4. IMPLEMENTASI SISTEM

Pada bab ini akan membahas mengenai penjelasan implementasi sistem dan teori yang akan digunakan sesuai dengan desain sistem. Implementasi tersebut meliputi alur yang telah dibuat di dalam desain sistem bab sebelumnya, alur meliputi *pre-processing*, *modeling*, evaluasi model, dan implementasi *website*.

4.1 Perangkat Lunak yang Digunakan

Pembuatan model *machine learning* dalam skripsi ini dibangun menggunakan Bahasa pemgrograman *python* versi 3.10.12 dengan bantuan *Google Collab* untuk menulis dan menjalankan *Source Code*. Ada beberapa *library python* yang digunakan dalam implementasi model *machine learning*. Pertama ada *library pandas* yang berfungsi untuk memanipulasi dan menganalisis data secara efisien menggunakan struktur data yang biasa disebut *DataFrame*. *Pandas* juga membantu untuk menulis dan membaca data dengan berbagai macam format *file*. Selanjutnya ada *library numpy* yang berfungsi untuk komputasi numerik. Lalu ada *library matplotlib* yang berfungsi untuk membuat grafik dan *visualisasi data*. Ada *library sns* yang merupakan *library* yang dibangun diatas *matplotlib* yang berguna untuk membuat *visualisasi data* yang lebih indah dan informatif. Kemudian ada *library joblib* yang berfungsi untuk meyimpan dan memanggil model yang sudah dibuat. Lalu ada *library scikit-learn* yang berfungsi untuk meyediakan algoritma-algoritma *machine learning* yang digunakan dalam pembuatan model. Dan yang terakhir ada *library warnings* untuk membantu mengelola pesan peringatan yang muncul selama eksekusi program.

Lalu untuk pembuatan *website* dalam skripsi ini ditulis dan dijalankan menggunakan *Microsoft Visual Studio Code*. Bagian *front-end website* dibangun menggunakan *Vue 3 JS* sebagai *framework JavaScript* dibantu dengan *Vite* sebagai *build tools* dan *Tailwind CSS* sebagai *framework CSS*nya. Kombinasi kedua teknologi tesebut dapat membantu untuk membuat *interface* yang interaktif dan efisien. Diperlukan *Node* JS dan *NPM* untuk membantu menginstall *package* dari *Vue 3 JS, Vite* dan *Tailwind CSS*. Kemudian untuk bagian *back-end website* dibangun menggunakan *Flask* dan beberapa *library* lainnya seperti *numpy, joblib, warnings* yang dibuat dengan pemrograman *python* versi 3.11.4.

Terakhir, untuk bagian interaksi *API website* dalam skripsi ini dibuat menggunakan *Axios JS* yang merupakan *library JavaScript*. *Axios JS* ini berfungsi untuk mengirimkan permintaan *HTTP*

dan menerima data yang diperlukan dengan kata lain *Axios JS* ini berguna sebagai sarana penghubung antara bagian *front-end* dan *back-end website*.

4.2 Implementasi Model *Machine Learning*

Setelah membahas tentang *library* serta *tools* yang digunakan dan telah selesai dilakukan instalasi, tahap selanjutnya adalah pembuatan model dan implementasi *website*. Pada bab ini akan membahas implementasi dari desain sistem yang telah dibuat pada bab 3 yang akan dirincikan dalam bentuk pemetaan Segmen Program pada Tabel 4.1.

Tabel 4.1 Pemetaan Segmen Program

Desain Sistem	Segmen Program	Keterangan
3.3.1 *Preprocessing*	4.1	*Preprocessing*
3.3.2 *Data Modeling*	4.2	Fungsi dalam Data *Modeling*
	4.3	*Support Vector Machine Modeling*
	4.4	*Support Vector Machine Tuned Modeling*
	4.5	*K-Nearest Neighbor Modeling*
	4.6	*K-Nearest Neighbor Tuned Modeling*
	4.7	*Random Forest Modeling*
	4.8	*Random Forest Tuned Modeling*
3.3.3 *Evaluate Model*	4.9	*Evaluate Model*
3.3.4 *Website Implementation*	4.10	*Main.js*
	4.11	*Main.css*
	4.12	*StrokePrediction.vue*
	4.13	*Home.vue*
	4.14	*App.vue*
	4.15	*Index.html*
	4.16	*app.py*
	4.17	*Axios.js*

4.2.1 Proses *Preprocessing Dataset*

Pada tahap ini akan dilakukan proses *preprocessing* terhadap *dataset*. *Preprocessing* yang akan dilakukan meliputi *feature selection, data scalling* dan *data splitting*. Pertama akan

dilakukan *feature selection* dengan cara membuat sebuah *heatmap correlation* yang berfungsi untuk menggambarkan korelasi antar fitur dalam *dataset* yang dapat membantu untuk mengukur sejauh mana hubungan antara fitur satu dengan yang lain. Fitur yang memiliki korelasi terbaik akan digunakan dalam proses selanjutnya. Selanjutnya, dilakukan *data scaling* pada *dataset*. *Data scaling* berguna untuk mengubah rentang nilai pada fitur agar seimbang dan mempermudah analisis, pada skripsi ini *data scaling* akan dilakukan dengan menggunakan fungsi *MinMaxScaler*. Setelah dataset telah melalui tahapan *data scaling*, akan dilakukan proses *Train Test Split* terhadap data yang siap digunakan untuk membangun model *machine learning*. Dalam skripsi ini akan dibagi *dataset* dengan porsi *data training* sebesar 80% dan *data testing* sebesar 20%. *Source Code* untuk menjalankan proses diatas dapat dilihat pada Segmen Program 4.1

Segmen Program 4.1 *Code Preprocessing Dataset*

```
#heatmap correlation
plt.figure(figsize=(20, 5))
sns.heatmap(df_cleaned.corr(), annot=True, cmap='Blues')

#final data after feature selection heatmap
final_df = df_cleaned[['sex', 'age', 'hypertension', 'heart_disease', 'avg_glucose_level', 'bmi',
'smoking_status', 'stroke']]
final_df.head()
print("Total Baris Final Dataset :", final_df.shape[0])
print("Total Kolom Final Dataset :", final_df.shape[1])

#data scaling
minmax_scaler = MinMaxScaler()
data_scaling = minmax_scaler.fit_transform(final_df.drop(['stroke'], axis=1))
df_scaled = pd.DataFrame(data_scaling, columns=final_df.drop(['stroke'], axis=1).columns)
df_scaled.shape
df_scaled.head()

#save datascaling
joblib.dump(minmax_scaler, Model_Directory + 'MinMaxScaler.save')


#data splitting
X = df_scaled.copy(deep=True) # feature
y = final_df['stroke'] # target
display(X)
display(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1,
stratify=df_cleaned['stroke'])
```

4.2.2 Proses *Data Modeling*

Tahap selanjutnya adalah *data modeling*, proses pertama akan dibuat sebuah fungsi *evaluate model* yang nantinya berguna untuk proses *testing* sekaligus menampilkan hasil prediksi seperti akurasi, presisi, *recall*, dan skor *F1* dari model yang telah di buat agar bisa dievaluasi lebih lanjut. Lalu akan dibuat juga fungsi *confusion matrix* yang berguna untuk membuat visualisasi matriks konfusi berdasarkan hasil prediksi. Akan ditampilkan juga jumlah *data training* dan *data testing*. *Source Code* untuk menjalankan proses diatas dapat dilihat pada Segmen Program 4.2

Segmen Program 4.2 *Code* Fungsi dalam *Data Modeling*

```
#fungsi evaluate model untuk testing & evaluasi hasil
def evaluate_model(model, X_test, y_test, name):
   model_pred = model.predict(X_test)

   accuracy = accuracy_score(y_test, model_pred)
   precision = precision_score(y_test, model_pred)
   recall = recall_score(y_test, model_pred)
   f1 = f1_score(y_test, model_pred)

   conf_mtr = tuple(confusion_matrix(y_test, model_pred).ravel())
   conf_mtr_val = []
   conf_mtr_val.append(conf_mtr)

   df_model = pd.DataFrame({'Model': [name], 'Accuracy':accuracy, 'Precision':precision,
'Recall':recall, 'F1':f1, '(TN, FP, FN, TP)':conf_mtr_val})
   return df_model

#fungsi untuk confusion matrix
def conf_custom(model):
   y_pred = model.predict(X_test)
   cm = confusion_matrix(y_test, y_pred)

   sns.heatmap(cm, annot=True, fmt='d')
   plt.title('Confusion Matrix')
   plt.xlabel('Predicted')
   plt.ylabel('Actual')
   plt.show()

#Jumlah data training & data testing
print("Total (Baris, Kolom) Fitur Data Latih", X_train.shape)
print("Total (Baris, Kolom) Target Data Latih", y_train.shape)
print("="*50)
print("Total (Baris, Kolom) Fitur Data Test", X_test.shape)
print("Total (Baris, Kolom) Target Data Test", y_test.shape)
```

```
display(X_train.head())
display(y_train.head())
```

4.2.2.1 *Support Vector Machine Modeling*

Metode pertama yang digunakan dalam proses *modeling* adalah *Support Vector Machine*. Model dilatih menggunakan algoritma *Support Vector Machine* dengan parameter *default*, selanjutnya model yang telah dilatih tersebut disimpan. Kemudian hasil dari model *Support Vector Machine* yang telah dilatih tesebut dites dan dievaluasi menggunakan fungsi *evaluate_model* yang telah dibuat sebelumnya untuk menampilkan hasil seperti akurasi, presisi, *recall*, skor *F1*. Selanjutnya, diperlihatkan juga visualisasi matriks konfusi menggunakan fungsi yang telah dibuat juga sebelumnya untuk mengetahui seberapa baik metode *Support Vector Machine* ini dalam memprediksi. *Source Code* untuk menjalankan proses diatas dapat dilihat pada Segmen Program 4.3.

Segmen Program 4.3 *Code Support Vector Machine Modeling*

```
#train model SVM with default parameter
#default C=1.0, kernel=rbf, gamma=scale
svm = SVC(random_state=1, probability=True)
svm.fit(X_train, y_train)

#save trained model
joblib.dump(svm, Model_Directory + 'svm_model.save')

#load trained model
svm_model = joblib.load(Model_Directory + 'svm_model.save')
model_name = 'Support Vector Machine'

#testing & evaluasi model
result_svm = evaluate_model(svm_model, X_test, y_test, model_name)
display(result_svm)

#display confusion matrix
ConfusionMatrixDisplay.from_estimator(svm_model, X_test, y_test, cmap='Blues')
```

4.2.2.2 *Support Vector Machine Tuned Modeling*

Metode selanjutnya yang digunakan dalam proses modeling adalah *Support Vector Machine Tuned*. Model dilatih menggunakan algoritma *Support Vector Machine Tuned*, artinya akan dicoba berbagai kombinasi parameter dan parameter terbaik akan digunakan untuk tahap selanjutnya. Untuk mencoba berbagai kombinasi parameter, digunakan teknik *GridSearchCV*

yang terdapat pada *library sckit learn*. Selanjutnya model yang telah dilatih tersebut disimpan. Kemudian hasil dari model *Support Vector Machine Tuned* yang telah dilatih tesebut dites dan dievaluasi menggunakan fungsi *evaluate_model* yang telah dibuat sebelumnya untuk menampilkan hasil seperti akurasi, presisi, *recall*, skor *F1*. Selanjutnya, diperlihatkan juga visualisasi matriks konfusi menggunakan fungsi yang telah dibuat juga sebelumnya untuk mengetahui seberapa baik metode *Support Vector Machine Tuned* ini dalam memprediksi. *Source Code* untuk menjalankan proses diatas dapat dilihat pada Segmen Program 4.4.

Segmen Program 4.4 *Code Support Vector Machine Tuned Modeling*

```
#kombinasi parameter yang akan dicoba
param_grid = {
   'C': [0.1, 1, 10],
   'kernel': ['linear', 'rbf'],
   'gamma': [0.01, 0.1, scale]
}
#train model SVM menggunakan kombinasi parameter
svm = SVC(random_state=1, probability=True)
svm_grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, verbose=3)
svm_grid_search.fit(X_train, y_train)

#save trained model
joblib.dump(svm_grid_search, Model_Directory + 'svm_tuned_model.save')

#load trained model
svm_tuned_model = joblib.load(Model_Directory + 'svm_tuned_model.save')

#print best parameter & score
print('Parameter Terbaik : ', svm_tuned_model.best_params_)
print('Score Terbaik : ', svm_tuned_model.best_score_)

#testing & evaluasi model
model_name = 'Support Vector Machine Tuned'
result_svm_tuned = evaluate_model(svm_tuned_model, X_test, y_test, model_name)
display(result_svm_tuned)

#display confusion matrix
ConfusionMatrixDisplay.from_estimator(svm_tuned_model, X_test, y_test, cmap='Blues')
```

4.2.2.3 *K-Nearest Neighbor Modeling*

Metode selanjutnya yang digunakan dalam proses *modeling* adalah *K-Nearest Neighbor*. Model dilatih menggunakan algoritma *K-Nearest Neighbor* dengan parameter *default*, selanjutnya model yang telah dilatih tersebut disimpan. Kemudian hasil dari model *K-Nearest*

*Neighbo*r yang telah dilatih tesebut dites dan dievaluasi menggunakan fungsi *evaluate_model* yang telah dibuat sebelumnya untuk menampilkan hasil seperti akurasi, presisi, *recall*, skor *F1*. Selanjutnya, diperlihatkan juga visualisasi matriks konfusi menggunakan fungsi yang telah dibuat juga sebelumnya untuk mengetahui seberapa baik metode *K-Nearest Neighbor* ini dalam memprediksi. *Source Code* untuk menjalankan proses diatas dapat dilihat pada Segmen Program 4.5.

Segmen Program 4.5 *Code K-Nearest Neighbor Modeling*

```
#train model KNN with default parameter
#default n_neighbors=5, weight=uniform, metric=minknowski

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)

#save trained model
joblib.dump(knn, Model_Directory + 'knn_model.save')

#load trained model
knn_model = joblib.load(Model_Directory + 'knn_model.save')

#testing & evaluasi model
model_name = 'K Nearest Neighbor'
result_knn = evaluate_model(knn_model, X_test, y_test, model_name)
display(result_knn)

#display confusion matrix
ConfusionMatrixDisplay.from_estimator(knn_model, X_test, y_test, cmap='Blues')
```

4.2.2.4 *K-Nearest Negihbor Tuned Modeling*

Metode selanjutnya yang digunakan dalam proses *modeling* adalah *K-Nearest Neighbor Tuned*. Model dilatih menggunakan algoritma *K-Nearest Neighbor Tuned*, artinya akan dicoba berbagai kombinasi parameter dan parameter terbaik akan digunakan untuk tahap selanjutnya. Untuk mencoba berbagai kombinasi parameter, digunakan teknik *GridSearchCV* yang terdapat pada *library sckit learn*. Selanjutnya model yang telah dilatih tersebut disimpan. Kemudian hasil dari model *K-Nearest Neighbor Tuned* yang telah dilatih tesebut dites dan dievaluasi menggunakan fungsi *evaluate_model* yang telah dibuat sebelumnya untuk menampilkan hasil seperti akurasi, presisi, *recall*, skor *F1*. Selanjutnya, diperlihatkan juga visualisasi matriks konfusi menggunakan fungsi yang telah dibuat juga sebelumnya untuk mengetahui seberapa baik

metode *K-Nearest Neighbor Tuned* ini dalam memprediksi. *Source Code* untuk menjalankan proses diatas dapat dilihat pada Segmen Program 4.6.

Segmen Program 4.6 *Code K-Nearest Neighbor Tuned Modeling*

```
#kombinasi parameter yang akan dicoba
param_grid = {
  'n_neighbors' : range(3, 11, 2),
  'weights' : ['uniform', 'distance'],
  'metric' : ['euclidean', 'manhattan', 'minkowski']
}

#train model KNN menggunakan kombinasi parameter
knn_grid_search = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid, n_jobs=-1,
verbose=3)
knn_grid_search.fit(X_train, y_train)

#save trained model
joblib.dump(knn_grid_search, Model_Directory + 'knn_tuned_model.save')

#load trained model
knn_tuned_model = joblib.load(Model_Directory + 'knn_tuned_model.save')

#print best parameter & score
print('Parameter Terbaik : ', knn_tuned_model.best_params_)
print('Score Terbaik : ', knn_tuned_model.best_score_)

#testing & evaluasi model
model_name = 'K Nearest Neighbor Tuned'
result_knn_tuned = evaluate_model(knn_tuned_model, X_test, y_test, model_name)
display(result_knn_tuned)

#display confusion matrix
ConfusionMatrixDisplay.from_estimator(knn_tuned_model, X_test, y_test, cmap='Blues')
```

4.2.2.5 *Random Forest Modeling*

Metode selanjutnya yang digunakan dalam proses *modeling* adalah *Random Forest*. Model dilatih menggunakan algoritma *Random Forest* dengan parameter *default*, selanjutnya model yang telah dilatih tersebut disimpan. Kemudian hasil dari model Random Forest yang telah dilatih tesebut dites dan dievaluasi menggunakan fungsi *evaluate_model* yang telah dibuat sebelumnya untuk menampilkan hasil seperti akurasi, presisi, *recall*, skor *F1*. Selanjutnya, diperlihatkan juga visualisasi matriks konfusi menggunakan fungsi yang telah dibuat juga sebelumnya untuk mengetahui seberapa baik metode *Random Forest* ini dalam memprediksi. *Source Code* untuk menjalankan proses diatas dapat dilihat pada Segmen Program 4.7.

Segmen Program 4.7 *Code Random Forest Modeling*

```
#train model RFC with default parameter
#default n_estimator=100, max_features=sqrt, max_depth=none, max_leaf_nodes=none
rfc = RandomForestClassifier(random_state=32)
rfc.fit(X_train, y_train) # proses training

#save trained model
joblib.dump(rfc, Model_Directory + 'random_forest_model.save')

#load trained model
model_rfc = joblib.load(Model_Directory + 'random_forest_model.save')

#testing & evaluasi model
model_name = 'Random Forest'
result_rfc = evaluate_model(model_rfc, X_test, y_test, model_name)
display(result_rfc)

#display confusion matrix
ConfusionMatrixDisplay.from_estimator(model_rfc, X_test, y_test, cmap='Blues')
```

4.2.2.6 *Random Forest Tuned Modeling*

Metode selanjutnya yang digunakan dalam proses modeling adalah *Random Forest Tuned*. Model dilatih menggunakan algoritma *Random Forest Tuned*, artinya akan dicoba berbagai kombinasi parameter dan parameter terbaik akan digunakan untuk tahap selanjutnya. Untuk mencoba berbagai kombinasi parameter, digunakan teknik *GridSearchCV* yang terdapat pada *library sckit learn*. Selanjutnya model yang telah dilatih tersebut disimpan. Kemudian hasil dari model *Random Forest Tuned* yang telah dilatih tesebut dites dan dievaluasi menggunakan fungsi *evaluate_model* yang telah dibuat sebelumnya untuk menampilkan hasil seperti akurasi, presisi, *recall*, skor *F1*. Selanjutnya, diperlihatkan juga visualisasi matriks konfusi menggunakan fungsi yang telah dibuat juga sebelumnya untuk mengetahui seberapa baik metode *Random Forest Tuned* ini dalam memprediksi. *Source Code* untuk menjalankan proses diatas dapat dilihat pada Segmen Program 4.8.

Segmen Program 4.8 *Code Random Forest Tuned Modeling*

```
#kombinasi parameter yang akan dicoba
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_features': ['sqrt', 'log2', None],
    'max_depth': [5, 10, None],
    'max_leaf_nodes': [5, 10, None],
}
```

```
#train model KNN menggunakan kombinasi parameter
rf_grid_search = GridSearchCV(RandomForestClassifier(random_state=32),
param_grid=param_grid, verbose = 3)
rf_grid_search.fit(X_train, y_train)

#save trained model
joblib.dump(rf_grid_search, Model_Directory + 'random_forest_model_tuned.save')


#load trained model
model_rf_tuned = joblib.load(Model_Directory + 'random_forest_model_tuned.save')

#print best parameter & score
print('Parameter Terbaik : ', model_rf_tuned.best_params_)
print('Score Terbaik : ', model_rf_tuned.best_score_)

#testing & evaluasi model
model_name = 'Random Forest Tuned'
result_rfc_tuned = evaluate_model(model_rf_tuned, X_test, y_test, model_name)
display(result_rfc_tuned)

#display confusion matrix
ConfusionMatrixDisplay.from_estimator(model_rf_tuned, X_test, y_test, cmap='Blues')
```

4.2.3 Proses *Evaluate Model*

Pada tahap ini, dilakukan penggabungan hasil *testing* dari model-model yang sudah dibuat pada bab sebelumnya. Hasil evaluasi tiap model akan digabungkan menggunakan fungsi *concat* yang diambil dari *library pandas* dan dibentuk sebuah *DataFrame* baru. Hasil penggabungan tersebut kemudian disimpan menjadi *file csv*. Selanjutnya hasil penggabungan tersebut juga akan ditampilkan menggunakan *plot* grafik batang yang menggambarkan seberapa besar akurasi masing-masing model. *Source Code* untuk menjalankan proses diatas dapat dilihat pada Segmen Program 4.9.

Segmen Program 4.9 *Code Evaluate Model*

```
#menggabungkan semua hasil testing model / evaluasi model
result_all_model = pd.concat([result_rfc, result_svm, result_knn, result_rfc_tuned,
result_svm_tuned, result_knn_tuned], ignore_index=True)
result_all_model = result_all_model.sort_values(['Accuracy','Precision', 'Recall', 'F1'],
ascending=False).reset_index(drop=True)


#save hasil penggabungan evaluasi model
result_all_model.to_csv(Data_Directory + 'result_all_model.csv', index=False)
#load hasil penggabungan evaluasi model
```

```
result_all_model = pd.read_csv(Data_Directory + 'result_all_model.csv')
result_all_model

#display bar plot akurasi evaluasi model
g = sns.barplot(x='Accuracy', y='Model',
        data=result_all_model,
        orient='h',
        width=0.8)
g.set_xlabel("Accuracy")
g = g.set_title("Accuracy Scores")
g.figure.set_size_inches(6, 4)
```

4.3 *Wesbite Implementation*

Pada tahap ini, dilakukan implementasi *website* sederhana untuk memprediksi penyakit stroke. Proses dimulai dengan membuat bagian *front-end* kemudian dilanjutkan dengan membuat bagian *back-end* dan yang terakhir adalah interaksi *API* yang dapat menghubungkan bagian *front-end* dengan *back-end website* melalui *API*.

4.3.1 Bagian *Front-End Website*

Pada tahap ini, dilakukan proses pembuatan bagian *front-end* dari *website*. Langkah pertama yang dilakukan untuk membuat *front-end* dari *website* ini adalah menginstall *Node.js* dan *NPM* (*Node Package Manager*) untuk membantuk mengatur *package JavaScript* yang diperlukan untuk pembuatan *website*. Langkah selanjutnya adalah meningstall *Vite* dan *Vue 3 package* yang berfungsi untuk membuat suatu proyek baru menggunakan *Vue 3 JS*. Lalu selanjutnya adalah menginstall *Tailwind CSS package* agar dapat digunakan sebagai *framework CSS*. Setelah semua *package* yang diperlukan terinstall, langkah selanjutnya adalah proses pembuatan proyek.

File pertama yang ada dalam sebuah proyek *Vue 3* adalah *Main.js*. *File* ini adalah kunci utama dalam proyek *Vue 3* karena *file* ini berguna untuk menginialisasikan dan menyusun seluruh aplikasi dalam proyek yang dibuat. *Source Code* untuk *file* ini dapat dilihat pada Segmen Program 4.10.

Segmen Program 4.10 *Source Code Main.js*

```
import './assets/main.css';
import { createApp } from 'vue';
import App from './App.vue';
```

```
const app = createApp(App);

app.mount('#app');
```

Setelah *file Main.js*, *file* selanjutnya yang ada dalam proyek skripsi ini adalah *Main.css*. *File* ini berguna untuk mengatur konfigurasi dasar, komponen dasar dan utilitas dari *Tailwind CSS* agar dapat digunakan dalam proyek ini. *Source Code* untuk *file* ini dapat dilihat pada Segmen Program 4.11.

Segmen Program 4.11 *Source Code Main.css*

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Lalu ada *file StrokePrediction.vue*, *file* ini adalah sebuah komponen yang dirancang untuk memberikan pengguna kemampuan untuk memprediksi risiko terkena penyakit stroke berdasarkan informasi yang dimasukkan ke dalam formulir. Validasi dilakukan untuk memastikan bahwa semua data yang diperlukan telah diisi sebelum melakukan prediksi. Hasil prediksi kemudian ditampilkan kepada pengguna. Komponen ini terdiri dari *HTML template*, *script Vue*, dan memanggil *Axios* menggunakan *instance getAPI* dan *endpoint "/stroke"* untuk berkomunikasi dengan *back-end API*. *Source Code* untuk *file* ini dapat dilihat pada Segmen Program 4.12.

Segmen Program 4.12 *Source Code StrokePrediction.vue*

```
<template>
 <div class="w-full max-w-5xl px-10 py-8 m-6 sm:m-12 bg-white rounded-lg shadow-xl">
  <div class="mx-auto space-y-6">
   <form action="">
    <h2 class="text-5xl font-bold">Prediksi Penyakit Stroke</h2>
    <p class="my-4 opacity-70">
     Aplikasi ini memungkinkan Anda untuk memprediksi kemungkinan terkena penyakit
stroke berdasarkan informasi yang Anda berikan. Silakan isi form yang disediakan dengan
detail mengenai faktor-faktor risiko yang relevan, seperti usia,
      riwayat kesehatan, gaya hidup, dan sebagainya.
    </p>
    <hr class="my-6" />
    <div class="grid lg:grid-cols-2 gap-4">
     <div>
      <label class="uppercase text-sm font-bold opacity-70">Jenis Kelamin</label>
      <select
```

```
          v-model.trim="strokedata.a"
          id="a"
          class="focus:bg-white w-full p-3 mt-2 bg-slate-200 rounded border-2 focus:border-
slate-600 focus:outline-none"
          :class="[error && error.field === 'a' ? 'border-red-500' : 'border-slate-200']"
         >
          <option value="" disabled selected>Pilih salah satu</option>
          <option value="1">Laki - Laki</option>
          <option value="0">Perempuan</option>
         </select>
         <p v-if="error && error.field === 'a'" class="text-red-500 text-sm">
          {{ error.message }}
         </p>
        </div>
        <div>
         <label class="uppercase text-sm font-bold opacity-70">Umur</label>
         <input
          v-model.number="strokedata.b"
          type="number"
          id="b"
          placeholder="0-103"
          class="focus:bg-white p-3 mt-2 w-full bg-slate-200 rounded border-2 focus:border-
slate-600 focus:outline-none"
          :class="[error && error.field === 'b' ? 'border-red-500' : 'border-slate-200']"
         />
         <p v-if="error && error.field === 'b'" class="text-red-500 text-sm">
          {{ error.message }}
         </p>
        </div>
        <div>
         <label class="uppercase text-sm font-bold opacity-70">Ada Riwayat Hipertensi?</label>
         <select
          v-model.trim="strokedata.c"
          id="c"
          class="focus:bg-white w-full p-3 mt-2 bg-slate-200 rounded border-2 focus:border-
slate-600 focus:outline-none"
          :class="[error && error.field === 'c' ? 'border-red-500' : 'border-slate-200']"
         >
          <option value="" disabled selected>Pilih salah satu</option>
          <option value="1">Ada</option>
          <option value="0">Tidak ada</option>
         </select>
         <p v-if="error && error.field === 'c'" class="text-red-500 text-sm">
          {{ error.message }}
         </p>
        </div>
        <div>
         <label class="uppercase text-sm font-bold opacity-70">Ada Riwayat Penyakit
Jantung?</label>
```

```
          <select
            v-model.trim="strokedata.d"
            id="d"
            class="focus:bg-white w-full p-3 mt-2 bg-slate-200 rounded border-2 focus:border-
slate-600 focus:outline-none"
            :class="[error && error.field === 'd' ? 'border-red-500' : 'border-slate-200']"
          >
            <option value="" disabled selected>Pilih salah satu</option>
            <option value="1">Ada</option>
            <option value="0">Tidak Ada</option>
          </select>
          <p v-if="error && error.field === 'd'" class="text-red-500 text-sm">
            {{ error.message }}
          </p>
        </div>
        <div>
          <label class="uppercase text-sm font-bold opacity-70">Kadar Gula (MG)</label>
          <input
            v-model.number="strokedata.e"
            type="number"
            id="e"
            placeholder="55-272"
            class="focus:bg-white p-3 mt-2 w-full bg-slate-200 rounded border-2 focus:border-
slate-600 focus:outline-none"
            :class="[error && error.field === 'e' ? 'border-red-500' : 'border-slate-200']"
          />
          <p v-if="error && error.field === 'e'" class="text-red-500 text-sm">
            {{ error.message }}
          </p>
        </div>
        <div>
          <label class="uppercase text-sm font-bold opacity-70">BMI</label>
          <input
            v-model.number="strokedata.f"
            type="number"
            id="f"
            placeholder="11-92"
            class="focus:bg-white p-3 mt-2 w-full rounded border-2 bg-slate-200 focus:border-
slate-600 focus:outline-none"
            :class="[error && error.field === 'f' ? 'border-red-500' : 'border-slate-200']"
          />
          <p v-if="error && error.field === 'f'" class="text-red-500 text-sm">
            {{ error.message }}
          </p>
        </div>
        <div>
          <label class="uppercase text-sm font-bold opacity-70">Apakah Anda Merokok?</label>
          <select
            v-model.trim="strokedata.g"
```

```
          id="g"
          class="focus:bg-white w-full p-3 mt-2 bg-slate-200 rounded border-2 focus:border-
slate-600 focus:outline-none"
          :class="[error && error.field === 'g' ? 'border-red-500' : 'border-slate-200']"
         >
          <option value="" disabled selected>Pilih salah satu</option>
          <option value="1">Iya</option>
          <option value="0">Tidak</option>
         </select>
         <p v-if="error && error.field === 'g'" class="text-red-500 text-sm">
          {{ error.message }}
         </p>
        </div>
       </div>

       <input @click="predict" type="button" class="py-3 px-6 my-5 bg-emerald-500 text-white
font-medium rounded hover:bg-indigo-500 cursor-pointer ease-in-out duration-300"
value="Predict" />

       <!-- RANDOM FOREST -->
       <div class="grid lg:grid-cols-2 gap-3 p-4 border-2" v-if="APIResult_rf !== null &&
APIResult_rf !== undefined">
        <p class="text-lg font-bold lg:col-span-2">Random Forest</p>
        <div>
         <div class="flex justify-between mb-1">
          <span class="text-base font-medium text-yellow-950 dark:text-white">Stroke
Probability</span>
          <span class="text-sm font-medium text-yellow-950 dark:text-white">{{
APIResult_rf_stroke_prob }}%</span>
         </div>
         <div class="w-full bg-gray-200 rounded-full h-3.5 dark:bg-gray-700">
          <div class="bg-yellow-950 h-3.5 rounded-full" :style="{ width:
APIResult_rf_stroke_prob + '%', transition: 'width 0.8s ease-in-out' }"></div>
         </div>
        </div>
        <div>
         <div class="flex justify-between mb-1">
          <span class="text-base font-medium text-yellow-950 dark:text-white">Not Stroke
Probability</span>
          <span class="text-sm font-medium text-yellow-950 dark:text-white">{{
APIResult_rf_not_stroke_prob }}%</span>
         </div>
         <div class="w-full bg-gray-200 rounded-full h-3.5 dark:bg-gray-700">
          <div class="bg-yellow-950 h-3.5 rounded-full" :style="{ width:
APIResult_rf_not_stroke_prob + '%', transition: 'width 0.8s ease-in-out' }"></div>
         </div>
        </div>
        <p class="text-lg font-bold lg:col-span-2">
```

```
        Berdasarkan hasil probabilitas diatas, maka kemungkinan pasien <b>{{ APIResult_rf
}}</b> lebih tinggi.
        </p>
      </div>
    </form>
   </div>
  </div>
</template>

<style></style>

<script>
import { getAPI } from '@/axios';
export default {
  name: 'Posts',
  data() {
    return {
      strokedata: {
        a: '',
        b: '',
        c: '',
        d: '',
        e: '',
        f: '',
        g: '',
      },
      APIResult_rf: undefined,
      APIResult_rf_not_stroke_prob: undefined,
      APIResult_rf_stroke_prob: undefined,
      error: null,
    };
  },
  methods: {
    refreshPage() {
      location.reload();
    },
    predict() {
      if (this.strokedata.a == '') {
        this.error = {
          field: 'a',
          message: 'Pilih salah satu',
        };
        return;
      }
      if (this.strokedata.b == '') {
        this.error = {
          field: 'b',
          message: 'Umur tidak boleh kosong',
        };
```

```
  return;
 }
 if (this.strokedata.b < 0) {
  this.error = {
   field: 'b',
   message: 'Umur tidak boleh minus',
  };
  return;
 }
 if (this.strokedata.b % 1 !== 0) {
  this.error = {
   field: 'b',
   message: 'Umur tidak boleh koma',
  };
  return;
 }
 if (this.strokedata.b > 103) {
  this.error = {
   field: 'b',
   message: 'Isi umur antara 1-103',
  };
  return;
 }
 if (this.strokedata.c == '') {
  this.error = {
   field: 'c',
   message: 'Pilih salah satu',
  };
  return;
 }
 if (this.strokedata.d == '') {
  this.error = {
   field: 'd',
   message: 'Pilih salah satu',
  };
  return;
 }
 if (this.strokedata.e == '') {
  this.error = {
   field: 'e',
   message: 'Kadar gula tidak boleh kosong',
  };
  return;
 }
 if (this.strokedata.e < 0) {
  this.error = {
   field: 'e',
   message: 'Kadar gula tidak boleh minus',
  };
```

```
  return;
 }
 if (this.strokedata.e < 55 || this.strokedata.e > 272) {
  this.error = {
   field: 'e',
   message: 'Isi kadar gula antara 55-272',
  };
  return;
 }
 if (this.strokedata.f == '') {
  this.error = {
   field: 'f',
   message: 'BMI tidak boleh kosong',
  };
  return;
 }
 if (this.strokedata.f < 0) {
  this.error = {
   field: 'f',
   message: 'BMI tidak boleh minus',
  };
  return;
 }
 if (this.strokedata.f < 11 || this.strokedata.f > 92) {
  this.error = {
   field: 'f',
   message: 'Isi BMI antara 11-92',
  };
  return;
 }
 if (this.strokedata.g == '') {
  this.error = {
   field: 'g',
   message: 'Pilih salah satu',
  };
  return;
 }
 this.error = null;
 getAPI
  .get('/stroke', {
   params: {
    sex: this.strokedata.a,
    age: this.strokedata.b,
    hypertension: this.strokedata.c,
    heart_disease: this.strokedata.d,
    avg_glucose_level: this.strokedata.e,
    bmi: this.strokedata.f,
    smoking_status: this.strokedata.g,
   },
```

```
      })
      .then((response) => {
        this.APIResult_rf = response.data.prediction_result_rf;
        this.APIResult_rf_not_stroke_prob =
Math.round(response.data.prediction_proba_rf[0][0] * 100);
        this.APIResult_rf_stroke_prob = Math.round(response.data.prediction_proba_rf[0][1] *
100);
        console.log('Recieved data successfully');
        console.log(response);
        console.log(response.data);
        console.log(response.data.prediction_proba_rf[0][0]);
        console.log(response.data.prediction_proba_rf[0][1]);
        console.log(response.data.prediction_result_rf);
      })
      .catch((err) => {
        console.log(err);
      });
   },
  },
};
</script>
```

Selanjutnya ada *file Home.vue*, *file* ini adalah sebuah halaman utama *website* yang berisi komponen yang telah dibuat sebelumnya. *File* ini memanggil komponen *StrokePrediction.vue* agar dapat menampilkan formulir yang telah dibuat pada halaman. Secara garis besar, *file* ini berguna sebagai wadah/kanvas untuk menampilkan komponen yang dipanggil. *Source Code* untuk *file* ini dapat dilihat pada Segmen Program 4.13.

Segmen Program 4.13 *Source Code Home.vue*

```
<script setup>
import StrokePrediction from '../components/StrokePrediction.vue';
</script>

<template>
  <div class="container-fluid">
    <div class="flex w-full items-center justify-center min-h-screen from-teal-100 via-teal-300 to-
teal-500 bg-gradient-to-br">
      <StrokePrediction />
    </div>
  </div>
</template>
```

Lalu ada *file App.vue*, *file* ini merupakan *file* penting dalam proyek *vue*. *File* ini berguna untuk memanggil dan mengatur halaman yang telah dibuat sebelumnya. Pada proyek ini hanya

ada 1 halaman yang dibuat yaitu *Home.vue*. Maka *file* ini akan memanggil halaman *Home.vue* sekaligus menjadikan halaman tersebut menjadi halaman utama. *Source Code* untuk file ini dapat dilihat pada Segmen Program 4.14.

Segmen Program 4.14 *Source Code App.vue*

```
<script setup>
import Home from './views/Home.vue';
</script>

<template>
  <Home />
</template>
```

Terakhir ada *file Index.html*, *file* ini merupakan titik utama dari pembuatan *website*. File ini berguna untuk memberikan struktur dasar yang diperlukan untuk menjalankan dan menampilkan aplikasi yang telah dibuat ke dalam *browser*. *Source Code* untuk *file* ini dapat dilihat pada Segmen Program 4.15.

Segmen Program 4.15 *Source Code Index.html*

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" href="/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite App</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.js"></script>
  </body>
</html>
```

4.3.2 Bagian *Back-End Website*

Pada tahap ini, dilakukan proses pembuatan bagian *back-end* dari *website*. *Bagian back-end* dari *website* ini menggunakan *Flask* dari Bahasa pemgrograman *python*. Serta menggunakan *library-library* yang diperlukan seperti *joblib, numpy,* dan *warnings*.

File bagian *back-end website* seluruhnya ada di dalam *file app.py*. *File* ini berisi kode dengan Bahasa pemgrograman *python* dengan *library-library* yang diperlukan yaitu *Flask* untuk *framework web, numpy, joblib* dan *warnings*. Secara garis besar, kode di dalam *file* ini berfungsi untuk menerima data yang dikirimkan oleh sisi *front-end* lalu kemudian data-data tersebut diolah dan diproses menggunakan model *machine learning* yang di panggil. Pada skripsi ini, data dari pengguna diolah menjadi bentuk *array* menggunakan *numpy*, lalu juga di *reshape* menjadi bentuk *array* 2 dimensi, selanjutnya data tersebut akan melalui proses *data scaling* menggunakan *MinMaxScaler*. Setelah proses pengolahan data selesai maka dilakukan proses prediksi menggunakan model *machine learning* yang dipaggil, dan hasil prediksi yang didapatkan dikirim kembali ke sisi *front-end*. *Source Code* untuk *file* ini dapat dilihat pada Segmen Program 4.16.

Segmen Program 4.16 *Source Code app.py*

```
from flask import Flask, jsonify, request
from flask_cors import CORS
import numpy as np
import joblib
import warnings
warnings.filterwarnings("ignore")

app = Flask(__name__)

cors = CORS(app)
app.config['CORS_HEADERS'] = 'Content-Type'

filename_scaler = './src/model machine learning/MinMaxScaler.save'
scaler = joblib.load(filename_scaler)
filename_rf = './src/model machine learning/random_forest_model.save'
filename_rf_tuned = './src/model machine learning/random_forest_model_tuned.save'

@app.route("/stroke", methods=["GET"])
def strokepred():
   if request.method == 'GET':
      a = float(request.args.get('sex'))
      b = float(request.args.get('age'))
      c = float(request.args.get('hypertension'))
      d = float(request.args.get('heart_disease'))
      e = float(request.args.get('avg_glucose_level'))
      f = float(request.args.get('bmi'))
      g = float(request.args.get('smoking_status'))
```

```
    final_features = ([[a, b, c, d, e, f, g]])
    input_data_as_numpy_array = np.array(final_features)
    input_data_reshape = input_data_as_numpy_array.reshape(1,-1)
    sample_scale = scaler.transform(input_data_reshape)
    print("final_features no scale: ", final_features)
    print("input_data_as_numpy_array", input_data_as_numpy_array)
    print("input_data_reshape", input_data_reshape)
    print("sample_scale : ", sample_scale)

    model_rf = joblib.load(filename_rf_tuned)

    prediction_rf = model_rf.predict(sample_scale)
    prediction_proba_rf = model_rf.predict_proba(sample_scale)

    if prediction_rf[0] == 0:
      result_rf = str("Tidak Stroke")
    if prediction_rf[0] == 1:
      result_rf = str("Stroke")

    response = {
      "prediction_result_rf": result_rf,
      "prediction_proba_rf": prediction_proba_rf.tolist(),  # Convert to list for JSON
serialization
    }

    print("Hasil Prediksi Random Forest :", result_rf)
    return jsonify(response)

if __name__ == '__main__':
  app.run(debug=True)
```

4.3.3 Interaksi *API*

Pada tahap ini, dilakukan interaksi *API* menggunakan *Axios JS* untuk menghubungkan bagian *front-end* dan *back-end website*. Sehingga bagian *front-end* dapat membuat permintaan *HTTP* dan dapat menanggapi data yang diterima dari bagian *back-end*.

Konfigurasi interaksi *API* dalam skripsi ini ada di dalam *file Axios.js*. *File* ini berisi *instance getAPI* yang mencakup *baseURL* yang ditetapkan yaitu *'http://127.0.0.1:5000'* dengan *timeout* sebesar 1000 milidetik, *URL* ini merupakan *URL* dasar tempat *back-end Flask* berjalan. *Instance getAPI* tersebut diekspor agar dapat digunakan. Dengan menggunakan *getAPI*, proyek dapat dengan mudah berkomunikasi dengan bagian *back-end Flask*. *Source Code* untuk *file* ini dapat dilihat pada Segmen Program 4.17.

Segmen Program 4.17 *Source Code Axios.js*

```
import axios from 'axios';

const getAPI = axios.create({
  baseURL: 'http://127.0.0.1:5000',
  timeout: 1000,
});

export { getAPI };
```