PROYECTOS DE INGENIERÍA Y GESTIÓN DEL SOFTWARE

REPORT NUMBER	**8**
TEAM NUMBER	**24**
TITLE	**Functional / Static / Accessibility analysis**
DATE	**04/05/25**

Option 2: Static code analysis

In case you have used an automatic tool for static analysis, indicate it:

SonarQube

Member	Issue or group of issues	Solution
Francisco	**Hard-coded Password** The analysis detected hard-coded credentials within the **`authController.ts`** file. Storing sensitive information like passwords directly in the source code poses a significant security risk. If the source code were compromised, these credentials would be exposed, potentially leading to unauthorized access to user accounts and system resources.	The hard-coded password literals were removed from **`authController.ts`.** Secrets management is now handled through environment variables (**`process.env`**), ensuring sensitive data is not embedded in the codebase. Furthermore, user passwords are no longer stored in plaintext. Instead, they are securely hashed using **`bcrypt`** with appropriate salting and peppering techniques before being persisted in the database. The authentication flow (registration and login) has been updated accordingly to handle hashed passwords.

Instructions:

Each team must select only one of the 3 options and delete the tables of the options which have not been selected.

If option 1 is selected: Each member of the team must select a user story and test the acceptance tests identified for that story, indicating which ones have been passed and making the necessary changes for those which have not been passed.

If option 2 is selected: A static code analysis must be performed. Automatic tools can be used (e.g., SonarLint, SonarQube...). Each member of the team must address one issue or group of related issues and solve the problems associated with it.

If option 3 is selected: An accessibility analysis must be performed. Automatic tools such as Accessibility Scanner, or features such as TalkBack can be used to identify issues. Each member of the team must address one issue or group of related issues and solve the problems associated with it.

Antonio	**Denial of Service (DoS) via Catastrophic Backtracking Regex** A regular expression used for email validation in the **`Login.tsx`** component was identified as susceptible to catastrophic backtracking. Maliciously crafted email inputs could cause this regex to consume excessive CPU resources, potentially leading to a **Denial of Service (DoS)** condition where the application becomes unresponsive.	The vulnerable regular expression in **`Login.tsx`** has been replaced. We opted to delegate email validation to the battle-tested **`validator.js`** library. This library provides efficient and secure validation functions, mitigating the risk of DoS attacks related to regex processing and ensuring robust email format checking.
Antonio	**Use of Insecure Pseudorandom Number Generator** The application utilized **`Math.random()`** for generating random numbers in components like **`VehicleMap.tsx`** and associated utility functions. **`Math.random()`** produces predictable sequences, making it unsuitable for security-sensitive contexts where unpredictability is crucial (e.g., generating session tokens, unique IDs, or cryptographic keys). Using it could lead to guessable values and potential security breaches.	All instances of **`Math.random()`** used in security-sensitive contexts have been replaced with cryptographically secure pseudorandom number generators. We now use **`crypto.getRandomValues()`** for web environments and **`crypto.randomInt()`** / **`crypto.randomBytes()`** for Node.js environments. To ensure consistency and maintainability, this logic has been centralized into a dedicated helper utility, addressing all identified hotspots (approximately 40 instances) across the codebase.

Instructions:

Each team must select only one of the 3 options and delete the tables of the options which have not been selected.

If option 1 is selected: Each member of the team must select a user story and test the acceptance tests identified for that story, indicating which ones have been passed and making the necessary changes for those which have not been passed.

If option 2 is selected: A static code analysis must be performed. Automatic tools can be used (e.g., SonarLint, SonarQube...). Each member of the team must address one issue or group of related issues and solve the problems associated with it.

If option 3 is selected: An accessibility analysis must be performed. Automatic tools such as Accessibility Scanner, or features such as TalkBack can be used to identify issues. Each member of the team must address one issue or group of related issues and solve the problems associated with it.