



Клиент-серверная часть программной системы «Игра в Шахматы»

Программные средства разработки интеллектуальных систем

A close-up photograph of a dark-colored chess piece, likely a king or queen, standing on a chessboard. The board's alternating light and dark squares are visible in the foreground and background, creating a sense of depth. The lighting is dramatic, highlighting the piece's form against the dark background.

Клиент-серверная часть программной системы «Игра в Шахматы»

Программные средства разработки интеллектуальных систем

Студенты СПбГЭТУ (ЛЭТИ), гр. 1308:

- Мельник Даниил,
- Томилов Даниил,
- Лепов Алексей.

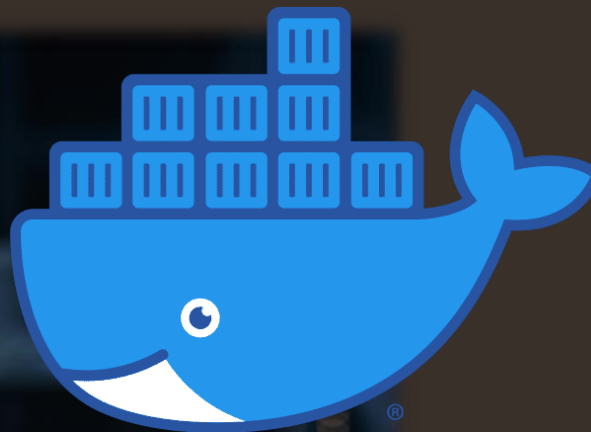
Постановка задач

Необходимо реализовать такую клиент-серверную систему, которая способна предоставить программно **бесперебойную** работу относительно **сервера** и рабочий **интерфейс** для игры в шахматы **клиентам**.



Выбранные технологии

- Docker
- MySQL
- VS Code
- PyCharm



Архитектура БД

Таблица «ПОЛЬЗОВАТЕЛИ»

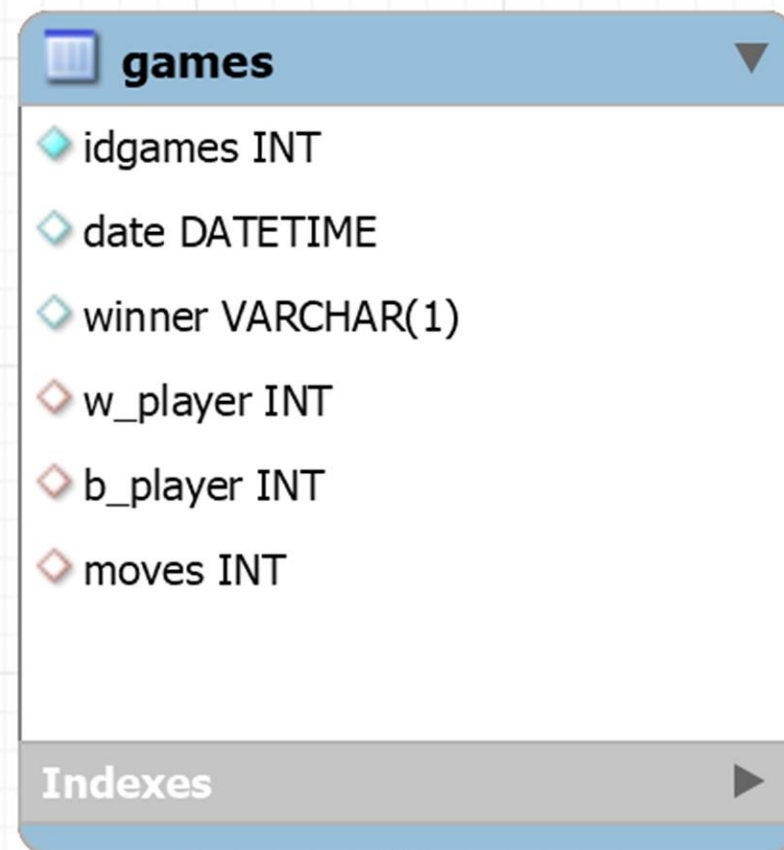
- id
- Имя пользователя
- Почта
- Пароль
- Рейтинг

users	
idusers	INT
username	VARCHAR(20)
email	VARCHAR(255)
password	VARCHAR(20)
score	INT
Indexes	

Архитектура БД

Таблица «Игры»

- id
- Дата
- Победитель
- Игрок за белые
- Игрок за черные



games	
◆ idgames	INT
◆ date	DATETIME
◆ winner	VARCHAR(1)
◇ w_player	INT
◇ b_player	INT
◇ moves	INT

Indexes ▶

Архитектура БД

Таблица «Ходы»

- id
- файл
- Оставшееся время белых
- Оставшееся время черных



moves



idmoves INT



file BLOB



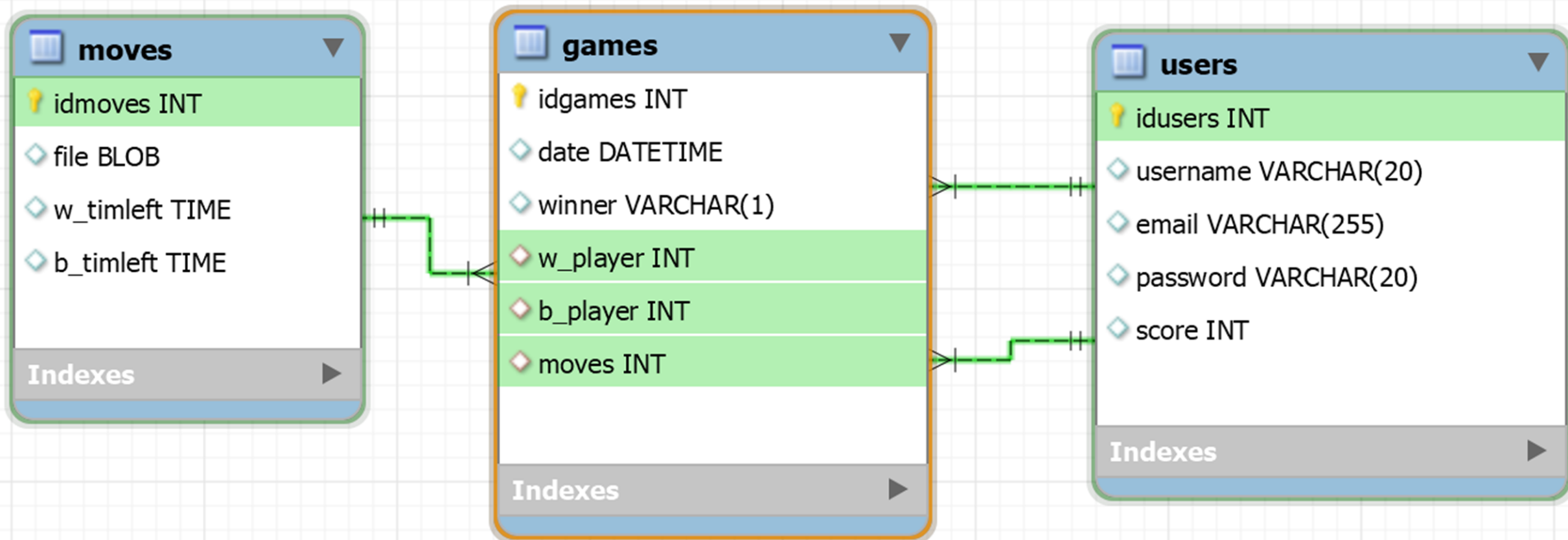
w_timleft TIME



b_timleft TIME

Indexes

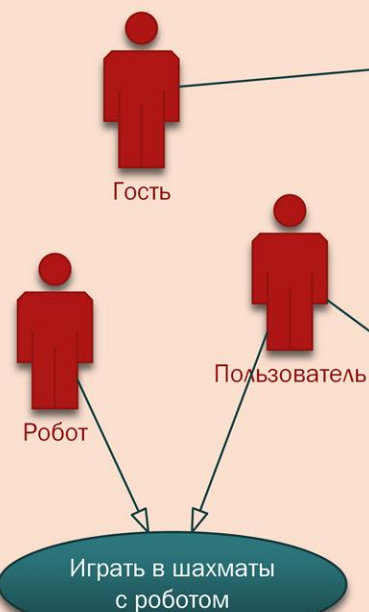
ER-диаграмма



USE CASE – ДИАГРАММА

Программная система «Игра в Шахматы»

Клиентская часть

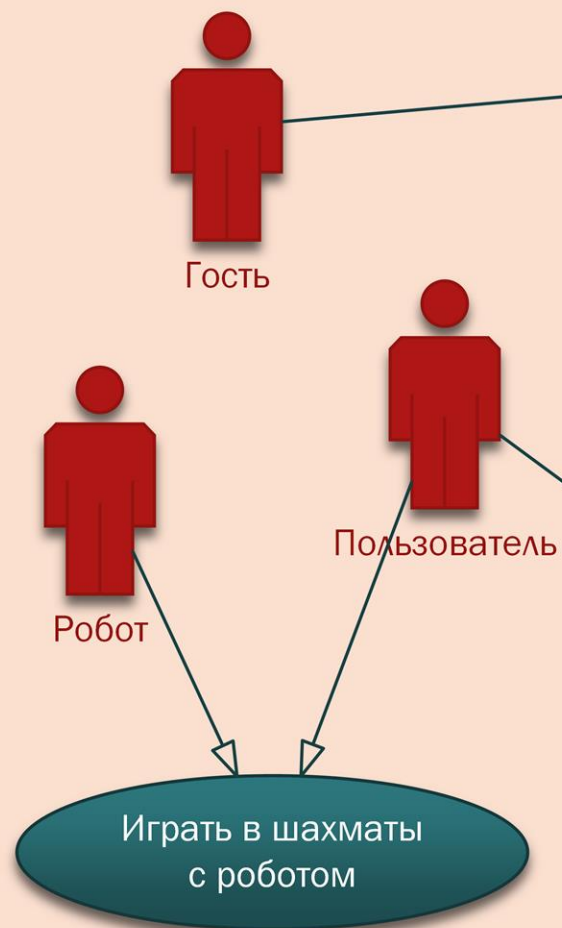


Серверная часть

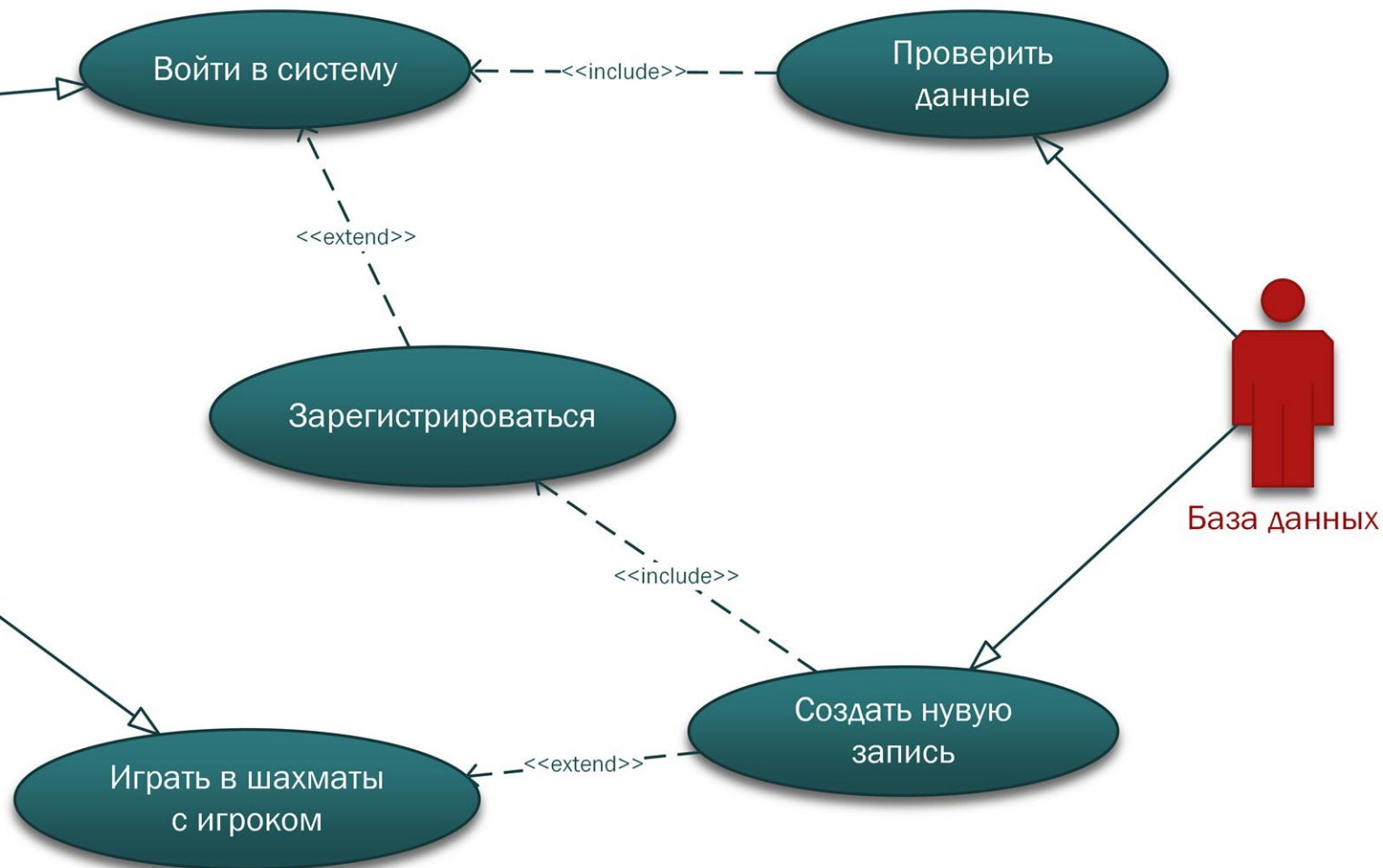


Программная система «Игра в Шахматы»

Клиентская часть



Серверная часть



Реализуемое API

За основу реализации
клиент-серверной
архитектуры взят
концепт **Flask-socketio**.



Клиент № 1

client1.py > ...

```
1  import socket
2  import codecs
3
4  def show_board(L):
5      for i in range(8):
6          print(L[i])
7
8  def connect_with_host(port):
9      sock = socket.socket()
10     sock.connect(('localhost', port))
11     print("connection was made")
12     return sock
13
14 def receive_board(sock):
15     for i in range(8):
16         print((sock.recv(2024)))
17
```

```
13
14 def receive_board(sock):
15     for i in range(8):
16         print((sock.recv(2024)))
17
18 sock=connect_with_host(9090)
19 for i in range(6):
20     tag = codecs.decode(sock.recv(1024))
21     if (tag=='1'):
22         movestr=input("твой ход: ")
23         print(movestr)
24         sock.send(codecs.encode(movestr))
25         print('sent')
26     else:
27         print("ход соперника")
28         sock.recv(1024)
29
30 sock.close()
```


Клиент № 2

```
client2.py > ...
1  import socket
2  import codecs
3
4  def show_board(L):
5      for i in range(8):
6          print(L[i])
7
8  def connect_with_host(port):
9      sock = socket.socket()
10     sock.connect(('localhost', port))
11     print("connection was made")
12     return sock
13
14 def receive_board(sock):
15     for i in range(8):
16         print((sock.recv(2024)))
```

```
15     for i in range(6):
16         print((sock.recv(2024)))
17
18     sock=connect_with_host(9091)
19     for i in range(6):
20         tag = codecs.decode(sock.recv(1024))
21         if (tag=='2'):
22             movestr=input("твой ход: ")
23             sock.send(codecs.encode(movestr))
24         else:
25             print("ход соперника")
26
27     ty=sock.recv([1024])
28
29     sock.close()
30
```

Сервер

```
server.py > Board > __init__
1  import socket
2  import codecs
3
4  def is_port_in_use(port: int) -> bool:
5      import socket
6      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
7          return s.connect_ex(('localhost', port)) == 0
8
9
10 class Board:
11     def __init__(self) -> None:
12         self.positions = []
13         self.moovs = []
14         self.nummov = 0
15         pass
16
17     def add_mov(self, message):
18         self.moovs.append(message)
19
20     def set_board(self):
21         self.positions=[['R','H','B','Q','K','B','H','R'],['p','p','p','p','p','p','p','p'],['*','*','*','*','*','*','*','*'],['*','*','*','*','*','*','*','*'],
22             ['*','*','*','*','*','*','*','*'],['*','*','*','*','*','*','*','*'],['p','p','p','p','p','p','p','p'],['R','H','B','K','Q','B','H','R']]
23
24     def show_board(self):
```


server.py > Board > __init__

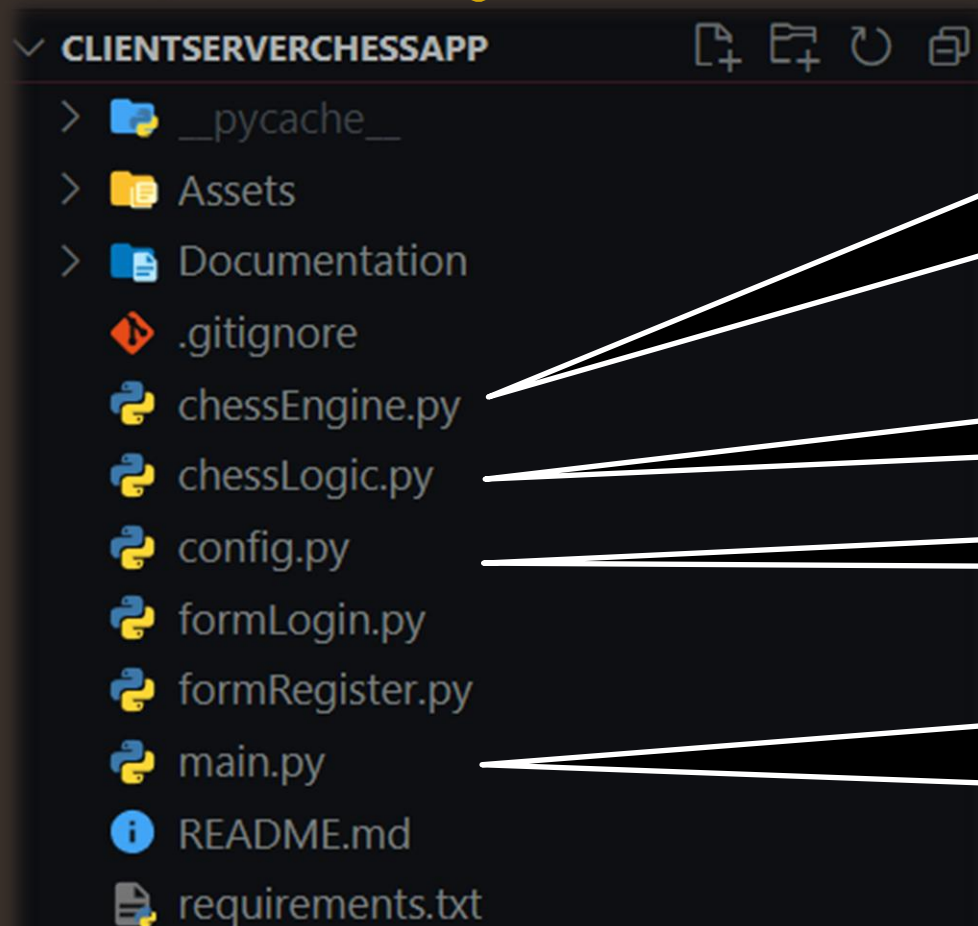
```
1  import socket
2  import codecs
3
4  def is_port_in_use(port: int) -> bool:
5      import socket
6      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
7          return s.connect_ex(('localhost', port)) == 0
8
9
10 class Board:
11     def __init__(self) -> None:
12         self.positions = []
13         self.moovs = []
14         self.nummov = 0
15         pass
16
17     def add_mov(self, message):
18         self.moovs.append(message)
19
20     def set_board(self):
21         self.positions=[['R','H','B','Q','K','B','H','R'],['p','p','p','p','p','p','p','p'],[
22             ['*','*','*','*','*','*','*','*'],['*','*','*','*','*','*','*','*'],[
23
```

```
23
24     def show_board(self):
25         for i in range(8):
26             print(self.positions[i])
27
28     def give_board(self):
29         return self.positions
30
31     def make_mov(self, x1, y1, x2, y2):
32         buf = self.positions[x2][y2]
33         self.positions[x2][y2] = self.positions[x1][y1]
34         self.positions[x1][y1] = buf
35         self.show_board()
36         self.nummov+=1
37     def update_mvs(self, move):
38         self.moovs.append(move)
39
40
41
42     def made_sock():
43         sock= socket.socket()
44         print("socket is made")
45         return sock
46
47     def connect(sock, port):
```

```
46
47 def connect(sock, port):
48     sock.bind(('', port))
49     sock.listen(1)
50     conn, addr = sock.accept()
51     print("connection is complete")
52     return conn, addr
53
54 def receive_from_client(conn):
55     data = conn.recv(1024)
56     print("data were received")
57     return data
58
59 def send_to_client(conn, message):
60     conn.send(message)
61     print("data were sent")
62
63 def send_board(conn, array):
64     for i in range(8):
65         array[i] = ''.join(str(x) for x in array[i])
66         conn.send(codecs.encode(array[i]))
67
68 board = Board()
69 board.set_board()
70 board.show_board()
```

```
69 board.set_board()
70 board.show_board()
71
72
73 sock1=made_sock()
74 sock2=made_sock()
75
76 conn1, addr1 = connect(sock1,9090)
77 conn2, addr2 = connect(sock2,9091)
78
79 for i in range(6):
80     tag=str(i%2+1)
81     send_to_client(conn1, codecs.encode(tag))
82     send_to_client(conn2, codecs.encode(tag))
83     if(tag=='1'):
84         move=codecs.decode(receive_from_client(conn1))
85     else:
86         move=codecs.decode(receive_from_client(conn2))
87     board.update_mvs(move)
88     movearr=move.split(' ')
89     print(movearr)
90     board.make_mov(int(movearr[0]),int(movearr[1]),int(movearr[2]),int(movearr[3]))
91     conn1.send(b'k')
92     conn2.send(b'k')
```


Файловая структура проекта



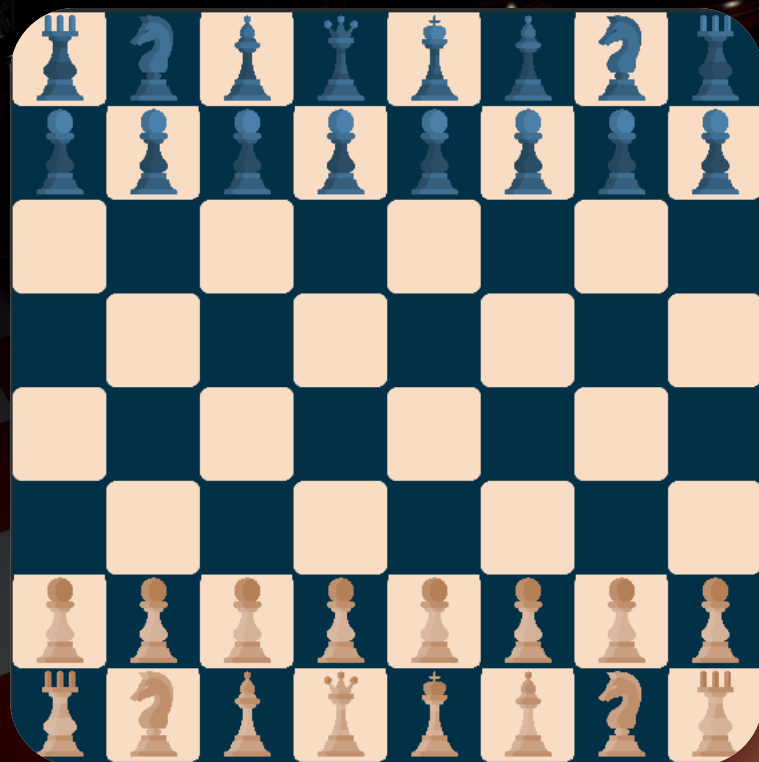
Программный модуль для работы с шахматной логикой

Программный модуль для обработки ходов робота

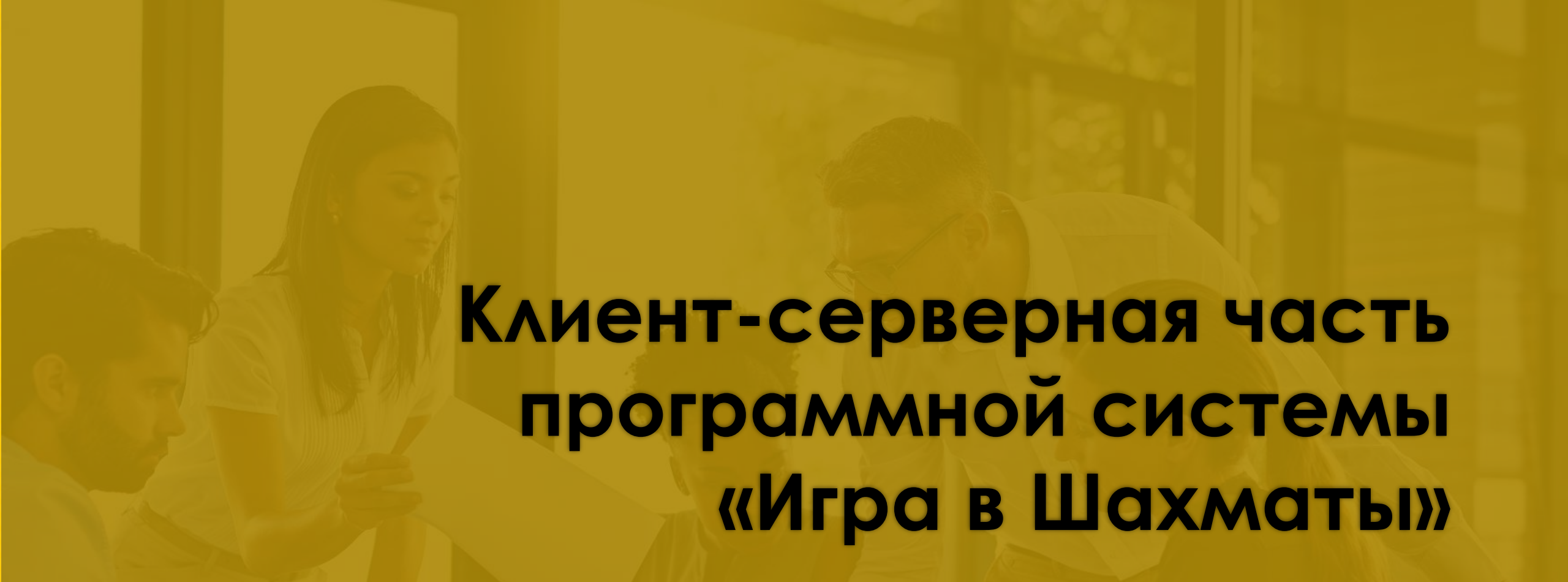
Файл конфигурации базы данных

Программный модуль инициализации и построения графического интерфейса

Пример работы



На данном этапе
разработки клиент-
серверная архитектура
находится на
финальной стадии
отладки



Клиент-серверная часть программной системы «Игра в Шахматы»

Студенты СПбГЭТУ (ЛЭТИ), гр. 1308:

- Томилов Даниил,
- Мельник Даниил,
- Лепов Алексей.