**על החברה**

**Valence ) ערכיות)**

valence Security

היא חברה שנוסדה בשנת 2021 וממוקמת בתל אביב. **החברה מתמחה באבטחת אפליקציות SaaS (תוכנה כשירות) על ידי ניהול סיכונים הקשורים לאינטגרציות צד-שלישי וחיבורי אפליקציות.**

הפלטפורמה של Valence מציעה פתרון מקיף לניהול מצב האבטחה של SaaS SaaS Security Posture Management (SSPM) וזיהוי ותגובה לאיומי זהות Identity Threat Detection and Response (ITDR).

**היא מאפשרת לארגונים לזהות ולתקן תצורות שגויות, לאכוף מדיניות אבטחה, ולעמוד בתקני תעשייה באפליקציות SaaS** קריטיות כמו Microsoft 365, Google Workspace, Salesforce, GitHub ו-Slack.

גישה ייחודית של החברה היא עבודה בשיתוף פעולה עם משתמשים עסקיים, על מנת להעניק הקשר לסיכונים ולסייע בצמצומם בתחומים כמו שיתוף נתונים, שרשרת אספקה, זהויות ותצורות שגויות. כך ניתן להבטיח אבטחה מבלי לפגוע בפרודוקטיביות העסקית.

באוקטובר 2022, Valence גייסה 25 מיליון דולר בסבב גיוס סדרה A בהובלת קרן ההשקעות של Microsoft, M12, מה שהעלה את סך הגיוסים שלה ל-32 מיליון דולר.

החברה זכתה להכרה כמובילה בתחום ה-SSPM ונבחרה כ-"Leader" בדו"ח.

**המשימה של Valence היא לאפשר לארגונים לאמץ ולנהל אפליקציות SaaS בצורה מאובטחת, על ידי מתן נראות, ניטור מתמשך, ואכיפת מדיניות אוטומטית המגנה על הרשת המקושרת של אפליקציות עסקיות.**

**שאלות?**

1.האתגרים שאתם עומדים בפניהם?

2.האם עובדים רק בארהב?

3.מבחינת עתיד בחברה אני רוצה להתפתח אם יש פה אפיק לגדול בו

4. כיצד נראה מבנה הצוות, ואילו שפות תכנות נמצאות בשימוש

5. כיצד אתם מגדירים את המועמד האידיאלי לתפקיד זה?

6.. כמה שלבי גיוס קיימים ומה כוללים?

אני סטודנט בשנה האחרונה לתואר באנגלית באוניברסיטת רייכמן, עם רקע וניסיון עשיר במגוון תחומים טכנולוגיים. במהלך שירותי הצבאי ביחידה טכנולוגית מובחרת, רכשתי ניסיון בעבודה בצוותים בין-תחומיים, פיתוח מוצרים מתקדמים, וחיפוש פתרונות טכנולוגיים חדשניים. עבדתי בעיקר על פיתוח צד-לקוח (Front-end) ומוצרים שזכו לשימוש נרחב.

לאחר תחילת הלימודים, התנסיתי בתפקידי מתכנת פולסטאק בסטארטאפ **Aware**, שם התמקדתי באינטגרציות עם RESTful APIs ואוטומציה של מידע, עם דגש על פיתוח צד-שרת (Back-end). בנוסף, הייתי חוקר במעבדה לחקר אינטראקציה אדם-מחשב אינטגרציות עם API C++, שם פיתחתי מוצר משותף שהוצג בכנס אקדמי בינלאומי בהוואי ובישראל. גם באוניברסיטת קיימברידג' עסקתי במחקר טכנולוגי בתחום הבלוקצ'יין.

מעבר לזה, הובלתי פרויקטים כמו פיתוח אפליקציית ניווט בקמפוס בשיתוף עם הדיקן, ופיתוח כלים לאיסוף נתונים מאמזון באמצעות **Beautiful Soup** ו-**Selenium**. החוויות האלו נתנו לי הבנה מעמיקה בפיתוח פולסטאק, מחקר אינטראקציות אדם-מחשב וניהול פרויקטים.

אני מאמין שהפתרון של **Valence** נותן מענה אמיתי לאתגרי האבטחה של העולם המחובר שבו אנחנו חיים היום. היכולת שלי לשלב חשיבה טכנולוגית עם עבודה בצוותים מגוונים, לצד הרקע שלי באינטגרציות API ואבטחת נתונים, הופכת אותי למתאים לתפקיד. אני רואה בתכנות כלי לשיפור העולם בתחומים כמו טכנולוגיות חדשניות ואבטחת מידע, ואני מאמין שהחזון שלכם ב-Valence תואם את הגישה שלי.

סטנדרט למתן הרשאות לאפליקציה אחת לגשת למידע של משתמש באפליקציה שנייה**, והוא עומד במרכז הבעיה ש-Valence Security ממוקדת בה**. **זהו מנגנון נפוץ לחיבור אפליקציות ויצירת ה- Business Application Mesh, בדרך כלל בלי נראות או פיקוח מצד צוותי אבטחה בארגון**, **וכך נוצר משטח תקיפה חדש ולא ממופה**. השאלה פותחת בראיון בעצם מיני-דיון שבו החברה מציגה את הבעיה שהיא באה לפתור, ומאפשרת למראיינים לבחון מה המועמדים חושבים על סוגיות שנתקלו בהן ואיזה תהליך מחשבתי הם עוברים.

**1. What Are RESTful APIs?**

**REST** (Representational State Transfer) is an architectural style for designing networked applications. RESTful APIs use HTTP methods **to access and manipulate resources**.

**Key Features of RESTful APIs:**

1. **Stateless**: Each request contains all the information the server needs to fulfill the request. The server doesn't store client session data.
2. **Resource-Based**: Resources (like users, products) are represented with URLs.
3. **Standard HTTP Methods**:
   * GET: Retrieve data.
   * POST: Create data.
   * PUT: Update/replace data.
   * PATCH: Partially update data.
   * DELETE: Remove data.

**JSON as a Standard Format**: Most APIs use JSON for request and response bodies.

**2. How RESTful APIs Work**

A typical workflow involves:

1. **Endpoint**: **The URL** for accessing the resource, e.g., https://api.example.com/users.
2. **Headers**: Metadata for the request, e.g., **Authorization**: Bearer token.
3. **Request Body**: (For POST/PUT/PATCH) Data sent to the server.
4. **Response**: **Server sends back the result in JSON** (or another format).

**3. HTTP Status Codes**

Memorize common status codes:

* **2xx (Success)**:
  + 200 OK: Request successful.
  + 201 Created: Resource successfully created.
* **4xx (Client Errors)**:
  + 400 Bad Request: Invalid input.
  + 401 Unauthorized: Authentication required.
  + 403 Forbidden: Access denied.
  + 404 Not Found: Resource doesn’t exist.
* **5xx (Server Errors)**:
  + 500 Internal Server Error: Something went wrong on the server.
* **response.raise\_for\_status()** is a method provided by the requests library in Python. It is used to check if an HTTP request resulted in a successful response (status codes in the 200 range). If the response has an error status code (e.g., 4xx or 5xx), it raises an HTTPError exception.

**4. Authentication Mechanisms**

**API Keys:**

* Simple, included in headers or query parameters.

**2. OAuth2:**

OAuth2 (Open Authorization 2.0) is a widely-used **protocol for granting access to resources on behalf of a user without sharing their credentials**. It's commonly used in APIs to secure data access, allowing third-party applications to act on behalf of How Does OAuth2 Work?

1. **Redirect User:** **The app sends the user to a login page** (e.g., GitHub's login).
2. **User Grants Permission**: The user logs in and says, "Yes, I allow this app to access my data."
3. **Authorization Code:** **The app gets a code from the platform after the user grants permission.**
4. **Exchange Code for Token**: **The app sends the code to GitHub to get an access token.**
5. **Use Token:** **The app uses the token to access data (e.g., user info, repos) without asking for the user's password**.
6. **Token Expiry:** **If the token expires, the app can ask for a new one using a refresh token** (if available).

Why is OAuth2 Useful?

* **Secure:** Users don’t share their passwords.
* **Limited Access**: The app only gets the permissions (scopes) the user allows.
* **Revocable**: Users can revoke access at any time.

סטנדרט למתן הרשאות לאפליקציה אחת לגשת למידע של משתמש באפליקציה שנייה**, והוא עומד במרכז הבעיה ש-Valence Security ממוקדת בה**. **זהו מנגנון נפוץ לחיבור אפליקציות ויצירת ה- Business Application Mesh, בדרך כלל בלי נראות או פיקוח מצד צוותי אבטחה בארגון**, **וכך נוצר משטח תקיפה חדש ולא ממופה**. השאלה פותחת בראיון בעצם מיני-דיון שבו החברה מציגה את הבעיה שהיא באה לפתור, ומאפשרת למראיינים לבחון מה המועמדים חושבים על סוגיות שנתקלו בהן ואיזה תהליך מחשבתי הם עוברים.

* **איך זה עובד**: משתמש מקבל **Access Token** דרך שרת **Authorization**. טוקן זה משמש לאימות בקשות.
* **יתרון**: מאפשר הרשאות מבוקרות (Scopes), מתאים לגישה בין אפליקציות.
* **חסרון**: מסובך ליישום.
* **חולשה**: ניתן לנצל את התהליך ל**Token Hijacking** אם האפליקציה לא מאובטחת.

**דרכים נפוצות לחטיפת טוקן:**

1. **Man-in-the-Middle (MITM):**  
   **תוקף מאזין לתעבורה** בין הלקוח לשרת ומיירט את ה-Token.  
   **פתרון**: שימוש קבוע ב-**HTTPS** להצפנת התקשורת.
2. **Cross-Site Scripting (XSS):**  
   **אם האפליקציה מאפשרת הזרקת קוד זדוני, תוקף יכול לגשת ל-Token** המאוחסן ב-Local Storage או ב-Cookies.  
   **פתרון**: מניעת XSS על-ידי הסרת קלט מסוכן והגדרת כותרות אבטחה כמו Content-Security-Policy.
3. **Redirect URI לא מאובטח:**  
   אם ה-Redirect URI פגיע או לא מאומת, **תוקף יכול להסיט את ה-Token לדומיין זדוני.  
   פתרון**: אימות מדויק של **Redirect URIs** ואישורם מראש.
4. **דליפת טוקן דרך הדפדפן:**  
   שימוש לא נכון ב-CORS או שמירת ה-Token ב-Local Storage חושף אותו לתקיפות **XSS**.  
   **פתרון**: שמירת טוקנים ב-**HTTP-Only Cookies** למניעת גישה לסקריפטים זדוניים.
5. **Replay Attack:**  
   תוקף משתמש שוב **ב-Token שנגנב** כדי לגשת למשאבים לפני שפג תוקפו.  
   **פתרון**: הוספת חתימה ו-**Timestamp** לטוקן ושימוש ב-**Refresh Token** עם זמן חיים קצר.

**סיכום הפתרונות לאבטחת OAuth2:**

1. שימוש ב-**HTTPS** בכל התעבורה.
2. מניעת **XSS** ו-CSRF על-ידי כותרות אבטחה והגנה על קלטים.
3. הגדרת **Redirect URIs** מאובטחים ומדויקים.
4. שמירת הטוקנים ב-**HTTP-Only Cookies** ולא ב-Local Storage.
5. שימוש ב-**Scopes** מוגדרים מראש כדי למנוע הרשאות יתר.

**3. GDPR Principles**

מגן על מידע אישי ופרטיות. הוא חל על כל חברה המטפלת בנתונים של אזרחי האיחוד האירופי.

השימוש בנתונים צריך להיות חוקי, הוגן ושקוף, מוגבל למטרות ספציפיות, מצומצם לנדרש בלבד, מדויק ומעודכן, מאובטח מפני פריצות וגישה לא מורשית, ותוך שמירה על אחריות ועמידה בכללי ה **GDPR** -. אי עמידה בתאים האלה יכול להוביל לקנסות אדירים. GDPR.

**The key principles are:**

1. **Lawfulness, Fairness, Transparency**: להשתמש בנתונים באופן חוקי, הוגן ולספק שקיפות לגבי אופן השימוש בהם.
2. **Purpose Limitation**: לאסוף נתונים רק למטרות ספציפיות ולהשתמש בהם אך ורק למטרות אלו
3. **Data Minimization**: לאסוף רק את המידע ההכרחי
4. **Accuracy**: לשמור על המידע מדויק ועדכני.
5. **Storage Limitation**: לא לשמור מידע יותר מהנדרש
6. **Security (Integrity & Confidentiality)**: להגן על המידע מפני פריצות וגישה לא מורשית
7. **Accountability**: Show that you follow GDPR rules (e.g., maintain records, train staff).

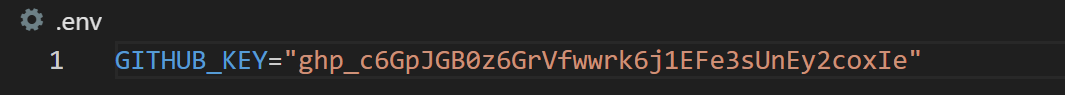
**User Rights**

* **Access**: See their data.
* **Rectification**: Fix errors.
* **Erasure**: Request deletion.
* **Portability**: Transfer data elsewhere.
* **Object**: Stop certain uses (e.g., marketing).

**Non-compliance can lead to huge fines.**

**4. dotnev:**

create .env file



Get the key

A screen shot of a computer program

Description automatically generated

The **dotenv library** makes it easier to load environment variables from a .env file into your application. Specifically, it:

1. Reads the key-value pairs from a .env file.
2. Loads them into your **environment variables** at runtime.
3. Allows you to use os.getenv() to access these variables.

**Why dotnev?**

**Using only os.getenv() is fine if the environment variables are already set at the system level** (e.g., via terminal export commands, server configurations, or CI/CD pipelines). However, during **local development** or **testing**, you might not want to set environment variables globally for your system. **This is where dotenv comes in.**

**http**

**Web Development and SaaS Fundamentals:**

**1. General Web Knowledge:**

**HTTP/HTTPS and REST APIs:**

* **HTTP/HTTPS:** HTTP (Hypertext Transfer Protocol) is the foundational **protocol for** **data communication on the web**. **HTTPS** (HTTP Secure) **adds a layer of security using** **TLS/SSL to encrypt data**, ensuring secure communication between the client and server..

**Client-Server Model**: The client (e.g., a browser or API consumer) sends a **request** to a server. The server processes the request and sends back a **response**.

* **Q: What are the advantages of using HTTPS over HTTP?**
* HTTPS encrypts data using TLS/SSL, which protects user information from being intercepted during transmission. It also provides data integrity and authentication, ensuring users are communicating with the intended **server.**

**DOM Manipulation and JavaScript:**

* DOM (Document Object Model) is a **programming interface** for HTML and XML documents. JavaScript can access and manipulate the DOM to change elements, styles, and content dynamically.
* Familiarize yourself with common DOM methods like document.querySelector(), element.addEventListener(), and document.createElement().

**Key Question & Answer:**

* Q: How does JavaScript interact with the DOM?
* A: JavaScript can manipulate the DOM using various methods and properties to select elements, modify their attributes, change their content, and listen for user interactions.

**2. Web Security Concepts:**

CSRF (Cross-Site Request Forgery):

* **CSRF is an attack where an attacker tricks a user into performing an action they didn’t intend to**, such as transferring funds or changing their profile information.
* Prevention: Use CSRF tokens to verify that the request originated from an authenticated user’s session.

**7. Interview Questions to Practice**

**Technical:**

* Write a Python function to authenticate with an API using OAuth2 and fetch data.
* Merge and filter data from two APIs into a single JSON output.
* Write code to retry an API call after failure (with exponential backoff).

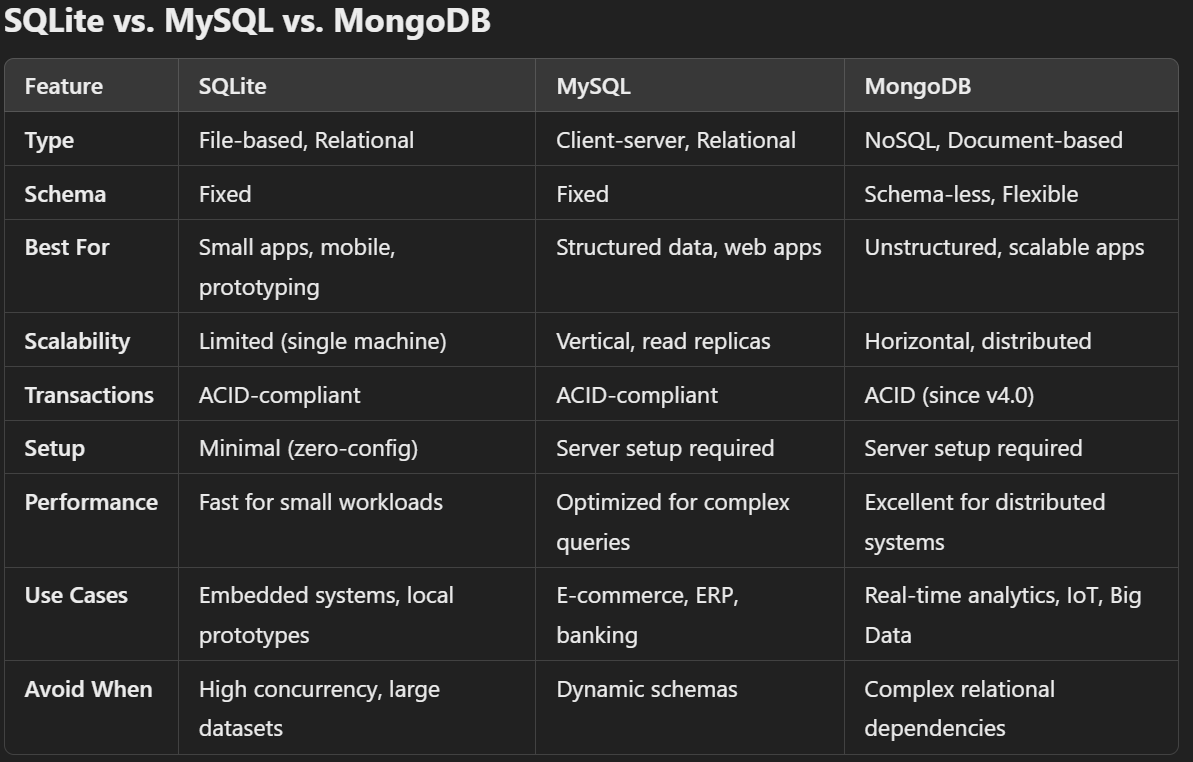
Store data: sql vs mysql

**8. Interview Questions to Practice**

**SQLite vs. MySQL vs. MongoDB**

Key Takeaways

* SQLite: Lightweight and great for small-scale apps.
* MySQL: Ideal for structured, relational data with strong schemas.
* MongoDB: Best for unstructured, dynamic data requiring high scalability.

****

s

**9.What is FastAPI?**

**FastAPI** הוא פריימוורק מודרני ומהיר לפיתוח APIs ב-Python. הוא פשוט לשימוש, מותאם לפרודקשן, ומתאים במיוחד ליצירת **RESTful APIs** ומיקרו-שירותים.

1. **ביצועים גבוהים**:
   * מבוסס על **Starlette** ו-**Pydantic** לאופטימיזציה ומהירות.
   * תומך בתכנות אסינכרוני (async/await) לניהול יעיל של בקשות.
2. **תיעוד אוטומטי**:
   * יוצר דוקומנטציה אינטראקטיבית באמצעות **Swagger UI** ו-**ReDoc**.
   * לדוגמה: גישה ל-/docs או /redoc.
3. **ולידציה של נתונים**:
   * משתמש ב-**Pydantic** לוולידציה ואוטומציה של קלט ופלט.
   * מבטיח שה-API מקבל רק נתונים בפורמט הנכון.
4. **תמיכה באסינכרוניות**:
   * מאפשר ניהול בקשות בו-זמנית בצורה לא חוסמת (non-blocking), אידיאלי לפעולות כמו שאילתות מסד נתונים.

A computer screen with text and images

Description automatically generated

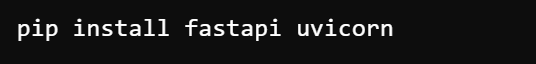
* **Endpoints**:
  + GET / → Returns a simple welcome message.
  + GET /items/{item\_id} → Returns an item based on item\_id and an optional query parameter.

A computer screen shot of a black screen

Description automatically generated

**Why Use FastAPI?**

* **Fast Development**: Minimal setup and automatic documentation save time.
* **Asynchronous Support**: Efficient handling of multiple requests.
* **Strong Typing**: Ensures type safety and reduces bugs.
* **Production Ready**: Ideal for deploying high-performance APIs



A black background with white text

Description automatically generated

**1. אילו שיטות אותנטיקציה קיימות על גבי HTTP?**

**שיטות נפוצות:**

**Basic Authentication1.**

* **איך זה עובד**: **שם משתמש וסיסמה נשלחים מקודדים ב-Base64** בכותרת (Authorization: Basic).
* **יתרון**: פשוט מאוד ליישום.
* **חסרון**: לא מאובטח כי הנתונים עוברים בצורה מקודדת אך לא מוצפנת.
* **חולשה**: רגיש ל-**Man-in-the-Middle** אם לא משתמשים ב-HTTPS.

**:Bearer Token (Token-Based Authentication)2.**

* **איך זה עובד**: הלקוח שולח **Token** בכותרת Authorization כמו ב(JWT)
* **יתרון**: אין צורך לשלוח שם משתמש וסיסמה בכל בקשה.
* **חסרון**: אם הטוקן נגנב, ניתן להשתמש בו עד פקיעתו.
* **חולשה**: רגיש ל-**Token Leakage** או שימוש לא נכון ב-CORS.

**:Mutual TLS Authentication3.**

* **איך זה עובד**: אימות מבוסס תעודות (Client Certificate + Server Certificate).
* **יתרון**: רמת אבטחה גבוהה.
* **חסרון**: מורכב לניהול והפצה של תעודות.

**2.אפליקציה ב-Cloud הפכה ליעד תקיפה, איך הייתם ניגשים?**

1. **שלב איסוף מידע (Reconnaissance)**:

* סריקות DNS לגילוי סאב-דומיינים.
* בדיקת תצורות Cloud פומביות (S3 Buckets פתוחים או Azure Blob Storage).

1. **בדיקות אבטחה נפוצות**:

* בדיקות **Misconfigurations** כמו הרשאות מוגזמות או תצורות שגויות.
* ניסיונות **Injection Attacks** ב-API.

1. **זיהוי נקודות גישה**:

* בדיקות ל-Credentials Hardcoding או שימוש בקובצי .env.
* גישה לא מאובטחת באמצעות CORS שגוי.

1. **ניצול פגיעויות**:

* ניצול רכיבים חיצוניים לא מעודכנים או פגיעויות בשרת ה-API.
* Brute-Force על טוקנים (אם אין Rate Limiting).

**3. מה זה PE וכיצד ניתן להפיק מידע?**

**Portable Executable (PE)**:

* פורמט קבצי הרצה של **Windows** (כגון .exe ו-.dll).
* מכיל **Headers** ו-**Sections** עם מידע על הקובץ, קוד ריצה, וכתובות בזיכרון.

**שליפת מידע בזמן ריצה**:

1. **דיבאגרים** כמו WinDbg או OllyDbg מספקים ניתוח בזמן ריצה.
2. **Dumping** מהזיכרון: אפשר לשחזר חלקים מהקוד או נתונים שהועמסו לזיכרון.
3. **Headers**: ניתן לקרוא מידע כמו הכתובת של Entry Point, Import Table ו-Export Table.

**4. מה זה קוד פולימורפי?**

* **הגדרה**: **קוד פולימורפי משנה את החתימה שלו בכל הפעלה, כך שאינו ניתן לזיהוי ע"י תוכנות אנטי-וירוס מבוססות חתימות.**
* **שימוש**: תוקפים משתמשים בו להסוואת **וירוסים** ו-**Malware**.
* **יתרון לפושעים**: קשה יותר לזהות ולהסיר את הקוד.
* **דוגמה**: **Ransomware** או קבצים עם פקודות מוצפנות שנטענות בזמן ריצה.

**6.מה זה CORS**

* **CORS (Cross-Origin Resource Sharing)**: מנגנון **שמאפשר לדפדפן לאשר גישה למשאבים** מדומיין אחד לדומיין אחר.

**איך זה עובד?**  
השרת שולח כותרות (Headers) כמו:

* Access-Control-Allow-Origin: מציין **מי יכול לגשת למשאבים**.
* Access-Control-Allow-Methods: **מציין אילו שיטות HTTP מותרות**.

**חולשות נפוצות**:

1. תצורה לא נכונה של Access-Control-Allow-Origin מאפשרת לכל דומיין לגשת.
2. **CORS Misconfigurations** מנוצלות לגניבת מידע רגיש מהדפדפן (למשל, Cookies).

A screenshot of a computer code

Description automatically generatedA screenshot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screen shot of a computer code

Description automatically generatedA screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated