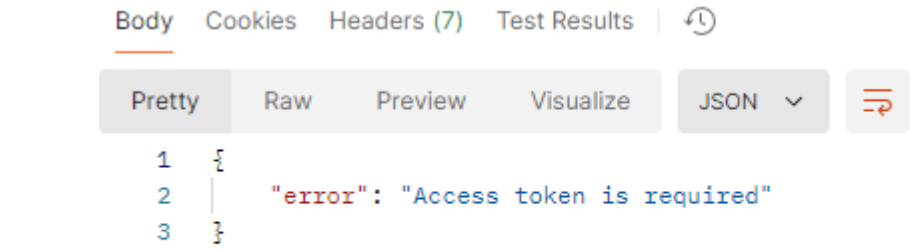
- **Bearer Token Requirement**:
  - The backend API endpoints are protected and require a valid Bearer token for access.
  - This setup ensures that only authenticated machines, like the frontend, can make requests to the backend API securely.
- **User and Role Separation**:
  - The token is separate for **user authentication** and **user roles**.
  - Role-based access control is enforced on frontend pages (e.g., Admin page), ensuring unauthorized users cannot access protected resources.
- **Public Endpoints**:
  - Certain endpoints (e.g., public APIs) may be set to allow open access without requiring authentication, but sensitive endpoints remain protected.
- **Special Token Endpoint**:
  - A special endpoint, such as **/api/accesstoken**, is used for requesting tokens from Auth0.
  - This endpoint requires the use of a **secret key** to ensure only authorized systems can request tokens.

Middleware.ts

```
// Validate API key for `/api/accesstoken` route
if (pathname.startsWith('/api/accesstoken')) {
  const apiKey = request.headers.get('accesstokenapikey');
  if (apiKey !== process.env.NEXT_PUBLIC_ACCESS_TOKEN_API_KEY) {
    console.error(`[${timestamp}] Unauthorized access attempt on: ${pathname} with invalid API key`);
    return NextResponse.json({ error: 'Unauthorized: Invalid or missing API key' }, { status: 401 });
  }
  return NextResponse.next();
}
```

```
// Validate Authorization header for other API routes
const token = request.headers.get('Authorization')?.split(' ')[1];
if (!token) {
  console.error(`[${timestamp}] Access token missing for API route: ${pathname}`);
  return NextResponse.json({ error: 'Access token is required' }, { status: 401 });
}
```

```
// Validate access token issuer and audience
try {
  const decoded = decodeJwt(token);
  if (decoded.iss === process.env.AUTH0_ISSUER_BASE_URL && decoded.aud === process.env.AUTH0_CLIENT_ID) {
    console.log(`[${timestamp}] Access token valid for user with sub: ${decoded.sub} for API route: ${pathname}`);
    return NextResponse.next();
  }
} catch (error) {
  console.error(`[${timestamp}] Invalid or expired access token for API route: ${pathname}`);
  return NextResponse.json({ error: 'Invalid or expired access token' }, { status: 403 });
}
}
```



Task: CSRF Token

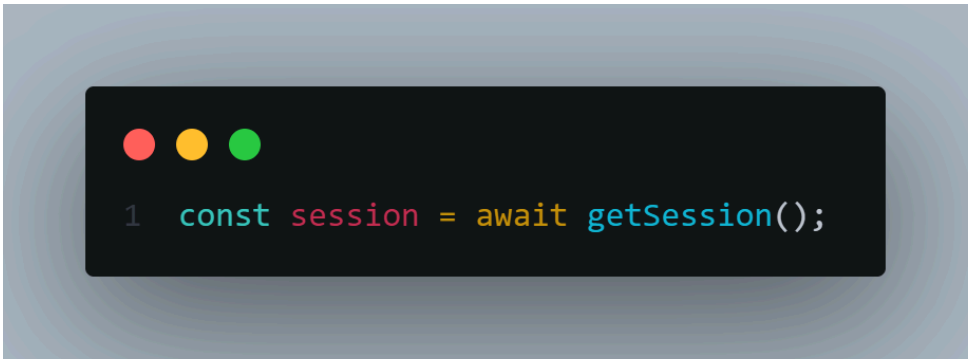
Description: By using **Machine-to-Machine (M2M) API protection** with Auth0, there's no need for a **CSRF token** because:

1. **M2M Communication**: The communication happens between trusted machines (e.g., your backend and frontend), not between a browser and a server, so there's no risk of Cross-Site Request Forgery (CSRF).
2. **Bearer Tokens**: Instead of relying on cookies, M2M protection uses **Bearer tokens** (like JWT), which are sent in the **Authorization header** of each request. This way, the server can verify that the request comes from an authenticated and authorized machine, not from an unauthorized user.
3. **No User Interaction**: CSRF attacks are a concern when a user is interacting with the web app (e.g., submitting forms or clicking links), but with M2M communication, there's no user directly involved in the request, so CSRF doesn't apply.

Task: Session Management

Description: The Auth0 SDK provides automatic session management services. Using the `api/auth/me` endpoint, we can easily retrieve the user's session. By calling a single line of code, we can manage and maintain the session state, simplifying authentication and session handling in our application. This eliminates the need for manual session management, as Auth0 handles user sessions seamlessly.

```
1  import { NextResponse } from 'next/server';
2  import { decodeJwt } from 'jose';
3  import { getSession } from '@auth0/nextjs-auth0/edge';
```

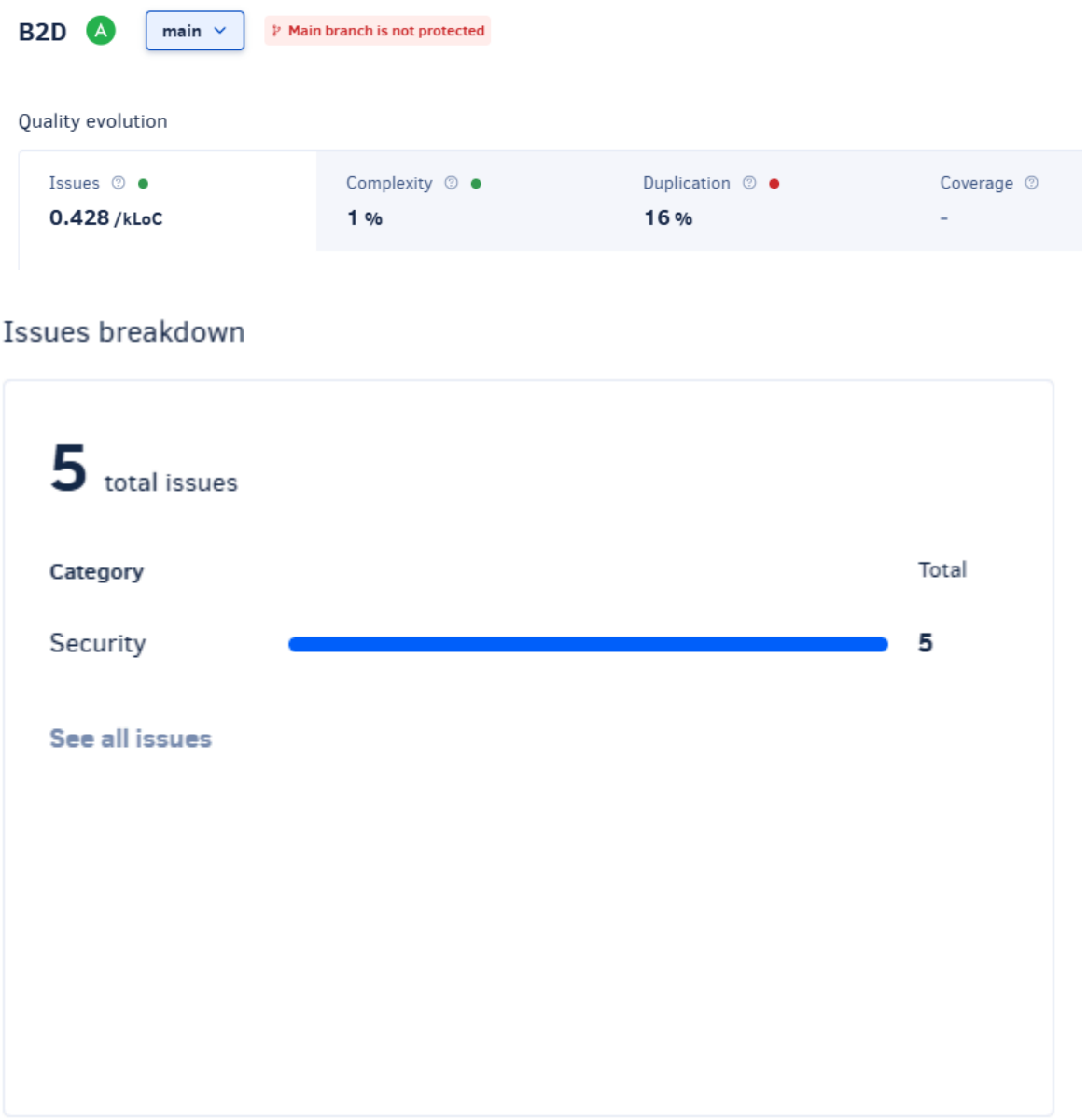


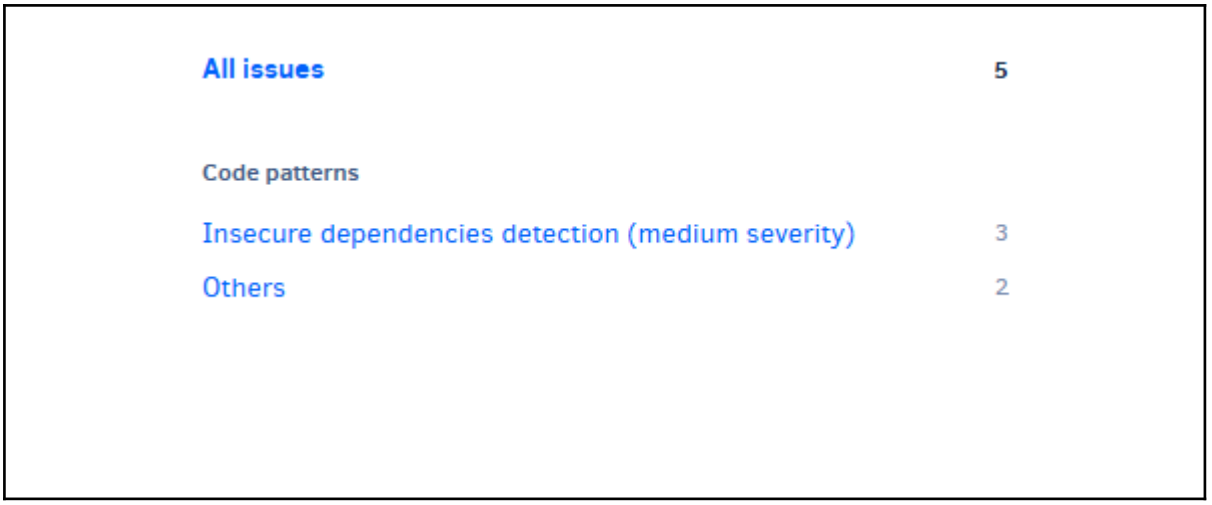
```
1  const session = await getSession();
```

Task: Security Code Analysis

Description: We use CodeScan tools, such as CodeQL or SonarQube, to perform automated security code analysis. This process helps identify potential vulnerabilities, code decay, and security issues in the codebase. By integrating these tools into the development workflow, we ensure clean code practices and maintain a secure application.

Currently, 5 security issues have been identified due to outdated dependencies that are no longer supported or maintained.

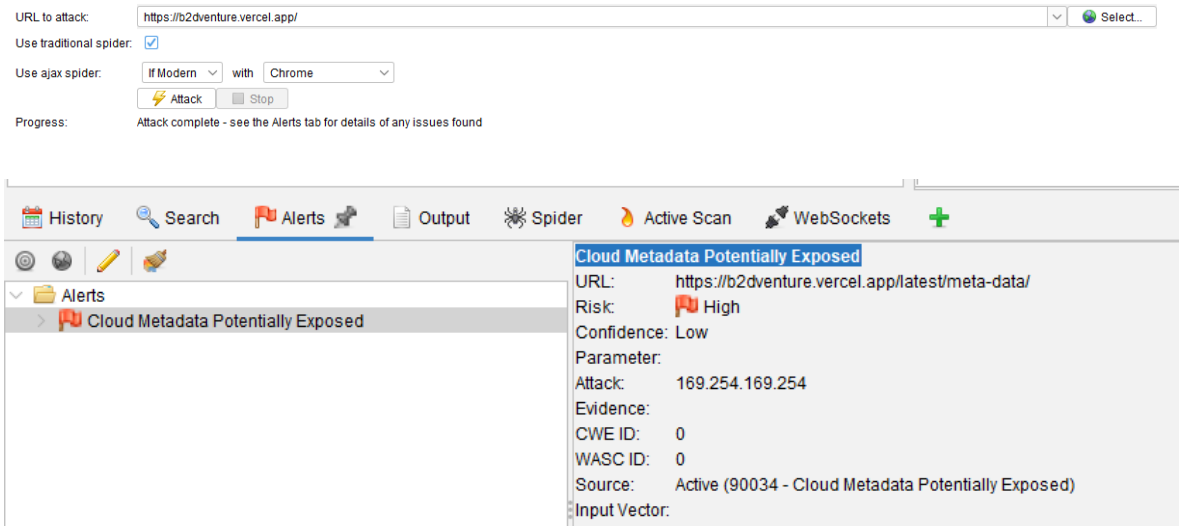



All issues
5
Code patterns
Insecure dependencies detection (medium severity)
3
Others
2

Task: Dynamic Application Security Testing

Description: Utilize DAST tools, such as OWASP ZAP, to scan for vulnerabilities in our website. During a recent scan, one alert was identified: "Cloud Metadata Potentially Exposed", which is typically caused by a misconfigured NGINX server. However, since our application is deployed using Vercel, which does not use NGINX, this issue is unlikely to affect our environment. Further verification confirms that Vercel's infrastructure provides adequate protection against such vulnerabilities.

Testing on Deploy website https://b2dventure.vercel.app/



Testing application

The **B2D Venture** project uses **Postman** as the primary tool for API testing. This involves validating security, authentication, and authorization mechanisms to ensure that all API endpoints are properly protected against unauthorized access and potential attacks. Postman helps simulate real-world scenarios and test for vulnerabilities in the API.