Secure Programming Project – Hospital Application

GROUP MEMBERS: HAJAR EL BOUTAHIRI

PHAM VU VAN THANH

Introduction

- Secure Hospital Application for Booking, Viewing and Managing Appointments and Users
- Support 5 Roles: Default Admin, Admin, Doctor, Patient and User
- Aim of the Program: By Building an Application From Scratch, we aimed to:
  - Enhance Software Engineering Skills
  - Learn Security Practices
  - Security Testing

Structure of program

- Backend:
  - FastAPI (Python)
- Frontend:
  - Angular (TypeScript, HTML, CSS)
- Database:
  - SQLite
- Docker:
  - Docker compose

Security practices

- Oauth2 + JWT for Authentication
  - Short Lived Access Tokens: 30 mins
- Log Out with Token Blacklist Implementation
  - Invalid Token, even though it still didn't expire
  - Revoke Token Whenever it is Needed
- Password Encryption on Database Using Bcrypt
  - Bcrypt a Password Hashing Function
  - One Way Hashing
- Sensitive Data Encryption Using Fernet
  - Fernet a Symmetric Encryption and Data Authentication Algorithm
  - Provides Confidentiality, Integrity and Authentication
- HTTPS Implementation
  - Used mkcert Tool to Generate a Self-Signed Certificate and Private Key

Security practices

- Role Based Access Control
  - Default Admin, Admin, Doctor, Patient and User
  - Separate Views and Allowed Actions
- Scheduled Data and Access Validity Check
  - Deactivate Expired Users and their Related Records: Every Day at Midnight
  - Update Appointments Status: Every 30 mins
- Input Validation
  - Ensure User Input is Correct and Valid
- Error Handling
  - Handle Expected Errors
- Exception Handling
  - Handle Unexpected Errors

HTTPS Implementation

Backend: "main.py"

```python
# allow requests from the frontend
app.add_middleware(
    CORSMiddleware,
    allow_origins=["https://localhost:4200"],
    allow_credentials=True,       # Allow cookies or credentials if needed
    allow_methods=["*"],          # Allow all HTTP requests
    allow_headers=["*"],          # Allow all headers
)
# initialize database
init_db()

# Include API routes
app.include_router(api_router)


# # run the server with https
if __name__ == "__main__":
    uvicorn.run(
    app,
    host="0.0.0.0",
    port = 8432,
    ssl_keyfile="../Certificate/key.pem",
    ssl_certfile="../Certificate/cert.pem",
    lifespan = "on",
)
```

Frontend: "angular.json"

```json
"serve": {
  "builder": "@angular-devkit/build-angular:dev-server",
  "configurations": {
    "production": {
      "buildTarget": "app:build:production"
    },
    "development": {
      "buildTarget": "app:build:development",
      "ssl": true,
      "sslKey":"../Certificate/key.pem",
      "sslCert":"../Certificate/cert.pem"
    }
  },
  "defaultConfiguration": "development"

},
```

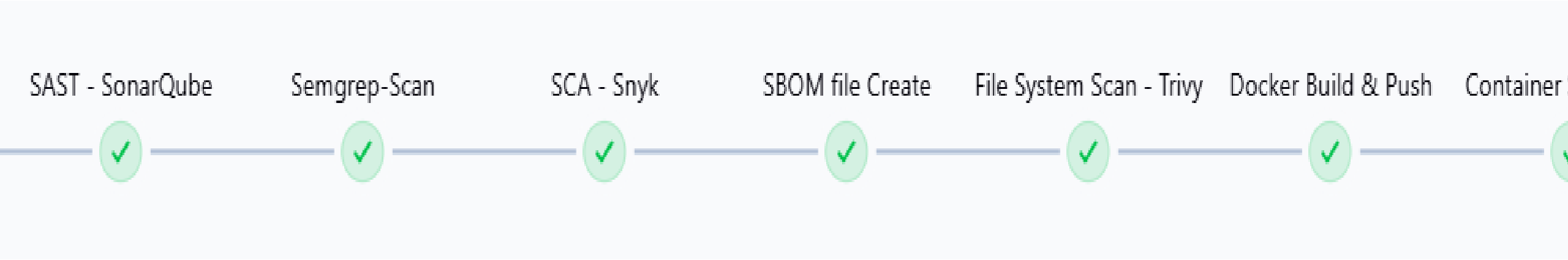
Oauth2 + JWT Implementation

```python
# User login and return token
@router.post("/login/")
async def login_for_access_token(
    form_data: OAuth2PasswordRequestForm = Depends(),
    db: Session = Depends(get_db)
    ) -> Token:
    user = authenticate_user(form_data.username, form_data.password, db)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect Username or Password",
            headers={"WWW-Authenticate": "Bearer"},
        )
    access_token_expires = timedelta(minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(
        data={"sub": str(user.user_id)}, expires_delta=access_token_expires
    )
    return Token(access_token=access_token, token_type="bearer")
```

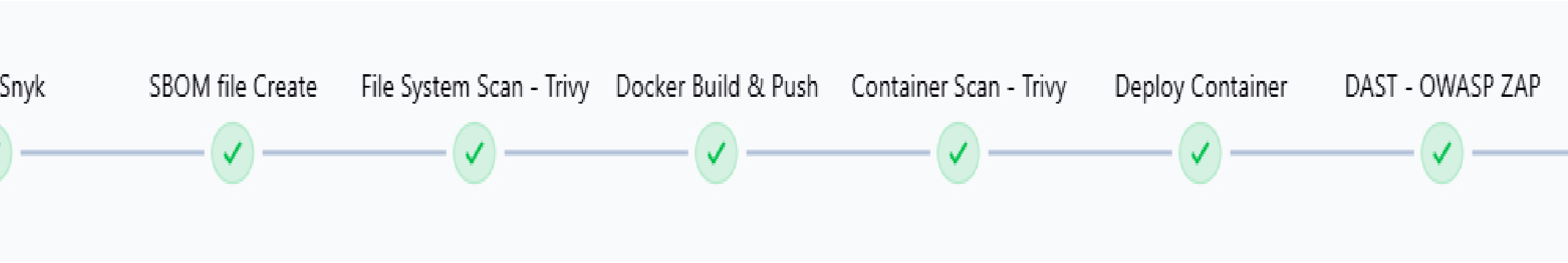
```python
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/auth/login")
```

```python
async def get_current_user(token: str = Depends(oauth2_scheme), db: Session = Depends(get_db)):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )
    # verify expired session token
    try:
        if is_logged_out(token, db):
            raise HTTPException(status_code=400, detail=authentication_error)
        payload = jwt.decode(token, settings.JWT_SECRET_KEY, algorithms=[settings.ALGORITHM])
        user_id = payload.get("sub")
        if not user_id:
            raise credentials_exception
        token_exp = payload.get("exp")
        if datetime.~~utcnow~~().timestamp() > token_exp:
            raise HTTPException(
                status_code=status.HTTP_401_UNAUTHORIZED,
                detail="Expired Session",
                headers={"WWW-Authenticate": "Bearer"},
            )
        token_data = TokenData(user_id=user_id)
    except JWTError:
        raise credentials_exception
```

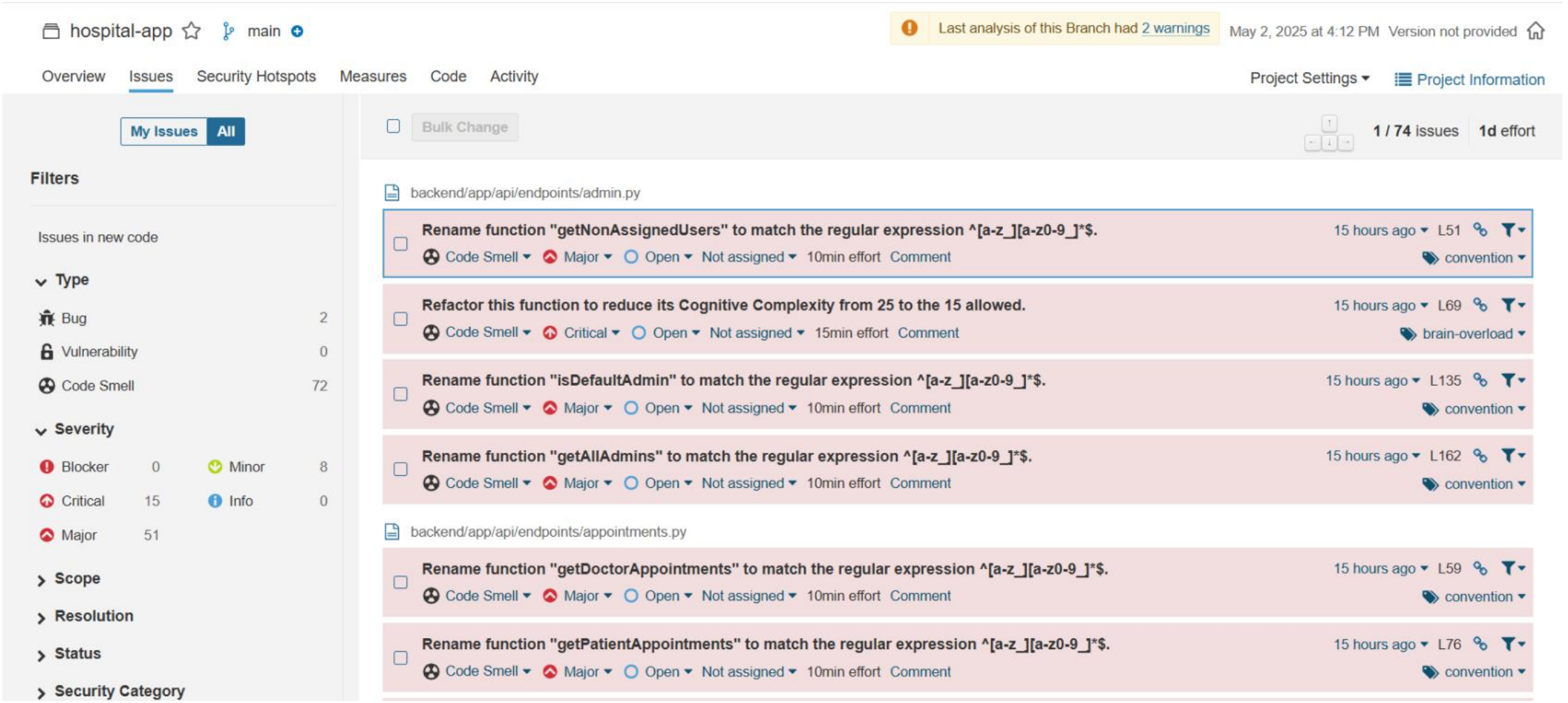
Security tests



Security tests



Security tests - Result



Name	Risk Level	Number of Instances
CSP: Failure to Define Directive with No Fallback	Medium	2
Content Security Policy (CSP) Header Not Set	Medium	3
Missing Anti-clickjacking Header	Medium	3
Sub Resource Integrity Attribute Missing	Medium	7
Dangerous JS Functions	Low	1
Insufficient Site Isolation Against Spectre Vulnerability	Low	12
Permissions Policy Header Not Set	Low	10
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	6
X-Content-Type-Options Header Missing	Low	9
Authentication Request Identified	Informational	1
Information Disclosure - Sensitive Information in URL	Informational	4
Information Disclosure - Suspicious Comments	Informational	8
Modern Web Application	Informational	1
Non-Storable Content	Informational	2
Storable and Cacheable Content	Informational	5
Storable but Non-Cacheable Content	Informational	5

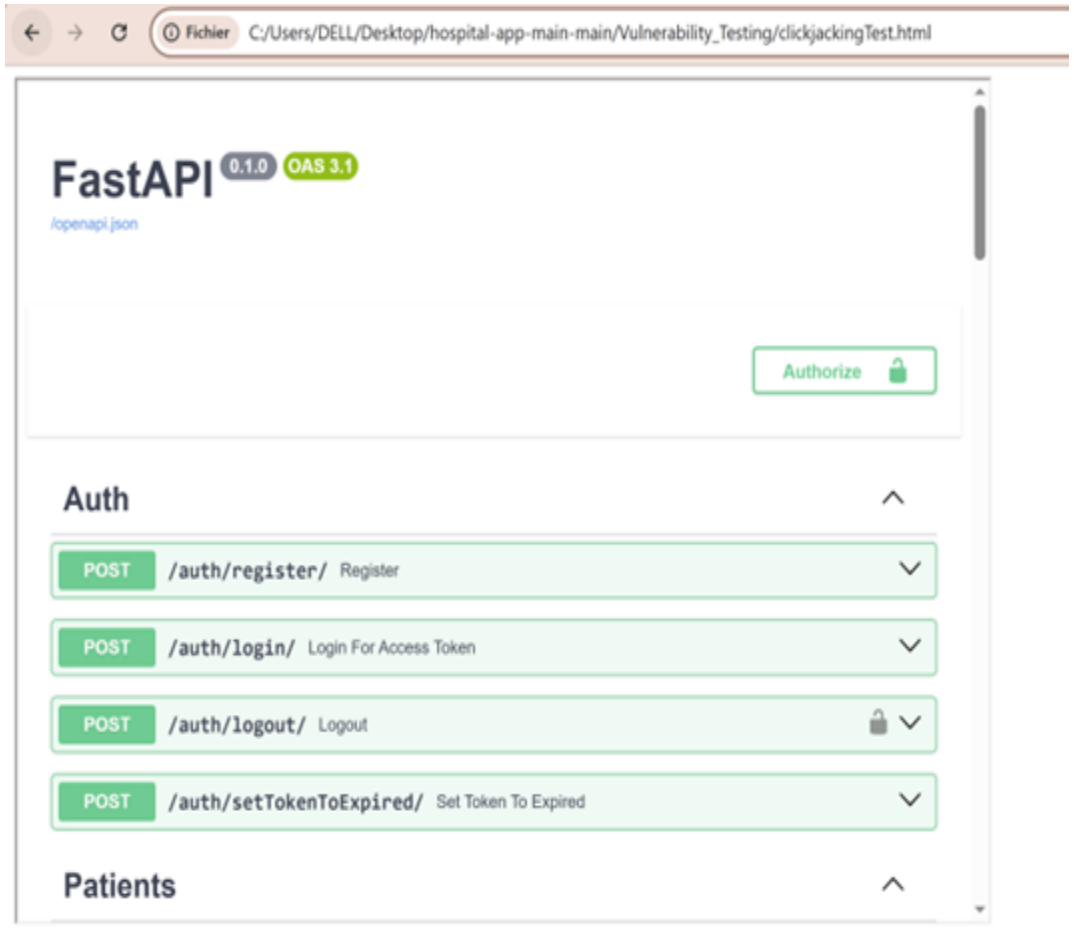
Enhanced Security

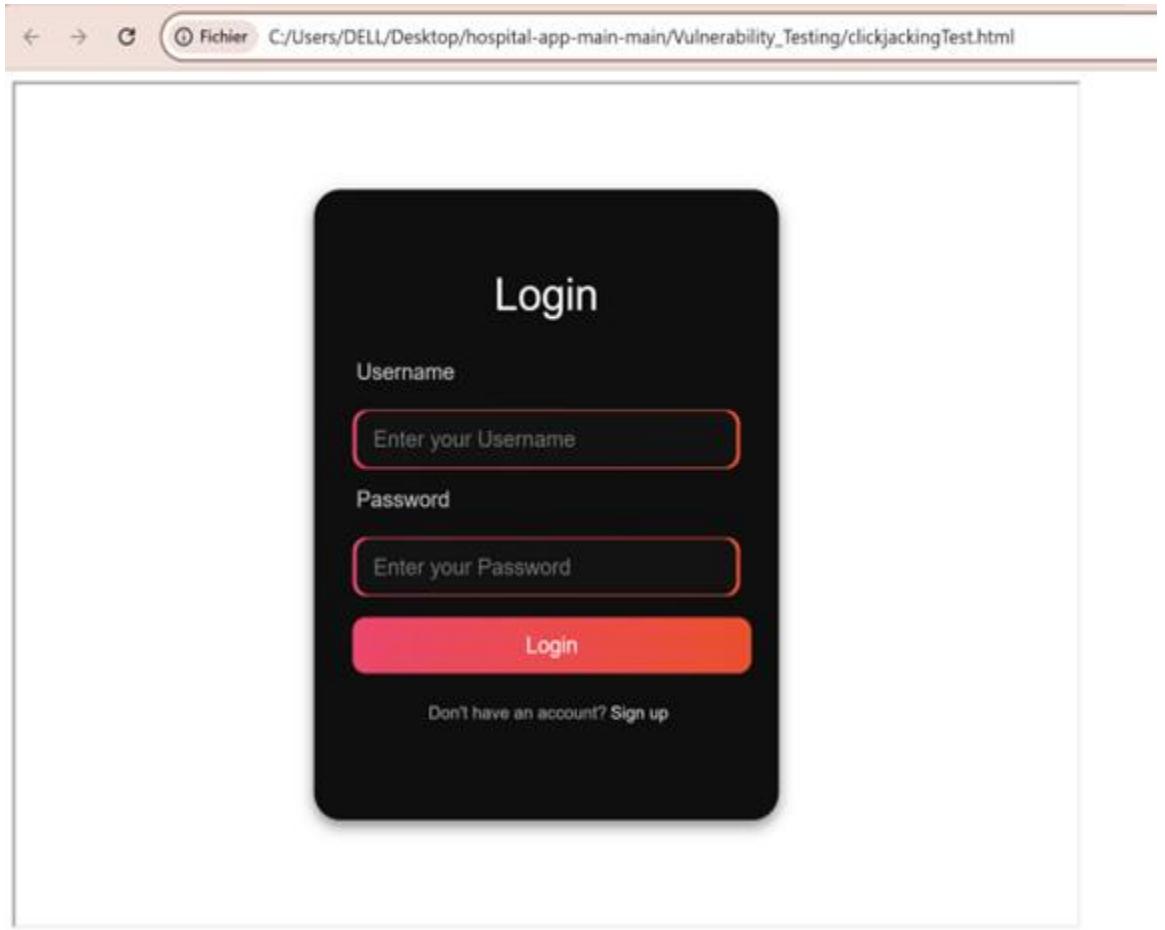
- Set Security Headers to Sensitive Files (" Dynamic Files")
  - Anti-clickjacking Header
  - Permissions Policy Header
  - X-Content-Type-Options Header
  - Site Isolation Against Spectre Vulnerability
  - Content Security Policy (CSP) Header
  - Caching
  - "X-Powered-By" (Removed Header)
- Added Error Handling Statements
- Sensitive Data Exposure Through Headers
  - Switch GET to POST Requests
- Code Refactoring & Function Renaming

Missing Anti-clickjacking Header Exploit

```html
<!DOCTYPE html>
<html>
<head>
  <title>Clickjacking Vulnerability Test</title>
</head>
<body>

  <!-- ***************** To Test Backend  ***************** -->
  <iframe src="https://localhost:8432/docs" width="800" height="600"></iframe>


  <!-- ***************** To Test Frontend  ***************** -->
  <iframe src="https://localhost:4200" width="800" height="600"></iframe>

</body>
</html>
```

Missing Anti-clickjacking Header Exploit

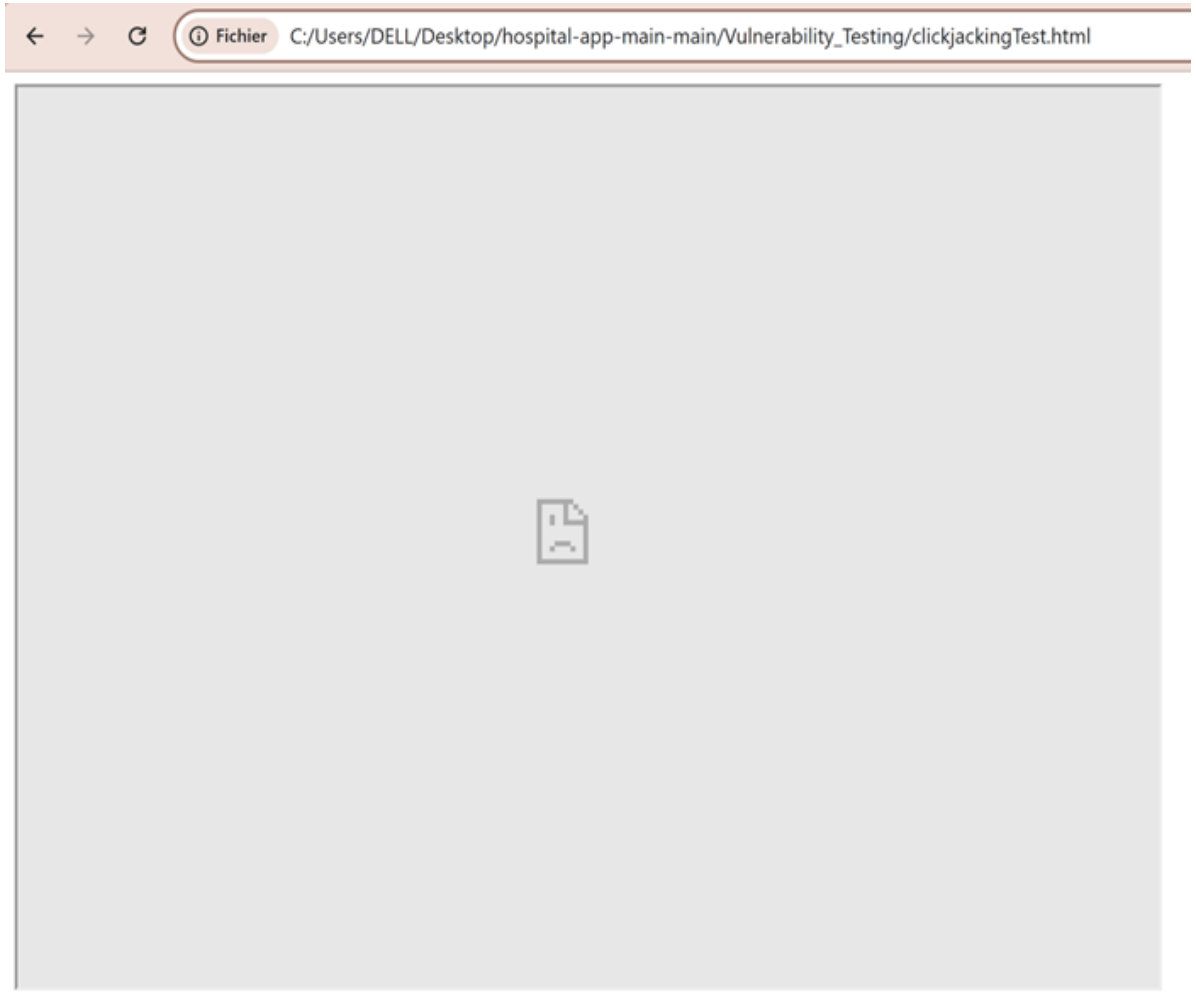




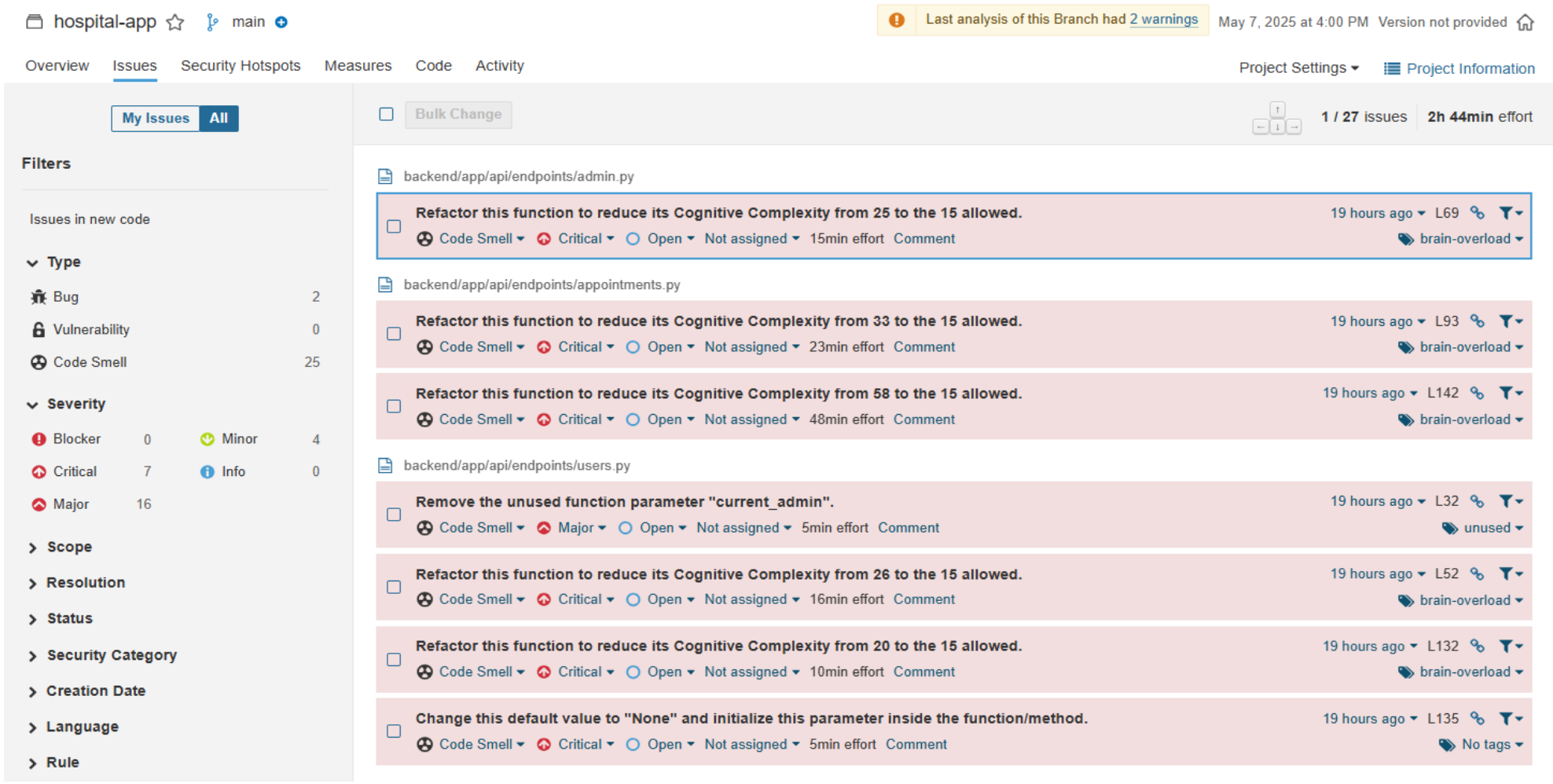
Solution Implementation

```python
class secureHeader(BaseHTTPMiddleware):
    Tabnine | Edit | Test | Explain | Document
    async def dispatch(self, request: Request, call_next):
        response: Response = await call_next(request)
        if request.url.path == "/docs":  # to allow only FASTAPI UI
            return response
        response.headers["X-Frame-Options"] = "SAMEORIGIN" # Missing
        return response

app.add_middleware(secureHeader)
```

```javascript
const app = express();

Tabnine | Edit | Test | Explain | Document
app.use((req, res, next) => {
  res.setHeader('X-Frame-Options', 'SAMEORIGIN');
  next();
});
```



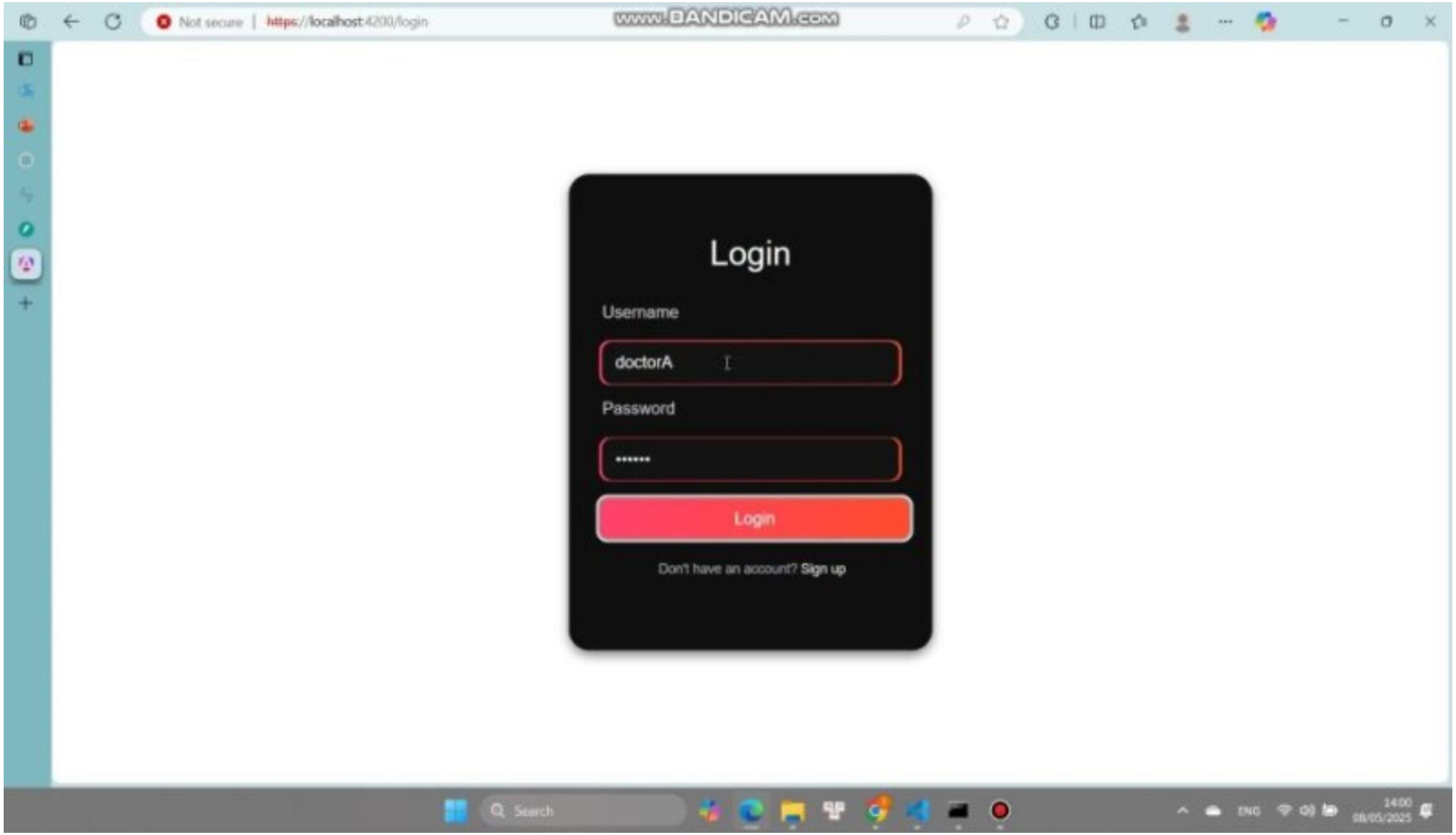
Security tests (Second time) - Result



Name	Risk Level	Number of Instances
CSP: Failure to Define Directive with No Fallback	Medium	2 -> 1
Content Security Policy (CSP) Header Not Set	Medium	3
~~Missing Anti-clickjacking Header~~	~~Medium~~	~~3~~
Sub Resource Integrity Attribute Missing	Medium	7 -> 1
Dangerous JS Functions	Low	1
Insufficient Site Isolation Against Spectre Vulnerability	Low	12 -> 8
Permissions Policy Header Not Set	Low	10 -> 8
~~Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)~~	~~Low~~	~~6~~
X-Content-Type-Options Header Missing	Low	9 -> 8
Authentication Request Identified	Informational	1
Information Disclosure - Sensitive Information in URL	Informational	4
Information Disclosure - Suspicious Comments	Informational	8 -> 7
Modern Web Application	Informational	1
Non-Storable Content	Informational	2
~~Storable and Cacheable Content~~	~~Informational~~	~~5~~
Storable but Non-Cacheable Content	Informational	5

OWASP ZAP report

Demo


Not secure | https://localhost:4200/login
www.BANDICAM.com
Login
Username
doctorA
Password
Login
Don't have an account? Sign up
Search

Further improvement

- Add Security Headers to Static Files
- Encrypt the Entire Database
- Resolve All Vulnerabilities in The Scanning Report
- Add Logs Auditing
- Implement Refresh Token
- Implement Change Password Functionality

AI Usage

- ChatGPT
- Find information: framework, security practices, vulnerabilities, etc.
- Assist coding and debugging
- Provide best-practice to avoid vulnerabilities

Thank you

Questions ?