**Comprehensive Instructions for Creating strategy\_calculators.py**

**Project Context**

The strategy\_calculators.py file is a critical component of the Heage Betting App, which integrates with Betfair and Smarkets APIs to identify profitable betting opportunities. This component will calculate profits for various betting strategies across 500 unique football bet types, with a focus on accuracy and consistency.

**Core Requirements**

1. **Overall Structure**:
   * Create a StrategyCalculators class with static methods for each bet type
   * Implement helper methods for input validation, lay stake calculation, profit scenario calculation, and standardized result creation
   * Use the Decimal type for all financial calculations to ensure precision
   * Include comprehensive error handling with custom exceptions
2. **Base Utility Methods**:
   * validate\_inputs: Validates required parameters and types
   * calculate\_lay\_stake: Calculates the appropriate lay stake with caching for performance
   * calculate\_profit\_scenarios: Handles multi-outcome bet scenarios
   * create\_standard\_result: Creates a standardized output format for all bet types
3. **Implement All Bet Types**:
   * Start with implementing methods 1-50 as specified in the uploaded documents
   * Ensure consistent naming pattern: calculate\_[bet\_type\_name]
   * Use standardized parameter formats and return structures
4. **Technical Specifications**:
   * All methods must handle both binary outcome bets and multi-outcome bets
   * Financial calculations should use Decimal with 2 decimal place precision
   * Include proper logging for all operations, warnings, and errors
   * Use type hints throughout the code
   * Ensure compatibility with Python 3.8+

**Method Templates**

**Binary Outcome Bet Template**

@staticmethod

def calculate\_bet\_type\_name(

back\_odds: Union[int, float, Decimal],

lay\_odds: Union[int, float, Decimal],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds': back\_odds,

'lay\_odds': lay\_odds,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds', 'lay\_odds', 'stake'],

{k: (int, float, Decimal) for k in inputs.keys()}

)

stake\_d = inputs['stake']

lay\_stake = StrategyCalculators.calculate\_lay\_stake(stake\_d, inputs['back\_odds'], inputs['lay\_odds'], inputs['commission'])

profit = (stake\_d \* (inputs['back\_odds'] - 1) \* (1 - inputs['back\_commission'])) - (

lay\_stake \* (inputs['lay\_odds'] - 1) \* (1 - inputs['commission'])

)

return StrategyCalculators.create\_standard\_result(

bet\_type="Bet Type Name",

profit\_scenarios={"win": profit},

lay\_stakes={"win": lay\_stake},

stake=stake\_d,

commission=inputs['commission'],

back\_commission=inputs['back\_commission']

)

**Multi-Outcome Bet Template**

@staticmethod

def calculate\_multi\_outcome\_bet\_type(

back\_odds\_dict: Dict[str, Union[int, float, Decimal]],

lay\_odds\_dict: Dict[str, Union[int, float, Decimal]],

stake: Union[int, float, Decimal],

commission: Union[int, float, Decimal] = Decimal('0.02'),

back\_commission: Union[int, float, Decimal] = Decimal('0.00')

) -> Dict[str, Union[str, bool, Decimal, Dict]]:

inputs = {

'back\_odds\_dict': back\_odds\_dict,

'lay\_odds\_dict': lay\_odds\_dict,

'stake': stake,

'commission': commission,

'back\_commission': back\_commission

}

StrategyCalculators.validate\_inputs(

inputs,

['back\_odds\_dict', 'lay\_odds\_dict', 'stake'],

{'stake': (int, float, Decimal), 'back\_odds\_dict': (dict,), 'lay\_odds\_dict': (dict,), 'commission': (int, float, Decimal), 'back\_commission': (int, float, Decimal)}

)

stake\_d = inputs['stake']

lay\_stakes = {outcome: StrategyCalculators.calculate\_lay\_stake(stake\_d, back\_odds\_dict[outcome], lay\_odds\_dict[outcome], inputs['commission']) for outcome in back\_odds\_dict}

profit\_scenarios = StrategyCalculators.calculate\_profit\_scenarios(stake\_d, back\_odds\_dict, lay\_odds\_dict, lay\_stakes, inputs['commission'], inputs['back\_commission'])

return StrategyCalculators.create\_standard\_result("Multi-Outcome Bet Type", profit\_scenarios, lay\_stakes, stake\_d, inputs['commission'], inputs['back\_commission'])

**Specific Fixes Required**

1. **Fix Bet Type 9 (calculate\_match\_to\_go\_to\_penalties)**:
   * In the validate\_inputs call, change 'lay\_ods\_dict' to 'lay\_odds\_dict' in the type\_checks dictionary
2. **Update Method Name for Bet Type 5**:
   * Change "calculate\_home\_team\_win\_either\_half" to "calculate\_home\_team\_to\_win\_either\_half" for consistency
   * Update the bet\_type string in create\_standard\_result accordingly
3. **Complete All 50 Bet Types**:
   * Ensure all 50 methods from bets 1-50 are implemented following the consistent pattern
   * Follow the naming conventions and parameter structures in the existing methods

**Quality Standards**

1. **Code Style**:
   * Use consistent indentation (4 spaces)
   * Follow PEP 8 naming conventions
   * Use descriptive variable names
   * Add inline comments for complex logic
2. **Error Handling**:
   * Validate all inputs before processing
   * Return appropriate error messages for invalid inputs
   * Handle edge cases (division by zero, negative odds, etc.)
3. **Performance Optimization**:
   * Use lru\_cache for frequently called methods
   * Avoid redundant calculations
   * Optimize multi-outcome bet calculations
4. **Documentation**:
   * Include docstrings for all methods
   * Document parameters, return values, and exceptions
   * Add example usage where helpful

**Implementation Notes**

1. Review the provided sample code for bet types 1-40 and ensure your implementation is consistent with this pattern.
2. Implement bet types 41-50 following the same structure.
3. Fix the identified issues (esp. in Bet Type 9) and ensure consistency in naming (esp. for Bet Type 5).
4. Ensure all methods return a standardized result structure using the create\_standard\_result helper.
5. Test all bet types with sample data to verify correct calculation.

The complete implementation should provide a robust, accurate calculation engine for all 50 bet types while maintaining consistent interfaces and error handling throughout.