DOI: https://doi.org/10.36910/6775-2524-0560-2023-52-12
УДК 004.056.53
Поліщук Микола Миколайович, к.т.н., доцент
https://orcid.org/0000-0002-1218-5925
Семенюк Олександр Вікторович, магістр
Поліщук Лілія Олексіївна, магістр
https://orcid.org/0000-0001-9282-1639
Ломакін Микола Валентинович, магістр
Луцький національний технічний університет, м. Луцьк, Україна

МОЖЛИВОСТІ АВТОРИЗАЦІЇ ТА ЗАХИСТУ ДАНИХ КОРИСТУВАЧА ПІД ЧАС РОЗРОБКИ ХМАРНИХ ВЕБ-ДОДАТКІВ ДЛЯ ІОТ

Поліщук М.М., Семенюк О. В., Поліщук Л.О., Ломакін М.В. Можливості авторизації та захисту даних користувача під час розробки хмарних веб-додатків для IoT. У статті розглядаються проблеми безпеки та захисту інформації, які створює поширення Інтернету речей. Досліджуються різні підходи та технології для запровадження надійних та безпечних протоколів авторизації пристроїв та збереження конфіденційних даних користувачів. На основі розглянутих технологій і протоколів було спроектовано і розроблено архітектуру системи для керування розумним IoT пристроєм.
Ключові слова: авторизація, автентифікація, безпека даних, інтернет речей, мобільні додатки.

Polishchuk M.M., Semenyuk O.V., Polishchuk L.O., Lomakin M.V. Possibilities of authorization and protection of user data during the development of cloud web applications for IoT. The article examines the security and information protection problems created by the spread of the Internet of Things. Various approaches and technologies are being explored for implementing reliable and secure device authorization protocols and preserving confidential user data. Based on the considered technologies and protocols, the system architecture for controlling a smart IoT device was designed and developed.
Key words: authorization, authentication, data security, Internet of Things, mobile applications.

Вступ. Інтернет речей (IoT) – революційна технологія, що об'єднує різні пристрої і системи, від побутових до промислових, у безперебійну мережу. Ця екосистема принесла безпрецедентну зручність, ефективність та автоматизацію. Однак IoT має проблему з безпекою - потрібні надійні заходи для захисту конфіденційних даних користувачів. Зростання пристроїв IoT призвело до багатьох вразливостей, які потребують уваги, таких як несанкціонований доступ, витік даних і порушення конфіденційності. Щоб забезпечити стале зростання та успішне впровадження IoT, необхідно ефективно вирішити ці проблеми з безпекою.

Постановка наукової проблеми. Інтернет речей приніс значний прогрес, але він також представляє різні проблеми безпеки. Слабка автентифікація, ненадійне шифрування та відсутність оновлень безпеки роблять пристрої IoT сприйнятливими до несанкціонованого доступу та витоку даних. Фізичний доступ до пристроїв викликає занепокоєння щодо втручання, а атаки на відмову в обслуговуванні можуть порушити основні служби [1]. Проблеми сумісності, обізнаності користувачів, і маніпулювання даними ще більше ускладнюють проблему. Вразливість ланцюга постачання оновлень та веб-атаки додають ризиків. Вирішення цих проблем вимагає від виробників впровадження надійних заходів безпеки, від користувачів — пильності в управлінні пристроями, а також промислової співпраці щодо стандартів безпеки та найкращих практик.

Виклад основного матеріалу й обґрунтування отриманих результатів дослідження.

Автентифікація – процес розпізнавання користувачів для отримання доступу до пристроїв, систем, мереж, серверів, веб-сайтів або програм. Мета автентифікації полягає у підтвердженні ідентичності користувача. Наприклад, користувач А не може переглядати особисті дані користувача В, що підтверджується процесом автентифікації. Це підвищує безпеку, оскільки неавторизовані особи не мають доступу до конфіденційних даних. Основний метод автентифікації - комбінація імені користувача та пароля [2, 3].

Існує кілька методів, за допомогою яких ми можемо досягти надійної автентифікації для захисту зв'язку пристроїв IoT:

- Одностороння аутентифікація – одна сторона аутентифікується перед іншою, але остання не вимагає автентифікації.
- Двостороння аутентифікація – обидві сторони автентифікуються одна перед одною.
- Тристороння аутентифікація – центральний орган автентифікує обидві сторони та допомагає їм автентифікувати одна одну.

- Розподілена автентифікація – використовує метод розподіленої прямої автентифікації між сторонами зв'язку.
- Централізована автентифікація – використовує централізований сервер або довірену третю сторону для керування сертифікатами автентифікації.

Протокол X.509 - найбезпечніший метод цифрової автентифікації, який базується на моделі ланцюжка довіри сертифікатів. Використання сертифікатів X.509 полегшує масштабування виробництва та спрощує доставку обладнання. Інфраструктура відкритих ключів (PKI) містить деревоподібну структуру серверів і пристроїв, підтримуючи список довірених кореневих сертифікатів [4].

Кожен сертифікат містить відкритий ключ пристрою, підписаний закритим ключем ЦС з унікальним "відбитком" для чіткої ідентифікації за допомогою алгоритму, такого як RSA.

Авторизація — це процес надання дозволу на доступ до певного ресурсу. Дозволи в цьому контексті визначають дії, які користувач може виконувати над ресурсом [2].

У деяких випадках авторизація пов'язана з ідентифікацією, наприклад, як під час посадки в літак, пасажиру потрібні посадковий талон і паспорт для підтвердження особи та дозволу на політ.

Комп'ютерні системи використовують авторизацію так само, як і реальні приклади, описані вище. IAM (Ідентифікація та керування доступом) — це IT-дисципліна, яка включає авторизацію та автентифікацію для контролю того, хто має доступ до системних ресурсів і встановлення привілеїв клієнта [4]. Правила авторизації у комп'ютерних системах дозволяють об'єктам виконувати завдання, які не дозволені іншим об'єктам, подібно до процесу контролю доступу в реальному світі.

Існує кілька методів авторизації, які можуть бути використані в контексті Інтернету речей [4]:

- API ключі – до запиту додається ключ (токен), який перевіряється на сервері на валідність. Вразливий, якщо ключ потрапить до сторонніх осіб.
- Базова авторизація – використовує імена користувачів і паролі, як простий підхід. Дані перевіряються з авторизованим списком на сервері.
- HMAC – використовує секретний ключ для підпису запитів і захисту від несанкціонованого втручання. Краще поєднувати з HTTPS для захисту від атак типу "людина посередині".
- OAuth – складний метод авторизації, що дозволяє стороннім програмам отримувати доступ до обмежених ресурсів. Надає токени доступу стороннім програмам, які потім використовуються для авторизації запитів від імені користувача.

Для обробки великої кількості даних, створених пристроями, у сфері Інтернету речей використовуються різні бази даних. Деякі з них доволі вузького призначення, пристосовані спеціально для використання в хмарі або адаптовані для роботи з конкретними структурами даних [5].

Найпопулярнішими рішення для невеликих систем і домашніх мереж є NoSQL бази даних, оскільки зазвичай вони не вимагають детального налаштування, підтримують гнучку модель даних, яка легко адаптується до змін, та мають низьку криву навчання, що буде важливо для початківців. Особливо це стосується хмарних рішень, таких як Firestore та MongoDB Atlas, оскільки вони пропонують вже готову інфраструктуру, яка вимагає мінімальних налаштувань для початку роботи [6].

На основі проведених досліджень та аналізу їх результатів, наведених вище, було спроектовано та розроблено архітектуру проекту, який демонструє можливі шляхи реалізації авторизації та керування даними у контексті Інтернету речей.

Проект являє собою просту систему керування IoT пристроями, зокрема, розумним кондиціонером, який має доступ до локальної домашньої мережі через технологію WiFi.

Для збереження та керування даними було прийнято рішення використовувати базу даних Google Firestore для збереження метрик, а також списку доданих пристроїв.

Для реалізації реєстрації та автентифікації було обрано платформу Firebase. Вона проста у використанні та має добре задокументовне SDK. Авторизації запитів від користувачів виконується за допомогою протоколу JSON Web Token (JWT).

Проект передбачає наявність мобільного додатка, через який і буде здійснюватись взаємодія із серверною частиною та IoT пристроєм. Для розробки цього додатка було обрано фреймворк Flutter.

Для імітації роботи IoT пристрою було створено окреме програмне забезпечення – набір скриптів написаних на TypeScript, які виконують певні запити та відповідають на запити ззовні, керують внутрішнім станом тощо. Структура папок цього сервісу зображена на рисунку 1.

Папка data містить файли конфігурації та файли, які зберігають значення внутрішнього стану пристрою. Папка keys містить файли, які потрібні для авторизації запитів – публічні ключі, сертифікати, серійний номер пристрою. Виконання сервісу починається у файлі main.ts, який викликає функції із файлі у папці src, в яких і міститься основна логіка.

У файлі settings_server.ts описано логіку зміни налаштувань пристрою. Код у цьому файлі налаштовує сервер TCP у Node.js за допомогою модуля net для обробки запитів клієнтів і керування подіями, пов'язаними з пристроєм. Сервер взаємодіє з двома класами, StateManager і DataClient, відповідальними за керування станом пристрою та логуванням метрик відповідно.

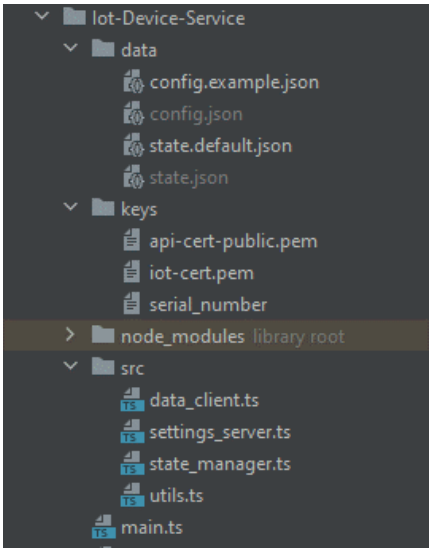

Рисунок 1 – Структура папок сервісу IoT пристрою

Список доступних подій наведено у переліку Action:

```
enum Action {
  PW_ON,
  PW_OFF,
  SELF_CLEAN_ON,
  SELF_CLEAN_OFF,
  SET_WORK_MODE,
  TURBO_ON,
  TURBO_OFF,
  WIFI_ON,
  WIFI_OFF,
  DISP_LED_ON,
  DISP_LED_OFF,
  GET_TEMP,
  GET_NOISE,
  GET_PW_CON,
  METRICS_LOGGING_ON,
  METRICS_LOGGING_OFF,
  CURRENT_SETTINGS,
}
```

Сервер прослуховує подію "connection", яка виникає, коли клієнт підключається. Після підключення сервер обробляє дві події:

– Подія "data": коли дані отримані від клієнта, сервер визначає тип запиту (requestData.type) і виконує різні дії залежно від типу сигналу. Для кожної дії сервер викликає різні методи з класів stateManager і dataClient. Після обробки запиту відповіді надсилаються назад клієнту шляхом запису даних у сокет за допомогою writeToSocket(), а потім сокет закривається за допомогою socket.end().

– Подія "close": Ця подія спрацьовує, коли з'єднання клієнта закрито. Сервер виводить у консоль повідомлення про це.

Функція writeToSocket(socket, data) використовується для запису даних у вигляді рядка JSON у вказаний сокет, обробляючи можливі помилки під час запису:

```
function writeToSocket(socket: Socket, data: any) {
  socket.write(JSON.stringify({ data }), (err) => {
    if (err) {
      console.error(
        `ERROR: Error occurred while writing to socket
${socket.remoteAddress}:${socket.remotePort}`,
      );
      console.error(err);
    }
  });
}
```

Код у файлі state_manager.ts оголошує клас під назвою StateManager, який відповідає за керування станом гіпотетичного пристрою. Стан можна зберегти у файлі JSON. Це виконано за допомогою пакету lowdb/node, який дозволяє створити невелику локальну базу даних у JSON файлі та робити різні запити до неї.

Клас містить методи для зміни налаштувань і таймер для реєстрації показників. Приклад методів зміни стану наведено нижче:

```
async wifiOn() {
  this.db.data.wifi = true;
  await this.db.write();
}

async wifiOff() {
  this.db.data.wifi = false;
  await this.db.write();
}
```

Виміри поточних метрик проводяться за допомогою генерації випадкового числа:

```
private startMetricsLogging() {
    this.metricsInterval = setInterval(async () => {
      this.db.data.temperature_c = this.measureTemperature();
      this.db.data.noise = this.measureNoiseLevel();
      this.db.data.pw_consumption = this.measurePwConsumption();

      await this.db.write();
    }, METRICS_POLLING_INTERVAL);
  }

  private measureTemperature() {
    return randomFloat(24, 27);
  }

  private measureNoiseLevel() {
    return randomFloat(50, 55);
  }

  private measurePwConsumption() {
    return randomFloat(0.81, 0.83);
  }
```

У файлі data_client.ts оголошено клас DataClient для інкапсуляції функціональності логування метрик. У класі наявні методи loggingOn() та loggingOff(), які, відповідно, вмикають та вимикають надсилання метрик до зовнішнього API.

Метод loggingOn() перевіряє наявність посилання на зовнішнє API у файлі конфігурації і виводить помилку, якщо його немає. Після цього метод завантажує вміст файла із серійним номером пристрою у приватне поле класу serialNumber. Це серійний номер потім відправляється у заголовку

X-Serial-Number і служить додатковим кроком під час авторизації пристрою. Після цього викликається метод loggingOff(), у випадку якщо логування вже було запущено раніше, і встановлюється новий інтервал для логування у поле loggingInterval. Під час логування, метод отримує потрібні метрики із екземпляру класу StateManager, робить запит для перевірки сертифікату і надсилає метрики на задану в конфігурації адресу.

Перевірка сертифікату реалізована так:

```
private async verifyCertificate() {
    if (!this.apiPublicKey) this.apiPublicKey = await getApiPublicKey();
    if (!this.iotCert) this.iotCert = await getIotCert();

    const formData = new FormData();

    formData.append(
      "public_key",
      new Blob([this.apiPublicKey]),
      API_PUBLIC_KEY_FILENAME,
    );
    formData.append("iot_cert", new Blob([this.iotCert]), IOT_CERT_FILENAME);

    await axios.post(config.endpoints.log_metrics, formData, {
      headers: {
        "X-Verify-Certs-Request": true,
      },
    });
  }
```

Запит для логування поточних метрик виглядає наступним чином:

```
const temperature_c = await this.stateManager.getTemperature();
const noise = await this.stateManager.getNoiseLevel();
const pw_consumption = await this.stateManager.getPwConsumption();

try {
  await this.verifyCertificate();
  await axios.post(
    config.endpoints.log_metrics,
    {
      temperature_c,
      noise,
      pw_consumption,
    },
    { headers: { "X-Serial-Number": this.serialNumber } },
  );
  console.log(
    `LOG: Written metrics data to database: ${JSON.stringify({
      temperature_c,
      noise,
      pw_consumption,
    })}`,
  );
} catch (e) {
  console.error(
    `ERROR: Could not log metrics: ${
      e.response?.data?.error || e.message
    }`,
  );
}
```

Сервіс API написано на мові програмування TypeScript та з використанням бібліотеки express. Структура папок зображена на рисунку 2. Основний файл, з якого починається виконання програми розташований в корені проекту і називається api.ts. В ньому створюється екземпляр самого додатка, підключаються проміжні програми (middleware), шляхи до кінцевих точок (endpoints) та запускається HTTP сервер.

У вихідному коді програми містяться також сертифікати для авторизації запитів до IoT девайса та файл конфігурації для Firebase SDK.

У папці services знаходяться класи для взаємодії із зовнішніми джерелами даних. Наприклад SettingsService – для взаємодії із сервером налаштувань IoT пристрою. DevicesService – для керування пристроями в базі даних Firestore. CertificatesService – завантаження та верифікація сертифікатів.

Окремо зупинимось на проміжних програмах для авторизації та верифікації запитів від пристрою – authMiddleware, verifyCertificateMiddleware та verifyDeviceMiddleware.

Програма authMiddleware перевіряє навність та валідність JWT токена у запитах до API від мобільного додатка. Запит проходить далі тільки якщо токен наявний і валідний.

Програма verifyCertificateMiddleware перевіряє валідність сертифікатів пристрою і API під час запиту. Файл публічного ключа API та сертифікат пристрою приходить і тілі запиту. Файл сертифікату API та публічного ключа пристрою міститься в папці certs в корені API сервісу.

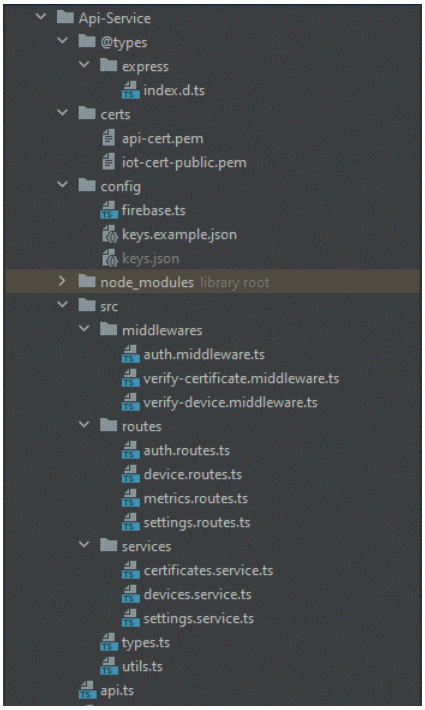

Рисунок 2 – Структура папок API сервісу

Програма authMiddleware:

```
export async function authMiddleware(
  req: Request,
  res: Response,
  next: NextFunction,
) {
  const accessToken = (req.headers.authorization as string)?.split(" ")?.[1];
  req.user = null;

  if (!accessToken)
    return res
      .status(401)
      .json({ error: "Access token is invalid or expired" });

  try {
    req.user = await firebaseAdmin.auth().verifyIdToken(accessToken);
    next();
  } catch (e) {
    console.log(e);
    return res
      .status(401)
      .json({ error: "Access token is invalid or expired" });
  }
}
```

Програма verifyCertificateMiddleware:

```
export async function verifyCertificateMiddleware(
  req: Request,
  res: Response,
  next: NextFunction,
) {
  const files = (
    req.headers["x-verify-certs-request"] === "true" ? req.files || null : null
  ) as FilesObject | null;
  if (!files) return next();

  const pk_api = files.public_key[0];
  const iotCert = files.iot_cert[0];
  if (!pk_api)
    return res.status(401).json({ error: "API: Public key not found" });

  if (!iotCert)
    return res.status(401).json({ error: "IoT: Certificate not found" });

  const pk_iot = await certService.getIotPublicKey();

  if (!pk_iot)
    return res.status(401).json({ error: "IoT: Public key not found" });

  const apiCert = await certService.getApiCertificate();
  if (!apiCert)
    return res.status(401).json({ error: "API: Certificate not found" });

  try {
    const x509_api = new X509Certificate(apiCert);
    const pkObj_api = createPublicKey(pk_api.buffer);

    const x509_iot = new X509Certificate(iotCert.buffer);
    const pkObj_iot = createPublicKey(pk_iot);

    const valid = x509_api.verify(pkObj_api) && x509_iot.verify(pkObj_iot);

    if (!valid)
      return res
        .status(401)
        .json({ error: "Certificate or public key is not valid" });

    return res.status(200).end();
  } catch (e) {
    return res
      .status(401)
      .json({ error: "Certificate or public key is not valid" });
  }
}
```

Програма verifyDeviceMiddleware авторизує запит від IoT пристрою до API. Вона перевіряє навяність заголовка X-Serial-Number, який містить серійний номер пристрою, у запиті, а також наявність цього серійного номера в базі даних:

```
export async function verifyDeviceMiddleware(
  req: Request,
  res: Response,
  next: NextFunction,
) {
  const serialNumber = req.headers["x-serial-number"] as string;

  if (!serialNumber) {
    return res.status(403).json({ error: "No serial number provided" });
  }

  try {
    const devices = await devicesService.getBySerialNumber(serialNumber);
```

```
    const exists = !!devices.length;

    if (!exists) {
      return res.status(403).json({ error: "Unknown device" });
    }

    next();
  } catch (e) {
    console.log(e);
    return res
      .status(500)
      .json({ error: "Error while processing the request" });
  }
}
```

Додаток для керування налаштуванням IoT пристрою розроблено на фреймворку Flutter. Основна одиниця для побудови інтерфейсів у цьому фреймворку називається «віджет» (Widget). Виконання починається із файла main.dart, розташованого в корені проекту.

Після запуску додатка відкривається головний екран (рис. 3), на якому міститься нижня панель навігації для переходу на інші віджети. Панель навігації містить посилання такі віджети: «Метрики», «Додати пристрій», «Налаштування». Крім цього, при запуску додаток перевіряє наявність збережного в локальному сховищі JWT токена. Якщо його немає, користувача автоматично переадресовує на екран автентифікації в додаток (рис. 4). З цього екрану можна також перейти на екран реєстрації користувача (рис. 5). Така ж переадресація відбудеться, якщо токен наявний, але сервер не зміг підтвердити його валідність і надіслав відповідну помилку.

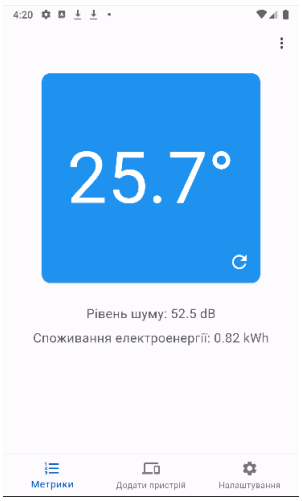

Рисунок 3 – Головний екран додатка

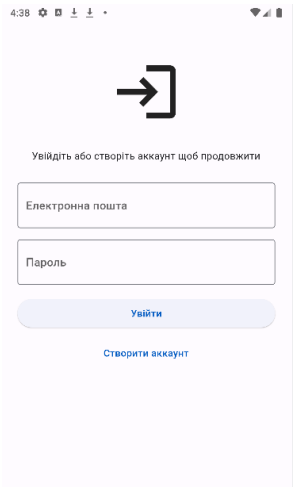

Рисунок 4 – Екран автентифікації

Віджет «Метрики» містить у собі картку з інформацією про температуру та інформацію про рівень шуму і споживання електроенергії. Щоб отримати актуальну інформацію можна натиснути на відповідну кнопку на картці. Крім цього, інформація оновлюється автоматично кожну хвилину.

Екран «Додати пристрій» (рис. 6) містить форму, через яку користувач може додати новий пристрій у свій аккаунт, щоб потім періодично отримувати з нього метрики і зберігати їх у базу даних.

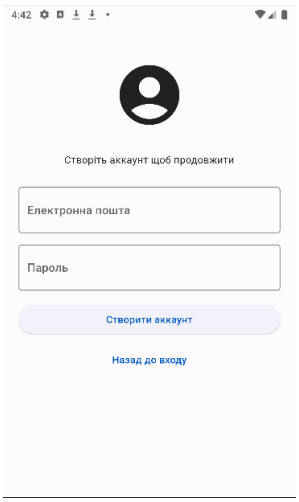

Рисунок 5 – Екран реєстрації

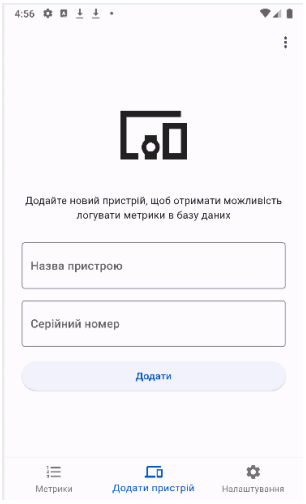

Рисунок 6 – Екран додавання пристрою

Екран «Налаштування» містить список налаштувань IoT пристрою, які можна змінити (рис. 7). Після переходу на цей екран, додаток робить запит до API, щоб отримати список налаштувань і їх поточні значення. При зміні налаштувань через інтерфейс додаток відсилає запит до API з відповідним сигналом. На цьому ж екрані користувач може вимкнути живлення пристрою та керувати станом логування метрик в базу даних.

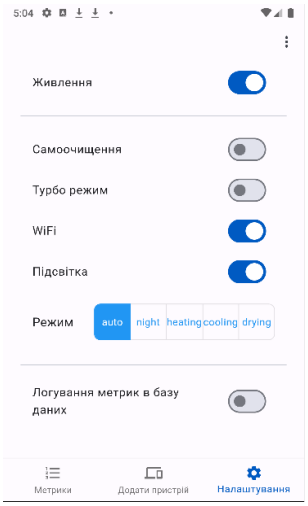

Рисунок 7 – Екран налаштувань

Висновок. Дослідження важливих аспектів безпеки Інтернету речей показало, що швидке розширення цієї технології має численні переваги, але також породжує серйозні проблеми безпеки. Було розглянуто різні методи автентифікації та авторизації, а також вивчили проблеми, пов'язані зі значним поширенням IoT пристроїв.

В роботі було розроблено програмну архітектуру, яка використовує досліджені технології для авторизації та автентифікації IoT пристроїв та користувачів, а також для безпечного зберігання даних і конфіденційності. Архітектура включає сервіс, що імітує поведінку IoT пристрою, API-сервіс в якості брокера повідомлень між клієнтом та пристроєм, і мобільний додаток для управління пристроєм.

Для авторизації API та пристрою використали технологію X.509 сертифікатів, а для авторизації користувача - протокол JSON Web Token (JWT) і стратегію входу за допомогою електронної пошти та пароля. Механізми автентифікації та реєстрації були реалізовані на платформі Firebase, а дані користувачів зберігались безпечно у хмарній базі даних Firestore від Google.

Список бібліографічного опису

1. Rose K. The internet of things: An overview / K. Rose, S. Eldridge, L. Chapin., 2015. – 53 c.
2. Authorization schemes for internet of things: requirements, weaknesses, future challenges and trends / K.Abid, A. Awais, A. Mansoor, S. Jadran., 2022. – (8).

3. Systematic Review of Authentication and Authorization Advancements for the Internet of Things / M.Trnka, A. S. Abdelfattah, A. Shrestha, M. Coffey., 2022.
4. Mohammad A. User Authentication and Authorization Framework in IoT Protocols / A. Mohammad, H. Al-Refai, A. A. Alawneh., 2022. – 147 с.
5. James A. Challenges for Database Management in the Internet of Things / A. James, C. Joshua., 2014. – 320 с. – (5).
6. Internet of Things Data Storage Infrastructure in the Cloud Using NoSQL Databases / [V. Brunno, M. Guilherme, F. Mauri та ін.]., 2017. – 743 с. – (4).
7. Поліщук М.М., Костючко С.М., Христинець М.О. Порівняння методів оптимізації нейронних мереж на прикладі задачі класифікації зображень // Науковий журнал "Комп'ютерно-інтегровані технології: освіта, наука, виробництво" – Луцьк: Видавництво ЛНТУ. – Вип. 37. – 2019. – С. 43-52.
8. Поліщук, М., Повстяна, Ю., Ящук А., Ліщина, Н., Потейчук, М. Система радіоелектронної боротьби на базі Arduino UNO R3. Комп'ютерно-інтегровані технології: освіта, наука, виробництво, (38), С. 10-16.
9. Поліщук, М., Гринюк, С. (2020). Використання технології шифрування інформації для безпечної передачі в мережі. Комп'ютерно-інтегровані технології: освіта, наука, виробництво, (39), 122-126.
10. O. Maksymovych, T.Solyar, A.Sudakov, I.Nazar, M.Polishchuk. 2021. Determination of stress concentration near the holes under dynamic loadings, Naukovyi Visnyk Natsionalnoho Hirnychoho Universytetu, 3, pp. 19-25.
11. Костючко, С., Багнюк, Н., КузьмичО., Поліщук, М., & Кирилюк, Л. (2021). Біометрична ідентифікація засобами Python та Raspberry Pi. Комп'ютерно-інтегровані технології: освіта, наука, виробництво, (42), С. 142-146.
12. Kostiuchko, S., Polishchuk, M., Zabolotnyi, O., Tkachuk, A., Twarog, B. The Auxiliary Parametric Sensitivity Method as a Means of Improving Project Management Analysis and Synthesis of Executive Elements / Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST this link is disabled, 2021, 395 LNICST, pp. 174–184.
13. Savaryn, P., Strekha, V., Brych, M., Kabak, V., Polishchuk, M. The Original Method of Controlling a Computer Using Distance Sensors. Proceedings - 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering, TCSET 2022, 2022, pp. 683–688.
14. S. Kostiuchko, O. Kuzmych, A. Aitouche, S. Grinyuk and O. Mekush, "Application of Parametric Sensitivity Method to Analysis of Automatic Mooring Winch with Electric Drive System," 2019 4th Conference on Control and Fault Tolerant Systems (SysTol), Casablanca, Morocco, 2019, pp. 294-299

References

1. Rose K. The internet of things: An overview / K. Rose, S. Eldridge, L. Chapin., 2015. – 53 с.
2. Authorization schemes for internet of things: requirements, weaknesses, future challenges and trends / K.Abid, A. Awais, A. Mansoor, S. Jadran., 2022. – (8).
3. Systematic Review of Authentication and Authorization Advancements for the Internet of Things / M.Trnka, A. S. Abdelfattah, A. Shrestha, M. Coffey., 2022.
4. Mohammad A. User Authentication and Authorization Framework in IoT Protocols / A. Mohammad, H. Al-Refai, A. A. Alawneh., 2022. – 147 с.
5. James A. Challenges for Database Management in the Internet of Things / A. James, C. Joshua., 2014. – 320 с. – (5).
6. Internet of Things Data Storage Infrastructure in the Cloud Using NoSQL Databases / [V. Brunno, M. Guilherme, F. Mauri та ін.]., 2017. – 743 с. – (4).
7. Поліщук М.М., Костючко С.М., Христинець М.О. Порівняння методів оптимізації нейронних мереж на прикладі задачі класифікації зображень // Науковий журнал "Комп'ютерно-інтегровані технології: освіта, наука, виробництво" – Луцьк: Видавництво ЛНТУ. – Вип. 37. – 2019. – С. 43-52.
8. Polishchuk, M., Povstyana, Yu., Yashchuk A., Lyshchyna, N., Poteychuk, M. Electronic warfare system based on Arduino UNO R3. Computer-integrated technologies: education, science, production, (38), pp. 10-16.
9. Polishchuk, M., Hryniuk, S. (2020). Use of information encryption technology for secure transmission on the network. Computer-integrated technologies: education, science, production, (39), 122-126.
10. O. Maksymovych, T. Solyar, A. Sudakov, I. Nazar, M. Polishchuk. 2021. Determination of stress concentration near the holes under dynamic loadings, Naukovyi Visnyk Natsionalnoho Hirnychoho Universytetu, 3, pp. 19-25.
11. Kostyuchko, S., Bagniuk, N., Kuzmych O., Polishchuk, M., & Kirilyuk, L. (2021). Biometric identification using Python and Raspberry Pi. Computer-integrated technologies: education, science, production, (42), pp. 142-146.Kostiuchko, S.,
12. Polishchuk, M., Zabolotnyi, O., Tkachuk, A., Twarog, B. The Auxiliary Parametric Sensitivity Method as a Means of Improving Project Management Analysis and Synthesis of Executive Elements / Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST this link is disabled, 2021, 395 LNICST, pp. 174–184.
13. Savaryn, P., Strekha, V., Brych, M., Kabak, V., Polishchuk, M. The Original Method of Controlling a Computer Using Distance Sensors. Proceedings - 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering, TCSET 2022, 2022, pp. 683–688.
14. S. Kostiuchko, O. Kuzmych, A. Aitouche, S. Grinyuk and O. Mekush, "Application of Parametric Sensitivity Method to Analysis of Automatic Mooring Winch with Electric Drive System," 2019 4th Conference on Control and Fault Tolerant Systems (SysTol), Casablanca, Morocco, 2019, pp. 294-299