**1. Структура узла (Node)**

Класс **Node** представляет собой узел, который:

* Имеет **node\_id** (идентификатор узла).
* Хранит текущую нагрузку (**load**), максимальную нагрузку (**max\_load**) и время последнего обновления (**last\_update**).
* Позволяет изменять нагрузку (**update\_load** и **decrease\_load**).
* Может возвращать свой статус в виде словаря (**get\_status**).

*def* load\_node\_loads():

    """Загружает состояние узлов из файла"""

    try:

        if os.path.exists(NODE\_LOADS\_FILE):

            with open(NODE\_LOADS\_FILE, "r") as file:

                node\_data = json.load(file)

                for node in NODES:

                    node\_info = node\_data.get(*str*(node.node\_id))

                    if node\_info:

                        node.load = node\_info["load"]

                        node.last\_update = datetime.strptime(

                            node\_info["last\_update"], "%Y-%m-%d %H:%M:%S"

                        )

    except *Exception* as e:

        logging.error(*f*"Ошибка при загрузке данных о загрузке узлов: {e}")

*def* save\_node\_loads():

    """Сохраняет текущее состояние узлов в файл"""

    try:

        node\_data = {

*str*(node.node\_id): {

                "load": node.load,

                "last\_update": node.last\_update.strftime("%Y-%m-%d %H:%M:%S")

            }

            for node in NODES

        }

        with open(NODE\_LOADS\_FILE, "w") as file:

            json.dump(node\_data, file, *indent*=4)

    except *Exception* as e:

        logging.error(*f*"Ошибка при сохранении данных о загрузке узлов: {e}")

**2. Создание списка узлов**

При запуске создаются 3 узла:

NODES = [Node(i) for i in range(3)]

Каждый узел начинает с нулевой нагрузкой.

**3. Выбор узла с наименьшей нагрузкой**

Функция **select\_node**() выбирает узел, у которого минимальная загрузка:

def select\_node():

return min(NODES, key=lambda node: node.load)

**4. Обновление нагрузки узла**

Когда выполняется проверка email, URL или IP, нагрузка увеличивается:

**node.update\_load()**

После завершения проверки нагрузка уменьшается:

**node.decrease\_load()**

**5. Сохранение и загрузка нагрузки**

Нагрузка узлов сохраняется в node\_loads.json:

*def* save\_node\_loads():

    """Сохраняет текущее состояние узлов в файл"""

    try:

        node\_data = {

*str*(node.node\_id): {

                "load": node.load,

                "last\_update": node.last\_update.strftime("%Y-%m-%d %H:%M:%S")

            }

            for node in NODES

        }

        with open(NODE\_LOADS\_FILE, "w") as file:

            json.dump(node\_data, file, *indent*=4)

    except *Exception* as e:

        logging.error(*f*"Ошибка при сохранении данных о загрузке узлов: {e}")

**При старте бота загрузка восстанавливается из файла:**

def load\_node\_loads():

with open(NODE\_LOADS\_FILE, "r") as file:

node\_data = json.load(file)

for node in NODES:

node\_info = node\_data.get(str(node.node\_id))

if node\_info:

node.load = node\_info["load"]

**6. Алгоритм "Эхо" и балансировка**

Функция echo\_algorithm собирает информацию о нагрузке узлов:

*async* *def* echo\_algorithm(*node\_id*, *load\_data*):

    """Имитация волнового алгоритма 'Эхо' для сбора данных о загрузке узлов"""

    node\_loads = load\_node\_loads()

    node\_loads[*node\_id*] = *load\_data*

    save\_node\_loads(node\_loads)

    logging.info(*f*"Собрана информация о загрузке узла {*node\_id*}: {*load\_data*}")

    # После сбора данных алгоритм принимает решение о балансировке

    await make\_balancing\_decision(node\_loads)

**После сбора данных вызывается make\_balancing\_decision(), которая решает, нужно ли перенести объекты между узлами:**

    # Функция для принятия решения о балансировке

*async* *def* make\_balancing\_decision(*node\_loads*):

    """Принятие решения о переносе объекта на основе текущих данных о загрузке"""

    # Пример логики балансировки: переносим объект с самого загруженного узла на менее загруженный

    sorted\_nodes = sorted(*node\_loads*.items(), key=*lambda* *x*: *x*[1], *reverse*=True)

    most\_loaded\_node = sorted\_nodes[0]

    least\_loaded\_node = sorted\_nodes[-1]

    # Принятие решения о переносе

    logging.info(*f*"Балансировка: перенос объекта с узла {most\_loaded\_node[0]} на узел {least\_loaded\_node[0]}")

    # Логика переноса объекта (например, отправка сообщения)

    await bot.send\_message(most\_loaded\_node[0], *f*"🔄 Ваш узел перегружен, объект будет перемещен.")

    await bot.send\_message(least\_loaded\_node[0], *f*"💻 Вы получили новый объект для обработки.")

После сбора данных вызывается make\_balancing\_decision(), которая решает, нужно ли перенести объекты между узлами:

*async* *def* make\_balancing\_decision(*node\_loads*):

    """Принятие решения о переносе объекта на основе текущих данных о загрузке"""

    # Пример логики балансировки: переносим объект с самого загруженного узла на менее загруженный

    sorted\_nodes = sorted(*node\_loads*.items(), key=*lambda* *x*: *x*[1], *reverse*=True)

    most\_loaded\_node = sorted\_nodes[0]

    least\_loaded\_node = sorted\_nodes[-1]

    # Принятие решения о переносе

    logging.info(*f*"Балансировка: перенос объекта с узла {most\_loaded\_node[0]} на узел {least\_loaded\_node[0]}")