DLP Healthcare Application

Final Report

Washington State University

Jacob Eckfeldt;

Reed Havens;

Hemendra Kathi;

Alexander Shirk;

CptS 423 Software Design Project II

Fall 2023

Instructor: Balasubramanian Kandaswamy

Table of Contents

I. Introduction

1. With the proliferation of business web applications across all industries, there is more user information being transmitted across the internet than ever before. Many fields, such as healthcare and finance, deal with the transmission of confidential user information on a regular basis, whether it is a patient's medical history or a customer's routing number. This bulk of private information necessitates secure methods of transmission and storage without major impediment to the user experience. Our team aims to explore the implementation of such security methodologies within the context of a healthcare web application to enable the fast and secure serving of web content to users of multiple classes. While many basic web applications have built in security parameters, whether through frameworks such as Django or databases such as PostgreSQL, we aim to explore the implementation of manual security measures against an array of assessed threats such as phishing, spoofing, tampering, elevation of privilege, and information disclosure to aid in data loss prevention.

   The team aims to explore the use and construction of the tools required to mitigate these threats within the context of a healthcare application utilized by patients, doctors, and nurses to review patient histories, upcoming or past appointments, write or read notes on a given appointment, and review patient information. In doing this, there are several key overarching objectives that define the nature of the project. First, the data must be transmitted, handled, and presented securely at all times. The primary focus of this project is in data loss prevention methodologies and tools – their effectiveness, cost, and form – and so the protection of data against the array of surveyed threats is the highest priority. Additionally, it will be imperative to explore and implement data access controls to ensure a given party is able to access only the data they are permitted.

   Second, the implementation of these DLP must not cause undue lag, discomfort, or disruption to the user. Any security methodology only functions when the end user is cooperative, and so it is imperative that the tools utilized do not negatively impact the user experience, no matter the user class, to any meaningful extent. The final product should serve as a functional user portal for doctors, nurses, and patients to interact with patient and appointment history.

2. Our primary client is Jason Burt, a Google PM. The final project will serve as an exploration and analysis of existing data loss prevention strategies within a healthcare context.

3. Jason Burt, Google

II. Team Members & Bios

Reed Havens is a senior at Washington State University pursuing a degree in Computer Science, and plans on graduating in the Fall of 2023. Reed has interests in machine learning and web design and plans to explore these interests further following graduation. Reed has previous experience with web design in his university projects. Additionally, he has experience in Python and cybersecurity from courses and projects undertaken at WSU.

Hemendra Kathi is a senior at Washington State University pursuing a degree in Computer Science, and plans on graduating in the Fall of 2023. Hemendra has interests in networks and cybersecurity and plans to explore these interests further following graduation.Hemendra has taken previous classes relating to website development and cybersecurity. Additionally,he has also completed a 3 month course online about IT fundamentals to further cement his understanding.

Jacob Eckfeldt is a senior at Washington State University pursuing a degree in Computer Science, and plans on graduating in the Fall of 2023. Jacob has interests in Mechanical applications and web design and plans to explore these interests further following graduation. Jacob has previous experience with web design in his university projects.

Alexander Shirk is a senior at Washington State University pursuing a degree in Computer Science, and plans on graduating in the Fall of 2023. Alex has interests in systems programming and Linux and plans to explore these interests further following graduation. Alex has previous experience in Python and AWS from his software engineering internship. Additionally, he has taken a variety of classes at WSU, including systems programming and operating systems.

III. Project Requirements Specification

III.1. Project Stakeholders

There are multiple stakeholders within WSU, Google, and assorted healthcare institutions.

Potential interested parties in project output would be healthcare institutions as well as web developers with respect to the security insights derived from the project. To appeal to these parties, it would be important for the front-end website to be clean and friendly and the implementation of security methodologies to be well documented and easy to interpret.

It would also benefit all stakeholders of the project to write clean and well-documented code that is easy to evaluate. The DLP project team will endeavor to treat these needs as prerequisites as we see to the preferences discussed above. Our primary client (Jason Burt, Google PM) will take priority, but the needs and preferences of other stakeholders will be carefully considered throughout the project lifecycle and software design processes.

III.2. Use Cases

Registration (Patient)

Pre-Condition	- Connected to web app
Post-Condition	- User information is saved in DB - User is automatically logged in
Basic Path	Navigate to login page User selects “Register” User inputs necessary info and submits
Alternative Path	1. User attempts to navigate to any web page within the app 2. Non-logged in users are redirected to login / register page 3. User registers
Related Requirements	

Schedule Appointments

Pre-Condition	- Connected to web app - User is logged in - >=1 Doctor and Patient in DB
Post-Condition	- Appointment information is saved in DB - Appointment info is accessible by both parties
Basic Path	1. Navigate to appointments page 2. User selects “Schedule Appointment” 3. User inputs necessary info (date, doctor/patient, notes) and submits
Alternative Path	1. User reviews past appointment and selects “Schedule Follow-Up Appointment” 2. Information is auto-filled 3. User inputs optional additional notes and submits
Related Requirements	- Registration -

Review Appointment

Pre-Condition	- Connected to web app - User is logged in - >=1 appointment in history
Post-Condition	- Any changes made are saved to DB
Basic Path	1. Navigate to “Review Appointments” page 2. User selects a past or future appointment 3. User inputs notes or modifications and submits
Alternative Path	-
Related Requirements	- Registration - Schedule Appointment

Add Patient Diagnosis (Doctor)

Pre-Condition	- Connected to web app - User is logged in as Doctor class - >=1 Patient users assigned to given Doctor
Post-Condition	- Any changes or notes made are saved to DB
Basic Path	1. Navigate to “Review Appointments” page 2. User selects a past or future appointment 3. User inputs notes or modifications and submits
Alternative Path	-
Related Requirements	- Registration - Schedule Appointment - Review Appointment

Edit Profile (Patient)

Pre-Condition	- Connected to web app - User is logged in as Patient class
Post-Condition	- Any changes or notes made are saved to DB
Basic Path	1. Navigate to home page 2. Select “Profile” from Navigation Bar 3. Select “Edit Profile” 4. Edit chosen fields and select “Save”
Alternative Path	-
Related Requirements	- Registration

Edit Profile (Doctor)

Pre-Condition	- Connected to web app - User is logged in as Doctor class

Post-Condition	- Any changes or notes made are saved to DB
Basic Path	5. Navigate to home page 6. Select “Patients” from Navigation Bar 7. Select a Patient 8. Select “Edit Profile” 9. Confirm password. 10. Edit chosen fields and select “Save”
Alternative Path	-
Related Requirements	- Registration

Add Diagnosis (Doctor)

Pre-Condition	- Connected to web app - User is logged in as Doctor class
Post-Condition	- Any changes or notes made are saved to DB
Basic Path	11. Navigate to home page 12. Select “Add Diagnosis” from Navigation Bar 13. Select a Patient 14. Add Diagnosis 15. Add Notes 16. Edit chosen fields and select “Save”
Alternative Path	-
Related Requirements	- Registration

Login (2FA)

Pre-Condition	- Connected to web app - User is not logged in - User has authenticator app with

	secret=’base32secretbase32’
Post-Condition	- User is authenticated
Basic Path	17. Navigate to login page 18. Enter username and password 19. Enter OTP code from any Authenticator app 20. Submit and Log in
Alternative Path	-
Related Requirements	- Registration

III.3. Functional Requirements

User Authentication

Description	The app should implement user authentication mechanism which allows users to register/log in.Only registered and validated users can access the application
Source	Internal requirements elicitation among members of the team.
Priority	Level 0: Essential and required functionality.

Appointment Scheduling

Description	The app should allow users to schedule appointments with the healthcare provider.The details necessary inclue date/time and doctor/nurse’s name.
Source	Internal requirements elicitation among members of the

	team.
Priority	Level 0: Essential and required functionality.

Appointment Notes and Modifications

Description	The app should allow patients and nurses to change the appointment details when needed.
Source	Internal requirements elicitation among members of the team.
Priority	Level 0: Essential and required functionality.

Patient Diagnosis

Description	Doctors should be able to provide their diagnosis,add medical notes and update the patient's profile
Source	Internal requirements elicitation among members of the team.
Priority	Level 0: Essential and required functionality.

Data Loss Prevention

Description	The app should implement multiple critical measures that will prevent data loss including encryption and access control
Source	Internal requirements elicitation among members of the team.
Priority	Level 0: Essential and required functionality.

III.4. Non-Functional Requirements

Policy Compliance: The application should comply with contemporary data security health care regulations such as HIPAA (Health Insurance Portability and Accountability Act).

User-Friendly Interface:The application has a simple intuitive layout for easy navigation and interaction.

Efficiency: The application's performance as a healthcare portal must not be significantly impacted by the implementations of DLP strategies.

Availability: refers to uptime of service, promised / expected uptime, robustness (shouldn't crash at bad user input, e.g. need input validation). Uptime expected e.g. 99.99% of time per month

Multi-Factor Authentication: In addition to the regular login credentials,the application needs the user to verify his identity via a one-time password.

Encryption: The application should encrypt select fields to prevent malicious actors from obtaining sensitive information in case of a data leak

Data Obfuscation: Sensitive information is only visible to the Doctor when he chooses to reveal it; until then, the information is obscured by the application.

Additional Verification: The application needs the doctor to verify his identity to edit the patient's information.

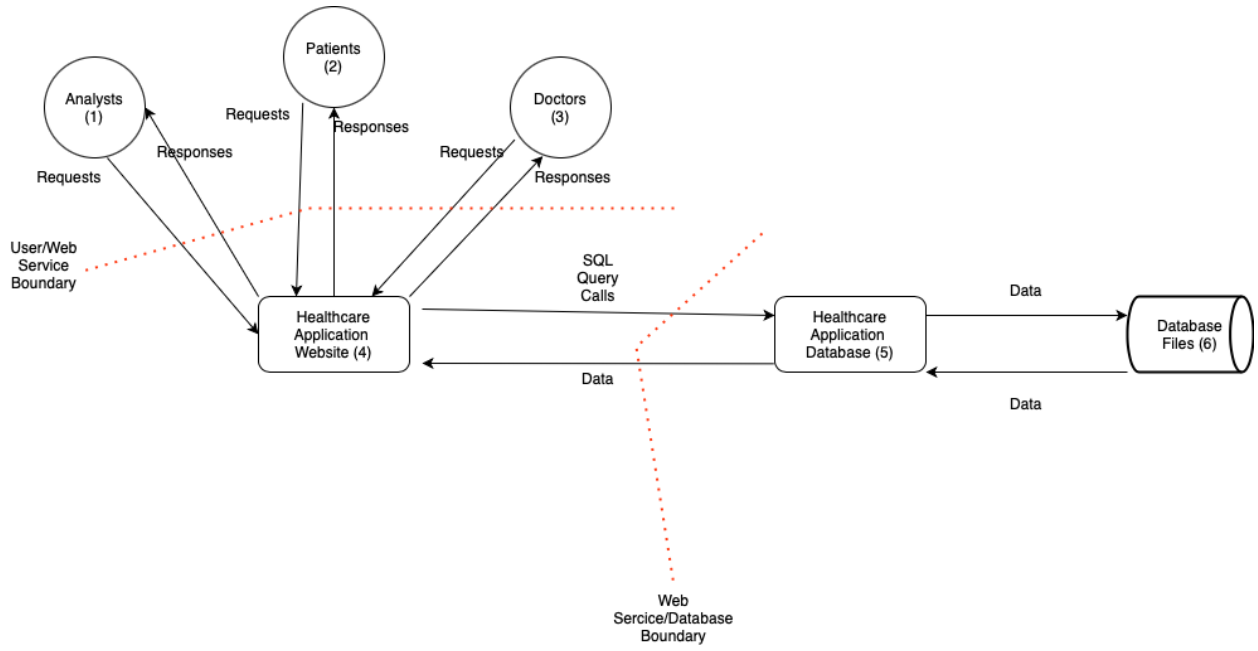
DDoS Mitigation: The application is equipped with mechanisms to detect and mitigate Distributed Denial of Service (DDoS) attacks, ensuring uninterrupted service delivery by effectively handling and mitigating the impact of excessive and malicious network traffic influxes.

IV. Software Design - From Solution Approach

IV.1. Architecture Design

IV.1.1. Overview

The HealthCare web app is doctor,patient portal that facilitates communication between the two user classes:Patient and Doctor.It offers doctors a wide range of functionality including a comprehensive dashboard to view appointments,access/edit patient records and adding a diagnosis.It allows the patients to keep track of their appointments and edit their personal data aiding their health care experience.The application utilizes Django as a Python-based web framework and makes use of its model-template-views architecture to develop the web application and its security features.The diagram below showcases the layout of the application.



IV.1.2. Data Models

Patient

Data model for a patient contains all information a doctor or other healthcare professional would need to know about a patient. Some information is relevant for only logistics while other fields are relevant for a doctor(s) diagnosis. A patient can have many doctors.

id	Automatically generated
username	Character Field: max length 100
password	Character Field: max length 25
address	Character Field: limit 200
age	Positive Integer Field: min val 0, max val 130
createdAt	Timestamp
lastUpdated	Timestamp
phone	Integer Field
ssn	Character Field: max length 9
sex	Character Field: limit 1
weight	Positive Integer Field: min value 1, max value 2000
symptoms	Text Field
allergies	Text Field
appointments	One To Many Field, one patient has many appointments
history	Text Field
diagnosis	Character Field: max length 500
diagNotes	Character Field: max length 1000

Doctor

A doctor can have many patients. We assume doctor information is stored elsewhere since it is not relevant for a doctor's information to be recorded in the scope of this app the same as a patient.

id	Automatically generated
fname	Character Field: max length 50
lname	Character Field: max length 50

specialty	Character Field: max length 50
patients	Many To Many Field to DoctorPortal.patient
appointments	One To Many Field, one doctor has many appointments

PatientDoctor

A relational model represents a doctor patient pair. Many pairs can exist that include the same patient and the same for a doctor. Restricts the access level of the interaction between the Patient and Doctor.

id	Automatically generated
staffID	Integer field: default -1
patientID	Integer field: default -1
updaterID	Integer field: default -1
AccessLevel	Either (‘A’, ‘Admin’) or (‘V’, ‘Viewer’), default ‘V’
lastUpdated	Timestamp
createdAt	Timestamp, automatically generated

Appointments

Appointments are created by the doctor and viewed by patients. They can be edited only by the doctor. Access level allows for later addition of staff/patient interactions.

id	Automatically generated
staffID	Integer field: default -1, one-to-many
patientID	Integer field: default -1, one-to-many
updaterID	Integer field: default -1
AccessLevel	Either (‘A’, ‘Admin’) or (‘V’, ‘Viewer’), default ‘V’
lastUpdated	Timestamp

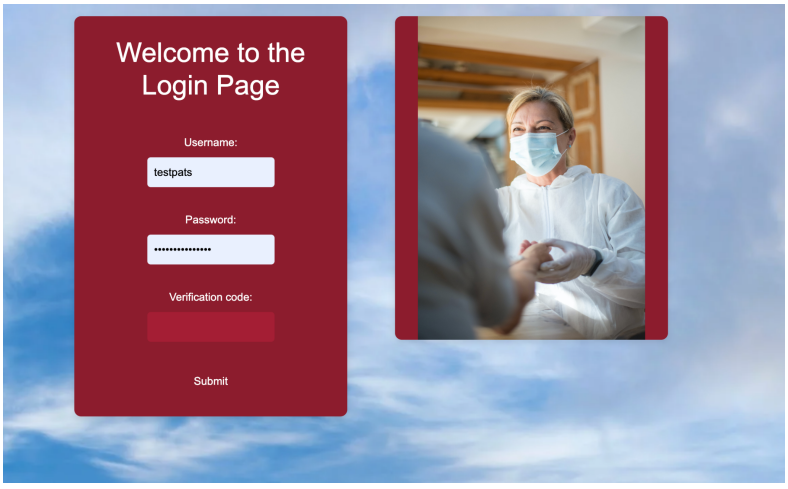
createdAt	Timestamp, automatically generated
clinicNotes	Str, variable length
patientNotes	Str, variable length

Diagnosis

Diagnosis are added by the doctor and viewed by patients.They can be edited only by the doctor

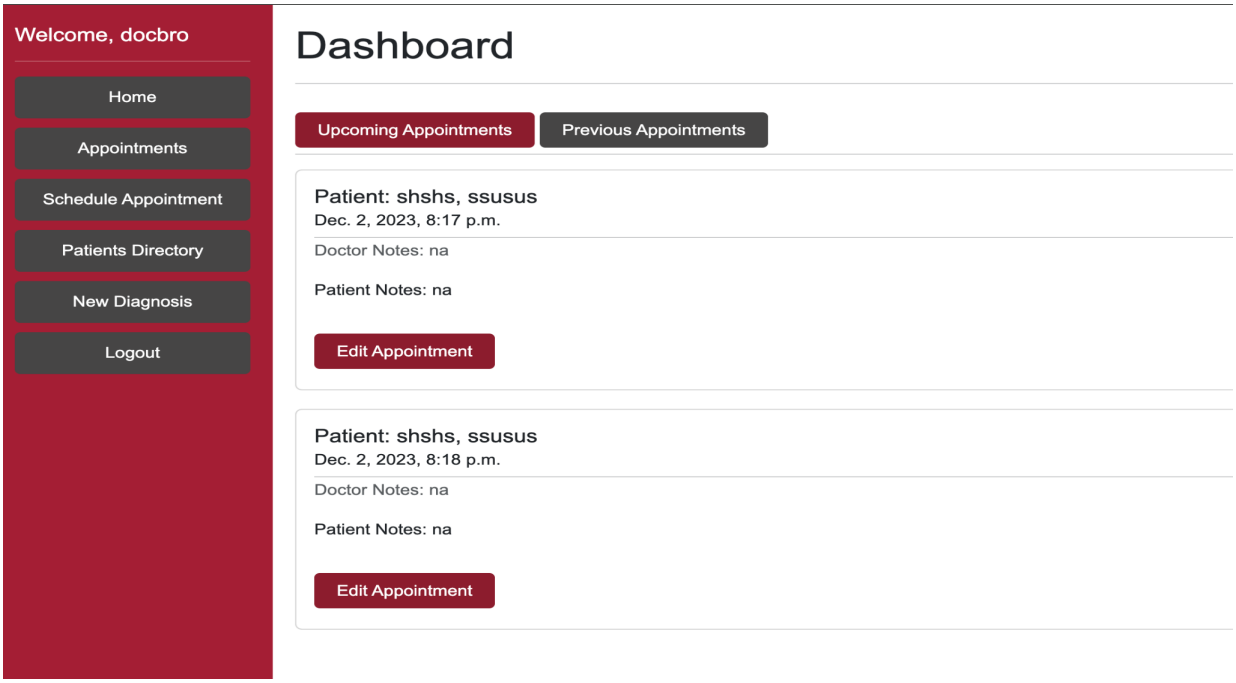
staffID	Integer field: default -1, one-to-many
patientID	Integer field: default -1, one-to-many
updaterID	Integer field: default -1
Diagnosis	Character field Max 200
DocNotes	Text Field
lastUpdated	Timestamp
createdAt	Timestamp, automatically generated

IV.3 Views

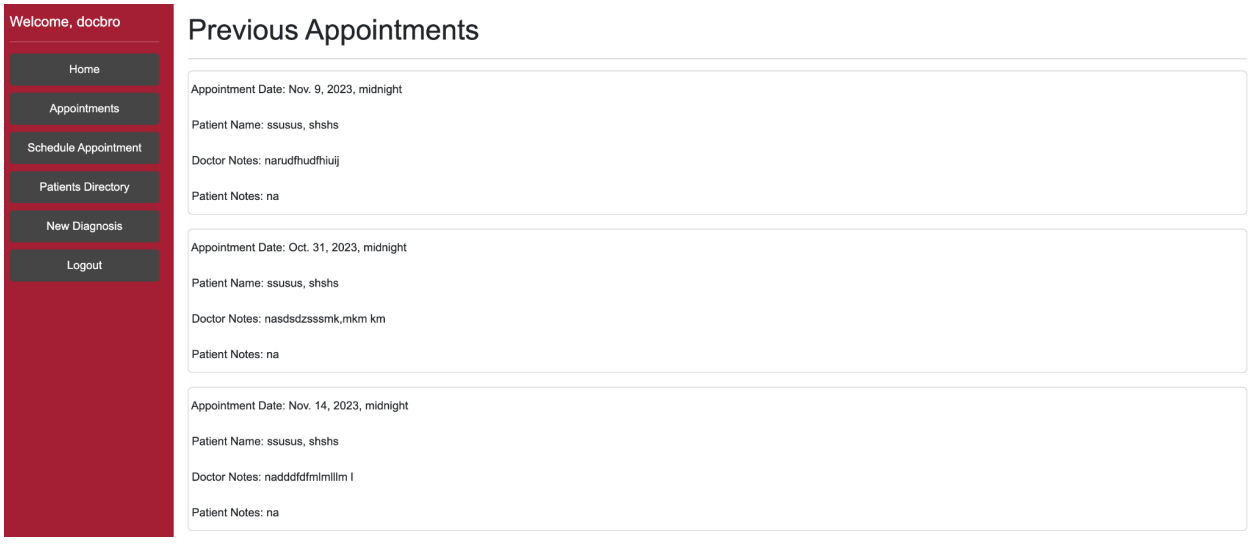


IV.3.1 Doctor

The homepage has the dashboard which by default shows the upcoming appointments. Previous appointment's information can be also accessed from here

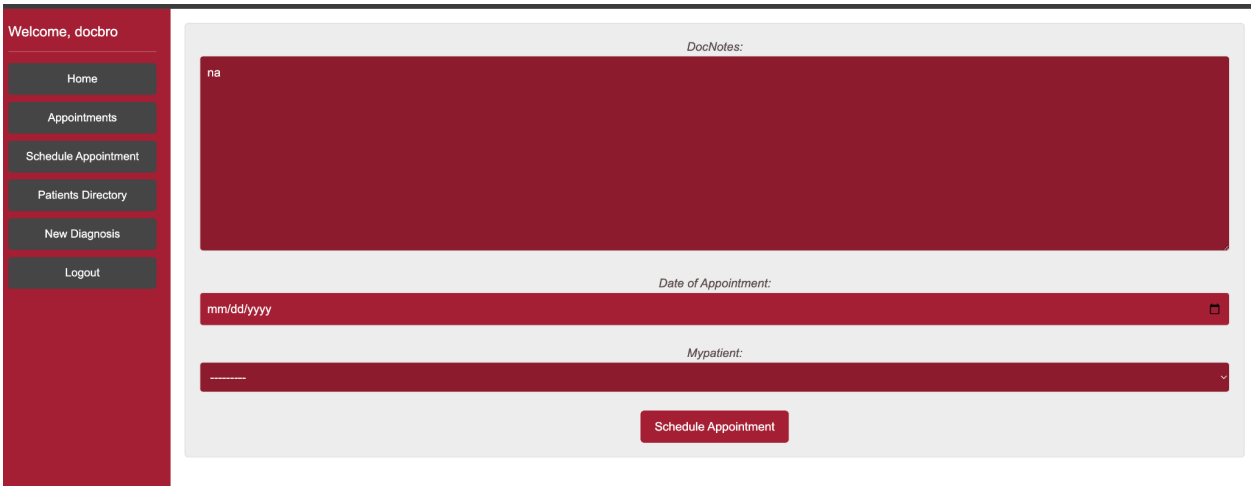


Home Page



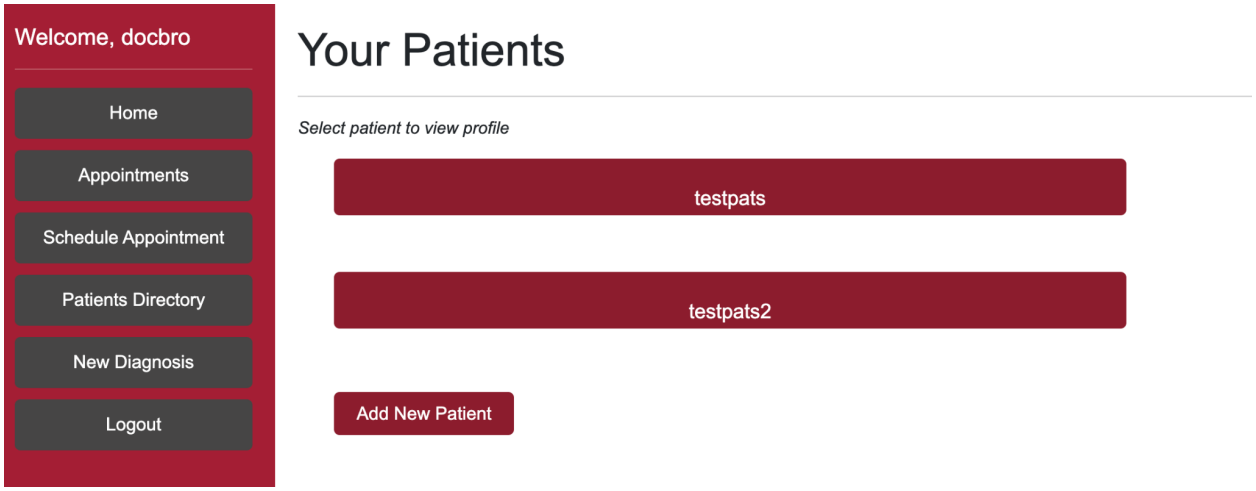
Previous Appointments Page

Appointments can be scheduled through the Schedule appointment page.



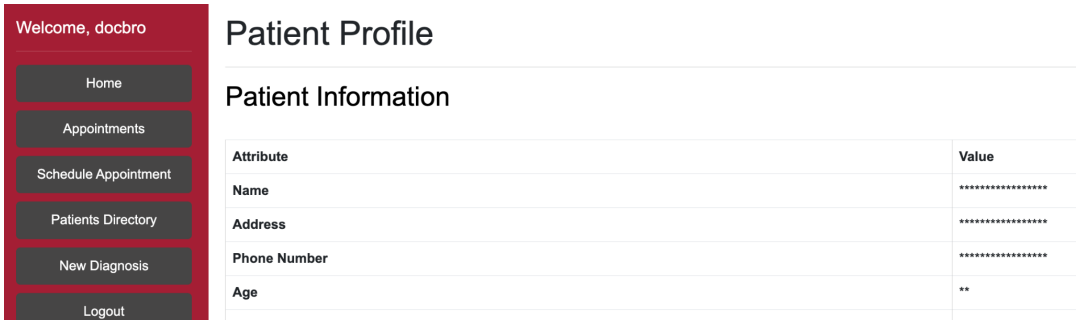
Schedule Appointment Page

Patients assigned to the doctor can be viewed through the Patient Directory.New patients can also be added from here.

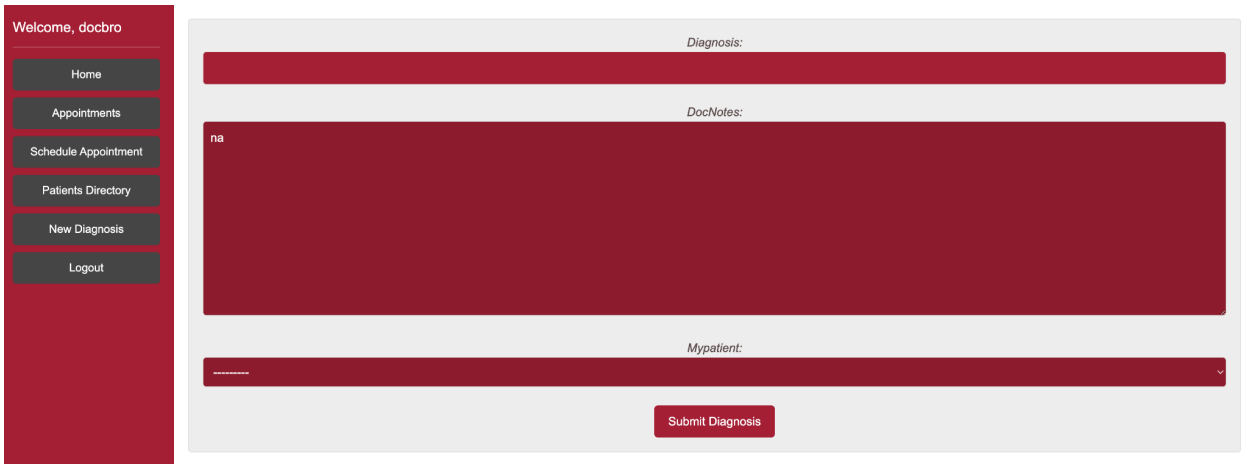


Patient Directory Page

Patient Profile can be accessed and edited from the Patient Directory



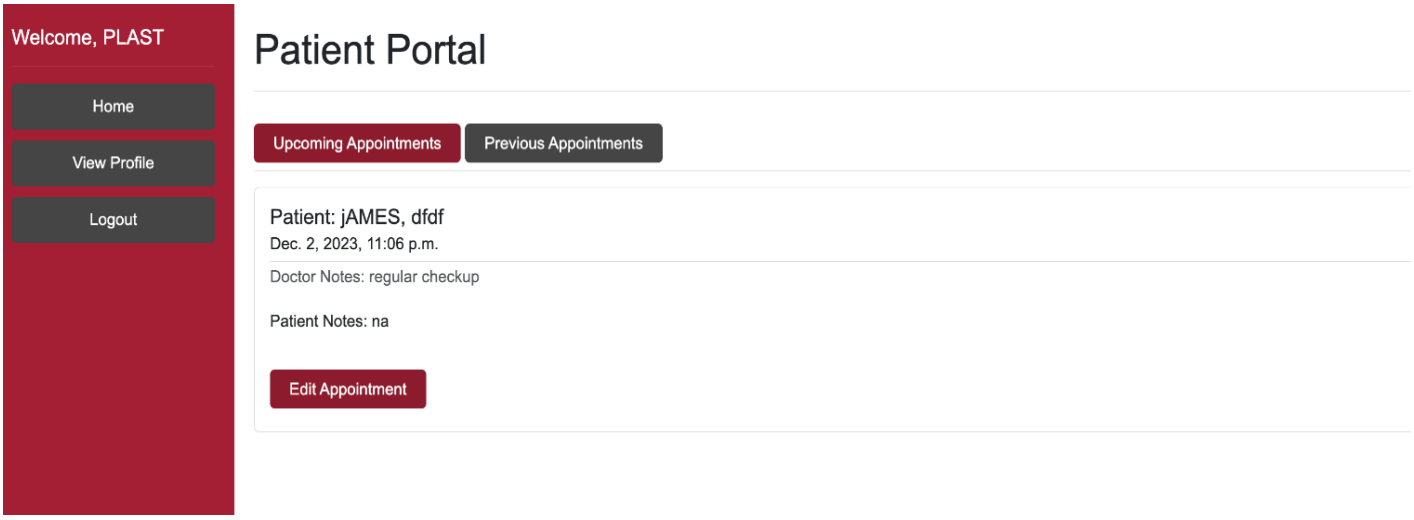
Latest Diagnosis can be quickly added through the New Diagnosis page



New Diagnosis Page

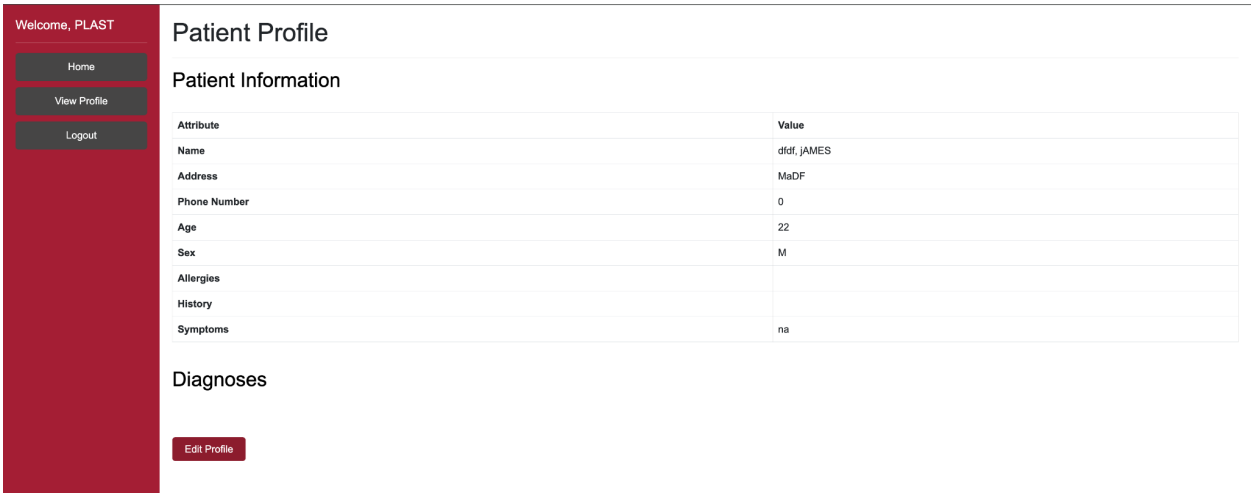
IV.3.2 Patient

Patient Home page shows upcoming appointments



Home Page

Patients can view and edit their personal information through view profile



Profile Page

IV.4 Controllers

The application uses Django's views as the controller component. These views handle the incoming HTTP requests and send the required response to the responsible controller function. The controller function executes the necessary code for processing that request and then coordinates with models and views to render the webpage.Below is the mapping between views and controller functions:

Patient

View	***Controller***	***Function***
register/	register	Registers a new user
'/'	login	Login Authentication
patientHome/	patientHome	Renders patient homepage
viewProfile/	viewProfile	Renders profile page
logoutPatient/	logoutPatient	Logs out the patient
viewFutureAppointmentsPatient/	viewFutureAppointmentsPatient	View upcoming appointments of the patient

editAppointmentPatient/<str:patientUsername>/<int:apptID>	editAppointmentPatient	Patient edits their appointment
editProfilePatient/	editProfilePatient	Edit patient profile by patient
viewPastAppointmentsPatient/	viewPastAppointmentsPatient	Patient views their past appointments

Doctor

View	***Controller***	***Function***
home/	doctorHome	Renders doctor homepage
addPatient/	addPatient	Add Patient page
addPatient/<str:patientUsername>/	addPatient2	Doctor adds a new patient
viewPatients/	viewPatients	Doctor views their patients
scheduleAppointmentDoctor/	scheduleAppointmentDoctor	Doctor schedules appointment
viewFutureAppointmentsDoctor/	viewFutureApppointmentsDoctor	Doctor views their upcoming appointments
viewPastAppointmentsDoctor/	viewPastAppointmentsDoctor	Doctor views their past appointments
addDiagnosis/	addDiagnosis	Doctor adds a diagnosis
logoutDoctor/	logoutDoctor	Doctor logs out
editProfileDoctor/<str:patientUsername>/	editProfileDoctor	Doctor edits a patient profile
editAppointmentDoctor/<str:patientUsername>/<int:apptID>/	editAppointmentDoctor	Doctor edits patient's appointment
viewProfileDoctor/<str:patientUsername>/	viewProfileDoctor	Doctor views their patient's complete profile
viewProfileDoctorSecure/<str:patientUsername>/	viewProfileDoctorSecure	Doctor sees obscured view of patient profile

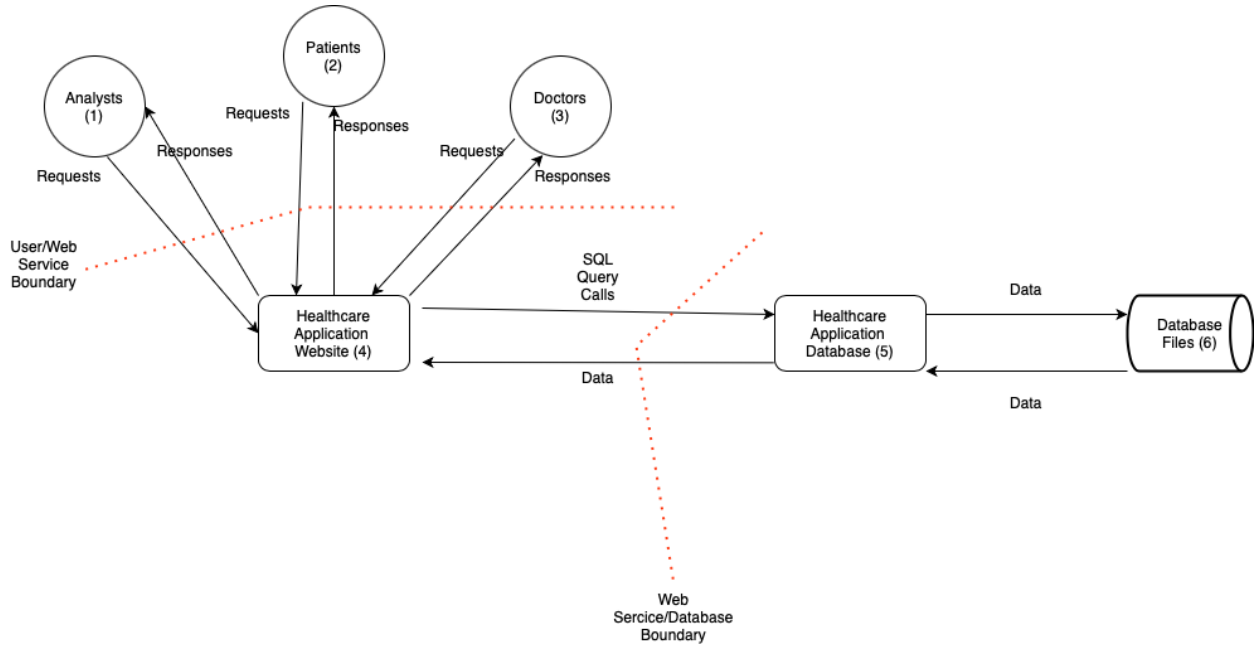
confirmPassword/<str:patient Username>	confirmPassword	Additional authentication for doctor

V. Threat Modeling & Mitigation

Threat Modeling & Analysis is important for accurately identifying security vulnerabilities and form the basis for developing Data loss prevention strategies.This section delves into the security threats originating from each of the subcomponents of our health care application.Furthermore, conclusions are drawn and the required prevention measures identified and implemented according to viability and priority.

V.1 Threat Modeling

STRIDE approach is used for threat modeling.It stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege.The threats identified are given below.



[Patients (User Class)]

Threat	Source	Target	Description

Spoofing	Patient(2)	Application Website (4)	A bad actor may acquire Patient credentials and pose as Patient to the Website
Elevation of Privilege	Patient(2)	Doctor (3)	A bad actor may attempt to elevate their privilege from Patient to Doctor by using existing privilege exploits

[Doctor (User Class)]

Threat	Source	Target	Description
Spoofing	Doctor(3)	Application Website (4)	A bad actor may acquire Doctor credentials and pose as an actual Doctor to the Website
Info Disclosure	Doctor(3)	Patient(2)	A bad actor who gains Doctor credentials can access patients information and also provide wrong diagnosis

[Healthcare Application Website]

Threat	Source	Target	Description
Spoofing	Doctor(3), Patient (2), Analyst (1)	Application Website (4)	A bad actor may acquire another's credentials and pose as a that individual to the Website, gaining access to private data
Spoofing	Application Website	Doctor(3), Patient (2),	A bad actor may

	(4)	Analyst (1)	serve erroneous content from the backend to legitimate users on the front end
Info Disclosure	Application Website (4)	Application Database (5)	Network eavesdropping may expose sensitive information as it crosses the WebService/Database boundary

[Healthcare Application Database]

Threat	Source	Target	Description
Tampering	Doctor(3), Patient (2), Analyst (1)	Application Database (5)	A bad actor may attempt to tamper with data stored directly on the DB through various methods
Info Disclosure	Application Website (4)	Application Database (5)	Through network eavesdropping, a bad actor may access network traffic going between the application and the database

V.2 Prevention Measures

Feature	Feature Description	Implementation	STRIDE Threats Mitigated
Multi-Factor Authentication	Doctors must enter a 6-digit code generated by	Prompts for a 6-digit code from an authenticator app. Compares the	Spoofing, Elevation of

	an authenticator app during login.	generated code with the website's code for login authentication.	Privilege
Additional Password Safeguards	Doctors must confirm their password to edit a Patient profile.	Requires Doctor's password re-entry for editing a Patient's profile. Hashes and compares the re-entered password with the stored hashed password for validation.	Elevation of Privilege,Info Tampering
Encryption	Field level encryption for select fields.	Utilizes encrypted_model_fields library with a field encryption key for specific fields within necessary models. Ensures critical patient/doctor details remain encrypted.	Information Disclosure
Data Obfuscation	Patient profile information obscured in Doctor view by default.	Loads Patient profile with asterisks for all fields except 'Name', requires manual click to 'Show Information'. Enhanced security against information disclosure.	Information Disclosure
Rotating Logs	Logs relevant information(user,type of change,date/time) upon change/security event.	Using Django's RotatingFileHandler, configure change.log for user-related changes and security.log for security events. Rotate files at 10MB with 5 backups.	Spoofing, Information Disclosure
DDoS Attack Test	Denial of service attack python script to test brute force defense.	Uses Python Socket Programming to simulate requests from a spoofed IP. Django Framework denies repeated requests from the script, ensuring resistance against DoS attacks.	Denial of Service (DoS)

VI. Test Case Specifications and Results

VI.1 Testing Overview

The objective of the tests within this program are meant to provide reasonable belief to the expectation that our code will execute consistently and perform as designed to fulfill the functional requirements. The series of automated and manual tests will help to ensure that the application continues to function as intended as new builds are deployed and features are added. The continued maintenance of the testing suite is integral to ensuring consumer data is handled securely and efficiently, particularly throughout the User Acceptance Testing process. High-quality tests for each functional requirement allows for greater flexibility and scope development as time goes on.

The project being developed in Python allows for the utilization of popular existing testing frameworks, such as Selenium and Pylint. Unit testing is performed locally for each feature before integration testing. The unit tests are performed manually on a local, disconnected database using Sqlite3, the default database provided by Django, to ensure data integrity within the primary database. These tests must be performed manually due to the interconnected MVC architecture of the web application. These tests and their results are documented below. Integration testing is performed upon completion of an individual feature; the feature branch is merged to main where the suite of tests will be run to ensure continued functionality, with the addition of the new feature's tests.

The testing process delivers four major deliverables. The first is a set of Specification Based Tests implemented primarily using Selenium; these correspond to the project's functional requirements and ensure the continued fulfillment of all stated use cases. The second deliverable is a set of Code Based Tests, which include Unit Testing and Integration Testing processes and results. The third deliverable is the User Acceptance Testing unit, which contains instructions, resources, and avenues for the sponsor and client to test the product and provide feedback. The final deliverable is the Code Style Analysis, which includes an overview of coding styles and Pylint output.

Developer Testing Process

1. Developer Creates Branch – The developer will create a branch from main to work on the new feature.
2. Developer Writes Code – The developer will write code for either a website feature or security feature on the new branch. Code is complete when a given functional requirement is fulfilled.
3. Write Test Cases – For security features, the developer will write at least two test cases per functional requirement to document intentional passing and failing of the given requirement. Basic website functionality may be relegated to manual testing or integrated within a larger test case.
4. Run Tests – The developer will run the new tests designed in the previous step and observe output.

5. Fix Bugs – If the tests failed in the previous step, the developer will identify the source of the error and fix it.
6. Perform Unit Tests – The developer will walk through the Unit Testing procedure to ensure the website is functional from start to finish, including the newly developed feature.
7. Developer Commits & Pushes Code – The developer will commit and push their code to their feature-specific branch created in Step 1.
8. Merge Branch – The developer will address merge conflicts and merge the feature branch with main.
9. Re-run Automated Tests – Developer will re-run basic suite of automated tests after merge to ensure continued functionality.

VI.2 Specification Based Tests

The majority of the Specification Based Tests are performed using automated tests with Selenium, while some must be performed manually – the nature of each test is indicated in the results table. These tests execute many basic website functionalities as they test the functional requirements – as a result, there are no dedicated tests for individual links or buttons, as these are all covered within the automated and manual tests below.

The testing environment requires the website to be run in one terminal, while the testing program is run in a different terminal window. The Selenium tests are configured to use an auto-install Chromedriver, which requires the user to have Chrome installed on their machine to function properly; the version of Chrome used is not relevant. There are no specific hardware requirements.

Test Results

The following tests correspond to the Use Cases listed in Section III.2.

Assumption: There is at least one Doctor and one Patient within the database already.

Test ID	Test Case	Expected Output	Actual Output	Pass / Fail	Auto / Manual
1	Register as a Patient	User is logged in as new user	User is logged in as new user	P	M
2	Schedule Appointment	Appointment is added to dashboard	Appointment is added to dashboard	P	A

		of patient and doctor	of patient and doctor		
3	Edit Appointments	Information is saved on both Patient and Doctor appts	Information is saved on both Patient and Doctor appts	P	A
4	Add Diagnosis	Information is stored within patient's profile	Information is stored within patient's profile	P	A
5	Edit Profile (Patient)	Eligible fields are modified and saved to profile.	Eligible fields are modified and saved to profile.	P	M
6	Edit Profile (Doctor) - 1	Password is confirmed, eligible fields are modified and saved to profile.	Password is confirmed, eligible fields are modified and saved to profile.	P	A
7	Edit Profile (Doctor) - 2	Password is wrong, user is redirected to patient list.	Password is wrong, user is redirected to patient list.	P	M

VI.3 Code Based Tests

Unit Tests

All security features implemented after the development of the baseline web application underwent thorough Unit and Integration testing. The goal of these tests were to ensure the functionality of the component in isolation, and the functionality of the component within the larger system, respectively. These tests were performed manually due to the interconnected nature of the web application's MVC architecture. Passing denotes meeting the behavior defined in the functional requirements.

Unit Test ID	Feature	Developer	Date	Pass / Fail
1	Logging	Hemendra Kathi	11/13/23	P
2	Encryption	Hemendra Kathi	11/14/23	P
3	DOS Protection	Jacob Eckfeldt	11/15/23	P

4	Information Obscuring	Reed Havens	11/14/23	P
5	2FA	Alexander Shirk	11/25//23	P

Integration Tests

Integration tests were performed manually following the completion of a feature's Unit Test. The purpose of these tests was to ensure the cohesive functionality of all application components working in unison after verifying a feature's functionality in isolation. As a standard procedure, the developer responsible for the feature Unit Test is responsible for the subsequent Integration Test against main, the stable program featuring the comprehensive application. Passing is denoted by the successful running of all listed automated and manual tests.

In each Integration Test, the responsible developer creates a new Patient user and a new Doctor user to walk through all listed Use Cases to ensure continued fulfillment of functional requirements. This set of manual testing is performed in addition to the suite of automatic tests outlined in the Specification Based Tests.

Integration Test ID	Feature	Merge Branch	Feature Branch	Developer	Date	Pass / Fail
1	Logging	main	main	Hemendra Kathi	11/13/23	P
2	Encryption	main	main	Hemendra Kathi	11/14/23	P
3	DOS Protection	main	homeregister	Jacob Eckfeldt	11/16/23	P
4	Information Obscuring	main	databseEdit	Reed Havens	11/15/23	P
5	2FA	main	mfa_pyotp	Alexander Shirk	11/27/23	P

Code Coverage: >90%

Through the list of tests outlined in the Specification Based Tests section, we achieve nearly full code coverage throughout the entire program, discounting deprecated features. All active code is contained within two views.py functions, and within these functions, we interact with all models and forms created and stored within the database. The automated and manual testing suite, powered primarily by Selenium, visits all non-deprecated view functions.

Branch Coverage: >50%

The branch coverage of the Specification Based Tests is relatively low, at a minimum of 50%. This is due to the nature of the web application design. All view functions within the program begin with an if-statement if the given user class is permitted to view the page – these statements have all been tested, but they do not fall within the regular testing suite. This is because there is little to test on the alternative branch options more than once, particularly because we do not make modifications to a user's patient or doctor status at any time.

VI.4 User Acceptance Testing

The goal of User Acceptance Testing is to elicit direct feedback from users regarding the look, feel, and functionality of the web application. To this end, the web application has been prepared for portable deployment and testing. The testing strategy below outlines the required resources, instructions for eliciting feedback, and directions on how to incorporate the feedback. The primary tester utilizing this process has been the project sponsor, Jason Burt. This plan details the instructions on how to apply the acceptance testing on a more general set of users.

A) Resources
    a) A running build of the web application
    b) An authenticator application
    c) A Google document for user feedback
B) Instructions
    a) A given user will be chosen to demo the web application.
    b) The user will be provided with a running version of the website.
        i) Note: The website build should include a previously-populated database.
    c) The user will walk through all listed use cases with minimal instruction beyond description of task, beginning with registration.
    d) Upon completion of all tasks or when the user is done participating, a Google document will be provided to gather feedback.
C) Revision
    a) The team will meet to review feedback forms gathered from test users.
    b) The group will consider the merits of feedback and proposed changes.

c) The group will decide on which changes to pursue
    i) If a new feature or sizable modification is elected, then the documentation will be updated to reflect this change.

This process is intended to elicit subjective and qualitative feedback, primarily. The application should function comfortably as a stand-alone healthcare user portal, and the additional security measures should not be visible or incur any cost to the user experience. The User Acceptance Testing process will help to ensure these objectives are met.

VI.5 Code Style Analysis

The web application code is written entirely in Python, and as such, follows PEP8 style guidelines to the greatest extent possible. In this manner, the coding structure followed ensures:

- Consistent indentation standards followed
- Proper spacing before and after class and function declarations
- Limitation of whitespace
- Up-to-date comments
- Maximum line length of 79 chars
- Concise import statements
- Many others

The goal of following the PEP8 guidelines is to maximize the readability of the project code. This ensures that future work may be done on the project effectively; additionally, it ensures that sponsors and stakeholders can readily understand the product at hand. Throughout all code-writing, developers followed PEP8 guidelines to the highest level possible. The primary exception within the program is the use of snake case variable naming, which the team used interchangeably with camel case naming.

Linter – Pylint

The project attempted to utilize Python linters, primarily Pylint, to assist in the enforcement of PEP8 coding standards, but this found mixed success. For reasons that could not be clarified, linters consistently marked database relations as syntactically invalid and nonexistent. For example, an attribute such as “Appointment.staffUser.fname” results in an error that ‘fname’ is not a member of the given staffUser Foreign Key. However, this is incorrect, as the Specification Based Tests demonstrate the validity of the database schema and its implementation within the program. This roadblock prevented the team from fully utilizing Pylint and other linters and presents a meaningful avenue for project improvement going forward.

VII. Projects and Tools Used

Tool/Library/Framework	Purpose
Django	Django is an open-source, Python-based web framework that was used to develop basic website functionality and process user data and requests. It also provides built-in authentication methods.
PyOTP	PyOTP is a one-time-password authentication library for Python that allows the usage of third-party authenticator apps within the MFA authentication process.
Selenium	Automated testing framework for use in various programming languages – Python-specific implementation was used in this application.

Languages Used			
Python	CSS	HTML	

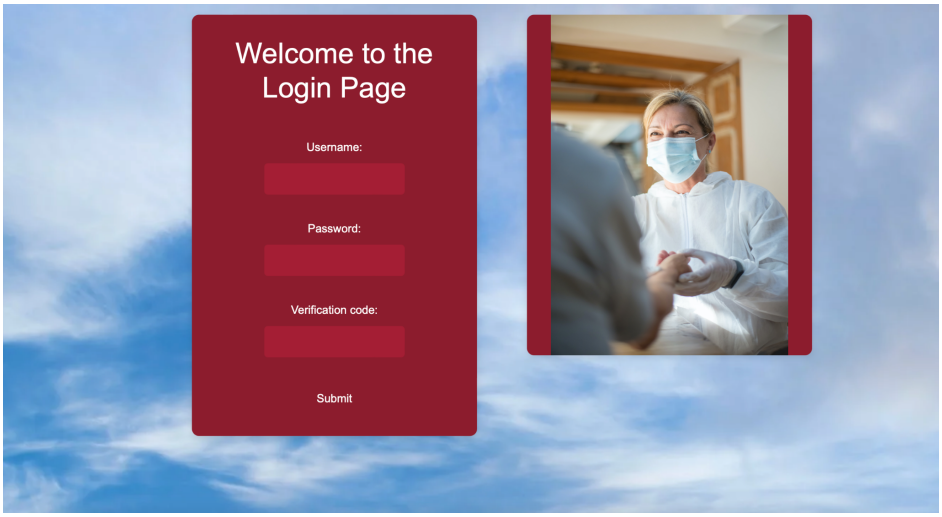
VIII. Description of Final Prototype

VIII.1 Final Prototype Summary

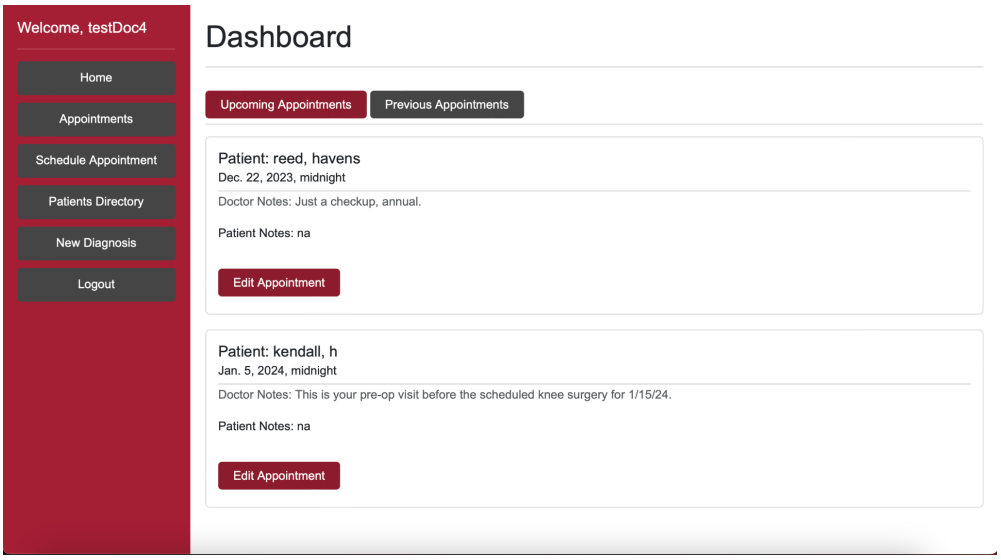
The Healthcare DLP Web Application is a standalone healthcare user portal that simulates an end-user experience for both patients and doctors, with a focus on the protection of sensitive user data transmitted and stored throughout the standard operations of the user portal. To this end, the DLP Web Application offers extensive functionality to both Doctors and Patients, allowing:

- Doctors to:
    - Securely login with 2FA
    - Schedule and edit appointments
    - Add patients to their personal active registry
    - Add or edit Patient diagnoses
    - Securely view and edit select fields of a Patient profile
- Patients to:
    - Securely login with 2FA
    - Schedule and edit appointments
    - Review past and future appointment history
    - Edit select fields of personal profile

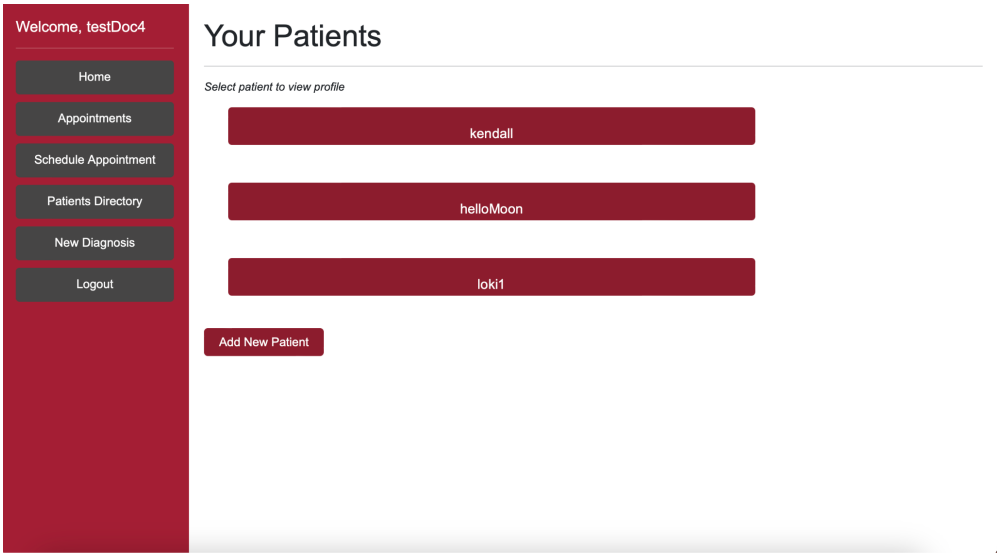
This summary will primarily cover the Doctor perspective, as all Patient functionality is available to the Doctor class, while the reverse is untrue. For any user, the first screen encountered following registration will be the Login screen, where users will enter their chosen Username and Password. Next, the user will enter the 6-digit time-based one-time-passcode (TOTP) from their chosen authenticator app to login.



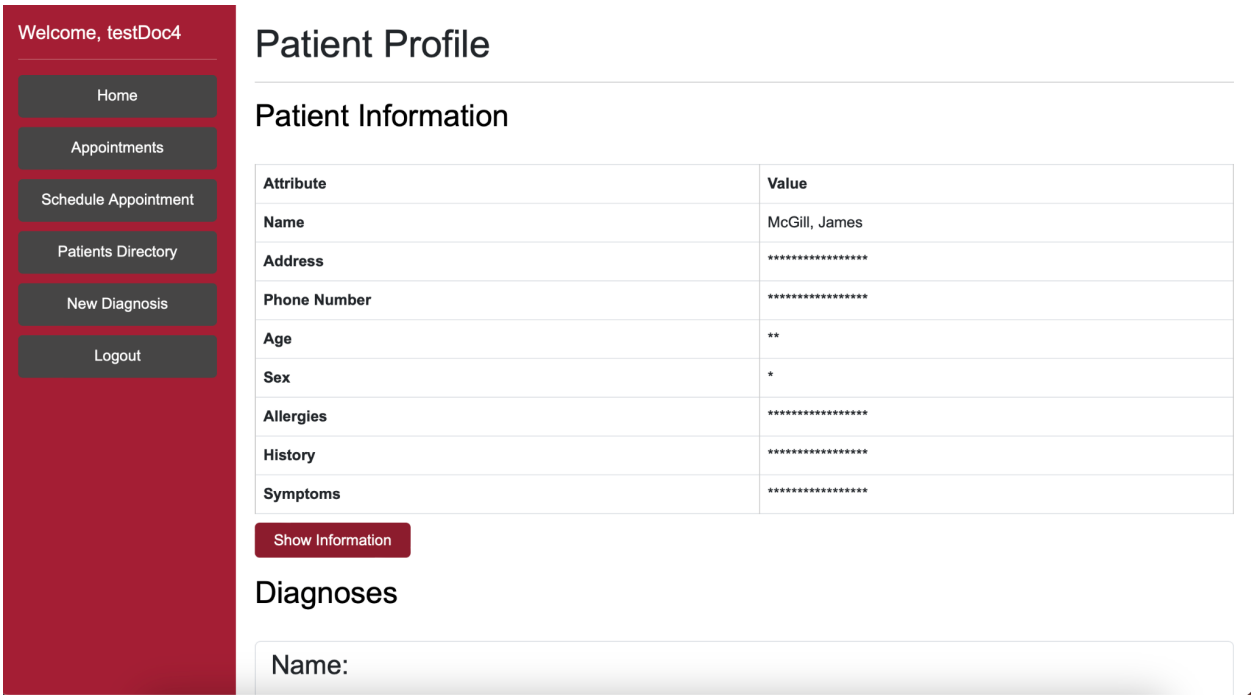
[Doctor]: Following login, from the Doctor's perspective, they are presented with a dashboard page that allows them to manage their suite of Patients. From here, they can review immediate upcoming appointments, or they can navigate to another page from the sidebar.



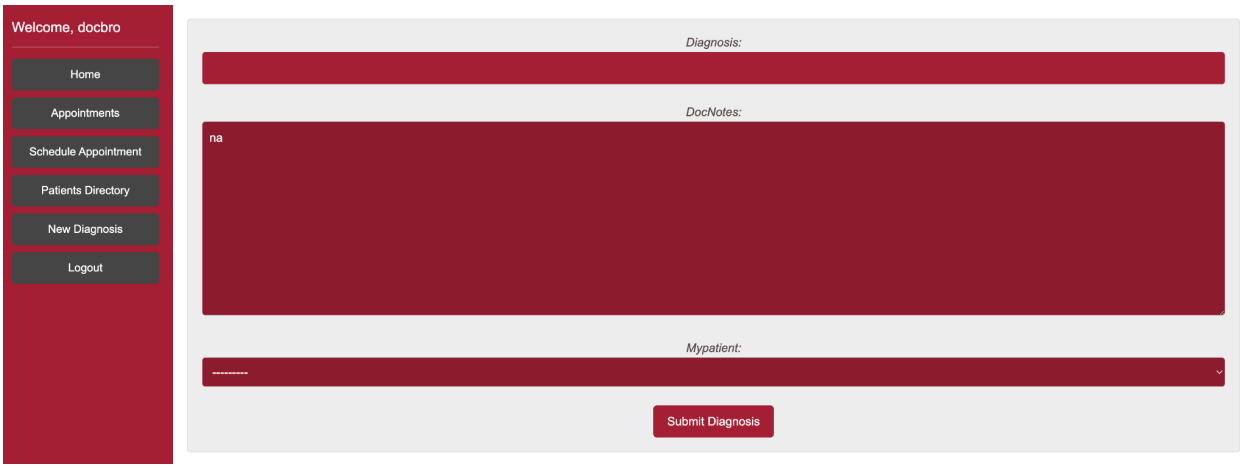
[Doctor]: First, the Doctor will first need to add Patients to their registry in order to interact with them. To do this, they navigate to “Patients Directory” and select “Add Patient” to select a Patient from the list of all existing, un-assigned Patients. Or, they may choose to view a Patient profile from this list of existing Patients, if any are already assigned.



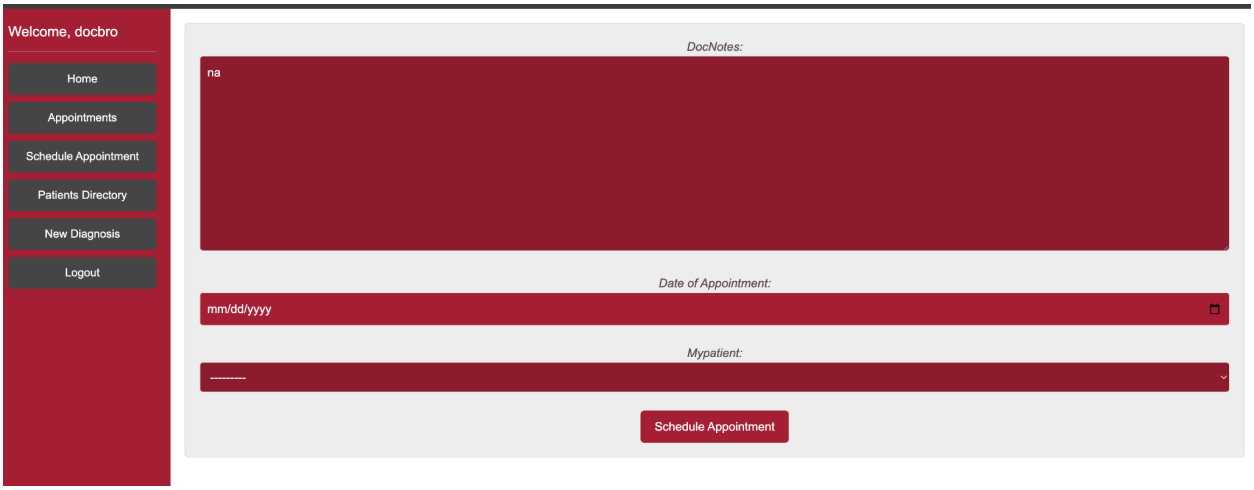
[Doctor]: Here, the Doctor may select to view a Patient profile, where they may view and edit select information. When viewing a Patient profile, the view will be as pictured below, where all fields aside from first and last name are obfuscated by default. The Doctor may select “Show Information” to reveal the hidden text temporarily.



With a Patient assigned, the Doctor may choose to add a New Diagnosis. They will select the New Diagnosis option and find the screen pictured below. Here, they may enter the name of the Diagnosis, the description, and the associated Patient. The information will be saved to the Patient profile.

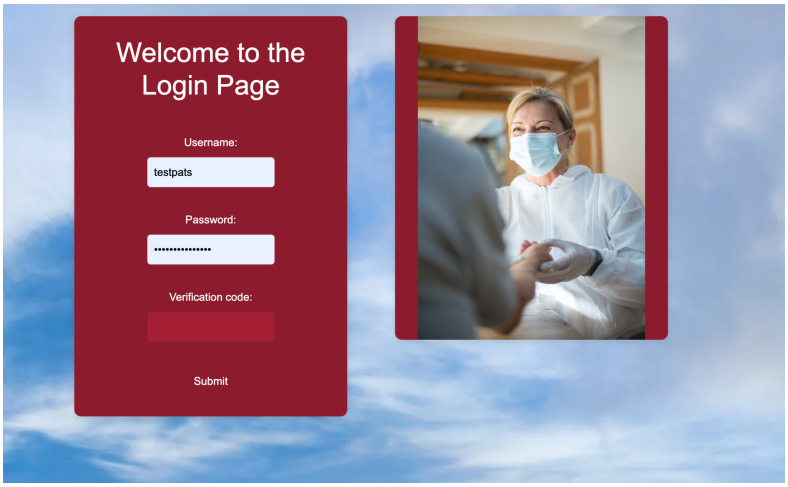


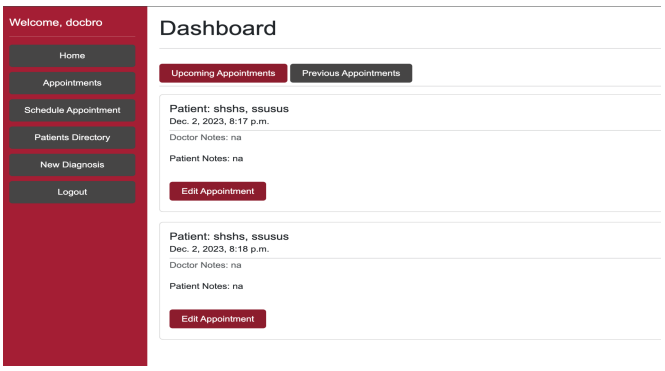
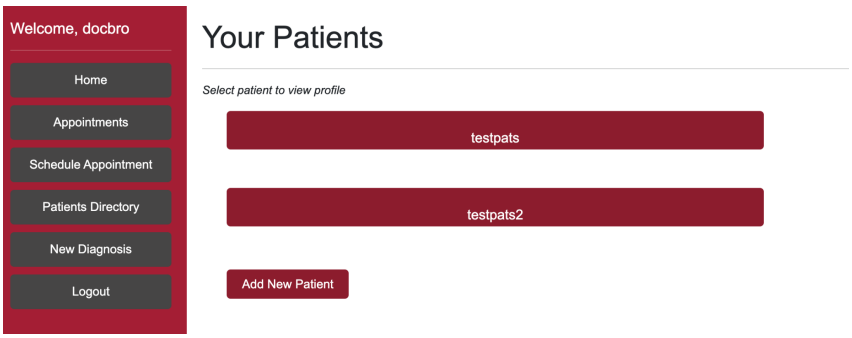
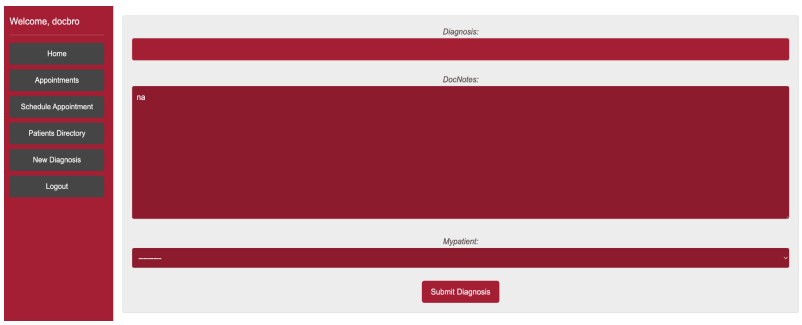
Now, they may wish to schedule an appointment with this Patient to discuss the new diagnosis. The Doctor will select “Schedule Appointment” from the sidebar and find the screen pictured below, where they will enter the date of the appointment, notes for the Patient’s reading, and the Patient’s name.

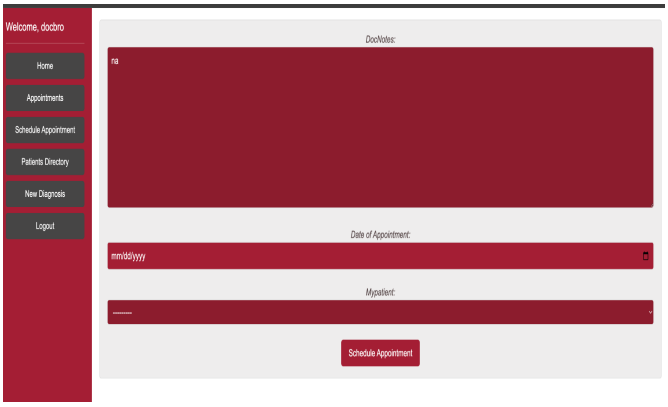


Now, the Doctor has added a new Patient, viewed the information on their profile, added a new Diagnosis to the Patient, and then Scheduled an Appointment about this diagnosis. This walkthrough provides an overview of the main user functionality of the final prototype.

VIII.2 Major Use Cases

Step ID	Step Description	User Interface
1	User logs in with 2FA using a third-party authenticator app.	

2	User views all upcoming appointments.	
3	User Adds Patient to Patient List	
4	User Adds New Diagnosis to New Patient	

5	User Schedules New Appointment	
6	User Edits Profile	

IX. Project Delivery Status

The final product is as a locally hosted website, run through a Django framework in a Python virtual environment. This website is available to download from the project's GitHub repository.

Product Location Information

- The latest release of the project can be found at the team's GitHub repository:

    - https://github.com/JEckfeldt/HealthCareDLP
- The main branch of this repository will always contain a stable version of the website.
- The main branch utilizes the default sqlite3 database to ensure maximum portability and deployability. The PostgreSQL database-equipped version of the website is available on the branch main_with_postgres.

Installation and Set-Up Instructions

1. Download source code and open up a terminal
2. Setup an environment for the project
    a. "py -m venv project-name" will create a folder project name with environment info (myvenv or venv suggested, as they are already in the git ignore)
    b. Activate the environment go to project-name/Scripts and run activate or activate.bat
    c. You should see (project-name) on the far left of your terminal
    d. Make sure you run activate every time you open a new terminal to work inside your env (You can run into version control issues if not)
3. Install the python reqs
    a. Make sure you are in the same dir as requirements.txt
    b. Run "pip install -r requirements.txt"
    c. This gives you all the packages used for the project

Prerequisites

1. All primary prerequisite requirements can be fulfilled by installing requirements.txt.
    a. Make sure you are in the same dir as requirements.txt
    b. Run "pip install -r requirements.txt"
    c. This gives you all the packages used for the project
2. The website and framework are compatible with any browser.
3. The Selenium tests require Google Chrome, but are version-independent of the browser.
4. To properly run main_with_postgres, download and install any version of PgAdmin4 or other PostgreSQL manager. The database migrations will not function properly if this application is not locally installed.

X. Conclusions and Future Work

X.1 Limitations and Recommendations

Time and Scope

The primary limitations experienced in the development of this product were time and scope. The standard development window for a project of this type is two academic semesters, which equates to approximately eight months for the project lifecycle. Due to external reasons beyond the development team's control, the original project sponsor had to exit the project between semesters. This resulted in a hard restart to the project lifecycle, allowing for only a four month planning and development window. Due to this limitation in time, certain drawbacks in scope were made. These are discussed in the following section on Future Work.

X.2 Future Work

Website Hosting

Currently, the website is hosted locally. Given the limited time and monetary resources available to the team, the website was unable to be deployed externally. However, external deployment would enhance the utility and applicability of the project. It would allow the website to utilize external services such as Cloudflare, to allow for enhanced DDoS protections, cloud cybersecurity, and website diagnostic information.

2FA Secret Complexity

For baseline testing purposes, we use a constant secret for our OTP-2FA implementation. This is a major security risk in a real world application and should be addressed. To enhance complexity, we may generate a base secret randomly at the time of user generation; this secret will be supplied to the user, and they will seed their authenticator app with this secret. On login, the website and the authenticator app will use the same unique secret associated with the given user. This will enhance the complexity and security of the 2FA implementation, and bring the website closer in line with real-world analogs.

Capture of Additional Metrics

Currently, we only analyze the impact of our select field encryption. Here, we see a 6-14ms increase in server response times as compared to when we do not encrypt any fields. It would enhance the quality and impact of this project were additional metrics to be analyzed. For example, analysis may be performed on the impact of logging website events, as well as on the performance of the website amidst a DoS attack. These insights may help to inform further feature development or refinement.

Threat Detection and Response

The website's logging implementation captures all events on the website, and it relies on Django's provided threat detection response capabilities. It would improve the final product to develop and test a custom threat detection response system. This would require defining what an adverse log event may be, as well as the appropriate response to that event. This would involve the definition of multiple threat classes and response types. Such a system would bring the final application closer inline with real-world security systems, and would allow for further development of data loss prevention strategies.

Enhanced Testing Capabilities

Due to the reduction in time and scope, certain compromises were made in the scale of testing deployed throughout this project. It would benefit future project developments if additional automated tests were implemented, particularly to further segment tests into singular use cases. The current limitation in this regard is the generation of the one-time code, forcing manual entry of the code at the beginning of the test and allowing only one test to be run at once. However, this is more likely to be a logic error than a programming limitation, and thus can likely be improved upon.

XI. Acknowledgements

First, We would like to thank our sponsor Jason Burt working at Google for his constant guidance and mentorship throughout this project.We also express our heartfelt thanks to Professor Subu Kandaswamy for his continual support and invaluable assistance in streamlining the project's development process, culminating in a successful prototype.Lastly, we wish to acknowledge our esteemed guest mentors, Chad Thunberg

from Yubiko and Ben Ransford from Stripe, whose expertise in cybersecurity has been instrumental in shaping our project and acquainting us with essential industry practices.

XII. Glossary

2FA: Two-Factor Authentication - A security process that requires two different authentication methods for access.

DDoS: Distributed Denial of Service - A cyberattack aiming to disrupt online services by overwhelming them with traffic.

Encryption: Conversion of data into a code to prevent unauthorized access.

STRIDE: A threat modeling framework representing six types of security threats: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege.

DLP: Data Loss Prevention - Strategies and tools to protect sensitive data from unauthorized access, sharing, or theft.

HTTP: Hypertext Transfer Protocol - Protocol for transmitting data across the internet.
Obfuscation: Technique used to make code or data difficult to understand or interpret.

Selenium: A web browser automation tool primarily used for testing purposes.

MVC: Model-View-Controller - A software architectural pattern separating an application into three interconnected components.
HIPAA: Health Insurance Portability and Accountability Act - Legislation ensuring data privacy and security for protected health information (PHI) in the healthcare industry.

Django: A high-level Python web framework used for rapid development of secure web applications.

pyOTP: A Python library for generating one-time passwords used in two-factor authentication.

pyLint: A Python static code analysis tool used to identify errors and maintain code quality.

Python: A high-level programming language known for its simplicity and readability.

HTML: Hypertext Markup Language - Standard language for creating web pages.

CSS: Cascading Style Sheets - A style sheet language used for describing the presentation of a document written in HTML.

XIII. References

"Data Loss Prevention." *Imperva.Com*, www.imperva.com/learn/data-security/data-loss-prevention-dlp/. Accessed 8 Dec. 2023.

"Django." *Django Project*, www.djangoproject.com/. Accessed 8 Dec. 2023.

"Python Client Library | Google Cloud." *Google*, Google, cloud.google.com/python/docs/reference/dlp/latest. Accessed 8 Dec. 2023.

"Selenium with Python¶." *Selenium with Python - Selenium Python Bindings 2 Documentation*, selenium-python.readthedocs.io/. Accessed 8 Dec. 2023.

Singh, Vanshika, et al. "Data Leakage Detection and Prevention Using Cloud Computing." *SpringerLink*, Springer International Publishing, 1 Jan. 1970, link.springer.com/chapter/10.1007/978-3-031-13577-4_9.

"What Is Data Loss Prevention (DLP) and Why Is It Important?" *Securiti*, 4 Dec. 2023, securiti.ai/what-is-data-loss-prevention-dlp/.

"What Is Data Loss Prevention (DLP)?" *Microsoft Security*, www.microsoft.com/en-us/security/business/security-101/what-is-data-loss-prevention- lp. Accessed 8 Dec. 2023.

Wright, Drew. "A Technical Analysis of the Capital One Cloud Misconfiguration Breach." *Fugue*, www.fugue.co/blog/a-technical-analysis-of-the-capital-one-cloud-misconfiguration-breach. Accessed 8 Dec. 2023.

XIV. Appendix A - Project Management

The team's weekly schedule included meetings with the team to discuss what items were to be worked on that week, what was expected in the following week, and what blockers the team was experiencing, if any. Every Friday, the team met with the client, Jason Burt, to provide an update on the status of the project – these updates included items such as updates to the threat model, development of database design, and development of the prototype application. These meetings allowed the client to provide direct feedback to the status of major and minor deliverables. Additionally, the team met with their mentor, Subu Kandaswamy, on a monthly basis to check in on the progress of the project and discuss any concerns faced by the team. All communications were supplemented by frequent email correspondence between the team, client, and mentor throughout the project lifecycle.

Team Meetings

Each week, a minimum of 3 of the 4 core team members would meet via Discord to discuss the status of the project. The team reviewed what work was in development for the given week, what deliverables were expected at the upcoming client meeting, and what blockers were being experienced. These meetings served to keep the team informed on the progress of key deliverables, as well as to maintain strong understanding of project scope and requirements.

Client Meetings

Each Friday, the team would meet with the client, Jason Burt, via Google Meet for approximately 30 minutes. In these meetings, team members would review what they worked on in the previous week, as well as any issues or concerns they encountered. The client is provided with updates to the project and prototype, and direct feedback on these updates is received. Jason also brought in several key individuals to help inform our discussions and answer specific questions – Chad Thunberg from Yubiko and Ben Ransford from Stripe both met with the team on separate occasions to share feedback and insights.

Mentor Meetings

Each month, the team met with mentor Subu Kandaswamy. These meetings alternated between high-level status updates and in-depth code reviews, each occurring approximately three weeks apart, with one meeting of each type per sprint. Here, statuses of key deliverables were shared and recommendations were made on how to improve or expand certain areas, as well as on overall project guidance.