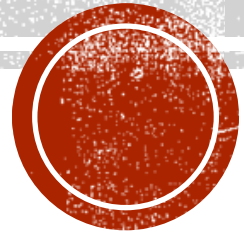
SECURE FILE SHARING APP

Secure Programming exercise work



AIM OF THE PROGRAM

- The program aims to demonstrate secure development practices for handling personal files online
- Users can register with a valid email and secure password, then log in to access their files
- Once authenticated, users can upload, download, view, and delete only their own files

TECHNOLOGIES USED

- Frontend: React
- Backend: Nodejs, Express
- Programming Language: TypeScript
- CI/CD Pipeline: GitHub Actions
- No database

SECURE PROGRAMMING

- User passwords are hashed using bcrypt
- Sessions are managed with JWTs which are stored HttpOnly cookies
- Protected endpoints
- File sanitizing
- CORS policies

```ts
// Registration route: validates inputs and securely hashes passwords
router.post(
  "/register",
  [
    body("email")
      .isEmail()
      .withMessage("Must be a valid email address"),
    body("password")
      .isLength({ min: 8 })
      .withMessage("Password must be at least 8 characters long"),
  ],
  async (req: Request, res: Response): Promise<any> => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const { email, password } = req.body;

    // Prevent duplicate account registration
    const existingUser = users.find(user => user.email === email);
    if (existingUser) {
      return res.status(400).json({ message: "User already exists" });
    }

    // Securely hash password before storing
    const passwordHash = await bcrypt.hash(password, 10);
    users.push({ email, passwordHash });

    return res.status(201).json({ message: "User registered successfully" });
  }
);
```

```
  "/login",
  async (req: Request, res: Response): Promise<any> => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const { email, password } = req.body;

    const user = users.find((u) => u.email === email);
    if (!user) {
      return res.status(401).json({ message: "Invalid email or password" });
    }

    const isPasswordValid = await bcrypt.compare(password, user.passwordHash);
    if (!isPasswordValid) {
      return res.status(401).json({ message: "Invalid email or password" });
    }

    const token = jwt.sign({ email: user.email }, JWT_SECRET, {
      expiresIn: "1d",
    });

    // Set HttpOnly cookie to securely store JWT
    res.cookie("token", token, {
      httpOnly: true,
      secure: true, // Only over HTTPS in production
      sameSite: "strict",
      maxAge: 1000 * 60 * 60 * 24, // 1 day
    });

    res.json({ message: "Login successful" });
```

```
// Logout route: clears authentication cookie
router.post("/logout", (req: Request, res: Response) => {
  res.clearCookie("token", {
    httpOnly: true,
    secure: true,
    sameSite: "strict",
  });

  res.json({ message: "Logged out successfully" });
});
```

```
// Secure download: validate filename to prevent path traversal attacks
router.get("/download/:filename", authenticate, (req: Request, res: Response) => {
  let file = req.params.filename;

  file = sanitizeFilename(file);

  const uploadsDir = path.resolve(__dirname, "../uploads");
  const filePath = path.join(uploadsDir, file);

  // Confirm the resolved path stays inside uploads directory
  if (!filePath.startsWith(uploadsDir)) {
    res.status(400).json({ message: "Invalid file path" });
    return
  }

  if (!fs.existsSync(filePath)) {
    res.status(404).json({ message: "File not found" });
    return
  }

  res.download(filePath);
});
```

TESTING

- GitHub Actions runs Dependency-Check and SonarQloud on every push ad publishes reports
- SonarCloud found vulnerabities that were fixed
- Dependency-Check found vulnerable packages that are dependencies of React

AI USAGE

- ChatGPT was used to create CSS and help with problems that came up while programming