Documentatie

## Configuratia hardware

MacBook Pro – ARM64 M1 Pro, 16 GB RAM

Masini Windows – Intel i9, i7, AMD Ryzen 7, 16/32GB RAM

## Configuratia software

Limbaj de programare: Python 3.11

Framework testare: PyTest 8.3.5

Tool mutanti: mutmut 3.2.3

Tool generare diagrama CFG: PyCFG 0.1

## Bucati de cod si rularea testelor

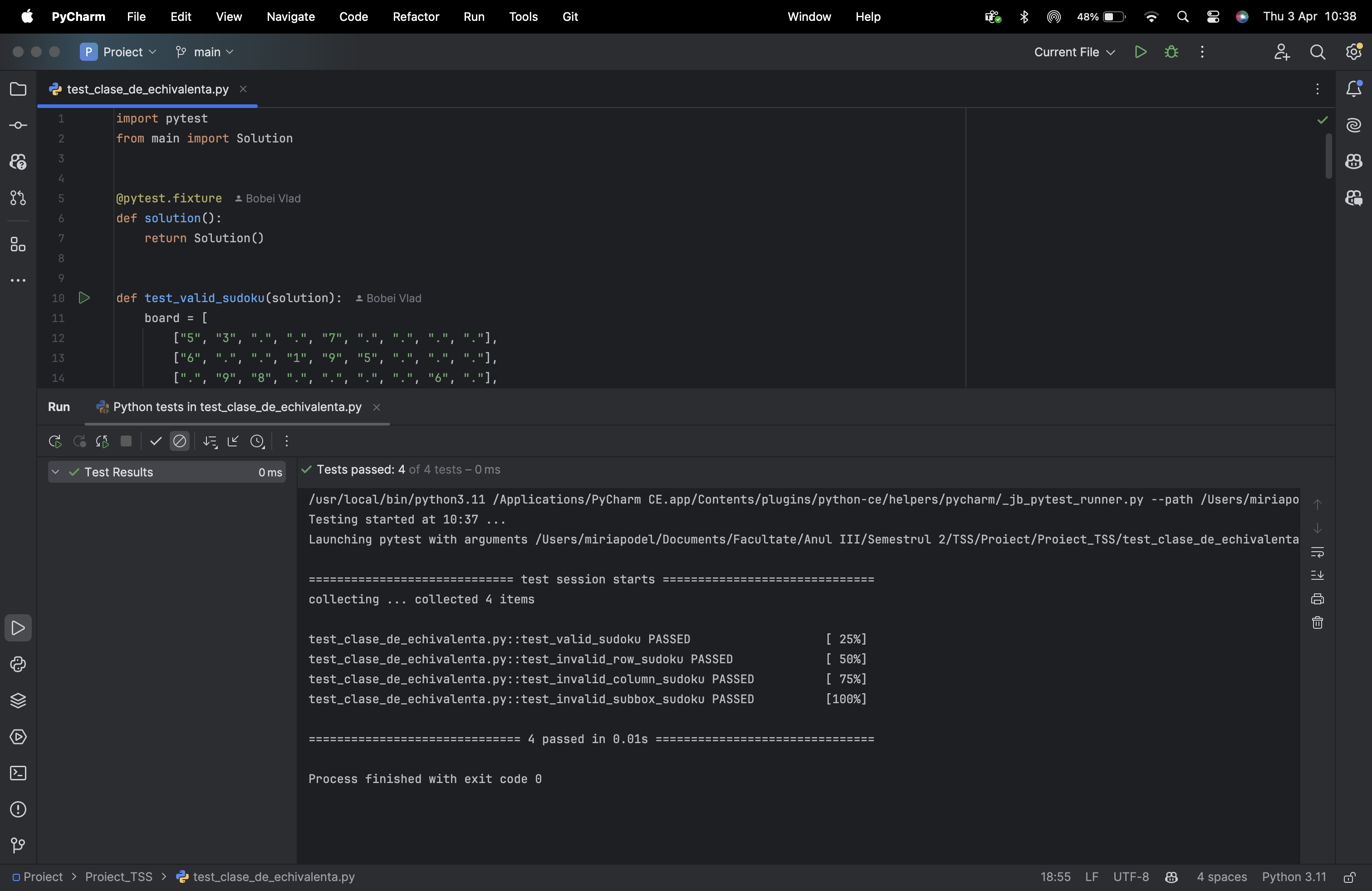
### Clasa principala care urmeaza a fi supusa testarii

class Solution:  
 def valid\_board\_structure(self, board):  
 valid\_characters = {".", "1", "2", "3", "4", "5", "6", "7", "8", "9"}  
  
 if not isinstance(board, list) or len(board) != 9:  
 return False  
  
 for row in board:  
 if not isinstance(row, list) or len(row) != 9:  
 return False  
  
 for cell in row:  
 if cell not in valid\_characters:  
 return False  
  
 return True  
  
  
 def isValidSudoku(self, board):  
  
 if not self.valid\_board\_structure(board):  
 return False  
 else:  
 columns = defaultdict(set)  
 rows = defaultdict(set)  
  
 def checkRowsAndColsForUniqueElems():  
 for i in range(9):  
 for j in range(9):  
 if board[i][j] == ".":  
 continue  
  
 if board[i][j] in rows[i] or board[i][j] in columns[j]:  
 return False  
  
 rows[i].add(board[i][j])  
 columns[j].add(board[i][j])  
  
 return True  
  
 def checkSubBox(noOfBox):  
 elems = set()  
 startingRow = (noOfBox // 3) \* 3  
 startingColumn = (noOfBox % 3) \* 3  
  
 for i in range(startingRow, startingRow + 3):  
 for j in range(startingColumn, startingColumn + 3):  
 if board[i][j] == ".":  
 continue  
  
 if board[i][j] in elems:  
 return False  
  
 elems.add(board[i][j])  
  
 return True  
  
 if not checkRowsAndColsForUniqueElems():  
 return False  
  
 for i in range(9):  
 if not checkSubBox(i):  
 return False  
  
 return True

### Exemple de testare a claselor de echivalenta [1], [3]

def test\_valid\_sudoku(solution):  
 board = [  
 ["5", "3", ".", ".", "7", ".", ".", ".", "."],  
 ["6", ".", ".", "1", "9", "5", ".", ".", "."],  
 [".", "9", "8", ".", ".", ".", ".", "6", "."],  
 ["8", ".", ".", ".", "6", ".", ".", ".", "3"],  
 ["4", ".", ".", "8", ".", "3", ".", ".", "1"],  
 ["7", ".", ".", ".", "2", ".", ".", ".", "6"],  
 [".", "6", ".", ".", ".", ".", "2", "8", "."],  
 [".", ".", ".", "4", "1", "9", ".", ".", "5"],  
 [".", ".", ".", ".", "8", ".", ".", "7", "9"]  
 ]  
 assert solution.isValidSudoku(board)  
  
  
def test\_invalid\_row\_sudoku(solution):  
 board = [  
 ["5", "3", ".", ".", "7", ".", ".", ".", "5"],  
 ["6", ".", ".", "1", "9", "5", ".", ".", "."],  
 [".", "9", "8", ".", ".", ".", ".", "6", "."],  
 ["8", ".", ".", ".", "6", ".", ".", ".", "3"],  
 ["4", ".", ".", "8", ".", "3", ".", ".", "1"],  
 ["7", ".", ".", ".", "2", ".", ".", ".", "6"],  
 [".", "6", ".", ".", ".", ".", "2", "8", "."],  
 [".", ".", ".", "4", "1", "9", ".", ".", "5"],  
 [".", ".", ".", ".", "8", ".", ".", "7", "9"]  
 ]  
 assert not solution.isValidSudoku(board)

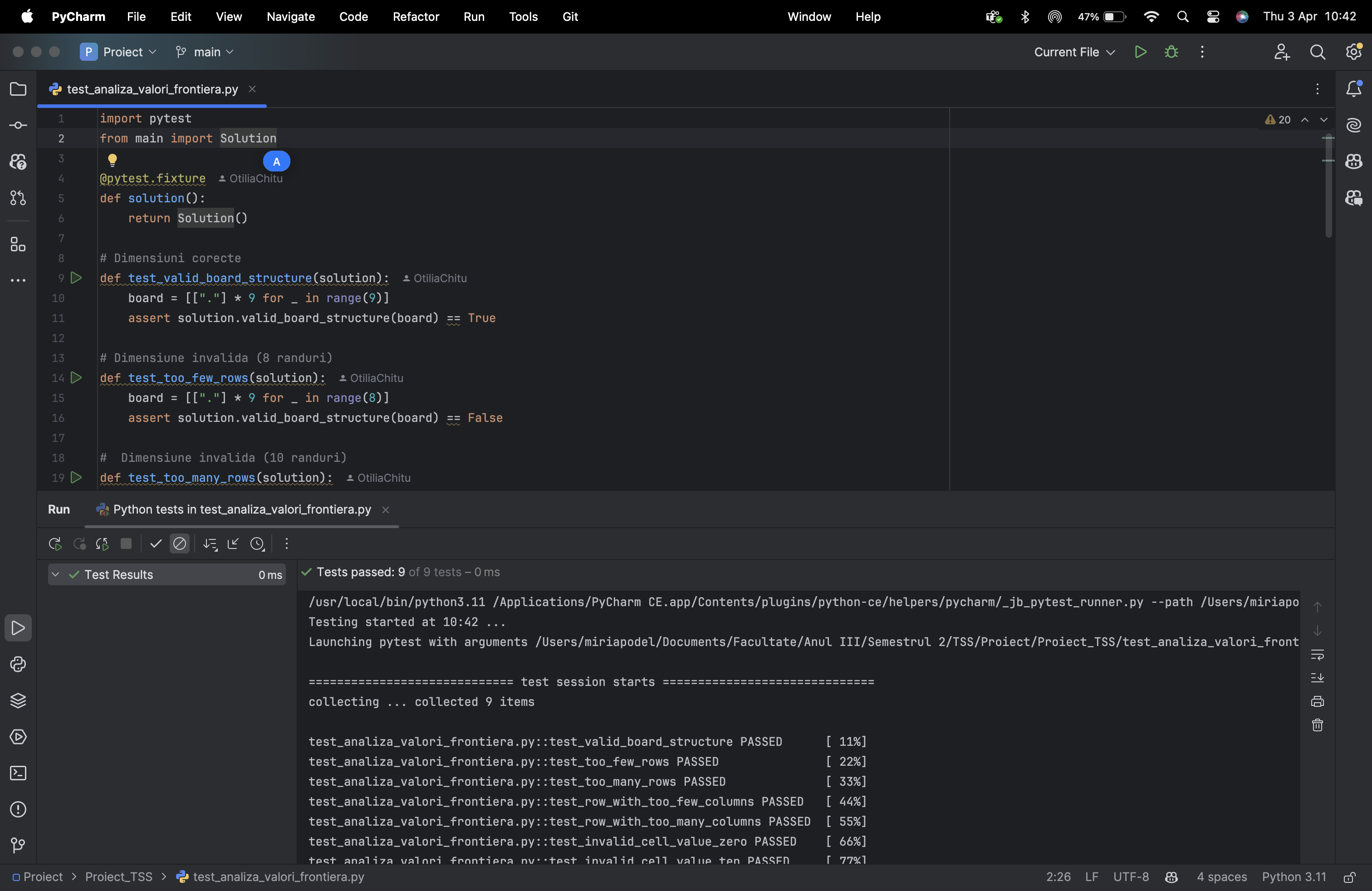
### Rularea tuturor testelor referitoare la clasele de echivalenta [3]



### Exemple de testare a valorilor la frontiera [3]

# Dimensiuni corecte  
def test\_valid\_board\_structure(solution):  
 board = [["."] \* 9 for \_ in range(9)]  
 assert solution.valid\_board\_structure(board) == True  
  
# Dimensiune invalida (8 randuri)  
def test\_too\_few\_rows(solution):  
 board = [["."] \* 9 for \_ in range(8)]  
 assert solution.valid\_board\_structure(board) == False

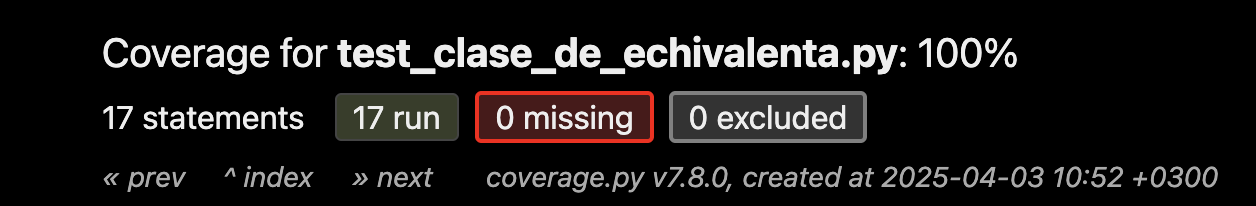
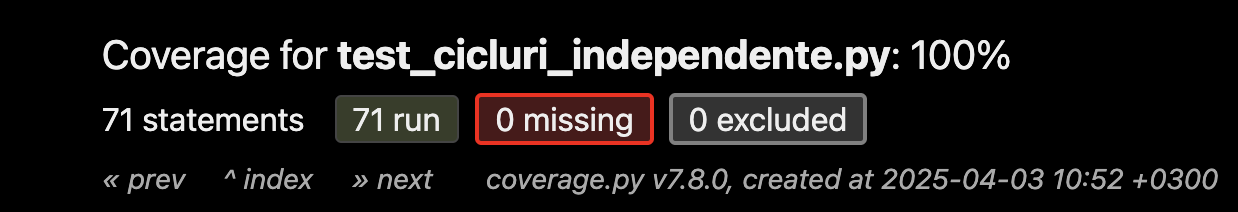
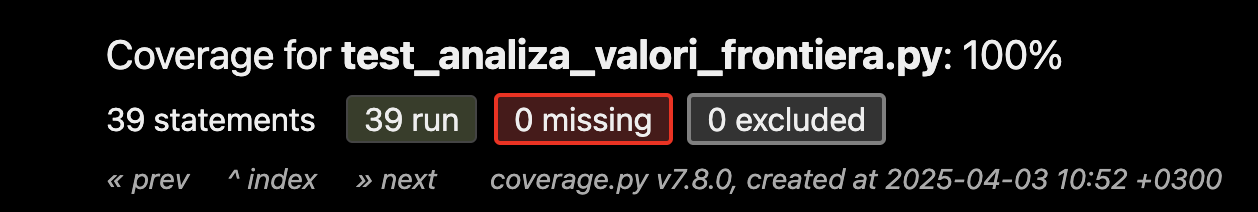
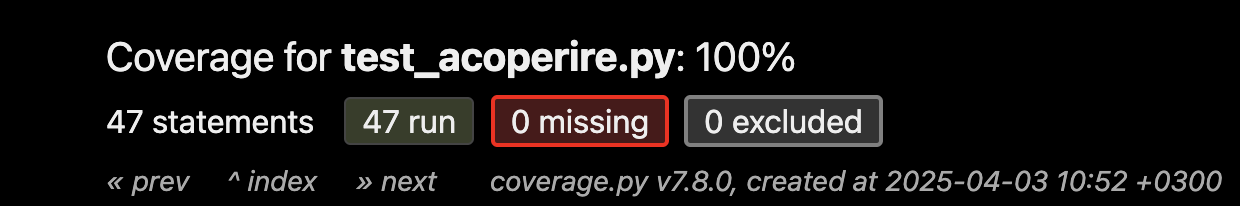
### Rularea tuturor testelor referitoare la valorile de frontiera [3]



### Exemple de testare a acoperirii [3]

def test\_invalid\_not\_list(solution):  
 board = "not a list"  
 assert solution.valid\_board\_structure(board) == False  
  
  
def test\_invalid\_row\_count(solution):  
 board = [["."] \* 9 for \_ in range(8)]  
 assert solution.valid\_board\_structure(board) == False

### Rapoarte HTML pentru acoperire [5]



### 

### Rularea tuturor testelor referitoare la acoperire [3]

A screenshot of a computer

AI-generated content may be incorrect.

### Exemple de testare a ciclurilor independente [3]

# 1. valid\_board\_structure 1st if FALSE(isinstance)  
def test\_not\_a\_list(solution):  
 board = "not\_a\_list"  
 assert solution.isValidSudoku(board) == False  
  
# 2. valid\_board\_structure 1st if FALSE(len)  
def test\_list\_len\_not\_9(solution):  
 board = [["."] \* 9] \* 8 # only 8 rows  
 assert solution.isValidSudoku(board) == False  
  
# 3. valid\_board\_structure 2nd if FALSE(isinstance)  
def test\_row\_not\_list(solution):  
 board = ["row"] \* 9 # rows are strings, not lists  
 assert solution.isValidSudoku(board) == False

### Rularea tuturor testelor referitoare la ciclurile independente [3]

A screenshot of a computer program

AI-generated content may be incorrect.

### 

### Diagrama CFG care ne-a ajutat sa identificam si sa testam ciclurile independente [2]

### Schita dupa care am creat numarul optim de teste pentru ciclurile independente

1. valid\_board\_structure 1st if FALSE(isinstance)

2. valid\_board\_structure 1st if FALSE(len)

3. valid\_board\_structure 2nd if FALSE(isinstance)

4. valid\_board\_structure 2nd if FALSE(len)

5. valid\_board\_structure 3rd if FALSE(cell invalid)

6. valid\_board\_structure -> checkRowsAndColsForUniqueElems continue FALSE(board in rows)

7. valid\_board\_structure -> checkRowsAndColsForUniqueElems continue FALSE(board in columns)

8. valid\_board\_structure -> checkRowsAndColsForUniqueElems NO continue FALSE(board in rows)

9. valid\_board\_structure -> checkRowsAndColsForUniqueElems NO continue FALSE(board in colums)

10. valid\_board\_structure -> checkRowsAndColsForUniqueElems continue -> checkSubBox continue FALSE(board in elems)

11. valid\_board\_structure -> checkRowsAndColsForUniqueElems continue -> checkSubBox NO continue FALSE(board in elems)

12. valid\_board\_structure -> checkRowsAndColsForUniqueElems NO continue -> checkSubBox continue FALSE(board in elems)

13. valid\_board\_structure -> checkRowsAndColsForUniqueElems NO continue -> checkSubBox NO continue FALSE(board in elems)

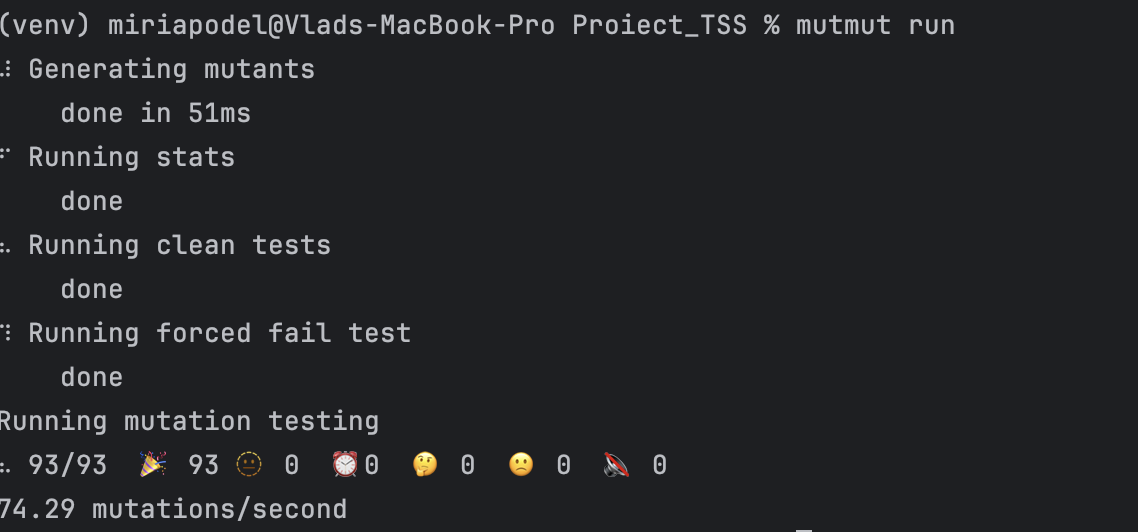
14. valid\_board\_structure -> checkRowsAndColsForUniqueElems continue -> checkSubBox continue TRUE

15. valid\_board\_structure -> checkRowsAndColsForUniqueElems continue -> checkSubBox NO continue TRUE

16. valid\_board\_structure -> checkRowsAndColsForUniqueElems NO continue -> checkSubBox continue TRUE

17. valid\_board\_structure -> checkRowsAndColsForUniqueElems NO continue -> checkSubBox NO continue TRUE

### Generarea mutantilor [4]



### Exemple de mutanti generati [4]

def xǁSolutionǁisValidSudoku\_\_mutmut\_20(self, board):  
  
 if not self.valid\_board\_structure(board):  
 return False  
 else:  
 columns = defaultdict(set)  
 rows = defaultdict(set)  
  
 def checkRowsAndColsForUniqueElems():  
 for i in range(9):  
 for j in range(9):  
 if board[i][j] == ".":  
 continue  
  
 if board[i][j] in rows[i] or board[i][None] in columns[j]:  
 return False  
  
 rows[i].add(board[i][j])  
 columns[j].add(board[i][j])  
  
 return True  
  
 def checkSubBox(noOfBox):  
 elems = set()  
 startingRow = (noOfBox // 3) \* 3  
 startingColumn = (noOfBox % 3) \* 3  
  
 for i in range(startingRow, startingRow + 3):  
 for j in range(startingColumn, startingColumn + 3):  
 if board[i][j] == ".":  
 continue  
  
 if board[i][j] in elems:  
 return False  
  
 elems.add(board[i][j])  
  
 return True  
  
 if not checkRowsAndColsForUniqueElems():  
 return False  
  
 for i in range(9):  
 if not checkSubBox(i):  
 return False  
  
 return True

def xǁSolutionǁisValidSudoku\_\_mutmut\_61(self, board):  
 if not self.valid\_board\_structure(board):  
 return False  
 else:  
 columns = defaultdict(set)  
 rows = defaultdict(set)  
  
 def checkRowsAndColsForUniqueElems():  
 for i in range(9):  
 for j in range(9):  
 if board[i][j] == ".":  
 continue  
  
 if board[i][j] in rows[i] or board[i][j] in columns[j]:  
 return False  
  
 rows[i].add(board[i][j])  
 columns[j].add(board[i][j])  
  
 return True  
  
 def checkSubBox(noOfBox):  
 elems = set()  
 startingRow = (noOfBox // 3) \* 3  
 startingColumn = (noOfBox % 3) \* 3  
  
 for i in range(startingRow, startingRow + 3):  
 for j in range(startingColumn, startingColumn + 3):  
 if board[i][j] == ".":  
 continue  
  
 if board[i][j] in elems:  
 return False  
  
 elems.add(board[i][None])  
  
 return True  
  
 if not checkRowsAndColsForUniqueElems():  
 return False  
  
 for i in range(9):  
 if not checkSubBox(i):  
 return False  
  
 return True

### Referinte:

[1] OpenAI, ChatGPT, <https://chatgpt.com/>, Data generarii: 1-2 Aprilie 2025

[2] PyPi, PyCFG, <https://pypi.org/project/pycfg/>, Data accesarii: 2 Aprilie 2025

[3] PyTest, <https://docs.pytest.org/en/stable/>, Data accesarii: 1-2 Aprilie 2025

[4] mutmut, <https://mutmut.readthedocs.io/en/latest/>, Data accesarii: 2 Aprilie 2025

[5] Coverage, <https://coverage.readthedocs.io/en/7.8.0/>, Data accesarii: 1 Aprilie 2025

[6] Predut Sorina, Cursuri TSS, <https://drive.google.com/drive/folders/18CVua5zkJeaY1UsDjJIJgHjMqYxs_d7R>, Data accesarii: 1-2 Aprilie 2025