# ДОДАТОК А

Код Брокера

```
import asyncio
import websockets
import json
import logging
from typing import Dict, Set, Optional, Any
from datetime import datetime
import hashlib

# Налаштування логування
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s – %(name)s – %(levelname)s – %(message)s'
)
logger = logging.getLogger('Broker')



class MessageBroker:
    def __init__(self, host='localhost', port=8765, password='broker123'):
        self.host = host
        self.port = port
        self.password = password

        # Словник клієнтів: websocket –> client_info
        self.clients: Dict[websockets.WebSocketServerProtocol, Dict[str, Any]] = {}

        # Словник підписок: topic –> set(websockets)
        self.subscriptions: Dict[str, Set[websockets.WebSocketServerProtocol]] = {}
```

```
        # Статистика
        self.message_count = 0
        self.start_time = datetime.now()

        logger.info(f"Broker initialized on {host}:{port}")

    def _hash_password(self, password: str) -> str:
        """Хешування пароля для безпеки"""
        return hashlib.sha256(password.encode()).hexdigest()

    def _authenticate(self, password: str) -> bool:
        """Перевірка пароля"""
        return self._hash_password(password) == self._hash_password(self.password)

    async def _send_message(self, websocket: websockets.WebSocketServerProtocol,
message: Dict[str, Any]):
        """Відправка повідомлення клієнту"""
        try:
            await websocket.send(json.dumps(message))
        except websockets.exceptions.ConnectionClosed:
            logger.warning(f"Failed to send message – connection closed")
        except Exception as e:
            logger.error(f"Error sending message: {e}")

    async def _broadcast_to_subscribers(self, topic: str, message: Dict[str, Any]):
        """Відправка повідомлення всім підписникам топіка"""
        if topic not in self.subscriptions:
            logger.debug(f"No subscribers for topic: {topic}")
            return
```

```
        subscribers = self.subscriptions[topic].copy()  # Копія для безпеки
        logger.info(f"Broadcasting to {len(subscribers)} subscribers of topic '{topic}'")

        # Відправляємо повідомлення всім підписникам
        for subscriber in subscribers:
            if subscriber in self.clients:
                await self._send_message(subscriber, {
                    'type': 'message',
                    'topic': topic,
                    'data': message,
                    'timestamp': datetime.now().isoformat()
                })

    async def _handle_auth(self, websocket: websockets.WebSocketServerProtocol, data:
Dict[str, Any]) -> bool:
        """Обробка аутентифікації"""
        password = data.get('password', '')
        client_name = data.get('client_name', 'Unknown')

        if self._authenticate(password):
            # Успішна аутентифікація
            self.clients[websocket] = {
                'name': client_name,
                'authenticated': True,
                'connected_at': datetime.now(),
                'subscriptions': set()
            }

            await self._send_message(websocket, {
                'type': 'auth_response',
```

```
            'success': True,
            'message': f'Welcome, {client_name}!'
        })

        logger.info(f"Client '{client_name}' authenticated successfully")
        return True
    else:
        # Невдала аутентифікація
        await self._send_message(websocket, {
            'type': 'auth_response',
            'success': False,
            'message': 'Authentication failed'
        })

        logger.warning(f"Authentication failed for client '{client_name}'")
        return False

async def _handle_subscribe(self, websocket: websockets.WebSocketServerProtocol,
data: Dict[str, Any]):
    """Обробка підписки на топік"""
    if websocket not in self.clients or not self.clients[websocket]['authenticated']:
        await self._send_message(websocket, {
            'type': 'error',
            'message': 'Not authenticated'
        })
        return

    topic = data.get('topic', '')
    client_name = self.clients[websocket]['name']
```

```
        if not topic:
            await self._send_message(websocket, {
                'type': 'error',
                'message': 'Topic cannot be empty'
            })
            return

        # Додаємо підписку
        if topic not in self.subscriptions:
            self.subscriptions[topic] = set()

        self.subscriptions[topic].add(websocket)
        self.clients[websocket]['subscriptions'].add(topic)

        await self._send_message(websocket, {
            'type': 'subscribe_response',
            'success': True,
            'topic': topic,
            'message': f'Successfully subscribed to {topic}'
        })

        logger.info(f"Client '{client_name}' subscribed to topic '{topic}'")

    async def _handle_unsubscribe(self, websocket:
websockets.WebSocketServerProtocol, data: Dict[str, Any]):
        """Обробка відписки від топіка"""
        if websocket not in self.clients or not self.clients[websocket]['authenticated']:
            return

        topic = data.get('topic', '')
```

```
        client_name = self.clients[websocket]['name']

        if topic in self.subscriptions and websocket in self.subscriptions[topic]:
            self.subscriptions[topic].remove(websocket)
            self.clients[websocket]['subscriptions'].discard(topic)

            # Видаляємо порожні топіки
            if not self.subscriptions[topic]:
                del self.subscriptions[topic]

            await self._send_message(websocket, {
                'type': 'unsubscribe_response',
                'success': True,
                'topic': topic,
                'message': f'Successfully unsubscribed from {topic}'
            })

            logger.info(f"Client '{client_name}' unsubscribed from topic '{topic}'")

    async def _handle_publish(self, websocket: websockets.WebSocketServerProtocol,
data: Dict[str, Any]):
        """Обробка публікації повідомлення"""
        if websocket not in self.clients or not self.clients[websocket]['authenticated']:
            await self._send_message(websocket, {
                'type': 'error',
                'message': 'Not authenticated'
            })
            return

        topic = data.get('topic', '')
```

```
message = data.get('message', {})
client_name = self.clients[websocket]['name']

if not topic:
    await self._send_message(websocket, {
        'type': 'error',
        'message': 'Topic cannot be empty'
    })
    return

# Додаємо метадані до повідомлення
enriched_message = {
    **message,
    'sender': client_name,
    'broker_timestamp': datetime.now().isoformat()
}

# Відправляємо підписникам
await self._broadcast_to_subscribers(topic, enriched_message)

# Підтверджуємо відправнику
await self._send_message(websocket, {
    'type': 'publish_response',
    'success': True,
    'topic': topic,
    'message': 'Message published successfully'
})

self.message_count += 1
logger.info(f"Client '{client_name}' published message to topic '{topic}' (total:
```

```
{self.message_count})")

    async def _handle_status(self, websocket: websockets.WebSocketServerProtocol):
        """Відправка статусу брокера"""
        uptime = datetime.now() – self.start_time

        status = {
            'type': 'status_response',
            'broker_info': {
                'uptime_seconds': int(uptime.total_seconds()),
                'connected_clients': len(self.clients),
                'total_messages': self.message_count,
                'active_topics': list(self.subscriptions.keys()),
                'start_time': self.start_time.isoformat()
            }
        }

        await self._send_message(websocket, status)

    async def _cleanup_client(self, websocket: websockets.WebSocketServerProtocol):
        """Очищення даних клієнта при відключенні"""
        if websocket not in self.clients:
            return

        client_name = self.clients[websocket]['name']
        client_subscriptions = self.clients[websocket]['subscriptions'].copy()

        # Видаляємо з усіх підписок
        for topic in client_subscriptions:
            if topic in self.subscriptions and websocket in self.subscriptions[topic]:
```

```
                self.subscriptions[topic].remove(websocket)
                # Видаляємо порожні топіки
                if not self.subscriptions[topic]:
                    del self.subscriptions[topic]

        # Видаляємо клієнта
        del self.clients[websocket]

        logger.info(f"Client '{client_name}' disconnected and cleaned up")

    async def handle_client(self, websocket: websockets.WebSocketServerProtocol):
        """Основний обробник клієнта"""
        client_address =
f"{websocket.remote_address[0]}:{websocket.remote_address[1]}"
        logger.info(f"New connection from {client_address}")

        try:
            # Очікуємо повідомлення від клієнта
            async for message in websocket:
                try:
                    data = json.loads(message)
                    message_type = data.get('type', '')

                    logger.debug(f"Received message type: {message_type} from
{client_address}")

                    if message_type == 'auth':
                        await self._handle_auth(websocket, data)
                    elif message_type == 'subscribe':
                        await self._handle_subscribe(websocket, data)
```

```
            elif message_type == 'unsubscribe':
                await self._handle_unsubscribe(websocket, data)
            elif message_type == 'publish':
                await self._handle_publish(websocket, data)
            elif message_type == 'status':
                await self._handle_status(websocket)
            elif message_type == 'ping':
                await self._send_message(websocket, {'type': 'pong'})
            else:
                await self._send_message(websocket, {
                    'type': 'error',
                    'message': f'Unknown message type: {message_type}'
                })

        except json.JSONDecodeError:
            await self._send_message(websocket, {
                'type': 'error',
                'message': 'Invalid JSON format'
            })
        except Exception as e:
            logger.error(f"Error handling message: {e}")
            await self._send_message(websocket, {
                'type': 'error',
                'message': 'Internal server error'
            })

    except websockets.exceptions.ConnectionClosed:
        logger.info(f"Client {client_address} disconnected")
    except Exception as e:
        logger.error(f"Unexpected error with client {client_address}: {e}")
```

```
    finally:
        await self._cleanup_client(websocket)

async def start_server(self):
    """Запуск WebSocket сервера"""
    logger.info(f"Starting broker server on {self.host}:{self.port}")

    server = await websockets.serve(
        self.handle_client,
        self.host,
        self.port,
        ping_interval=20,
        ping_timeout=10
    )

    logger.info("Broker server started successfully")

    # Виводимо періодичну статистику
    async def print_stats():
        while True:
            await asyncio.sleep(30)  # Кожні 30 секунд
            uptime = datetime.now() – self.start_time
            logger.info(f"Stats: {len(self.clients)} clients, "
                        f"{len(self.subscriptions)} topics, "
                        f"{self.message_count} messages, "
                        f"uptime: {int(uptime.total_seconds())}s")

    # Запускаємо статистику в фоні
    asyncio.create_task(print_stats())
```

```
        # Очікуємо завершення сервера
        await server.wait_closed()

    def stop_server(self):
        """Зупинка сервера (для використання в GUI)"""
        logger.info("Stopping broker server...")


if __name__ == "__main__":
    # Створення та запуск брокера
    broker = MessageBroker(host='localhost', port=8765, password='broker123')

    try:
        asyncio.run(broker.start_server())
    except KeyboardInterrupt:
        logger.info("Broker stopped by user")
    except Exception as e:
        logger.error(f"Broker error: {e}")
```

**ДОДАТОК Б**

Демонстраційний матеріал

| № докумен-та | Позначення | Найменування | Додаткові відомості |
|---|---|---|---|
| | | Текстові документи | |
| | | | |
| 1 | ГЮИК. 503320.021ПЗ | Пояснювальна записка | А4, 88 с. |
| | | | |
| | | Додаткові матеріали | |
| | | | |
| 2 | | Код Брокера | А4, 1 с. |
| 3 | | Демонстраційний матеріал | А4, 15 с. |

| Змін. | Арк. | Номер докум. | Підпис | Дата | ГЮИК. 503320.021ПЗ | | | |
|---|---|---|---|---|---|---|---|---|
| Розроб. | | Прокопенко Н.В. | | | Розробка мікросервісної системи обміну даними між програмними модулями у розподіленому середовищі | Літера | Аркуш | Аркушів |
| Перевір. | | Іванов Л.С. | | | | | | 1 |
| | | | | | | Кафедра КІТАМ ХНУРЕ | | |
| Н.контр. | | Стародубцев М.Г. | | | | | | |
| Затв. | | Невлюдов І.Ш. | | | | | | |