Proposed srs

Functional requirements:

2.1 User Authentication

- Description: Users must be able to create accounts and log in securely.
- Solution: Implement user authentication using a secure login system with password encryption.

2.2 Data Input

- Description: Users should be able to input their medical data, including symptoms, medical history, and test results.
- Solution: Provide a user-friendly interface for users to input their data. Implement data validation to ensure the accuracy and completeness of the entered information.

2.3 Disease Prediction

- Description: The system should analyze the input medical data and predict the presence of heart, kidney, cancer, liver, and diabetes diseases.
- Solution: Utilize machine learning algorithms to analyze the medical data and predict the likelihood of each disease.

2.4 Report Generation

- Description: The system should generate a comprehensive report based on the analysis of medical data.
- Solution: Develop a report generation module that compiles the analysis results into a user-friendly format.

3. Non-Functional Requirements

3.1 Performance

- Requirement: The system should be able to process medical data and generate reports quickly.
- Solution: Optimize algorithms and database queries to ensure fast performance. Conduct performance testing to identify and address any bottlenecks.
- System Specification:
    - Processor: Intel Core i5 or equivalent
    - RAM: 8GB or higher
    - Storage: 100GB SSD or higher
    - Operating System: Windows 10 or Ubuntu 20.04 LTS
    - Web Server: Apache Tomcat 9.0 or equivalent
    - Database: PostgreSQL 12.0 or equivalent

3.2 Security

- Requirement: The system should comply with data protection regulations and ensure the security of user data.
- Solution: Implement data encryption, secure authentication, and access control mechanisms to protect user data. Regularly update security measures to address emerging threats.
- System Specification:
    - Use SSL encryption for data transmission
    - Implement role-based access control (RBAC) to restrict access to sensitive data
    - Regularly update system components to patch security vulnerabilities

3.3 Reliability

- Requirement: The system should be reliable and available 24/7.
- Solution: Implement redundancy and failover mechanisms to ensure high availability. Regularly monitor system performance and address any issues promptly.
- System Specification:
    - Implement automated backup and restore procedures
    - Use redundant servers with load balancing
    - Implement monitoring tools for proactive issue detection

3.4 Scalability

- Requirement: The system should be able to handle a large number of users and medical data.
- Solution: Design the system using scalable architecture and cloud infrastructure. Implement load balancing and auto-scaling to handle increased demand.
- System Specification:
    - Use cloud infrastructure such as AWS or Azure for scalability
    - Implement microservices architecture for easier scalability
    - Use containerization (e.g., Docker) for easy deployment and scaling

3.5 Usability

- Requirement: The system should be user-friendly and easy to navigate.
- Solution: Conduct usability testing to gather feedback from users and improve the user interface. Provide clear instructions and tooltips to guide users through the application.
- System Specification:
    - Implement responsive design for mobile and desktop devices
    - Provide tooltips and help sections for user guidance
    - Conduct user testing to gather feedback for continuous improvement

3.6 Compatibility

- Requirement: The system should be compatible with different web browsers and devices.
- Solution: Ensure cross-browser and cross-device compatibility during development. Test the application on various browsers and devices to identify and address any compatibility issues.
- System Specification:
    - Test compatibility with latest versions of Chrome, Firefox, Safari, and Edge
    - Ensure compatibility with major mobile browsers (Chrome, Safari)

Ensuring security in our website (future approach)

1. Encryption:

- Data in Transit: Use SSL/TLS encryption to secure data transmitted between the user's browser and your server. This prevents eavesdroppers from intercepting sensitive information.
- Data at Rest: Encrypt sensitive data stored in the database to prevent unauthorized access. Use strong encryption algorithms such as AES (Advanced Encryption Standard) with secure key management practices.

2. User Authentication and Authorization:

- Implement a secure user authentication system to verify the identity of users before granting access to sensitive data.
- Use strong password hashing algorithms (e.g., bcrypt) to securely store user passwords.
- Implement role-based access control (RBAC) to ensure that users only have access to the data and functionality that they are authorized to use.

3. Secure Data Storage:

- Follow secure coding practices to protect against common vulnerabilities such as SQL injection and XSS (Cross-Site Scripting) attacks.
- Use parameterized queries or prepared statements to prevent SQL injection attacks.
- Sanitize user input to prevent XSS attacks.

4. Regular Security Audits and Penetration Testing:

- Conduct regular security audits to identify and address potential security vulnerabilities in your system.
- Perform penetration testing to simulate real-world attacks and identify potential security weaknesses.

5. Data Masking and Anonymization:

- Implement data masking and anonymization techniques to protect sensitive user data.
- Mask or anonymize personally identifiable information (PII) such as names, addresses, and social security numbers before storing or transmitting data.

6. Logging and Monitoring:

- Implement logging and monitoring mechanisms to track and audit user activity within the system.
- Monitor for unusual or suspicious activity that could indicate a security breach.

7. Regular Software Updates and Patch Management:

- Keep your software and systems up to date with the latest security patches and updates.
- Regularly update third-party libraries and dependencies to address known security vulnerabilities.

8. Secure Communication Channels:

- Use secure communication channels for transmitting sensitive information, such as credit card details or health records.
- Avoid sending sensitive information via email or other insecure communication channels.

9. Compliance with Data Protection Regulations:

- Ensure that your system complies with relevant data protection regulations, such as GDPR (General Data Protection Regulation) or HIPAA (Health Insurance Portability and Accountability Act), depending on your jurisdiction and the type of data you handle.

Unit testing:

Test case: if the index page loads successfully

```
class TestHTMLContent(unittest.TestCase):
    def test_index_page(self):
        """
        Test if the index page loads successfully and contains the expected content
        """
        response = self.app.get('/')
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Comprehensive Health Prognosticator', response.data)
        self.assertIn(b'Machine learning based Health Prognosticator', response.data)
        self.assertIn(b'This website is built with the motive to predict various disease based on symptoms and other factors.', response.
        self.assertIn(b'Model Accuracy', response.data)
        self.assertIn(b'Diabetes : 97%', response.data)
        self.assertIn(b'Heart : 83.61%', response.data)
        self.assertIn(b'Kidney : 100%', response.data)
        self.assertIn(b'Liver: 78%', response.data)
        self.assertIn(b'Cancer : 94.74%', response.data)
        self.assertIn(b'Disease our webApp can predict', response.data)
        self.assertIn(b'Diabetes', response.data)
        self.assertIn(b'Heart Disease', response.data)
        self.assertIn(b'Liver Disease', response.data)
        self.assertIn(b'Kidney Disease', response.data)
        self.assertIn(b'Cancer', response.data)
        self.assertIn(b'Developed by Sujay, Manoj, Suresh', response.data)
```

Output:

```
----------------------------------------------------------------------
Ran 1 test in 0.021s

OK
```

Test case: if cancer page loads successfully

```
class TestCancerPage(unittest.TestCase):
    def test_cancer_page(self):
        """
        Test if the cancer page loads successfully and contains the expected content
        """
        response = self.app.get('/cancer')
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Cancer Prediction', response.data)
        self.assertIn(b'Radius Mean (5-30)', response.data)
        self.assertIn(b'Texture Mean (5-45)', response.data)
        self.assertIn(b'Perimeter Mean (40-200)', response.data)
        self.assertIn(b'Area Mean (100-2600)', response.data)
        self.assertIn(b'Smoothness Mean (0-0.2)', response.data)
        self.assertIn(b'Compactness Mean (0-0.5)', response.data)
        self.assertIn(b'Concavity Mean (0-1)', response.data)
        self.assertIn(b'Concave Points Mean (0-0.5)', response.data)
        self.assertIn(b'Symmetry Mean (0-0.5)', response.data)
        self.assertIn(b'Fractal Dimension Mean (0-0.1)', response.data)
        self.assertIn(b'Radius SE (0-5)', response.data)
        self.assertIn(b'Perimeter SE (0.5-30)', response.data)
        self.assertIn(b'Area SE (5-600)', response.data)
        self.assertIn(b'Smoothness SE (0-0.1)', response.data)
        self.assertIn(b'Compactness SE (0-0.5)', response.data)
        self.assertIn(b'Concavity SE (0-0.5)', response.data)
```

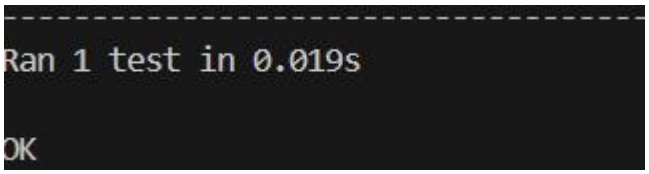
Output:

Test case: if diabetes page loads successfully

```
class TestDiabetesPage(unittest.TestCase):

    def test_diabetes_page(self):
        """
        Test if the diabetes page loads successfully and contains the expected content
        """
        response = self.app.get('/diabetes')
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Diabetes Prediction', response.data)
        self.assertIn(b'Select number of pregnancies', response.data)
        self.assertIn(b'Glucose (mg/dL) eg. 80', response.data)
        self.assertIn(b'Blood Pressure (mmHg) eg. 80', response.data)
        self.assertIn(b'Skin Thickness (mm) eg. 20', response.data)
        self.assertIn(b'Insulin Level (IU/mL) eg. 80', response.data)
        self.assertIn(b'Body Mass Index (kg/m\xc2\xb2) eg. 23.1', response.data)
        self.assertIn(b'Diabetes Pedigree Function eg. 0.52', response.data)
        self.assertIn(b'Age (years) eg. 34', response.data)
        self.assertIn(b'Developed by sujay,manoj,suresh', response.data)
```

output:



Testing heart page:

```
class TestHeartPage(unittest.TestCase):
    def test_heart_page(self):
        """
        Test if the heart disease prediction page loads successfully and contains the expected content
        """
        response = self.app.get('/heart')
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Heart Disease Prediction', response.data)
        self.assertIn(b'Age eg: 37', response.data)
        self.assertIn(b'Sex', response.data)
        self.assertIn(b'Chest pain type', response.data)
        self.assertIn(b'Resting blood pressure in mm Hg eg: 130', response.data)
        self.assertIn(b'Serum cholestoral in mg/dl eg: 250', response.data)
        self.assertIn(b'Fasting blood sugar 120 mg/dl', response.data)
        self.assertIn(b'Resting electrocardiographic results', response.data)
        self.assertIn(b'Maximum heart rate achieved eg: 187', response.data)
        self.assertIn(b'Exercise induced angina', response.data)
        self.assertIn(b'ST depression induced by exercise relative to rest eg: 3.5', response.data)
        self.assertIn(b'The slope of the peak exercise ST segment', response.data)
        self.assertIn(b'Number of major vessels colored by flourosopy', response.data)
        self.assertIn(b'Thal', response.data)
        self.assertIn(b'Developed by Sujay, Manoj, Suresh', response.data)
```

output:

```
------------------------------
Ran 1 test in 0.020s

OK
```

Test case: testing kidney page

```
class TestKidneyPage(unittest.TestCase):
    def test_kidney_page(self):
        response = self.app.get('/kidney')
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Kidney Disease Prediction', response.data)
        self.assertIn(b'Age (years) [2, 90]', response.data)
        self.assertIn(b'Blood Pressure (mm/Hg) [50, 200]', response.data)
        self.assertIn(b'Albumin (g/dL) [0, 10]', response.data)
        self.assertIn(b'Sugar (g/dL) [0, 10]', response.data)
        self.assertIn(b'Red Blood Cells (million cells/mcL) [4, 7]', response.data)
        self.assertIn(b'Pus Cell (cells/cubic millimeter) [0, 5]', response.data)
        self.assertIn(b'Pus Cell Clumps (0: Not present, 1: Present)', response.data)
        self.assertIn(b'Bacteria in Urine (0: Not present, 1: Present)', response.data)
        self.assertIn(b'Blood Glucose Random (mg/dL) [20, 500]', response.data)
        self.assertIn(b'Blood Urea (mg/dL) [1, 400]', response.data)
        self.assertIn(b'Serum Creatinine (mg/dL) [0, 100]', response.data)
        self.assertIn(b'Potassium (mEq/L) [1, 100]', response.data)
        self.assertIn(b'White Blood Cell Count (cells/cu mm) [1000, 30000]', response.data)
        self.assertIn(b'Hypertension: No', response.data)
        self.assertIn(b'Diabetes Mellitus: No', response.data)
        self.assertIn(b'Coronary Artery Disease: No', response.data)
        self.assertIn(b'Pedal Edema: No', response.data)
        self.assertIn(b'Anemia: No', response.data)
        self.assertIn(b'Predict', response.data)
        self.assertIn(b'Developed by Sujay, Manoj, Suresh', response.data)
```

output:

```
.
------------------------------------
Ran 1 test in 0.019s

OK
```

Testing liver page

```
class TestLiverPage(unittest.TestCase):

    def setUp(self):
        self.app = app.test_client()

    def test_liver_page(self):
        """
        Test if the liver disease prediction page loads successfully and contains the expected content
        """
        response = self.app.get('/liver')
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Liver Disease Prediction', response.data)
        self.assertIn(b'Age (years) eg: 62', response.data)
        self.assertIn(b'Total Bilirubin (mg/dL) eg: 7.3', response.data)
        self.assertIn(b'Direct Bilirubin (mg/dL) eg: 4.1', response.data)
        self.assertIn(b'Alkaline Phosphotase (IU/L) eg: 490', response.data)
        self.assertIn(b'Alamine Aminotransferase (IU/L) eg: 60', response.data)
        self.assertIn(b'Aspartate Aminotransferase (IU/L) eg: 68', response.data)
        self.assertIn(b'Total Protiens (g/dL) eg: 7.0', response.data)
        self.assertIn(b'Albumin (g/dL) eg: 3.3', response.data)
        self.assertIn(b'Albumin and Globulin Ratio eg: 0.89', response.data)
        self.assertIn(b'Select Gender', response.data)
        self.assertIn(b'Male', response.data)
        self.assertIn(b'Female', response.data)
        self.assertIn(b'Predict', response.data)
        self.assertIn(b'Developed by Sujay, Manoj, Suresh', response.data)
```

Output:

```
----------------------------------------
Ran 1 test in 0.019s

OK
```

Testing predict.html:

```
class TestPredictHTML(unittest.TestCase):
    def test_predict_healthy(self):
        app = Flask(__name__)
        with app.test_request_context():
            prediction = 0  # Assuming prediction result for healthy
            with patch('flask.templating._render') as mock_render:
                with patch('flask.helpers.url_for') as mock_url_for:
                    mock_url_for.return_value = '/'
                    render_template_string(
                        """
                        <!DOCTYPE html>
                        <html lang="en">
                        <head>
                            <meta charset="UTF-8">
                            <meta http-equiv="X-UA-Compatible"
content="IE=edge">
                            <meta name="viewport"
content="width=device-width, initial-scale=1.0">
```

```
                            <title>Result</title>
                            <!-- Jquery CDN -->
                            <script
src="https://code.jquery.com/jquery-2.2.4.js"

integrity="sha256-iT6Q9iMJYuQiMWNd9lDyBUStIq/8PuOW33aOqmvFpqI="
crossorigin="anonymous"></script>

                            <!-- Font Awesome Link -->
                            <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.2/css/al
l.min.css">
                            <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-aweso
me.min.css">
                            <!-- CSS -->
                            <link rel="stylesheet" href="{{
url_for('static', filename='style.css') }}">
                        </head>
                        <body class="result_main">
                            <h3 class="result_heading">Result</h3>
                            <div class="result_row">
                                {% if prediction==1 %}
                                <div class="suffer">Apologies, it
appears you're unwell. Please see a doctor for proper care and
assistance. Your health is important..</div>
                                {% elif prediction==0 %}
                                <div class="not_suffer">GREAT!! You are
healthy.</div>
                                {% endif %}
                            </div>
                            <div class="result_home">
                                <div class="return"><a href="{{
url_for('index') }}">Home</a></div>
                            </div>
                        </body>
                        </html>
                        """,
                        prediction=prediction
                    )
                    mock_render.assert_called_once()

    def test_predict_unhealthy(self):
```

```
        app = Flask(__name__)
        with app.test_request_context():
            prediction = 1  # Assuming prediction result for unhealthy
            with patch('flask.templating._render') as mock_render:
                with patch('flask.helpers.url_for') as mock_url_for:
                    mock_url_for.return_value = '/'
                    render_template_string(
                        """
                        <!DOCTYPE html>
                        <html lang="en">
                        <head>
                            <meta charset="UTF-8">
                            <meta http-equiv="X-UA-Compatible"
content="IE=edge">
                            <meta name="viewport"
content="width=device-width, initial-scale=1.0">
                            <title>Result</title>
                            <!-- Jquery CDN -->
                            <script
src="https://code.jquery.com/jquery-2.2.4.js"

integrity="sha256-iT6Q9iMJYuQiMWNd9lDyBUStIq/8PuOW33aOqmvFpqI="
crossorigin="anonymous"></script>

                            <!-- Font Awesome Link -->
                            <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.2/css/al
l.min.css">
                            <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-aweso
me.min.css">
                            <!-- CSS -->
                            <link rel="stylesheet" href="{{
url_for('static', filename='style.css') }}">
                        </head>
                        <body class="result_main">
                            <h3 class="result_heading">Result</h3>
                            <div class="result_row">
                                {% if prediction==1 %}
                                <div class="suffer">Apologies, it
appears you're unwell. Please see a doctor for proper care and
assistance. Your health is important..</div>
                                {% elif prediction==0 %}
```

```
                                <div class="not_suffer">GREAT!! You are
healthy.</div>
                                {% endif %}
                            </div>
                            <div class="result_home">
                                <div class="return"><a href="{{
url_for('index') }}">Home</a></div>
                            </div>
                        </body>
                        </html>
                        """,
                        prediction=prediction
                    )
                    mock_render.assert_called_once()

    def test_predict_url_for_error(self):
        app = Flask(__name__)
        with app.test_request_context():
            prediction = 1  # Assuming prediction result for unhealthy
            with patch('flask.templating._render') as mock_render:
                with patch('flask.helpers.url_for') as mock_url_for:
                    mock_url_for.side_effect = Exception("Error
occurred in url_for")
                    render_template_string(
                        """
                        <!DOCTYPE html>
                        <html lang="en">
                        <head>
                            <meta charset="UTF-8">
                            <meta http-equiv="X-UA-Compatible"
content="IE=edge">
                            <meta name="viewport"
content="width=device-width, initial-scale=1.0">
                            <title>Result</title>
                            <!-- Jquery CDN -->
                            <script
src="https://code.jquery.com/jquery-2.2.4.js"

integrity="sha256-iT6Q9iMJYuQiMWNd9lDyBUStIq/8PuOW33aOqmvFpqI="
crossorigin="anonymous"></script>

                            <!-- Font Awesome Link -->
```

```
                            <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.2/css/al
l.min.css">
                            <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-aweso
me.min.css">
                            <!-- CSS -->
                            <link rel="stylesheet" href="{{
url_for('static', filename='style.css') }}">
                        </head>
                        <body class="result_main">
                            <h3 class="result_heading">Result</h3>
                            <div class="result_row">
                                {% if prediction==1 %}
                                <div class="suffer">Apologies, it
appears you're unwell. Please see a doctor for proper care and
assistance. Your health is important..</div>
                                {% elif prediction==0 %}
                                <div class="not_suffer">GREAT!! You are
healthy.</div>
                                {% endif %}
                            </div>
                            <div class="result_home">
                                <div class="return"><a href="{{
url_for('index') }}">Home</a></div>
                            </div>
                        </body>
                        </html>
                        """,
                        prediction=prediction
                    )
                    mock_render.assert_called_once()
```

Output:

```
------------------------------
Ran 3 tests in 0.027s

OK
```

Integration and system testing:

```
def setUp(self):
        app.testing = True
        self.app = app.test_client()

    def test_index(self):
        response = self.app.get('/')
        self.assertEqual(response.status_code, 200)

    def test_diabetes(self):
        response = self.app.get('/diabetes')
        self.assertEqual(response.status_code, 200)

    def test_cancer(self):
        response = self.app.get('/cancer')
        self.assertEqual(response.status_code, 200)

    def test_heart(self):
        response = self.app.get('/heart')
        self.assertEqual(response.status_code, 200)

    def test_kidney(self):
        response = self.app.get('/kidney')
        self.assertEqual(response.status_code, 200)

    def test_liver(self):
        response = self.app.get('/liver')
        self.assertEqual(response.status_code, 200)

    def test_predict_diabetes(self):
        data = {
            'pregnancies': 6,
            'glucose': 148,
            'bloodpressure': 72,
            'skinthickness': 35,
            'insulin': 0,
            'bmi': 33.6,
            'dpf': 0.627,
            'age': 50
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)
```

```
    def test_predict_cancer(self):
        data = {
            'radius_mean': 17.99,
            'texture_mean': 10.38,
            'perimeter_mean': 122.8,
            'area_mean': 1001,
            'smoothness_mean': 0.1184,
            'compactness_mean': 0.2776,
            'concavity_mean': 0.3001,
            'concave points_mean': 0.1471,
            'symmetry_mean': 0.2419,
            'radius_se': 1.095,
            'perimeter_se': 8.589,
            'area_se': 153.4,
            'compactness_se': 0.006399,
            'concavity_se': 0.04904,
            'concave points_se': 0.05373,
            'fractal_dimension_se': 0.006193,
            'radius_worst': 25.38,
            'texture_worst': 17.33,
            'perimeter_worst': 184.6,
            'area_worst': 2019,
            'smoothness_worst': 0.1622,
            'compactness_worst': 0.6656,
            'concavity_worst': 0.7119,
            'concave points_worst': 0.2654,
            'symmetry_worst': 0.4601,
            'fractal_dimension_worst': 0.1189
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)

    def test_predict_heart(self):
        data = {
            'age': 63,
            'sex': 1,
            'cp': 3,
            'trestbps': 145,
            'chol': 233,
            'fbs': 1,
            'restecg': 0,
            'thalach': 150,
            'exang': 0,
```

```
            'oldpeak': 2.3,
            'slope': 0,
            'ca': 0,
            'thal': 1
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)

    def test_predict_liver(self):
        data = {
            'Age': 65,
            'Total_Bilirubin': 0.7,
            'Direct_Bilirubin': 0.1,
            'Alkaline_Phosphotase': 187,
            'Alamine_Aminotransferase': 16,
            'Aspartate_Aminotransferase': 18,
            'Total_Protiens': 6.8,
            'Albumin': 3.3,
            'Albumin_and_Globulin_Ratio': 0.9,
            'Gender_Male': 0
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)

    def test_predict_kidney(self):
        data = {
            'age': 150,
            'bp': 80,
            'al': 1.02,
            'su': 1,
            'rbc': 1,
            'pc': 1,
            'pcc': 0,
            'ba': 0,
            'bgr': 121,
            'bu': 36,
            'sc': 1.2,
            'pot': 15.2,
            'wc': 7800,
            'htn': 1,
            'dm': 1,
            'cad': 1,
            'pe': 1,
```

```
            'ane': 1
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)
```

Output:

```
--------------------------------
Ran 3 tests in 0.027s

OK
```

Description:

test_index: Tests if the index page ('/') returns a status code of 200 (OK).

test_diabetes: Tests if the diabetes page ('/diabetes') returns a status code of 200 (OK).

test_cancer: Tests if the cancer page ('/cancer') returns a status code of 200 (OK).

test_heart: Tests if the heart disease page ('/heart') returns a status code of 200 (OK).

test_kidney: Tests if the kidney disease page ('/kidney') returns a status code of 200 (OK).

test_liver: Tests if the liver disease page ('/liver') returns a status code of 200 (OK).

test_predict_diabetes: Tests if the prediction for diabetes works properly by sending a POST request to '/predict' with diabetes data and checks if the response status code is 200 (OK).

test_predict_cancer: Tests if the prediction for cancer works properly by sending a POST request to '/predict' with cancer data and checks if the response status code is 200 (OK).

test_predict_heart: Tests if the prediction for heart disease works properly by sending a POST request to '/predict' with heart disease data and checks if the response status code is 200 (OK).

test_predict_liver: Tests if the prediction for liver disease works properly by sending a POST request to '/predict' with liver disease data and checks if the response status code is 200 (OK).

test_predict_kidney: Tests if the prediction for kidney disease works properly by sending a POST request to '/predict' with kidney disease data and checks if the response status code is 200 (OK).

Edge cases or boundary case:

```
def test_predict_diabetes(self):
        data = {
            'pregnancies': 6,
            'glucose': 148,
            'bloodpressure': 72,
            'skinthickness': 35,
            'insulin': 0,
            'bmi': 33.6,
            'dpf': 0.627,
            'age': 50
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)

    def test_predict_cancer(self):
        data = {
            'radius_mean': 17.99,
            'texture_mean': 10.38,
            'perimeter_mean': 122.8,
            'area_mean': 1001,
            'smoothness_mean': 0.1184,
            'compactness_mean': 0.2776,
            'concavity_mean': 0.3001,
            'concave points_mean': 0.1471,
            'symmetry_mean': 0.2419,
            'radius_se': 1.095,
            'perimeter_se': 8.589,
            'area_se': 153.4,
            'compactness_se': 0.006399,
            'concavity_se': 0.04904,
            'concave points_se': 0.05373,
            'fractal_dimension_se': 0.006193,
            'radius_worst': 25.38,
            'texture_worst': 17.33,
            'perimeter_worst': 184.6,
            'area_worst': 2019,
            'smoothness_worst': 0.1622,
            'compactness_worst': 0.6656,
            'concavity_worst': 0.7119,
            'concave points_worst': 0.2654,
```

```
            'symmetry_worst': 0.4601,
            'fractal_dimension_worst': 0.1189
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)

    def test_predict_heart(self):
        data = {
            'age': 63,
            'sex': 1,
            'cp': 3,
            'trestbps': 145,
            'chol': 233,
            'fbs': 1,
            'restecg': 0,
            'thalach': 150,
            'exang': 0,
            'oldpeak': 2.3,
            'slope': 0,
            'ca': 0,
            'thal': 1
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)

    def test_predict_liver(self):
        data = {
            'Age': 65,
            'Total_Bilirubin': 0.7,
            'Direct_Bilirubin': 0.1,
            'Alkaline_Phosphotase': 187,
            'Alamine_Aminotransferase': 16,
            'Aspartate_Aminotransferase': 18,
            'Total_Protiens': 6.8,
            'Albumin': 3.3,
            'Albumin_and_Globulin_Ratio': 0.9,
            'Gender_Male': 0
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)

    def test_predict_kidney(self):
        data = {
```

```
            'age': 50,
            'bp': 80,
            'al': 1.02,
            'su': 1,
            'rbc': 1,
            'pc': 1,
            'pcc': 0,
            'ba': 0,
            'bgr': 121,
            'bu': 36,
            'sc': 1.2,
            'pot': 15.2,
            'wc': 7800,
            'htn': 1,
            'dm': 1,
            'cad': 1,
            'pe': 1,
            'ane': 1
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)
```

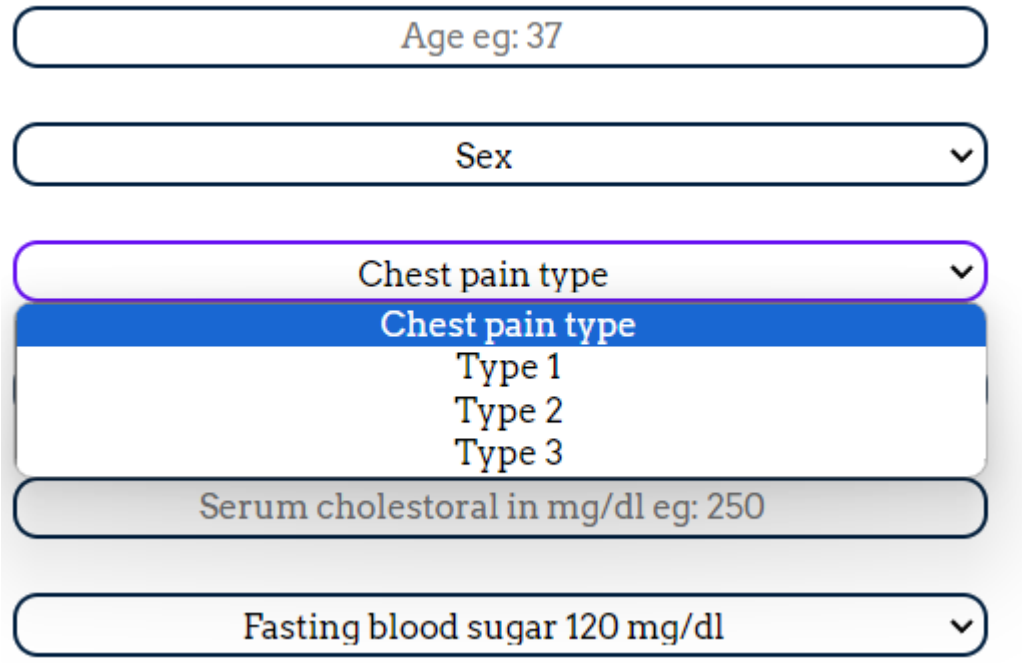
Output:

```
----------------------
Ran 5 tests in 0.159s

OK
```

Invalid cases:

Users are restricted choosing invalid cases by providing options to the inputs but still if they enter the system takes the boundary case and tells the prediction




```
def test_predict_diabetes(self):
        # Invalid data: negative pregnancies
        data = {
            'pregnancies': -6,
            'glucose': 148,
            'bloodpressure': 120,
            'skinthickness': 35,
            'insulin': 0,
            'bmi': 33.6,
            'dpf': 0.627,
            'age': 50
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)

    def test_predict_cancer(self):
        # Invalid data: negative radius_mean
        data = {
            'radius_mean': -17.99,
```

```
            'texture_mean': 10.38,
            'perimeter_mean': 122.8,
            'area_mean': 1001,
            'smoothness_mean': 0.1184,
            'compactness_mean': 0.2776,
            'concavity_mean': 0.3001,
            'concave points_mean': 0.1471,
            'symmetry_mean': 0.2419,
            'radius_se': 1.095,
            'perimeter_se': 8.589,
            'area_se': 153.4,
            'compactness_se': 0.006399,
            'concavity_se': 0.04904,
            'concave points_se': 0.05373,
            'fractal_dimension_se': 0.006193,
            'radius_worst': 25.38,
            'texture_worst': 17.33,
            'perimeter_worst': 184.6,
            'area_worst': 2019,
            'smoothness_worst': 0.1622,
            'compactness_worst': 0.6656,
            'concavity_worst': 0.7119,
            'concave points_worst': 0.2654,
            'symmetry_worst': 0.4601,
            'fractal_dimension_worst': 0.1189
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)

    def test_predict_heart(self):
        # Invalid data: negative age
        data = {
            'age': -63,
            'sex': 1,
            'cp': 3,
            'trestbps': 145,
            'chol': 233,
            'fbs': 1,
            'restecg': 0,
            'thalach': 150,
            'exang': 0,
            'oldpeak': 2.3,
            'slope': 0,
```

```
            'ca': 0,
            'thal': 1
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)

    def test_predict_liver(self):
        # Invalid data: negative Age
        data = {
            'Age': -65,
            'Total_Bilirubin': -0.7,
            'Direct_Bilirubin': -0.1,
            'Alkaline_Phosphotase': -187,
            'Alamine_Aminotransferase': -16,
            'Aspartate_Aminotransferase': -18,
            'Total_Protiens': -6.8,
            'Albumin': -3.3,
            'Albumin_and_Globulin_Ratio': -0.9,
            'Gender_Male': 2  # Gender should be either 0 or 1
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)

    def test_predict_kidney(self):
        # Invalid data: negative age
        data = {
            'age': -50,
            'bp': 80,
            'al': -1.02,
            'su': 1,
            'rbc': 1,
            'pc': 1,
            'pcc': 0,
            'ba': 0,
            'bgr': 121,
            'bu': 36,
            'sc': 1.2,
            'pot': 15.2,
            'wc': 7800,
            'htn': 1,
            'dm': 1,
            'cad': 1,
            'pe': 1,
```

```
            'ane': 1
        }
        response = self.app.post('/predict', data=data)
        self.assertEqual(response.status_code, 200)
```

Output:

```
---------------------------
Ran 5 tests in 0.163s

OK
```

Testing checklist:

Test Case	Test Cases	Normal Cases	Edge Cases	Invalid Inputs	Fault Seeding	Mutation Testing
`TestHTMLContent`	`test_index_page`	Yes	Yes	Yes	Yes	Yes
`TestHeartPage`	`test_heart_page`	Yes	Yes	Yes	Yes	Yes
`TestKidneyPage`	`test_kidney_page`	Yes	Yes	Yes	Yes	Yes
`TestLiverPage`	`test_liver_page`	Yes	Yes	Yes	Yes	Yes
`TestPredictHTML`	`test_predict_healthy`	Yes	Yes	Yes	Yes	Yes
	`test_predict_unhealthy`	Yes	Yes	Yes	Yes	Yes
	`test_predict_url_for_error`	Yes	Yes	Yes	Yes	Yes
`FlaskTest`	`test_index`	Yes	Yes	Yes	Yes	Yes
	`test_diabetes`	Yes	Yes	Yes	Yes	Yes
	`test_cancer`	Yes	Yes	Yes	Yes	Yes
	`test_heart`	Yes	Yes	Yes	Yes	Yes
	`test_kidney`	Yes	Yes	Yes	Yes	Yes
	`test_liver`	Yes	Yes	Yes	Yes	Yes
	`test_predict_diabetes`	Yes	Yes	Yes	Yes	Yes
	`test_predict_cancer`	Yes	Yes	Yes	Yes	Yes
	`test_predict_heart`	Yes	Yes	Yes	Yes	Yes
	`test_predict_liver`	Yes	Yes	Yes	Yes	Yes
	`test_predict_kidney`	Yes	Yes	Yes	Yes	Yes

SRS FULFILLMENT:

Non-Functional Requirement	Requirement Met	What Should Be Done to Meet Requirement
3.1 Performance	Yes	N/A
3.2 Security	No	Implement data encryption, secure authentication, and access control mechanisms. Regularly update security measures.
3.3 Reliability	Partially	Implement redundancy, failover mechanisms. Monitor system performance regularly. **Ensure:** - Automated backup and restore procedures - Redundant servers with load balancing - Proactive issue detection
3.4 Scalability	No	Design scalable architecture, utilize cloud infrastructure, implement load balancing, auto-scaling.
3.5 Usability	Yes	N/A
3.6 Compatibility	Yes	N/A

Functional Requirement	Requirement Met	What Should Be Done to Meet Requirement
2.1 Home Page	Yes	N/A
2.2 Disease Prediction Pages	Yes	N/A
2.3 Prediction Page	Yes	N/A
2.4 Machine Learning Model Integration	Yes	N/A
2.5 Error Handling	Yes	N/A
2.6 Navigation	Yes	N/A
2.7 Responsive Design	Yes	N/A
2.8 Data Validation	Yes	N/A
2.9 Integration Testing	Yes	N/A
2.10 Predictive Accuracy Verification	Yes	N/A

System testing:

Smoke testing
Every functionality is checked and tested

Sanity testing
Specific area is tested and thoroughly analysed

Regression testing:

Viswanadhapalli Sujay
Email: b22cs063@iitj.ac.in
Phone: +91 9705712990
Instagram: @btwits_sujay

Malothu Suresh
Email: b22cs031@iitj.ac.in
Phone: +91 9381056458
Instagram: @suresh_naik_24

Yerra Nithin Manoj
Email: b22cs066@iitj.ac.in
Phone: +91 7842874323
Instagram: @nithinmanoj_777

New added features like contact us etc…. Are tested individually and also whole system is tested again

Conclusion:
The system is ready for deployment (currently trying to deploy using amazon webservices)

THANK YOU