In [26]:
```
# Import core libraries for data handling, modeling, and evaluation
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# Import scikit-learn tools for model building and evaluation
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report, cor
%matplotlib inline
```

In [27]:
```
# Load the SPECTF dataset directly from UCI ML Repository
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/spect
columns = ['Diagnosis'] + [f'F{i}' for i in range(1, 45)]
data = pd.read_csv(url, header=None, names=columns)

# Convert Diagnosis: 0 = Abnormal, 1 = Normal
data['Diagnosis'] = data['Diagnosis'].map({0: 0, 1: 1})

# Quick look at class distribution
print("Class distribution:\n", data['Diagnosis'].value_counts())
data.head()
```

```
Class distribution:
 1    40
0    40
Name: Diagnosis, dtype: int64
```

Out[27]:

	Diagnosis	F1	F2	F3	F4	F5	F6	F7	F8	F9	...	F35	F36	F37	F38	F39	F40	F41	F4
0	1	59	52	70	67	73	66	72	61	58	...	66	56	62	56	72	62	74	7
1	1	72	62	69	67	78	82	74	65	69	...	65	71	63	60	69	73	67	7
2	1	71	62	70	64	67	64	79	65	70	...	73	70	66	65	64	55	61	4
3	1	69	71	70	78	61	63	67	65	59	...	61	61	66	65	72	73	68	6
4	1	70	66	61	66	61	58	69	69	72	...	67	69	70	66	70	64	60	5

5 rows × 45 columns

In [28]:
```
# Split the features and target
X = data.drop('Diagnosis', axis=1)
y = data['Diagnosis']

# Train-test split with stratification to preserve class ratio
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
```

In [29]:
```
# Define hyperparameters for tuning
dt_params = {
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']
}

# Grid search with 5-fold cross-validation
dt_grid = GridSearchCV(DecisionTreeClassifier(random_state=42), dt_par
dt_grid.fit(X_train, y_train)

# Best estimator and predictions
dt_best = dt_grid.best_estimator_
dt_preds = dt_best.predict(X_test)
```

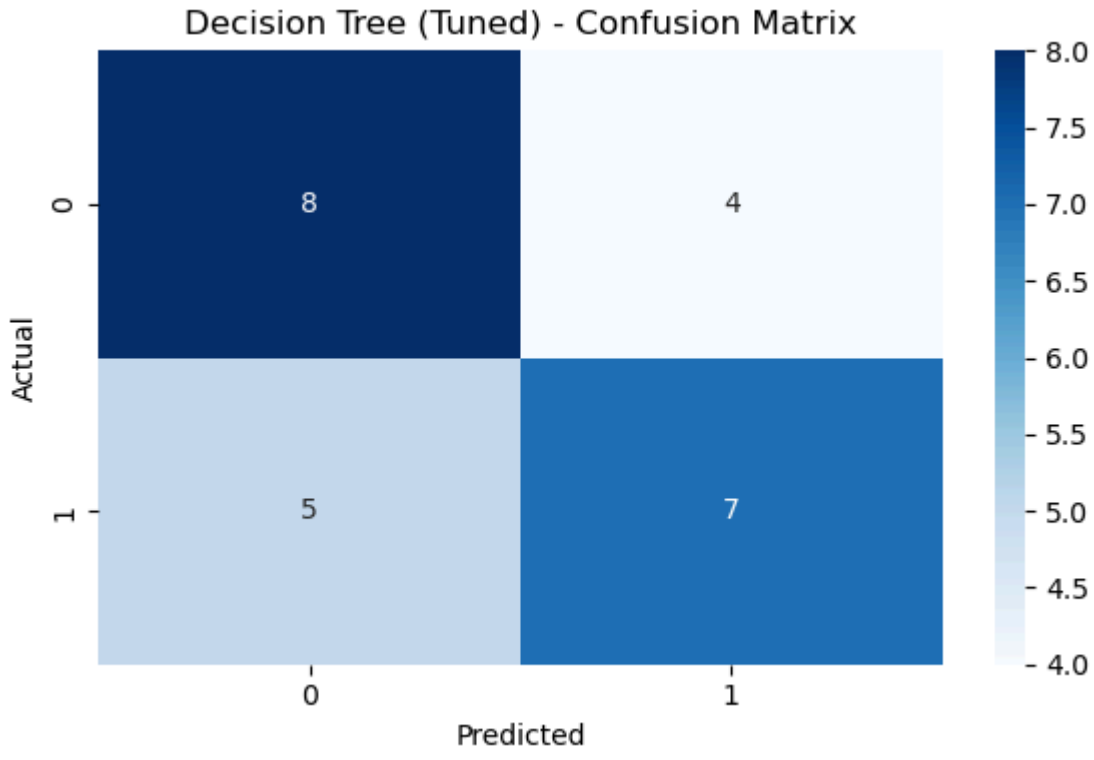
In [30]:
```
# Define hyperparameter grid
gb_params = {
    'n_estimators': [50, 100],
    'learning_rate': [0.05, 0.1],
    'max_depth': [3, 5]
}

# Grid search on Gradient Boosting
gb_grid = GridSearchCV(GradientBoostingClassifier(random_state=42), gb
gb_grid.fit(X_train, y_train)

# Best model and predictions
gb_best = gb_grid.best_estimator_
gb_preds = gb_best.predict(X_test)
```

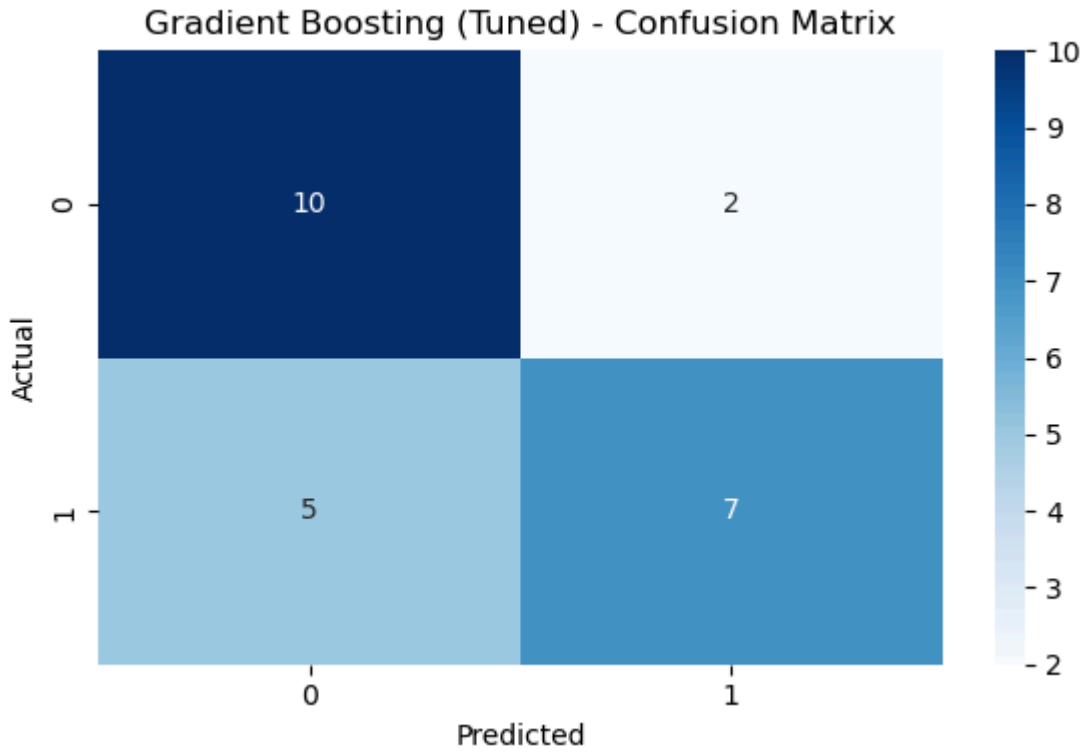
In [31]:
```
def evaluate(model_name, y_true, y_pred):
    print(f"\n📌 {model_name} Evaluation")
    print("Accuracy:", round(accuracy_score(y_true, y_pred), 3))
    print("Classification Report:\n", classification_report(y_true, y_

    # Plot confusion matrix heatmap
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 4))  # Optional: better sizing
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.title(f"{model_name} - Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.tight_layout()
```

In [32]:
```
# Evaluate Decision Tree
evaluate("Decision Tree (Tuned)", y_test, dt_preds)

# Evaluate Gradient Boosting
evaluate("Gradient Boosting (Tuned)", y_test, gb_preds)
```

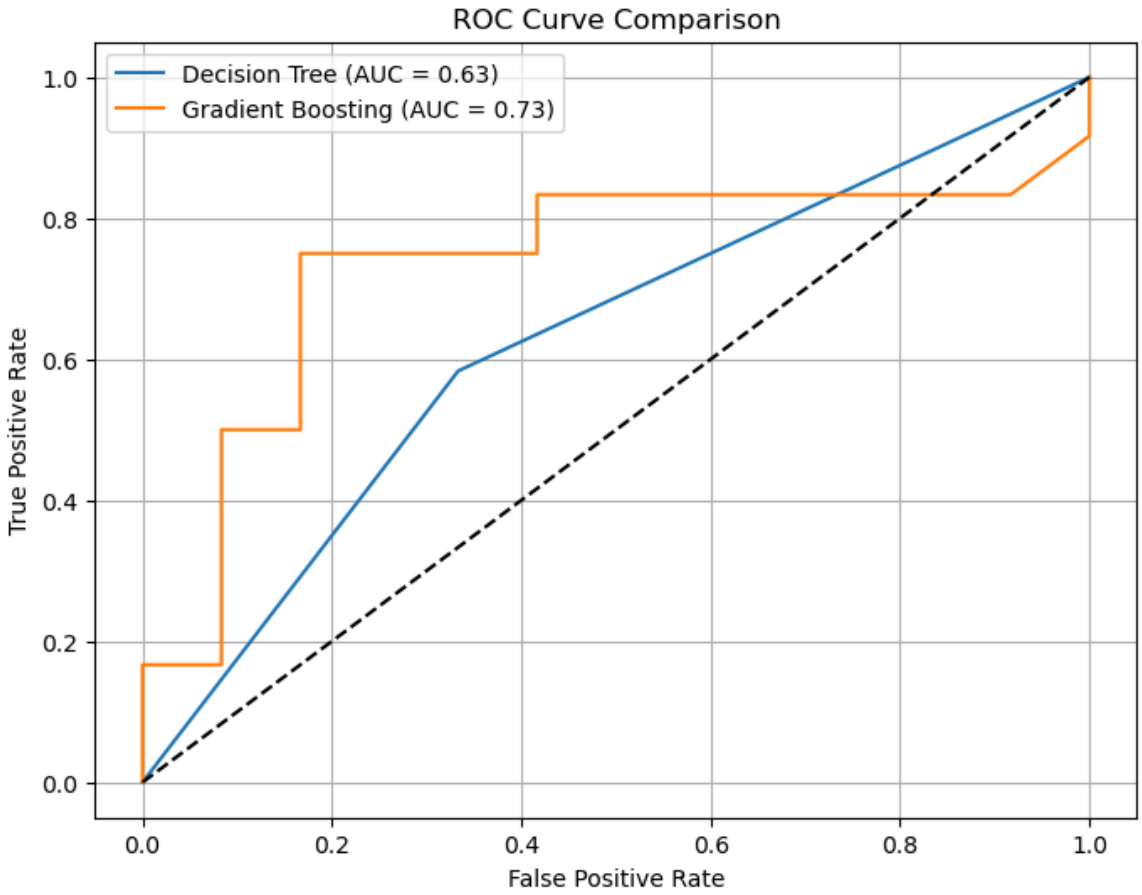
```
📌 Decision Tree (Tuned) Evaluation
Accuracy: 0.625
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.67      0.64        12
           1       0.64      0.58      0.61        12

    accuracy                           0.62        24
   macro avg       0.63      0.62      0.62        24
weighted avg       0.63      0.62      0.62        24
```

```
📌 Gradient Boosting (Tuned) Evaluation
Accuracy: 0.708
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.83      0.74        12
           1       0.78      0.58      0.67        12

    accuracy                           0.71        24
   macro avg       0.72      0.71      0.70        24
weighted avg       0.72      0.71      0.70        24
```

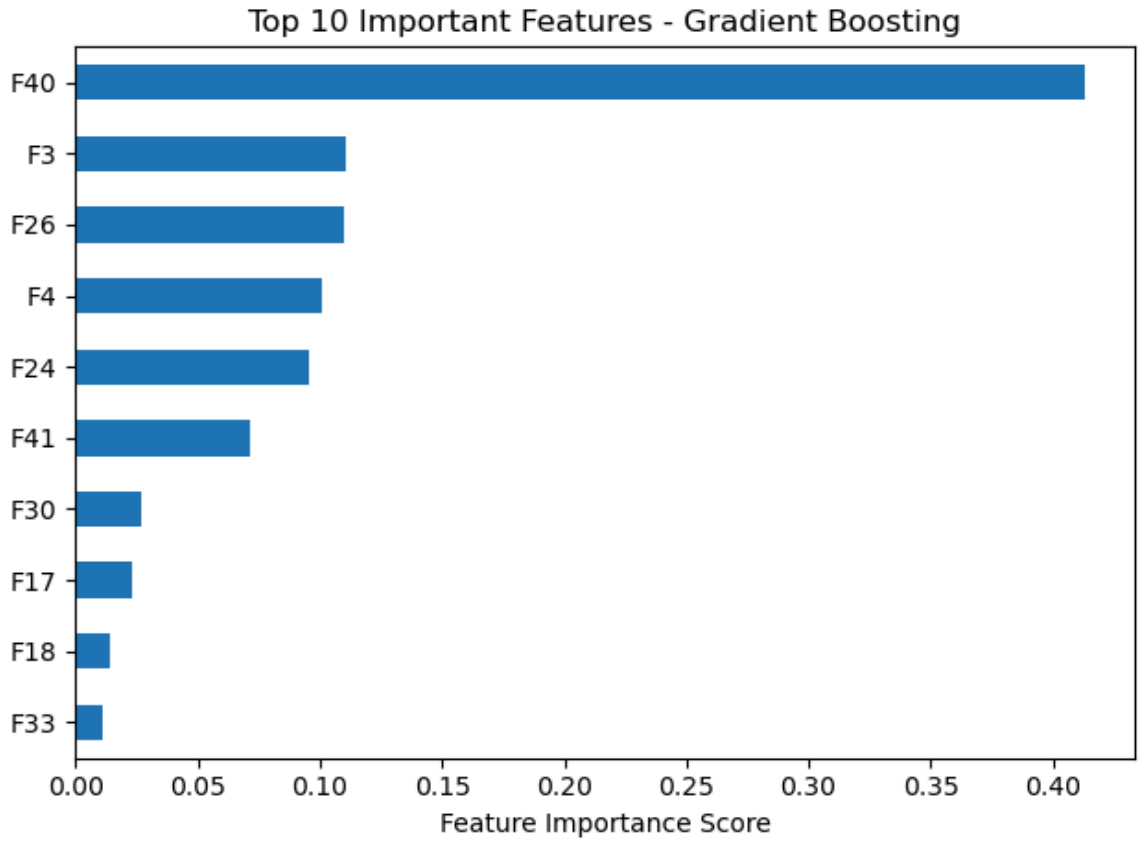



Gradient Boosting (Tuned) - Confusion Matrix
10
2
5
7
0
1
Actual
0
1
Predicted
10
9
8
7
6
5
4
3
2

In [33]:
```
# Get prediction probabilities for ROC curve
dt_probs = dt_best.predict_proba(X_test)[:, 1]
gb_probs = gb_best.predict_proba(X_test)[:, 1]

# Compute FPR/TPR for both models
fpr_dt, tpr_dt, _ = roc_curve(y_test, dt_probs)
fpr_gb, tpr_gb, _ = roc_curve(y_test, gb_probs)

# Plotting the ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_dt, tpr_dt, label=f"Decision Tree (AUC = {roc_auc_score(y
plt.plot(fpr_gb, tpr_gb, label=f"Gradient Boosting (AUC = {roc_auc_sc
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison")
plt.legend()
plt.grid()
plt.show()
```



In [34]:
```
# Visualize top 10 most important features from Gradient Boosting
importance = gb_best.feature_importances_
top_features = pd.Series(importance, index=X.columns).sort_values(asce

top_features.plot(kind='barh')
plt.title("Top 10 Important Features - Gradient Boosting")
plt.xlabel("Feature Importance Score")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```



In [35]:
```
# Create comparison summary
results = {
    'Model': ['Decision Tree', 'Gradient Boosting'],
    'Accuracy': [accuracy_score(y_test, dt_preds), accuracy_score(y_te
    'AUC Score': [roc_auc_score(y_test, dt_probs), roc_auc_score(y_tes
}

# Display as DataFrame
df_results = pd.DataFrame(results)
print(" Final Model Comparison:\n")
print(df_results)
```

```
 Final Model Comparison:

               Model  Accuracy  AUC Score
0      Decision Tree  0.625000   0.625000
1  Gradient Boosting  0.708333   0.732639
```