BIG DATA PROGRAMMING

HUSH - HUSH RECRUITER

DATE : 11 / 3 / 2024

TEAM 3 :
Shashank Barai
Aditya Dorle
Praneet Patnaik
Amogh Goudar
Rajat Pundir

DATA FETCHED FROM

TIMELINE

Login Page

Created a login page for decision-maker, hiring manager, and candidates using HTML and CSS

Decision-Maker Page

A page where the decision maker will enter the company's requirements for the hiring manager to fetch suitable candidates.

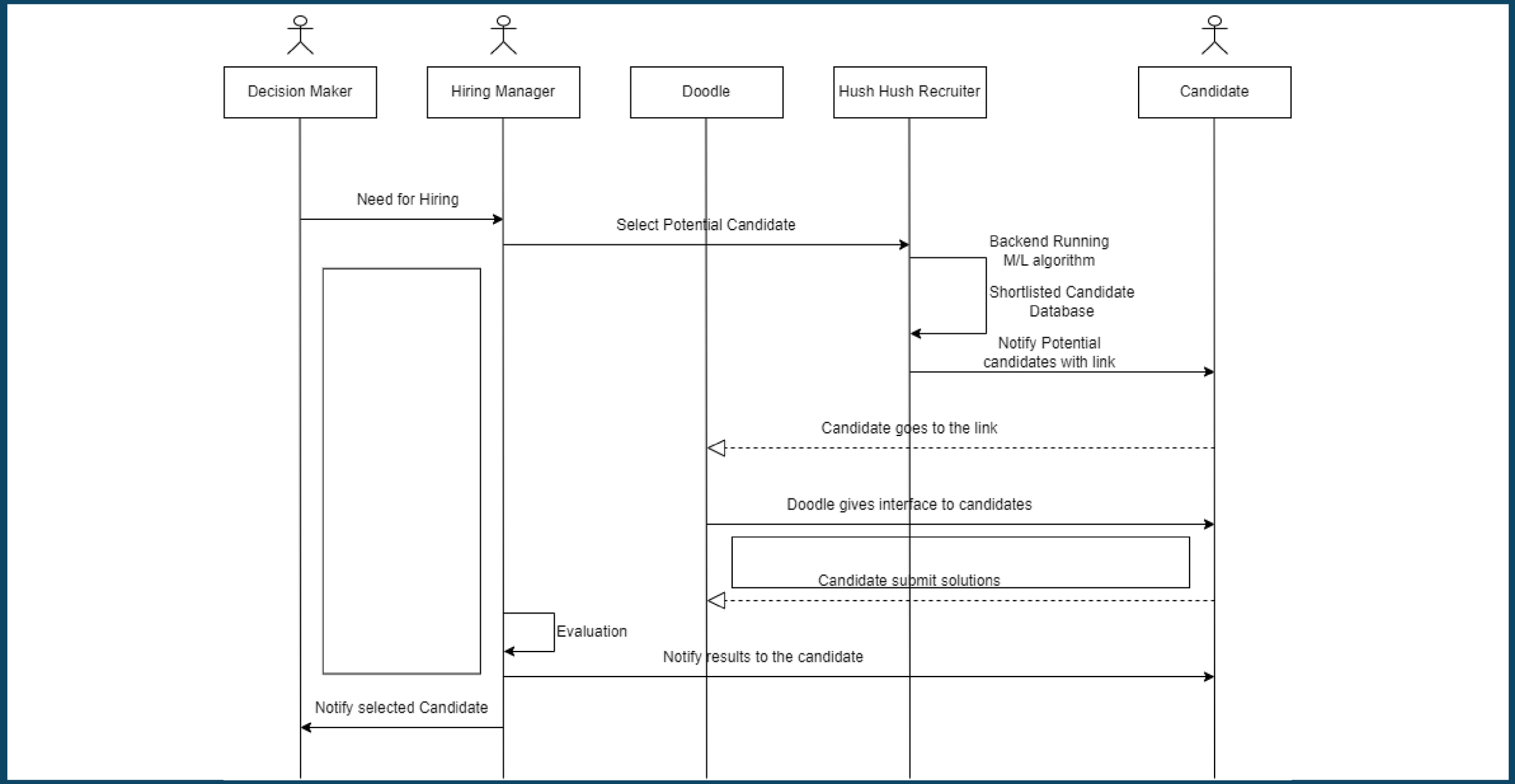
Hiring-Manager Page

Here the hiring manager fetches the potential candidates and an email is sent to them with a coding test link and the results of candidates to decision maker

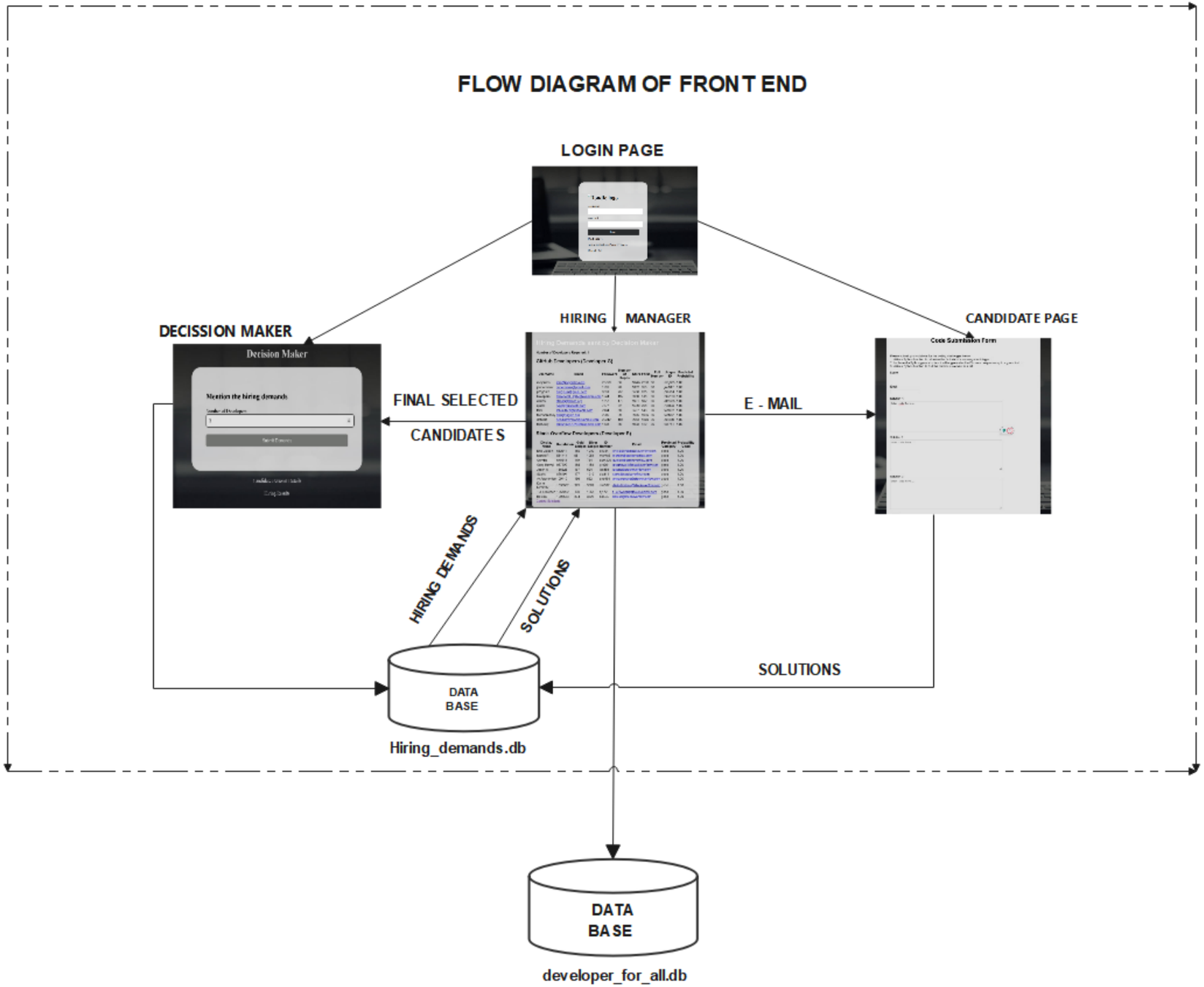
Candidate coding platform

The coding platform opens up from the link that the hiring manager sent and submit the codes which the hiring manager is able to see

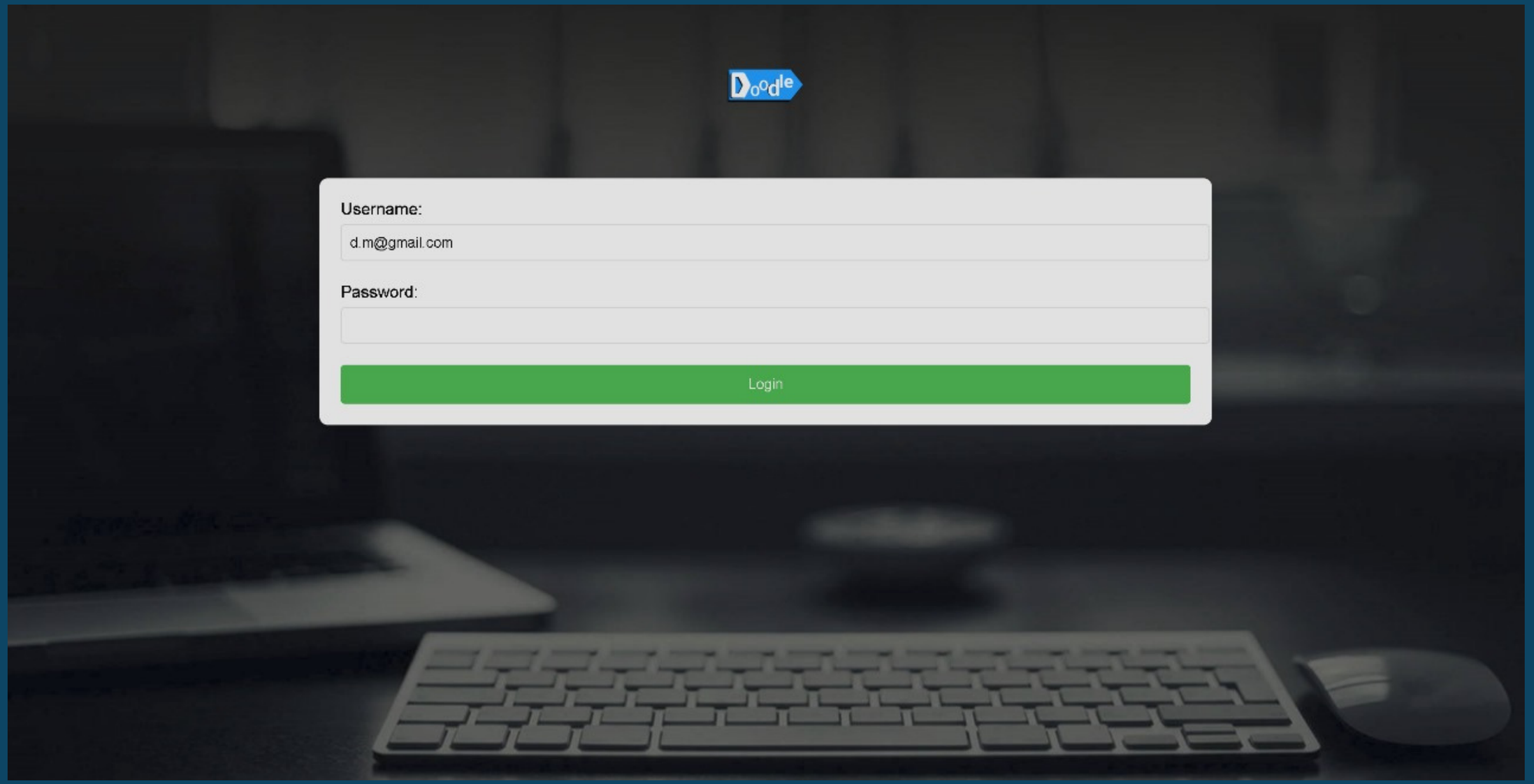
UPDATED SEQUENCE DIAGRAM



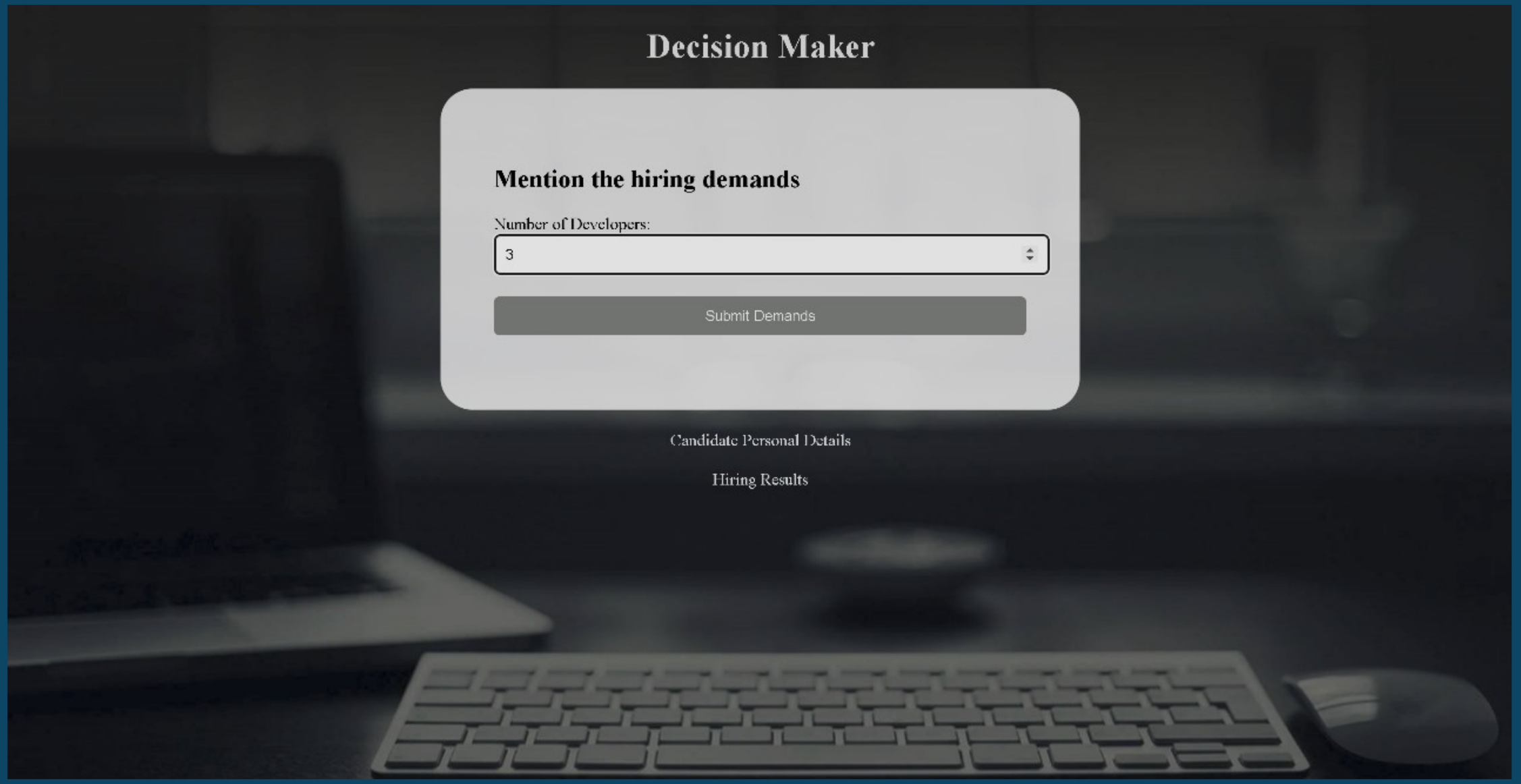
FLOW DIAGRAM OF FRONT END



DOODLE - LOGIN PAGE



DICISON MAKER PAGE



HIRING MANAGER PAGE

Hiring Demands sent by Decision Maker

Number of Developers Required: 1

GitHub Developers (Developer G)

Username	Email	Followers	Number of Repos	Stars	Forks	Pull Number	Unique ID	Predicted Probability
mojombo	tom@mojombo.com	23799	66	10262	2296	30	moj865	1.00
jnunemaker	nunemaker@gmail.com	1952	76	1662	353	30	jnu947	1.00
progrium	progrium@gmail.com	3631	250	3538	628	30	pro254	1.00
bkeepers	bkeepersLUq8V@example.com	1944	192	6660	516	30	bke120	1.00
atmos	atmos@atmos.org	1318	170	1901	502	30	atm355	1.00
ryanb	ryan@railscasts.com	7677	72	15098	2581	30	rya393	1.00
tobi	tobiuu9VX@example.com	2184	59	3507	1431	30	tob032	1.00
technomancy	phil@hagelb.org	2089	99	3675	1035	30	tec862	1.00
defunkt	defunktrwmQ2@example.com	22242	107	8663	1546	30	def580	1.00
topfunky	topfunkykOOKR@example.com	1333	86	1998	420	30	top751	1.00

Stack Overflow Developers (Developer S)

Display Name	Reputation	Gold Badges	Silver Badges	ID Number	Email	Predicted Category	Probability Good
Eric Lippert	652811	180	1249	eri384	ericlippert@stackoverflow.com	good	1.00
Martin R	534414	95	1251	mar485	martinr@stackoverflow.com	good	1.00
Gumbo	648341	110	784	gum300	gumbo@stackoverflow.com	good	1.00
Greg Hewgill	967802	186	1158	gre091	greghewgill@stackoverflow.com	good	1.00
Jason S	186628	167	624	jas358	jasons@stackoverflow.com	good	1.00
SLaks	876194	177	1919	sla314	slaks@stackoverflow.com	good	1.00
AnApprentice	109410	198	632	ana968	anapprentice@stackoverflow.com	good	1.00
Darin Dimitrov	1030032	273	3302	dar532	darindimitrov@stackoverflow.com	good	1.00
T.J. Crowder	1051082	193	1955	t.j172	t.j.crowder@stackoverflow.com	good	1.00
BalusC	1093553	374	3626	bal040	balusc@stackoverflow.com	good	1.00

Coding Solutions

Questions

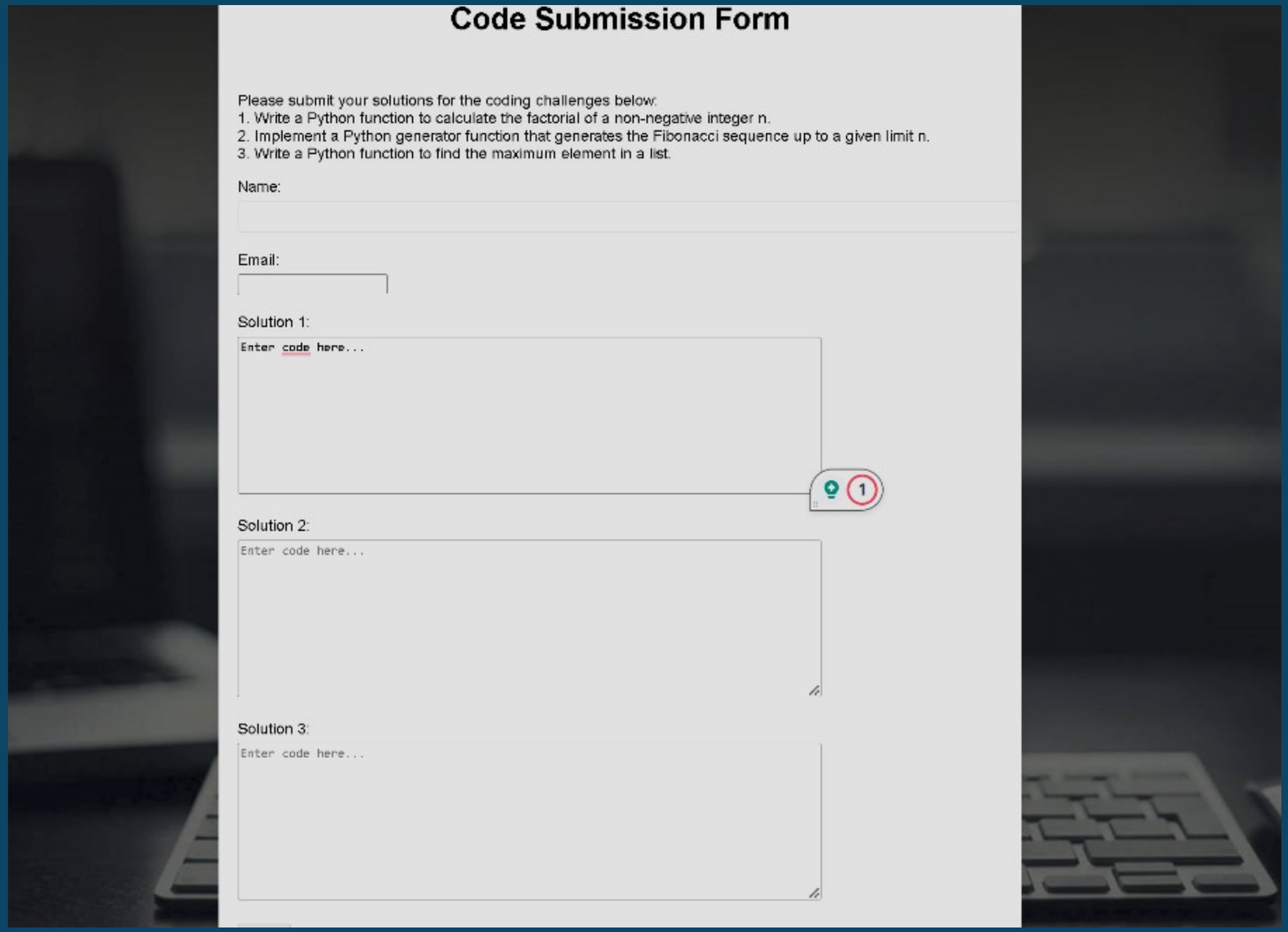
- Write a Python function to calculate the factorial of a non-negative integer n.
- Implement a Python generator function that generates the Fibonacci sequence up to a given limit n.
- Write a Python function to find the maximum element in a list.

Solutions

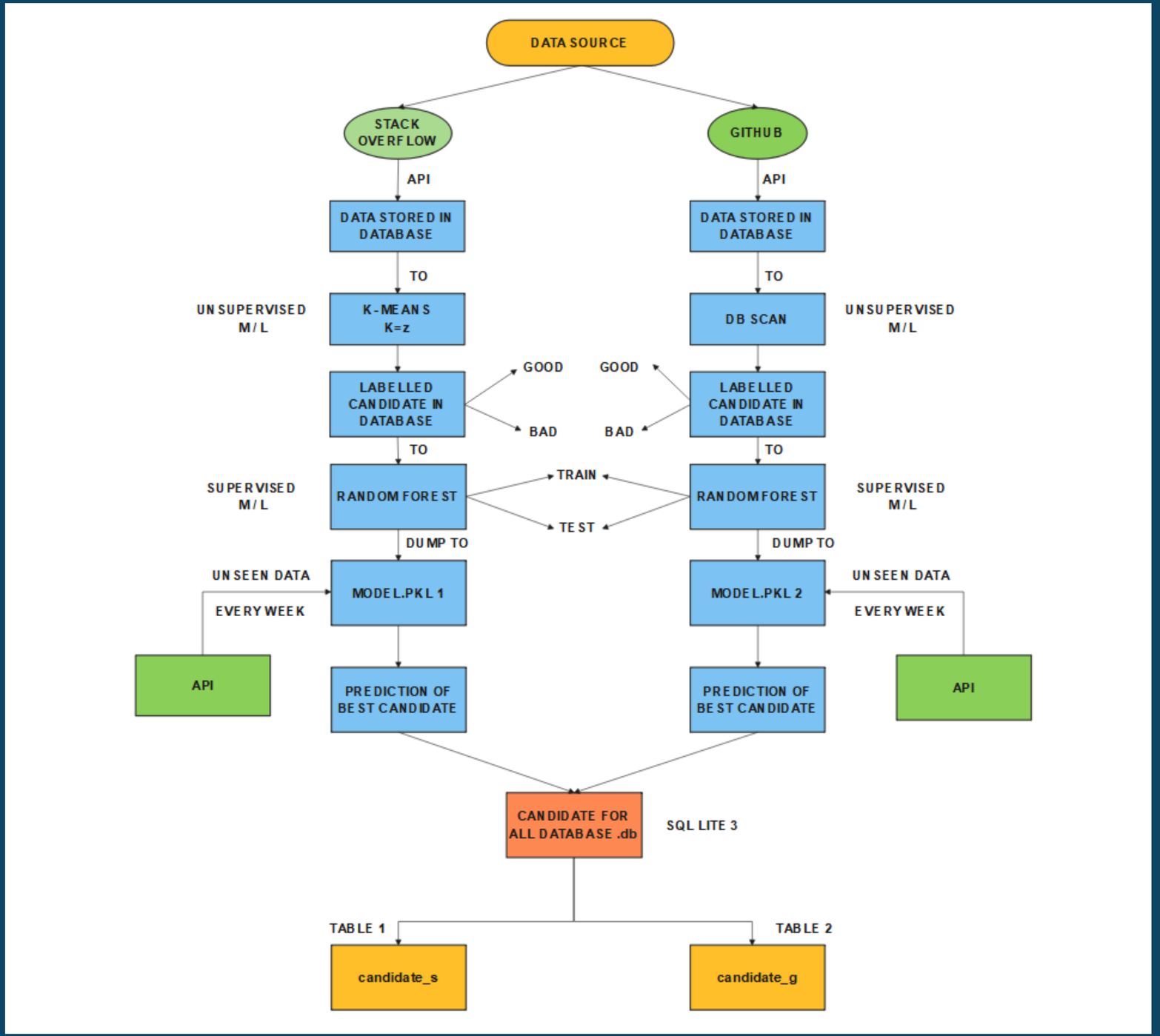
ID	Name	Email	Solution 1	Solution 2	Solution 3	Selected(Y/N)	Score
9	Praneet Patnaik	patnaik.praneet2001@gmail.com	g	g	g	Y	98
10	Praneet	patnaik.praneet@gmail.com	sd	sqdw	rfed	N	90
12	Patnaik	patnaik.praneet@rediff.com	some func()	func2()	func3()		

Submit Evaluations

CANDIDATE PAGE



BACKEND FLOW PROCESS

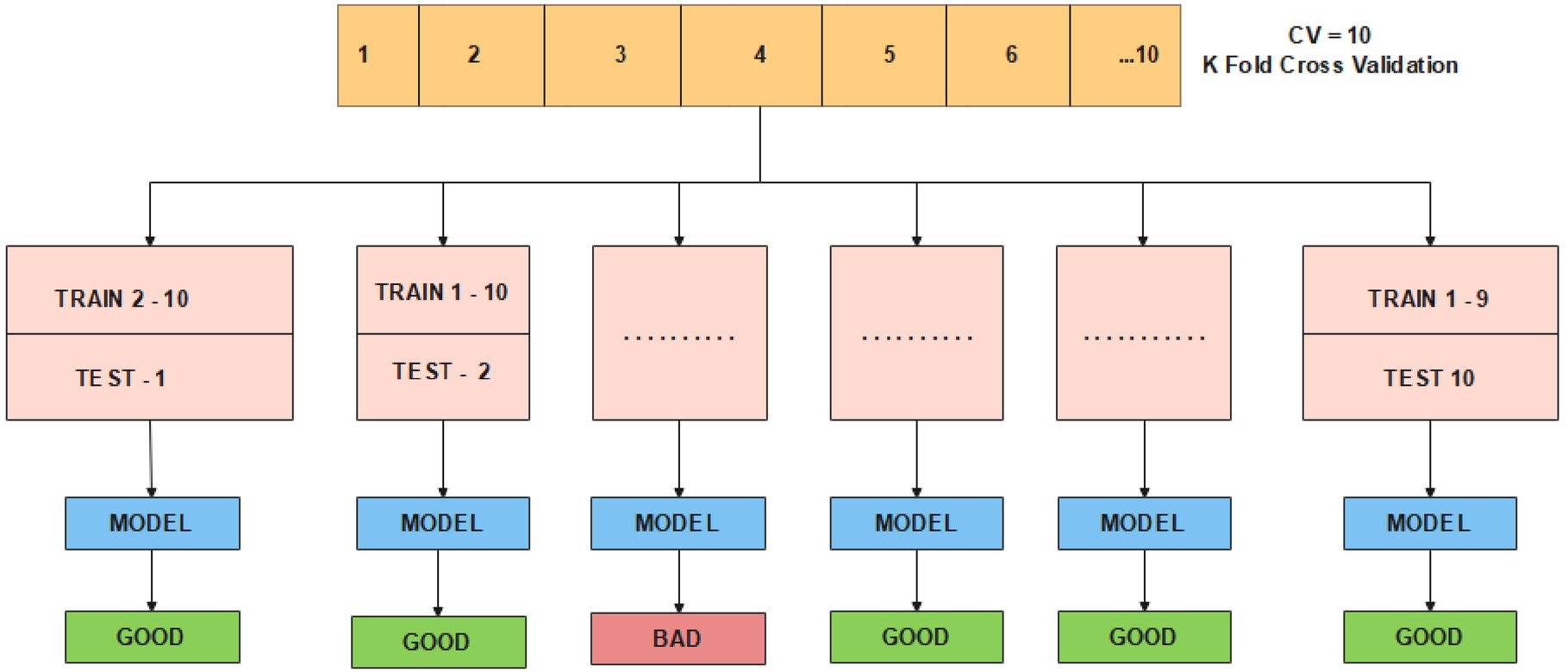


RANDOM FOREST MODEL

(Github cadidate selection)

MAIN DATA SET

	username	e- mail	follower	repo's	stars	forks	pullno
1							
2							
3							
4							

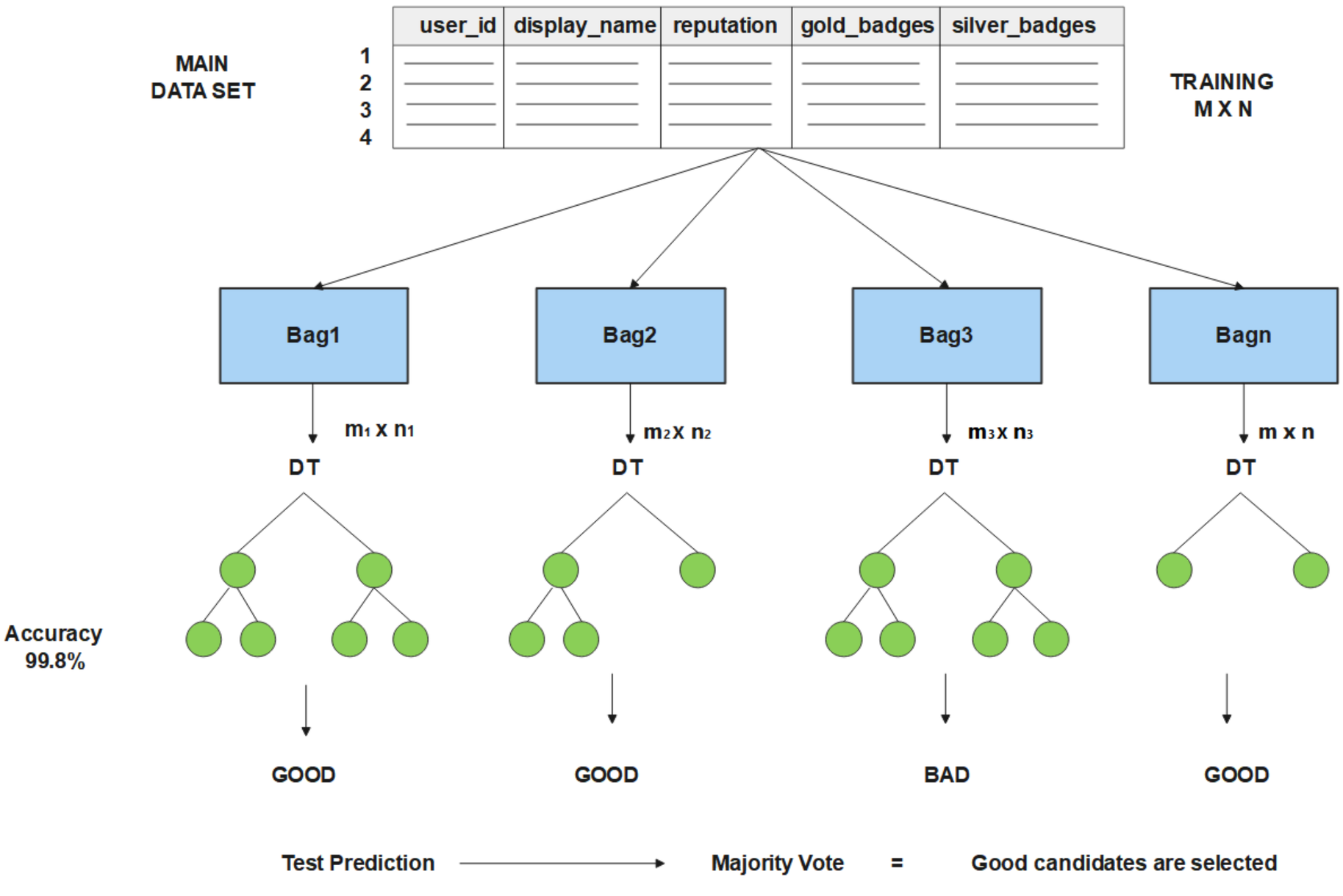


CV SCORE = 98.89 %

Test Pridiction ⟶ Majority Vote = Good Candidate Selected

RANDOM FOREST MODEL

(Stackover flow cadidate selection)



STACKOVERFLOW API DATA FETCH


```
import requests
import time
import pandas as pd
import sqlite3
```

```
api_url = "https://api.stackexchange.com/2.3/users"
```

Users

All user methods that take an `{ids}` parameter have a `/me` equivalent method that takes an `access_token` instead. These methods are provided for developer convenience, with the exception of plain /me, which is actually necessary for discovering which user authenticated to an application.

users	Get all users on the site.
users/{ids} /me	Get the users identified by a set of ids.
users/{ids}/answers /me/answers	Get the answers posted by the users identified by a set of ids.

Request Library using Request.get

```
user_id': user['user_id'],
        'display_name': user['display_name'],
        'reputation': user['reputation'],
        'gold_badges': user['badge_counts']['gold'],
        'silver_badges': user['badge_counts']['silver']
```

K - MEANS CLUSTERING


```
import sqlite3
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler


# Apply KMeans algorithm
    kmeans = KMeans(n_clusters=2, random_state=42)  # Assuming we want
to distinguish between 2 categories: good and bad
    kmeans.fit(df[['reputation', 'gold_badges', 'silver_badges']])

    # Calculate silhouette score
    silhouette_avg = silhouette_score(df[['reputation', 'gold_badges',
'silver_badges']], kmeans.labels_)
    print("Silhouette Score:", silhouette_avg)  # Print silhouette score


# Add cluster labels to the DataFrame
    df['cluster_label'] = kmeans.labels_

    # Calculate cluster centroids
    centroids = pd.DataFrame(kmeans.cluster_centers_,
columns=['reputation', 'gold_badges', 'silver_badges'])
```

```
Silhouette Score: 0.8058135534295623

Centroids:
       reputation  gold_badges  silver_badges
0   112111.569935    34.287582     201.149891
1   487783.658537   103.439024     836.512195

Good Criteria:
reputation       1
gold_badges      1
silver_badges    1
dtype: int64

Bad Criteria:
reputation       0
gold_badges      0
silver_badges    0
dtype: int64
```

RANDOM FOREST FOR STACKOVRLOW CANDIDATE SELECTION


```
import sqlite3
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import joblib
```

Code snippet

```
# Separate features (X) and target variable (y)
X = df[['reputation', 'gold_badges', 'silver_badges']]
y = df['candidate_type']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

random_forest_model = RandomForestClassifier(n_estimators=100,
random_state=42)
random_forest_model.fit(X_train_scaled, y_train)

# Save the trained model using joblib
joblib.dump(random_forest_model, 'random_forest_model.pkl')
```

```
Accuracy: 0.998

Classification Report:
              precision    recall  f1-score   support

         bad       1.00      1.00      1.00       471
        good       0.97      1.00      0.98        29

    accuracy                           1.00       500
   macro avg       0.98      1.00      0.99       500
weighted avg       1.00      1.00      1.00       500
```

Predcition With random_forest_model.pkl

```
X_unseen = unseen_df[['reputation', 'gold_badges', 'silver_badges']]
X_unseen_scaled = scaler.fit_transform(X_unseen)

# Make predictions on the unseen data
predictions = model.predict(X_unseen_scaled)

# Get probabilities of being "good"
probabilities = model.predict_proba(X_unseen_scaled)[:, 1]
```

GITHUB API FETH DATA & DBSCAN

Github API Fetch Data

```
import requests
import time
import json
```

API Link

```
https://api.github.com/users"
```

```
"username": user_data["login"],
            "email": user_data.get("email", "N/A"),
            "followers": user_data.get("followers", "N/A"),
            "number_of_repos": user_data.get("public_repos", "N/A"),
            "stars": sum([d["stargazers_count"] for d in repos]),
            "forks": sum([r['forks'] for r in repos]),
            "pull_number": len([r for r in repos if 'permissions' in r
and 'push' in r['permissions']])
```

Dbscan

Packages and Libraries

```
import sqlite3
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
from sklearn.model_selection import ParameterGrid
```

Code snippet

```
# Define parameter grid
param_grid = {'eps': [0.1, 0.5, 1.0, 1.5],
              'min_samples': [3, 5, 10, 15]}

best_score = -1
best_params = None

# Perform grid search
for params in ParameterGrid(param_grid):
    dbscan = DBSCAN(**params)
    cluster_labels = dbscan.fit_predict(X_scaled)
    silhouette_avg = silhouette_score(X_scaled, cluster_labels)
    if silhouette_avg > best_score:
        best_score = silhouette_avg
        best_params = params

# Perform DBSCAN clustering with best parameters
dbscan = DBSCAN(**best_params)
df['cluster'] = dbscan.fit_predict(X_scaled)
```

```
Best Silhouette Score: 0.7900454311092708
```

Labelling

```
df['category'] = df['cluster'].map({-1: 'good', 0: 'bad'})
```

RANDOM FOREST FOR GITHUB CANDIDATE SELECTION

Random Forest for Github Candidate selection

```
import sqlite3
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import classification_report
import pickle
```

Code snippet

```
# Separate features and target variable
X = df[['followers', 'number_of_repos', 'stars', 'forks',
'pull_number']]  # Features
y = df['category']  # Target variable


# Initialize Random Forest classifier with best parameters from grid
search
model = RandomForestClassifier(n_estimators=50, max_depth=None,
min_samples_split=2, min_samples_leaf=2, random_state=42)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
```

```
# Train Random Forest model on the training set
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)


# Display classification report
print("Classification Report:")
print(classification_rep)

# Calculate and display average CV score
cv_size = 10
cv_scores = cross_val_score(model, X_scaled, y, cv=cv_size)
print("Average CV Score:", cv_scores.mean())
```

```
Classification Report:
              precision    recall  f1-score   support

         bad       0.97      0.98      0.97       178
        good       0.84      0.73      0.78        22

    accuracy                           0.95       200
   macro avg       0.90      0.86      0.88       200
weighted avg       0.95      0.95      0.95       200

Average CV Score: 0.9808989898989898
```

```
'saved_model_and_data.pkl'   for Prediction
```

WHAT MORE COULD HAVE BEEN DONE ?

- Integrate Docker into the project environment to improve reliability, scalability, and maintainability, enhancing the development and deployment experience for all stakeholders

- Automate the data retrieval script to run periodically, adjusting the frequency based on the rate of new candidate additions and project needs.


THANK YOU