Backend Modules Documentation

Overview

This document covers all QML backend modules that provide data and signal interfaces between Python and QML UI components for FastReaction game.

---

1. FastReactionBackend (Active Screen)

Purpose

Backend for the active gameplay screen with real-time score and timer updates.

Main Class: `FastReactionBackend`

Inherits: `QObject`

Responsibilities:

- Update reaction score display (correct/wrong/miss counts)
- Update countdown timer (MM:SS format)
- Update total score display
- Update team name display
- Manage internal countdown timer
- Track game statistics

Signals

```
scoreChanged = pyqtSignal(str, arguments=['scoreValue'])
timeChanged = pyqtSignal(str, arguments=['timeValue'])
teamNameChanged = pyqtSignal(str, arguments=['teamName'])
correctCountChanged = pyqtSignal(int, arguments=['correctCount'])
wrongCountChanged = pyqtSignal(int, arguments=['wrongCount'])
missCountChanged = pyqtSignal(int, arguments=['missCount'])
timerValueChanged = pyqtSignal(int, float, arguments=['seconds',
'progress'])
countdownStarted = pyqtSignal()
countdownStopped = pyqtSignal()
```

Key Methods

Display Updates

```python
@pyqtSlot(str)
def set_score_value(self, score_value: str)
    """Update total score display"""

@pyqtSlot(int)
def set_score_int(self, score_value: int)
    """Update score from integer"""

@pyqtSlot(int)
def set_correct_count(self, count: int)
    """Update correct reactions count"""

@pyqtSlot(int)
def set_wrong_count(self, count: int)
    """Update wrong reactions count"""

@pyqtSlot(int)
def set_miss_count(self, count: int)
    """Update missed reactions count"""

@pyqtSlot(str)
def set_team_name(self, team_name: str)
    """Update team name display"""
```

Timer Control

```python
@pyqtSlot(int)
def set_timer_seconds(self, seconds: int)
    """Set timer to specific seconds (formats as MM:SS)"""

@pyqtSlot()
def start_countdown(self)
    """Start the countdown timer"""

@pyqtSlot()
def stop_countdown(self)
    """Stop the countdown timer"""

@pyqtSlot()
def reset_timer(self)
    """Reset timer to initial value"""
```

Usage Example

```python
# Create backend
backend = FastReactionBackend()
```

```
# Connect to QML
root_object.setProperty('backend', backend)

# Update game state
backend.set_team_name("Team Alpha")
backend.set_correct_count(15)
backend.set_wrong_count(3)
backend.set_miss_count(2)
backend.set_score_int(10)  # 15 correct - 3 wrong - 2 miss = 10
backend.set_timer_seconds(60)

# Start countdown
backend.start_countdown()
```

---

2. EnhancedLeaderboardBackend

Purpose

Backend for leaderboard display with top 5 teams and podium.

Main Class: `EnhancedLeaderboardBackend`

Inherits: `QObject`

Responsibilities:

- Manage top 5 player/team data
- Update podium (top 3 positions)
- Track last played team
- Sort and rank teams

Data Structure

```
{
    "name": "Team Alpha",
    "score": 45,
    "weighted_points": 225,
    "rank": 1,
    "team": "Team Alpha",
    "position": "player_table_item_1"
}
```

Signals

Player Table Signals

```
playerTableCellUpdated = pyqtSignal(int, str, int, int, int,
    arguments=['index', 'name', 'score', 'weighted_points', 'rank'])
playerTableNameUpdated = pyqtSignal(int, str, arguments=['index', 'name'])
playerTableScoreUpdated = pyqtSignal(int, int, arguments=['index',
'score'])
playerTableWeightedUpdated = pyqtSignal(int, int, arguments=['index',
'weighted_points'])
playerTableRankUpdated = pyqtSignal(int, int, arguments=['index', 'rank'])
```

Podium Signals

```
podiumUpdated = pyqtSignal(str, str, str, int, int, int, int, int, int,
    arguments=['gold_name', 'silver_name', 'bronze_name',
              'gold_score', 'silver_score', 'bronze_score',
              'gold_weighted', 'silver_weighted', 'bronze_weighted'])
goldPlayerUpdated = pyqtSignal(str, int, int, arguments=['name', 'score',
'weighted_points'])
silverPlayerUpdated = pyqtSignal(str, int, int, arguments=['name', 'score',
'weighted_points'])
bronzePlayerUpdated = pyqtSignal(str, int, int, arguments=['name', 'score',
'weighted_points'])
```

Last Player Signals

```
lastPlayerUpdated = pyqtSignal(str, int, int, int,
    arguments=['name', 'score', 'weighted_points', 'rank'])
lastPlayerNameUpdated = pyqtSignal(str, arguments=['name'])
lastPlayerScoreUpdated = pyqtSignal(int, arguments=['score'])
```

Key Methods

Update Player Table

```
@pyqtSlot(int, str)
def update_player_table_name(self, index: int, name: str)
    """Update player name at index (0-4)"""

@pyqtSlot(int, int)
def update_player_table_score(self, index: int, score: int)
    """Update player score at index"""

@pyqtSlot(int, int)
def update_player_table_weighted_points(self, index: int, weighted_points:
int)
```

```
    """Update weighted points at index"""

@pyqtSlot(int, int)
def update_player_table_rank(self, index: int, rank: int)
    """Update rank at index"""
```

Update Last Player

```
@pyqtSlot(str, int, int, int)
def update_last_player(self, name: str, score: int, weighted_points: int,
rank: int)
    """Update all last player data at once"""
```

Utility Methods

```
@pyqtSlot()
def sort_leaderboard_by_score(self)
    """Sort leaderboard by score and update ranks"""

@pyqtSlot()
def update_podium(self)
    """Update podium with current top 3"""
```

Usage Example

```
# Create backend
backend = EnhancedLeaderboardBackend()

# Update top 5
for i, team in enumerate(top_5_teams):
    backend.update_player_table_name(i, team['name'])
    backend.update_player_table_score(i, team['score'])
    backend.update_player_table_weighted_points(i, team['weighted_points'])
    backend.update_player_table_rank(i, i + 1)

# Update last played team
backend.update_last_player("Team Beta", 35, 175, 12)

# Update podium
backend.update_podium()
```

---

3. TeamScoreBackend

Purpose

Backend for final score display screen.

Main Class: `TeamScoreBackend`

Inherits: `QObject`

Responsibilities:

- Display team name
- Display final score

Signals

```
teamNameChanged = pyqtSignal(str, arguments=['teamName'])
scoreValueChanged = pyqtSignal(str, arguments=['scoreValue'])
```

Key Methods

```
@pyqtSlot(str)
def set_team_name(self, team_name: str)
    """Set team name display"""

@pyqtSlot(str)
def set_score_value(self, score_value: str)
    """Set score value (as string)"""

@pyqtSlot(int)
def set_score_int(self, score_value: int)
    """Set score value (from integer)"""

@pyqtSlot()
def reset_to_defaults(self)
    """Reset to default values"""
```

Usage Example

```
backend = TeamScoreBackend()

# Update final screen
backend.set_team_name("Team Alpha")
backend.set_score_int(45)
```

---

4. TeamNameBackend

Purpose

Backend for team name entry screen with bilingual support (Arabic/English).

Main Class: `TeamNameBackend`

Inherits: `QObject`

Responsibilities:

- Manage bilingual team names (Arabic and English)
- Manage 4 player names and avatars
- Update QML team entry screen

Data Structure

```
{
    "team_name_ar": "فريق سريع",          # Arabic team name
    "team_name_en": "Fast Team",          # English team name
    "players": [
        {"name": "Player 1", "avatar": "assets/avatar_img_13.png"},
        {"name": "Player 2", "avatar": "assets/avatar_img_14.png"},
        {"name": "Player 3", "avatar": "assets/avatar_img_15.png"},
        {"name": "Player 4", "avatar": "assets/avatar_img_16.png"}
    ]
}
```

Signals

```
teamNameArUpdated = pyqtSignal(str, arguments=["team_name_ar"])
teamNameEnUpdated = pyqtSignal(str, arguments=["team_name_en"])
playerNameUpdated = pyqtSignal(int, str, arguments=["index", "name"])
playerAvatarUpdated = pyqtSignal(int, str, arguments=["index",
"avatar_source"])
allPlayersUpdated = pyqtSignal(list, arguments=["players"])
```

Key Methods

Team Name Updates

```
@pyqtSlot(str, str)
def update_team_name_bilingual(self, team_name_ar: str, team_name_en: str)
    """Update both Arabic and English team names"""

@pyqtSlot(str)
def update_team_name_ar(self, team_name_ar: str)
    """Update Arabic team name only"""

@pyqtSlot(str)
def update_team_name_en(self, team_name_en: str)
    """Update English team name only"""
```

Player Updates

```
@pyqtSlot(int, str)
def update_player_name(self, index: int, name: str)
    """Update player name at index (0-3)"""

@pyqtSlot(int, str)
def update_player_avatar(self, index: int, avatar_source: str)
    """Update player avatar at index (0-3)"""
```

Usage Example

```
backend = TeamNameBackend()

# Update bilingual team name
backend.update_team_name_bilingual("الفريق السريع", "Fast Team")

# Update individual players
backend.update_player_name(0, "Ahmed")
backend.update_player_avatar(0, "assets/avatar_img_13.png")

backend.update_player_name(1, "Mohammed")
backend.update_player_avatar(1, "assets/avatar_img_14.png")
```

QML Integration Pattern

Connection in Python

```
# Create QQuickWidget
qml_widget = QQuickWidget()
qml_widget.setSource(QUrl.fromLocalFile("screen.qml"))
```

```
# Create backend
backend = FastReactionBackend()

# Connect backend to QML
root_object = qml_widget.rootObject()
if root_object:
    root_object.setProperty('backend', backend)
```

Connection in QML

```
Item {
    id: root

    // Backend property (injected from Python)
    property var backend: null

    // Connect to signals
    Connections {
        target: backend

        onScoreChanged: {
            scoreText.text = scoreValue
        }

        onCorrectCountChanged: {
            correctText.text = correctCount
        }

        onWrongCountChanged: {
            wrongText.text = wrongCount
        }
    }

    // Call backend methods
    Button {
        onClicked: backend.start_countdown()
    }
}
```

Signal Flow Diagram

```
Python Backend
    ↓
    emit signal (e.g., scoreChanged.emit("45"))
    ↓
Qt Meta-Object System
    ↓
```

```
QML Connections
    ↓
QML Property Update (scoreText.text = "45")
    ↓
UI Rendered
```

FastReaction Backend Features

FastReactionBackend Signals

Reaction Tracking:

- `correctCountChanged` - Track correct reactions
- `wrongCountChanged` - Track wrong reactions
- `missCountChanged` - Track missed reactions

Game State:

- `scoreChanged` - Total score updates
- `timeChanged` - Timer countdown
- `teamNameChanged` - Team name display

Score Display

FastReaction uses simple integer scores:

- Typical range: 0-100 points
- Real-time updates during gameplay
- Positive or negative values possible

Game Statistics

The backend tracks discrete reaction events:

- Correct reactions (+1 each)
- Wrong reactions (-1 each)
- Missed events (-1 each)
- Total score (sum of all events)

Thread Safety

All backend methods use Qt's signal/slot mechanism which is inherently thread-safe. Signals can be emitted from any thread and will be queued for execution on the main GUI thread.

Example:

```python
# Safe to call from worker thread
def worker_update_score(backend, score):
    backend.set_score_int(score)  # Will execute on main thread

# Create worker thread
thread = QThread()
thread.started.connect(lambda: worker_update_score(backend, 50))
thread.start()
```

Best Practices

1. **Use typed setters** - Prefer `set_score_int(45)` over `set_score_value("45")`
2. **Batch updates** - Update multiple properties before emitting signals
3. **Reset on game start** - Always reset counters when starting new game
4. **Validate inputs** - Check bounds before setting values
5. **Log state changes** - Log important backend state transitions for debugging