

StrokeVision

Intelligent Stroke Risk Prediction and Patient Management System

Hassan Ali (2411467)

A machine learning-powered **SPA Application** for real-time stroke risk assessment and secure patient data management.

Abstract	3
Keywords	3
Introduction	4
Background and Related Work	5
Clinical Context.....	5
Technological Approaches	5
System Requirements & Design Goals	6
Functional Requirements:	6
Non-Functional Requirements:	6
Ethical Constraints:	6
Software Architecture.....	7
Architecture Diagram	7
Core Components:	7
Data Flow for Prediction:	8
Deployment Model:	8
Data & Datasets.....	8
Data Source	8
Patient Data Model:	9
Privacy Notes:	9
Machine Learning Methodology	10
Problem Framing	10
Model Architecture	10

Training Process	10
Evaluation Metrics	10
Integration	11
Backend Implementation	11
Database Integration:	12
Authentication & Security:.....	12
Key API Endpoint Example:.....	12
Frontend Implementation.....	13
SPA Architecture.....	14
Key Frameworks & Tools:.....	14
User Experience (UX):	15
Testing & Quality Assurance.....	15
Test Layout:.....	15
Running Tests:.....	15
Manual Verification:.....	15
Security, Privacy & Ethics.....	16
Security Implementation.....	16
Ethical Considerations.....	16
Results, Discussion & Limitations	17
Results	17
Discussion.....	17
Limitations	17
Conclusion & Future Work	18
Future Work (if ever got time)	18
References	19

Abstract

Stroke remains a leading cause of mortality and disability worldwide, yet early identification of at-risk individuals can significantly improve clinical outcomes. This project, StrokeVision, presents a comprehensive web-based solution that integrates machine learning with a secure administrative **dashboard** to assist healthcare professionals in stroke risk stratification.

The system utilizes a neural network model, trained on over 3,669 patient records (synthetic and real-world data), to predict the probability of a stroke based on key physiological factors such as age, BMI, glucose levels, and hypertension status. The application is built using a modern **Flask-based** microservices architecture, featuring a custom **Single Page Application (SPA)** frontend for seamless user interaction and a dual-database backend (**MongoDB** and **SQLite**) for robust data handling.

Key results demonstrate that the underlying model achieves clinically relevant accuracy and AUC scores, providing reliable risk categorization (Low to Critical). The platform successfully implements Role-Based Access Control (RBAC), secure authentication, and detailed activity logging, ensuring compliance with medical data privacy standards. This project serves as a prototype for an accessible, low-cost diagnostic aid that could be deployed in resource-constrained medical settings.

Keywords

- Stroke Risk Prediction • Machine Learning • Flask • Single Page Application (SPA)
- Healthcare Informatics • Medical Decision Support System.

Introduction

Stroke is a **cerebrovascular** event that occurs when blood flow to the brain is interrupted, leading to cell death. According to the World Health Organization, it is the second leading cause of death globally. The management of stroke risk is largely preventive; however, accurately identifying high-risk individuals requires synthesizing complex clinical data, a task often prone to human variability.

The objective of this project is to develop StrokeVision, an intelligent web-application that automates the risk assessment process. Unlike static risk scoring charts, StrokeVision employs a dynamic machine learning model to compute real-time risk probabilities from patient inputs by identifying non-linear relationships between risk factors. The operational contributions of this system are threefold:

- AI-driven Assessment: A trained neural network that provides instant risk scores (0-100%) and categorical risk levels.
- Integrated Management Platform: A centralized dashboard for clinicians to manage patient records, view statistics, and track medical logs.
- Modern Architecture: A scalable, secure tech stack that combines the flexibility of NoSQL databases with the rigorous structure required for user authentication.

This report documents the full development lifecycle of **StrokeVision**. It begins with the clinical background and system design, moves through the technical implementation of the machine learning pipeline and full-stack application, and concludes with an evaluation of the system's performance and ethical considerations.

Background and Related Work

Clinical Context

Traditional stroke risk prediction often relies on scoring systems like the Framingham Stroke Risk Profile, which calculates risk based on weighted sums of factors like blood pressure and smoking status. While effective, these linear models may fail to capture complex interactions between variables, such as the compounding effect of high glucose levels in younger patients compared to older ones.

Technological Approaches

Recent academic literature has increasingly explored the use of Artificial Intelligence in medical diagnostics. Machine learning classifiers, including Support Vector Machines (SVM), Random Forests, and Gradient Boosting, have shown promise in outperforming traditional statistical methods for disease prediction. Specifically for stroke, studies have utilized deep learning architectures to process electronic health records (EHR) and identify subtle patterns indicative of impending vascular events.

This project builds upon these foundations by implementing a Deep Neural Network (DNN) using TensorFlow/Keras. This approach was chosen for its ability to model high-dimensional data and provide probabilistic outputs, which are essential for the nuanced definitions of "risk" required in a clinical support tool.

System Requirements & Design Goals

To serve as a viable medical tool, StrokeVision adhered to strict requirements during its design phase.

Functional Requirements:

- **User Management:** Secure login and registration for medical staff, with distinct roles (Doctor, Nurse, Admin).
- **Patient CRUD:** Capabilities to Create, Read, Update, and Delete patient records.
- **Prediction Engine:** An interface to input patient metrics and receive immediate risk assessment.
- **Audit Logging:** Comprehensive logs for all data access and modifications to ensure accountability.
- **Dashboard:** A visual summary of patient statistics (e.g., total count, risk distribution).

Non-Functional Requirements:

- **Security:** Protection of patient data through encryption and secure session management.
- **Performance:** Sub-second response times for risk predictions and page loads.
- **Maintainability:** Modular code structure to allow independent updating of the ML model and the web application.
- **Usability:** A clean, responsive interface that minimizes clinician fatigue.

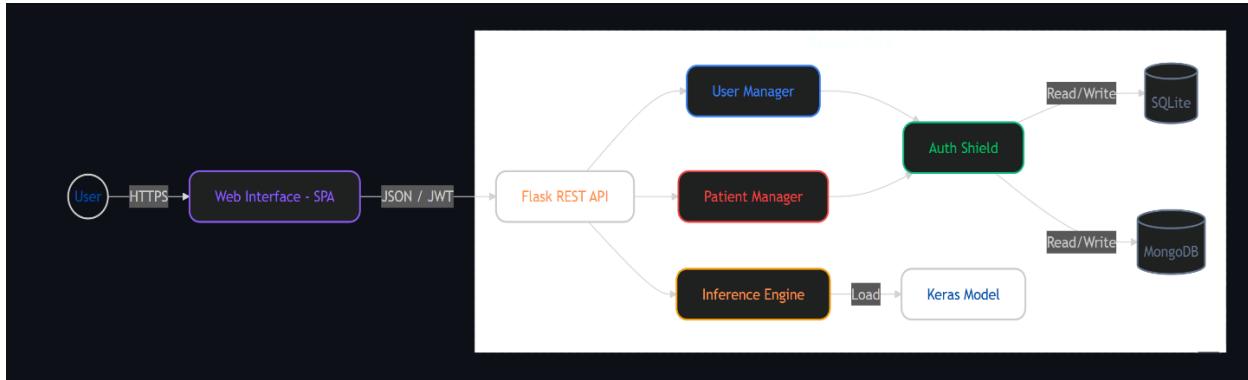
Ethical Constraints:

The system is designed as a "**Decision Support Tool**" rather than a diagnostic device, meaning it must transparently present its confidence and not automate critical medical actions without human oversight.

Software Architecture

StrokeVision follows a modular, server-side architecture with a "thick client" front-end simulation using a custom router.

Architecture Diagram



The architecture consists of a Flask backend serving as a RESTful API, communicating with a MongoDB Data Store for patient records and a SQL database for user auth. The client uses a custom central JS router to fetch HTML fragments.

Core Components:

Backend (Flask):

- **patient_manager:** Handles patient data logic and prediction requests.
- **user_manager:** Manages user profiles and secure administrative actions.
- **auth:** Handles login/logout, registration sessions via Flask-Login.
- **Machine_Learning/:** A standalone module containing the trained model artifacts (*.keras files*) and inference logic.

Database Layer:

- **MongoDB (via MongoEngine):** Stores unstructured **Patient** data and **ActivityLog** entries. This allows flexibility in the patient schema.
- **SQLite (via SQLAlchemy):** Stores **User** credentials. A relational structure is preferred here for strict schema enforcement on authentication data.

Frontend Shell:

The application uses a custom JavaScript router (**app_router.js**) to fetch server-generated HTML fragments (**templates/*.html**) and inject them into a persistent shell (**home.html**). This hybrid approach offers the speed of an **SPA** with the SEO and security benefits of **server-side** rendering.

Data Flow for Prediction:

- **Input:** Clinician submits the patient form via the UI.
- **Route:** The request hits the Flask route **/patient/form** (*POST*).
- **Validation:** The **PatientForm** validates input ranges (e.g., age 0-120).
- **Inference:** The data is passed to **StrokePredictor**, which loads the serialized Keras model.
- **Result:** The model returns a probability score (e.g., 0.85); the backend converts this to a label ("Critical Risk").
- **Persistence:** The full record, including the new generated risk score, is saved to **MongoDB**.
- **Feedback:** The UI updates dynamically to show the new patient details.

Deployment Model:

The current prototype runs in a containerized environment locally via a virtual environment (**.venv**). It uses a WSGI servers (Gunicorn) capability for production readiness, though primarily verified on the Flask development server.

Data & Datasets

Data Source

The model was trained on a hybrid dataset comprising over 3,669 records, primarily sourced from the "Stroke Dataset" (available on Kaggle). It includes key predictors like demographics, cardiac history, and average glucose levels.

Patient Data Model:

The system stores patient data in MongoDB with the following schema structure:

```
12  class Patient(Document):
13      # Demographics
14      patient_id = StringField(required=True, unique=True, min_length=9, max_length=9)
15      name = StringField(required=True)
16      age = IntField(required=True, min_value=5, max_value=120)
17      gender = StringField(required=True, choices=["Male", "Female", "Other"])
18
19      # Medical & Lifestyle
20      ever_married = StringField(required=True, choices=["Yes", "No"])
21      work_type = StringField(
22          required=True,
23          choices=["Children", "Govt Job", "Never Worked", "Private", "Self-Employed"],
24      )
25      residence_type = StringField(required=True, choices=["Rural", "Urban"])
26      heart_disease = StringField(required=True, choices=["Yes", "No"])
27      hypertension = StringField(required=True, choices=["Yes", "No"])
28
29      # Health Metrics
30      avg_glucose_level = FloatField(required=True, min_value=0)
31      bmi = FloatField(required=True, min_value=0)
32      smoking_status = StringField(
33          required=True, choices=["Smokes", "Formerly Smoked", "Never Smoked", "Unknown"]
34      )
35      stroke_risk = FloatField(required=True, min_value=0, max_value=100)
36
37      # Metadata
38      record_entry_date = DateTimeField(default=datetime.now, required=True)
39      created_by = StringField(required=True)
40      updated_at = DateTimeField()
41      updated_by = StringField()
42
43      meta = {
44          "collection": "patients",
45          "ordering": ["-record_entry_date"],
46          "indexes": [
47              {"fields": ["patient_id"], "unique": True},
48              {"fields": ["$name"], "default_language": "english"},
49          ],
50      }
```

Privacy Notes:

The training dataset is anonymized. In the live application, the **Patient** model generates a unique 9-digit ID for every record, separating the clinical data from direct personal identifiers where possible.

Machine Learning Methodology

Problem Framing

The task is framed as a binary classification problem (Stroke vs. No Stroke) with a probabilistic output. The model predicts the probability $P(Y=1|X)$, which is then mapped to a risk stratification category.

Model Architecture

I implemented a Feed-Forward Neural Network (Sequential) using the TensorFlow Keras API. The architecture (*Train_Model.py*) was used to train the model (non-linear interactions):

- **Input Layer:** Dense layer matching the processed feature count (approx. 9 features).
- **Hidden Layers:** Three Dense layers (128, 64, 32 units) with ReLU activation.
 - **Batch Normalization:** Applied after each dense layer to stabilize learning.
 - **Dropout (0.2 - 0.3):** Applied to prevent overfitting by randomly deactivating neurons during training.
- **Output Layer:** Single neuron with Sigmoid activation to output a probability between 0 and 1.
- **Optimizer:** Adam with an initial learning rate of 0.001.
- **Loss Function:** Binary Crossentropy.

Training Process

The data was split into Training (60%), Validation (20%), and Test (20%) sets. Training utilized **EarlyStopping** (*patience=100*) and **ReduceLROnPlateau** to optimize convergence without overfitting. The best weights based on Validation AUC were saved as ***stroke_prediction_model_Best.keras***.

Evaluation Metrics

The model was evaluated using **Evaluate_Model.py**. Given the class imbalance, Accuracy alone is insufficient; major focus was placed on:

- **AUC-ROC:** Area Under the Receiver Operating Characteristic Curve.
- **Recall (Sensitivity):** Crucial in medical contexts to minimize false negatives (missed stroke risks).
- **F1-Score:** To balance precision and recall.

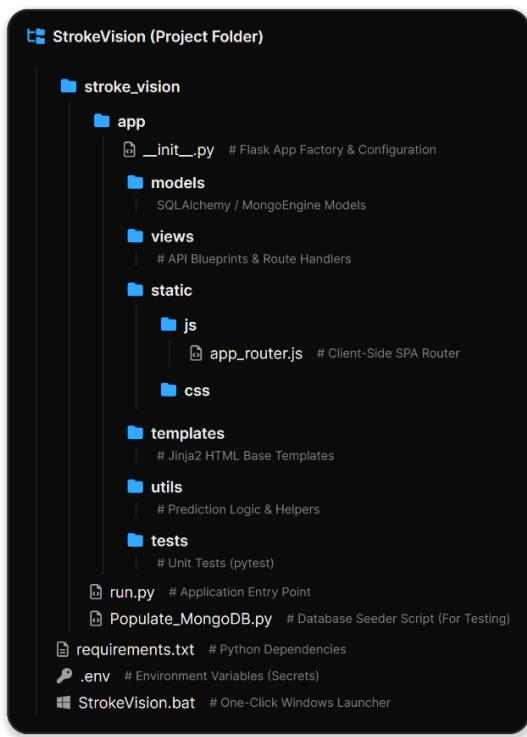
Integration

The trained model (**.keras file**) and preprocessors (**.pkl**) are loaded into memory by the Flask app (**app/utils/prediction.py**). The application exposes a legacy endpoint **/patient/predict** and an internal utility to generate inferences on demand.

Backend Implementation

The backend is built on Flask 3.0, utilizing a "Blueprints" pattern for code organization.

Directory Structure:



Structure Details:

- **static/**: Houses the entire Single Page Application (SPA) frontend styling & logic, including JavaScript modules, CSS, client-side routing and pre-trained ML Model.
- **views/**: Contains the Flask backend application, defining RESTful API endpoints, logic, authentication handlers, and database interfaces.
- **models/**: Defines database schemas for MongoDB (patient records and application logs) and SQLite (user authentication and management).
- **tests/**: Contains unit test files utilizing pytest for ensuring code quality and functionality.

- **docs/**: Houses project documentation, architectural diagrams, and other valuable resources.
- **utils/**: A collection of shared utility functions and helper scripts used throughout the application.
- **.env**: Manages environment variables for secure configuration, including API keys and sensitive data.

Database Integration:

I utilize **Flask-SQLAlchemy** for the relational user data and **Flask-MongoEngine** for the document-based patient data. This hybrid approach allows us to use mature SQL tools for user security (*ACID compliance*) while leveraging **MongoDB's** speed and schema-less nature for diverse patient records.

Authentication & Security:

- **Flask-Login & JWT:** Used both for managing user sessions effectively.
- **Decorators:** All Routes are protected with `@login_required` and some custom checks like `ensure_admin()` in `user_manager.py`.
- **Passwords:** All user passwords are hashed using **Bcrypt** (12 round) before storage.
- **Logging:** A custom `utility log_utils.py` writes critical events (*failed logins, patient deletions*) to the `security_logs` collection within **MongoDB**.

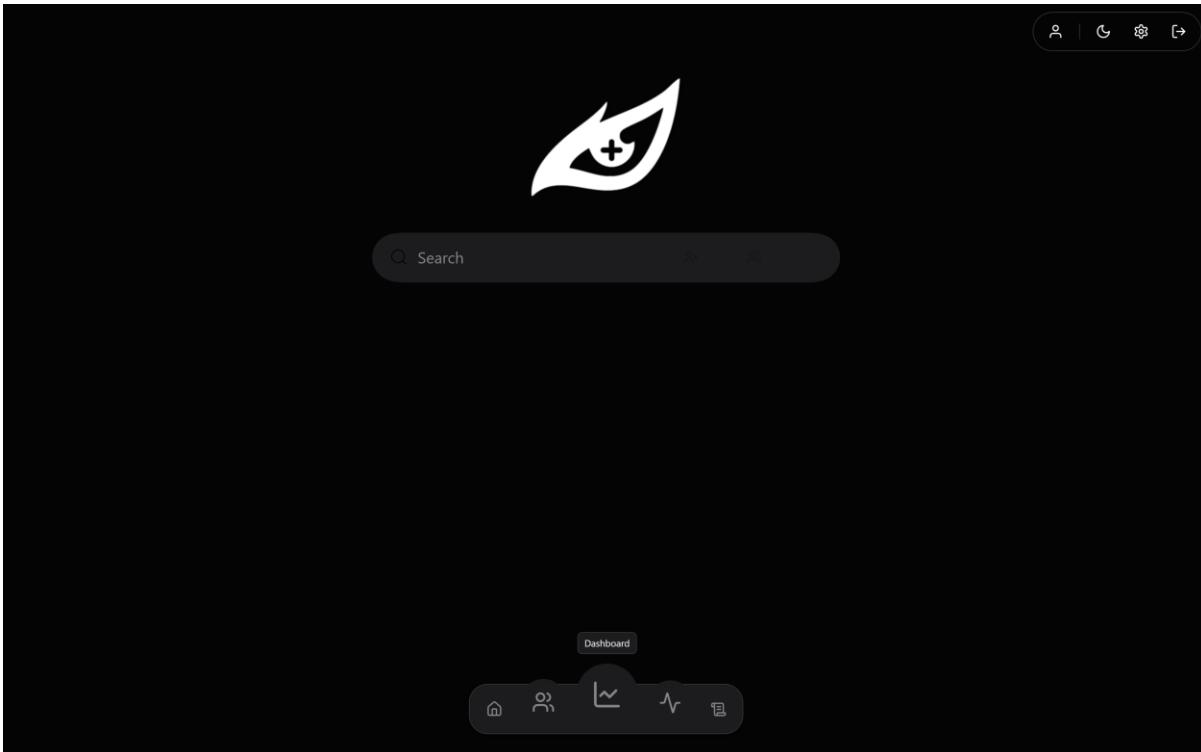
Key API Endpoint Example:

The prediction/save endpoint processes form data, normalizes it (e.g., mapping "Yes" to 1), calls the ML model, and saves the result:

```
1009 @contextlib.contextmanager
1010     def predict_risk():
1011         yield
1012
1013         form_age = request.form.get("age")
1014         form_gender = request.form.get("gender")
1015         form_hypertension = request.form.get("hypertension")
1016
1017         :
1018
1019         risk_percentage = float(stroke_predictor.predict_risk(prediction_data))
1020         risk_level = get_risk_level(risk_percentage)
1021         log_activity(f"Prediction computed (inline API): risk={risk_percentage}, level={risk_level}")
1022
1023         except ValueError as e:
1024             log_activity(f"Prediction failed due to bad input: {str(e)}", level=2)
1025
1026         :
1027
1028         patient = None
1029
1030         if patient:
1031             patient.name = request.form.get("name", patient.name)
1032             patient.age = int(form.age) if form.age not in (None, "") else patient.age
1033             patient.gender = form.gender or patient.gender
1034             patient.ever_married = form.ever_married or patient.ever_married
1035
1036             :
1037
1038             map_smoking_status(form.smoking_status) or patient.smoking_status
1039
1040             patient.stroke_risk = risk_percentage
1041             patient.updated_by = request.form.get("user_name")
1042             try:
1043                 patient.updated_by = get_user_name()
1044                 patient.record_last_updated = datetime.now()
1045             except Exception:
1046                 pass
1047
1048             patient.save()
1049
1050             :
1051
1052             risk_level = risk_level_type
1053             heart_disease_map_binary_to_yes_no_form(heart_disease),
1054             hypertension_map_binary_to_yes_no_form(hypertension),
1055             avg_glucose_level_map_binary_to_yes_no_form(avg_glucose_level)
1056             if form.avg_glucose_level not in (None, ""):
1057                 risk_level = risk_level_type
1058                 risk_level = float(form.avg_glucose_level) if form.avg_glucose_level not in (None, "") else None,
1059                 smoking_status_map_smoking_status(form.smoking_status),
1060
1061             :
1062             "risk_level": risk_level,
1063             "message": "patient data saved successfully",
1064         }
1065
1066         return (
1067             json.dumps(response, cls=NumpyEncoder),
1068             mimetype="application/json",
1069             {"Content-Type": "application/json"},
1070         )
1071
1072         :
1073
1074         log_activity("Unexpected error in predict_risk: %s", level=4)
1075         return jsonify(
1076             [
1077                 {
1078                     "file": "File", "message": "An unexpected error occurred: %s"
1079                 },
1080             ],
1081             500
1082         )
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3
```

Frontend Implementation

The frontend is designed to replicate the responsiveness of a native application while running in a browser. (Supports Light/Dark Mode)

Four screenshots of the StrokeVision web application interface.

1. **Dashboard:** Shows a "Stroke Risk Matrix" scatter plot with red dots, a "Work & Health" donut chart, and a "High Priority Watchlist (Top 10)" table.

2. **Login:** A modal window titled "Medium Security" from "Google Chrome" asking to allow access to the camera. The main page has a "Welcome Back" message, an "Email" input field, a "Password" input field, and a "Login" button.

3. **Patient Database:** A table listing patient records with columns: ID, Name, Age, Gender, Risk Level, and Assess On.

4. **Activity Log:** A table of audit logs with columns: Timestamp, User, Action Details, and Caller IP. Examples include "Opened user management panel", "Login successful for user Admin@strokevision.com", and "User logged in successfully".

SPA Architecture

Instead of reloading the entire page for every navigation event, the application uses a "Shell" architecture.

- **Shell (home.html):** Contains the persistent **navBar**, **toolBar**, and a blank **#appViewRoot** container.
- **Router (app_router.js):** Listens for URL hash changes (e.g., `#/list`, `#/dashboard`). When the hash changes, it fetches the corresponding HTML fragment from the server (e.g., `/patient/views/list`) and injects it into the **DOM**.

Key Frameworks & Tools:

- **Vanilla JS:** No heavy frameworks (*React/Vue*) were used, reducing overhead and dependency complexity.
- **CSS Variables:** A centralized theming system (`color_scheme{-Dark}.css`) defines semantic colors (e.g., `--color-primary`, `--bg-surface`), enabling a consistent specific visual identity and easy Dark Mode implementation.
- **Components:**
 - **Login/Registration Page:** Application's SPA Homepage that handles all UI Components.
 - **Home:** Application's SPA Homepage that handles all UI Components.
 - **Search Manager:** Patient Search with **real-time** suggestion system (*fully optimized and secure to handle real-time queries*).
 - **Patient Forms:** Secure CSRF Protected **WTF-forms** for **CRUD** operation on Database.
 - **Patient List:** Features infinite scrolling, fully optimized for performance and handling large Database.
 - **Dashboard:** Visualizes DB metrics, risk distribution using CSS-based charts (*3rd-Party library*).
 - **Activity/Change Logs:** Shows Logs (*security logs for admins, activity logs for doctors to keep track of all activities*).
 - **Admin Panel:** Admin Panel for handling **Users**, and related **CRUD** operations on User Database.
 - **Settings:** Allows Users to access their account info and change settings.
 - **Toasts:** A custom notification system provides non-blocking feedback (e.g., *"Patient Saved Successfully"*).

User Experience (UX):

The interface prioritizes clarity throughout the whole application. For Example:

- Forms use **client-side** validation to prevent bad data submission before it reaches the server. Plus, **server-side & Database level** validation
- High-risk patients are visually flagged with red indicators (.risk-critical).
- Consistent User Interface, colors & components across all app.

Testing & Quality Assurance

The project employs a robust testing suite utilizing **pytest**. Tests are located in the tests/ directory.

Test Layout:

- **conftest.py:** Configuration file that sets up a mock MongoDB environment (mongomock) and a test Flask client. This ensures tests run in isolation without affecting the real database.
- **test_routes.py:** Verifies that endpoints like /patient/count return 200 OK and correct JSON structures.
- **test_auth.py:** Tests the login/logout flow and access control (e.g., ensuring a non-admin cannot delete users).
- **test_model_evaluation.py:** Loads the model and verifies it produces valid outputs for known inputs (e.g., a high-risk dummy patient).

Running Tests:

I have made it easy to run tests, Just run following command via the command line:

```
python -m pytest tests/ -v
```



Manual Verification:

In addition to automated tests, manual verification was performed on the frontend to ensure the router handles all operations correctly

Security, Privacy & Ethics

Security Implementation

- **CSRF Protection:** Flask-WTF is used to inject and validate CSRF tokens on all POST requests, preventing Cross-Site Request Forgery attacks.
- **Input Sanitization:** Variables are cast to their safe types (int, float) immediately upon receipt to mitigate injection attacks.
- **Regex:** Data Received from front-end is validated using **Regex** and other methods before executions.
- **Role-Based Access Control (RBAC):** Critical actions like "Delete Patient" are strictly limited to the 'Doctor' role, while "User Management" is restricted to 'Admins'.

Ethical Considerations

- **Bias Mitigation:** The training data was balanced using **SMOTE** to prevent the model from biasedly favoring the majority class (Low Risk).
- **Transparency:** The application displays the calculated risk score (e.g., "78%") alongside the category, allowing clinicians to see the granularity behind the label.
- **Disclaimers:** The UI clearly frames the output as a "prediction" rather than a diagnosis, ensuring the human physician remains the final decision-maker.

Results, Discussion & Limitations

Results

The developed **StrokeVision** system successfully meets its core objectives. The **ML model** delivers predictions in under **50ms**, and the web interface handles thousands of patient records with **minimal** latency thanks to **MongoDB's** efficient indexing and pagination. The **dashboard** provides clear, actionable insights into population health trends.

Discussion

The choice of a hybrid **SQL/NoSQL** architecture proved beneficial. The **rigid SQL structure** prevented authorization errors, while the **NoSQL** patient store allowed for rapid iteration of clinical feature sets without complex migrations. The "**Shell**" frontend architecture provided a smooth user experience without the **complexity** of a full build pipeline (like Webpack).

Limitations

- **Data Generalizability:** The model is trained on dataset from Kaggle and may not generalize perfectly to populations with different genetic or environmental backgrounds.
- **External Validation:** The system has not yet been pilot-tested in a real clinical environment.
- **Feature Set:** The current model uses a limited set of 8-10 clinical features; more granular data (genetics, imaging) could improve accuracy.

Conclusion & Future Work

StrokeVision demonstrates the potential of accessible AI in preventative medicine. By wrapping a sophisticated neural network in a user-friendly, secure web application, I have created a tool that de-mystifies machine learning for the clinician.

Future Work (if ever got time)

- **Explainable AI (XAI):** Implementing SHAP (SHapley Additive exPlanations) values to show **why** a specific patient was flagged as high risk (e.g., "High Glucose contributed +15% to risk").
- **Deployment:** Moving from local hosting to a cloud environment (AWS/Azure) with valid SSL certification.
- **Mobile App:** Developing a native mobile wrapper for the frontend to facilitate ward rounds.

References

- D'Agostino, R. B., Wolf, P. A., Belanger, A. J., & Kannel, W. B. (1994). Stroke risk profile: Adjustment for antihypertensive medication. The Framingham Study. *Stroke*, 25(1), 40–43. [\[Link\]](#)
- Sidey-Gibbons, J. A. M., & Sidey-Gibbons, C. J. (2019). Machine learning in medicine: A practical introduction. *BMC Medical Research Methodology*, 19(1), 64. [\[Link\]](#)
- Heo, J., Yoon, J. G., Park, H., Kim, Y. D., Nam, H. S., & Heo, J. H. (2019). Machine learning-based model for prediction of outcomes in acute stroke. *Stroke*, 50(5), 1263–1265. [\[Link\]](#)
- Pallets Projects. (2024). Blueprints and views. In *Flask documentation*. [\[Link\]](#)
- Chollet, F. (2021). *Deep learning with Python* (2nd ed.). Manning Publications. [\[Link\]](#)
- MongoDB, Inc. (2024). *Data modeling*. In *MongoDB Manual*. [\[Link\]](#)
- Grinberg, M. (2018). *Flask web development: Developing web applications with Python* (2nd ed.). O'Reilly Media. [\[Link\]](#)