ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА
Факултет по изчислителна техника и автоматизация
Софтуерни и интернет технологии

Схематично представяне работата на примерни методи от библиотеката по дипломна работа на **тема** „Разработване на библиотека за изграждане на софтуерни системи работещи в реално време“

Изготвил: Велислав Василев Колесниченко

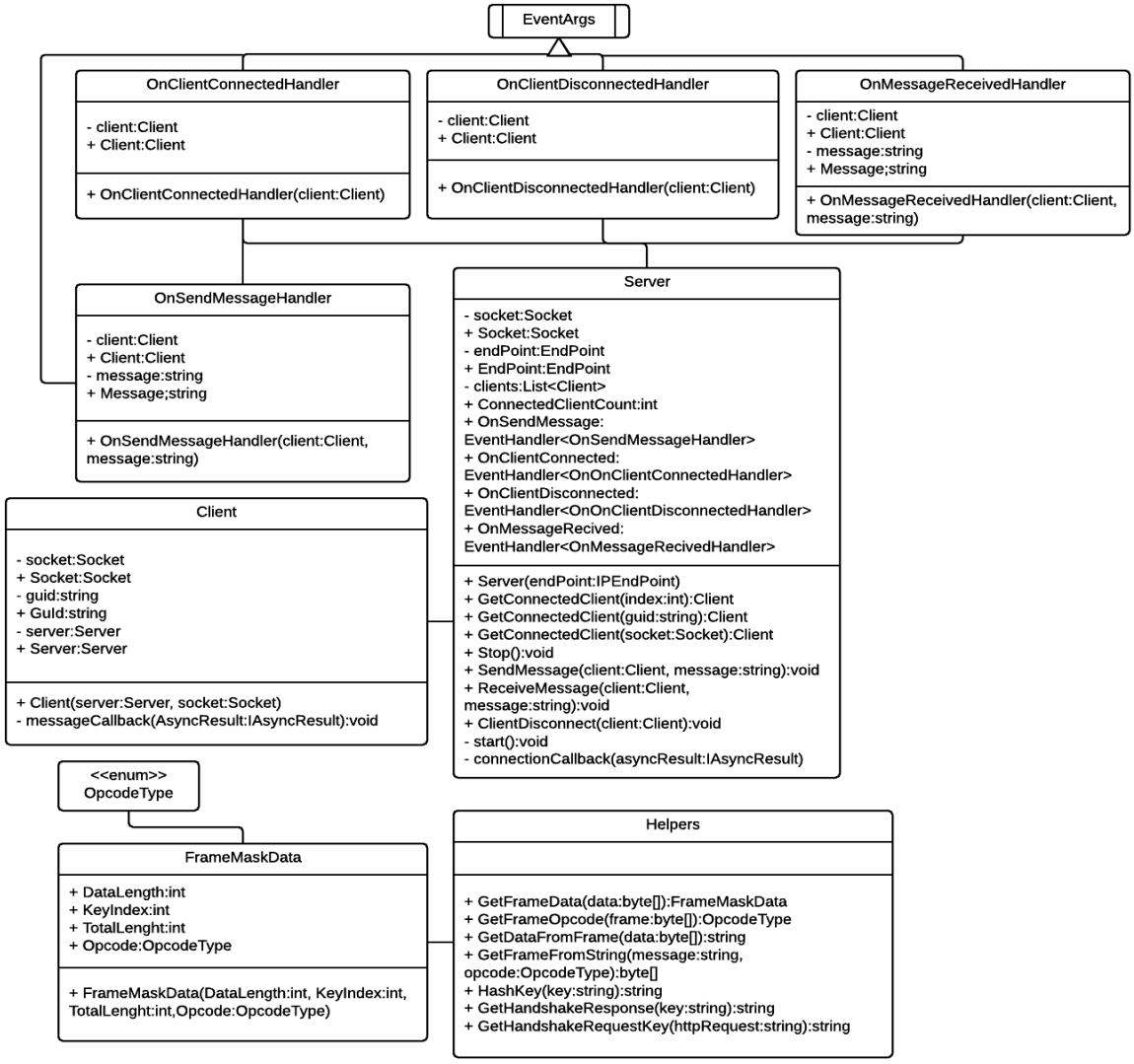
Ръководител: доц. Хр. Вълчанов

Специалност: Софтуерно инженерство

Факултетен номер: 61562005

ТУ Варна, 2017 г.

Библиотеката включва два основни модула сървър за обработване на логиката на приложенията и клиент за потребителското взаимодействие. Модулът за логиката се реализира със следните класове:



Фигура 1: Диаграма на класовете на библиотеката за създаване на приложения работещи в реално време

Библиотеката имплементира WebSocket протокола които използвам, за да реализирам предаването на данни в реално време. За да мога да използвам протокола имплементирам методи, за приемане на WebSocket пакет (представен в точка 3.1.4.3 на фигура 9)[1] и извличане от тях на значещата информация. Протоколът е имплементиран в класа Helpers които предоставя методи за управление на протокола:

- Метода GetHandshakeRequestKey проверява валидността на заглавната част на заявката за свързване ако тя е валидна и опита за

[1] От дипломната записка

свързване е за осъществяване на връзка през WebSocket се връща кода за сигурност в заглавната част на отговора.

```
public static string GetHandshakeRequestKey(string httpRequest)
{
   int keyStart = httpRequest.IndexOf("Dev-WebSocket-Key: ") + 19;
   string key = null;
   for (int i = keyStart; i < (keyStart + 24); i++)
   {
      key += httpRequest[i];
   }
   return key;
}
```

За да е валидна заявката заглавната и част (представен в точка 3.1.4.2 на фигура 7) [2] трябва да съдържа ключа „Sec-WebSocket-Key:“.

- Метода GetHandshakeResponse връща заглавната част на отговора за заявката за свързване от сървъра към клиента.

```
public static string GetHandshakeResponse(string key)
{
   return string.Format("HTTP/1.1 101 Switching Protocols\nUpgrade: WebSocket\nConnection: Upgrade\nSec-WebSocket-Accept: {0}\r\n\r\n", key);
}
```

Отговорът съдържа известие за клиента, което служи за прехвърляне на комуникацията от HTTP протокола към WebSocket протокола.

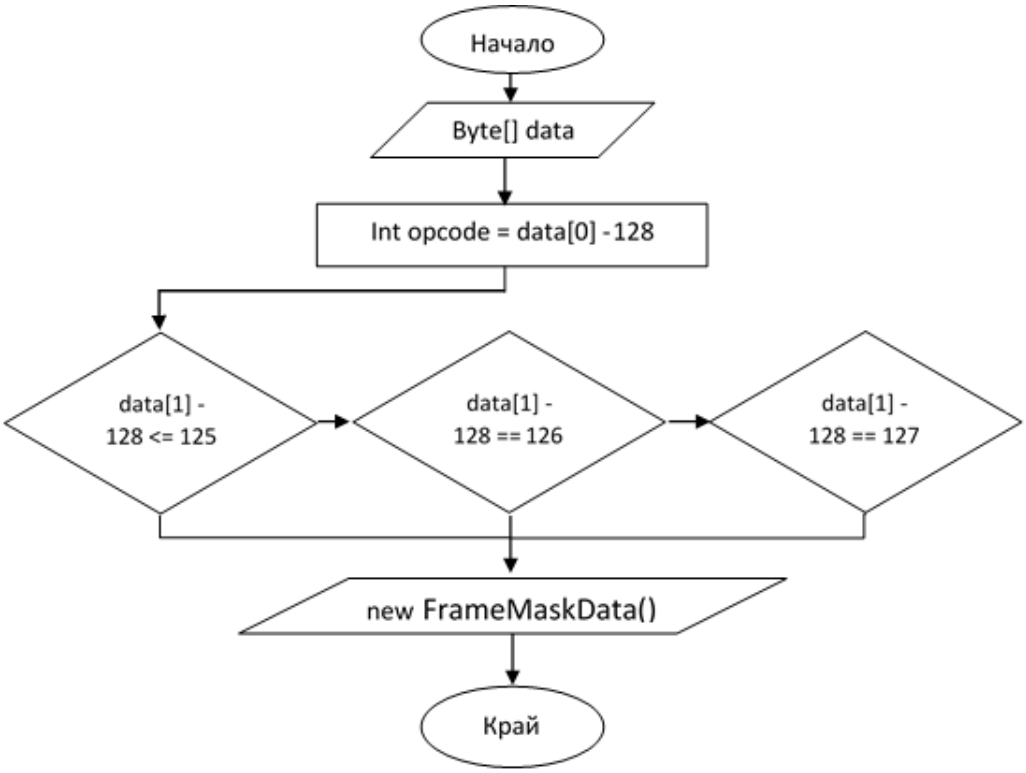
- Метода GetFrameOpcode извлича кода на пакета дефиниран в енумерацията OpcodeType. Основните стойностите на енумерацията са кодовете Text = 1 за текстово предавани данни и ClosedConnection = 8 за прекратена връзка.

```
public static OpcodeType GetFrameOpcode(byte[] frame)
{
   return (OpcodeType)frame[0] - 128;
}
```

От получения байтов пакет се извлича първия байт от които се изместват не значещите битове за да се извлече стойността на кода от четвъртия бит.

- Метода GetFrameData извлича допълнителните данни за пакета дължината на пакета вида и размера на предаваните данни.

[2] От дипломната записка



Фигура 2: Схема на обработването на WebSocket пакета

В зависимост от дължината на пакета между 10 и 46 бит се съдържа дължината на полезния товар в пакета. Следа като се определи размера на пакета кода му и размера на предаваните данни се създава обект с информацията за пакета.

- Методите GetDataFromFrame и GetFrameFromString служи съответно за прочитане на данните от пакета и образуване на пакет съдържащ данните за пренос.

Основният клас в библиотеката е Server той служи за създаване на услуги за управление на логиката на приложенията. Класът съдържа крайната точка на услугата в полето endPoint и сокета за връзка в полето socket, списък със свързаните клиенти в полето List<Client> client. Той дефинира събитията изпращане на съобщение OnSendMessage, свързване OnClientConnected, получаване на съобщение OnMessageReceived, разкачане OnClientDisconnected. Класът Server дефинира методите:

- SendMessage – метода служи за изпращане на съобщение до подаден клиент. Изпращането на съобщението пробужда събитие за изпратено съобщение което активира изпълнението на потребителските методи закачени към това събитие.
- ClientDisconnect – метода служи за прекратяване на връзка на клиент със сървър. Прекратяването на връзката пробужда събитие за изпратено

съобщение което активира изпълнението на потребителските методи закачени към това събитие.

- connectionCallback – метод вътрешен за класа служещ за обработване на заявката за свързване.

```
private void connectionCallback(IAsyncResult asyncResult)
{
//Стъпка 1
    Socket clientSocket = Socket.EndAccept(asyncResult);
    byte[] handshakeBuffer = new byte[1024];
     int handshakeReceived = clientSocket.Receive(handshakeBuffer);
    String requestKey = Helpers.GetHandshakeRequestKey(
                            Encoding.Default.GetString(handshakeBuffer));
          string hanshakeResponse = Helpers.GetHandshakeResponse(
                                                  Helpers.HashKey(requestKey));
          clientSocket.Send(Encoding.Default.GetBytes(hanshakeResponse));

//Стъпка 2
          Client client = new Client(this, clientSocket);
          clients.Add(client);

//Стъпка 3
          if (OnClientConnected == null) throw new Exception("Server error: event
OnClientConnected is not bound!");
          OnClientConnected(this, new OnClientConnectedHandler(client));
          Socket.BeginAccept(connectionCallback, null);
     }
```

Когато се получи заявка за свързване като първа стъпка се прочита заявката на клиента, който иска да се свържи и се проверява заглавието на заявката ако е валидна се извлича ключа на заявката и се образува отговора за подмяна на протокола за комуникация, при невалидна заявка от клиент не се осъществява връзка. Като втора стъпка се създава нов обект, клиент в списъка на сървъра със сокета на клиента. Третата стъпка е пробуждане на събитие за свързване което активира изпълнението на потребителските методи закачени към това събитие.

Класът Client съдържа описанието на свързаните клиенти към сървъра. В полето socket се съхранява информация за сокета на клиента, в полето guid се съхранява уникален идентификатор на клиента, в полето server се съхранява информация за сървъра към които е свързан клиента.

- Метода messageCallback е вътрешен за класа Client, който служи за обработка на получените съобщения от клиента. Когато възникне извикване първо се извлича пренасящия пакет. Като втора стъпка се извлича кода на пакета ако той е с код ClosedConnection или при невалиден код се извиква метода на сървъра за прекратяване на връзката. В третата стъпка се извличат данните от пакета и се пробужда събитието за получаване на съобщение от сървъра.

```
private void messageCallback(IAsyncResult AsyncResult)
{
//Стъпка 1
        Socket.EndReceive(AsyncResult);
        byte[] messageBuffer = new byte[255];
        int bytesReceived = Socket.Receive(messageBuffer);
//Стъпка 2
        OpcodeType opcode = Helpers.GetFrameOpcode(messageBuffer);
          if (opcode == OpcodeType.ClosedConnection) {
             Server.ClientDisconnect(this);
             return;
          }

//Стъпка 3
        Server.ReceiveMessage(this,
                                Helpers.GetDataFromFrame(messageBuffer));
        Socket.BeginReceive(new byte[] { 0 }, 0, 0, SocketFlags.None,
                                                          messageCallback, null);
}
```

Клиентският модул имплементира WebSocket Api на W3C. За целта е създаден JavaScript обекта WebSocketClient, който предоставя интерфейс за достъп до WebSocket Api. Обекта има полета:

- url за адреса на услугата
- wsInitSelector приема HTML елемент, който активира свързването към сървъра
- wsSendSelector приема HTML елемент, който активира изпращане на съобщение към сървъра
- OnOpen метод за обратно извикване, който се изпълнява при свързване към сървъра
- OnClose метод за обратно извикване, който се изпълнява при прекратяване на връзката към сървъра

- OnError метод за обратно извикване, който се изпълнява при възникнала грешка в комуникацията
- Send метод за обратно извикване, който образува съобщението за изпращане и се изпълнява при изпращане на съобщение към сървъра

Използване на библиотеката за приложение за споделяне сървърно на време:

Сървърна логика:

```
Server server = new Server(new IPEndPoint(IPAddress.Parse("127.0.0.1"), 8181));

server.OnClientConnected += (object sender, OnClientConnectedHandler e) =>
{
	Console.WriteLine("Client with GUID: {0} Connected!", e.Client.GuId);
};
server.OnClientDisconnected += (object sender, OnClientDisconnectedHandler e) =>
{
	Console.WriteLine("Client {0} Disconnected", e.Client.GuId);
 };
server.OnMessageReceived += (object sender, OnMessageReceivedHandler e) =>
{
	Console.WriteLine("Received Message: '{1}' for client: {0}", e.Client.GuId,
e.Message);
	while (true) {
		Thread.Sleep(1000);
		 for (int c = 0; c < e.Client.Server.ConnectedClientCount; c++)
		{
			server.SendMessage(e.Client.Server.GetConnectedClient(c),
                                                        DateTime.Now.ToString());
		}
	}
};
server.OnSendMessage += (object sender, OnSendMessageHandler e) =>
{
	Console.WriteLine("Sony message: '{0}' to client {1}", e.Message, e.Client.GuId);
};
```

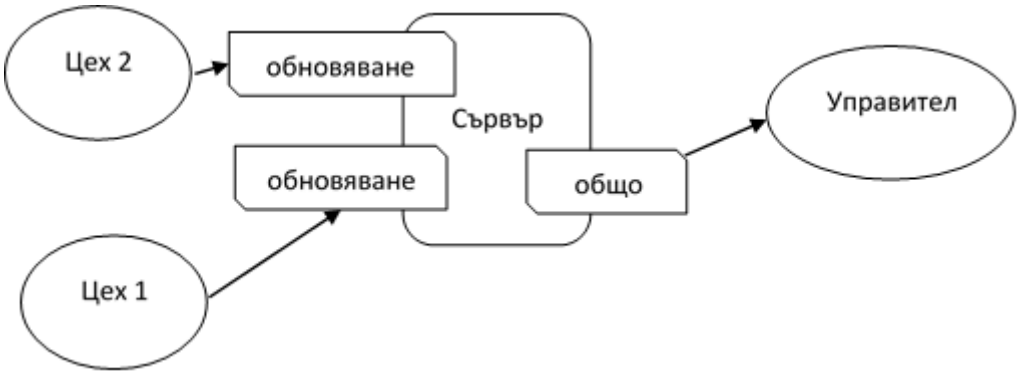
Клиентска логика:

```
var timer = new WebSocketClient({
	url: "ws://localhost:8181/",
	wsInitSelector: document.getElementById("demo"),
```

```
        wsSwndSelector: document.getElementById("send"),
        OnSend: function () { return "Start Timer"; },
        OnOpen: function () { console.log('WebSocket connection Open 1'); },
        OnMessage: function (e) { clear("test"); appendChild("test", e); },
});
```

Библиотеката използва реактивния модел на работа (представен в точка 2.3.1)[3] заимстван от функционалните езици за програмиране, където всеки обект е функция и всяка функция връща като резултат функция. Този модел се използва за четирите основни събития при комуникация свързване и разкачане на клиент, изпращане и получаване на съобщения. С този модел реализирам работата в реално време, за да може да се изпълни изискването за своевременна реакция. Начина на използване може да се види в следващите примери:

- Първият пример показва използването на библиотеката за получаване на общото количество извършена работа от два цеха от управителя. Двата цех произвежда една суровина независимо един от друг.



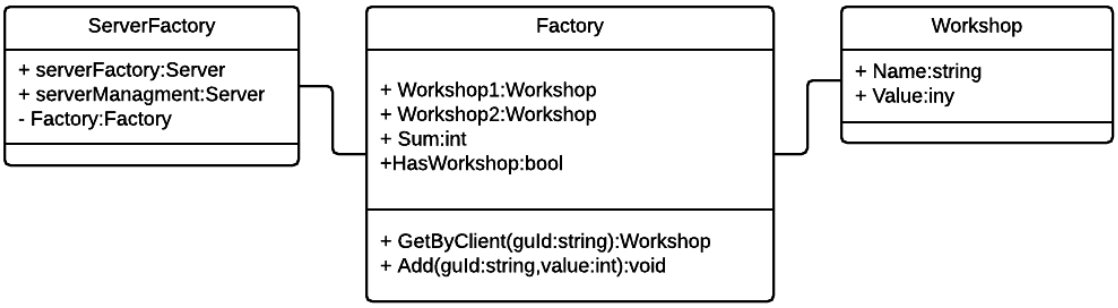
Фигура 3: Архитектура на приложение за производство във фабрика

Реализацията на схемата от фигура 3 включва:

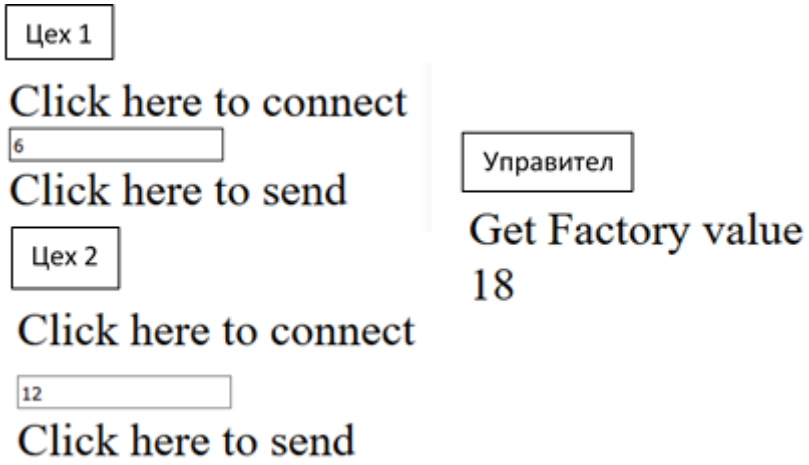
- Сървър имплементиращ две услуги една за обновяване на данните на произведеното от цеховете и една за получаване на общото изработено количество;

[3] От дипломната записка

- Клиент за цеховете, който изпраща данните към сървъра и клиент за управителя които получава общите данни.



Фигура 5: Диаграма на класовете на приложения за производство във фабриак



Фигура 4: Изглед на работата на приложения за производство във фабриак

Класът Workshop държи информация за цеховете в свойството Name името на цеха и в свойството Value произведеното от цеха. Класа Factory е фабриката в която са двата цеха свойствата Workshop1 и Workshop2 държат данните на двата цеха. Свойството Sum е общата извършена работа от двата цеха. Класа ServerFactory съдържа защитено свойство Factory с данните на фабриката и две свойства от тип Server за услугата за обновяване на данните на цеховете serverFactory и serverManagment за услугата по извличане на общата извършена работа.

Когато цех 1 обнови стойността на извършената работа, той изпраща съобщение към serverFactory, който когато получи новата стойност извиква обекта на съответния цех и обновява стойността му. След което в свойството Sum вече се съхранява стойността на новата обща сума, която се изпраща до свързаните към услугата serverManagment клиенти.

```
serverFactory.OnMessageReceived += (object sender, OnMessageReceivedHandler e)
=> {
        if (Factory.HasWorkshop) {
                Workshop Workshop = Factory.GetByClient(e.Client.GuId);
                Workshop.Value = Convert.ToInt32(e.Message);
        }
        Else {
                Factory.Add(e.Client.GuId, Convert.ToInt32(e.Message));
        }

        for (int c = 0; c < serverManagment.ConnectedClientCount; c++) {
                serverManagment
                        .SendMessage(serverManagment.GetConnectedClient(c),
                                        Factory.Sum.ToString());
        } };
```

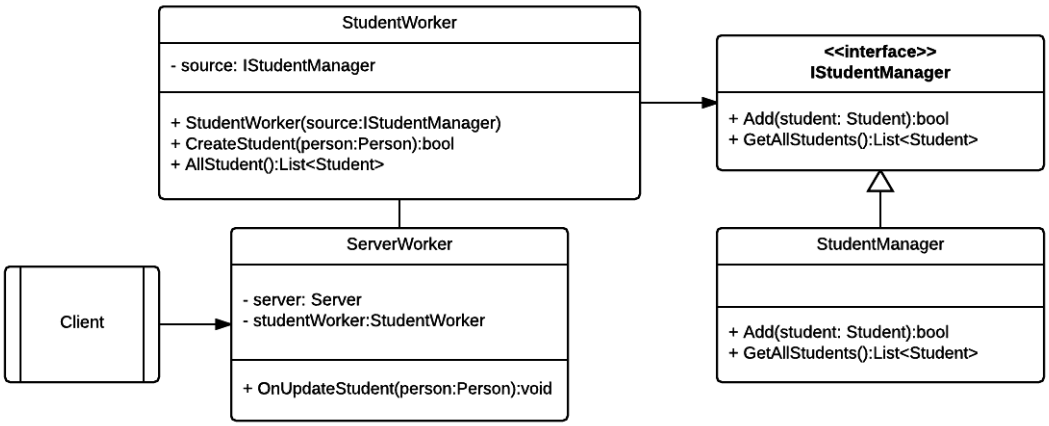
По този начин реализацията отговаря на изискването за своевременна реакция на системите обработващи данни в реално време.

Друг пример е използването му в информационна система за студенти. Целта на този пример е да се предава информация до няколко клиенти едновременно. Посредством реактивния модел при промяна да данните от един клиент всички останали получат обновленията.

Фигура 6: Архитектура на приложение за информационна система за студенти.

Реализацията на схемата от фигура 6 включва:

- Сървър имплементиращ услуга за обновяване на студентска информация, промяната на данните задейства изпращане до всички заинтересувани от промяната;
- Клиент който осъществява потребителското въздействие със системата.



Фигура 7: Диаграма на класовете на приложение за информационна система за студенти.

Интерфейсът IStudentManager дефинира действията, който могат да се извършват в системата. Класът StudentManager имплементира действията в системата. Класът StudentWorker служи за управление на студентската система. Класът StudentWorker съдържа свойство от тип Server което имплементира услугата за работа с информационната система за студенти.

Събитието на сървъра за свързване на нов клиент поражда изпращане на данните към клиента предизвикал събитието.

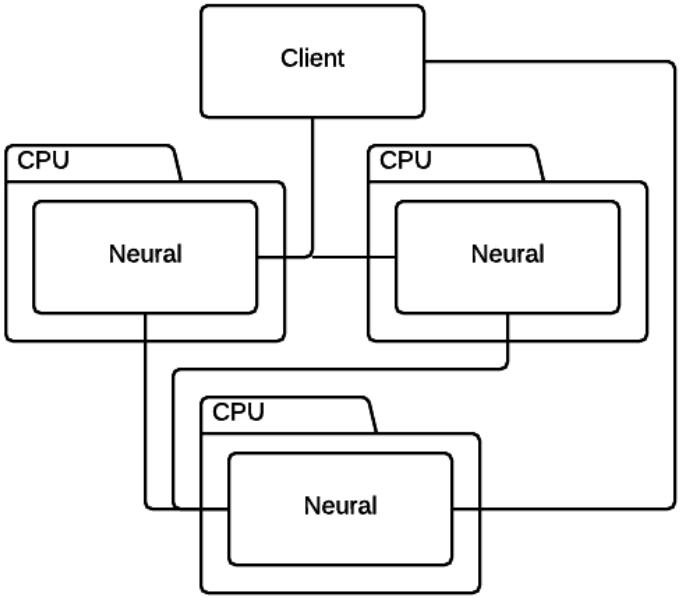
```
server.OnClientConnected += (object sender, OnClientConnectedHandler e) => {
        server.SendMessage(e.Client,
                        new JavaScriptSerializer().Serialize(manager.GetAllStudents()));
};
```

Когото се породи клиентско съобщение за обновяване на данните след обработката им се изпраща съобщение за обновяване до всички клиенти.

```
server.OnMessageReceived += (object sender, OnMessageReceivedHandler e) => {
        Student student = new JavaScriptSerializer().Deserialize<Student>(e.Message);
        if (student.Id == 0) {
                bool state = manager.Add(student);
        } else {
                bool state = manager.Save(student);
        }
        for (int c = 0; c < e.Client.Server.ConnectedClientCount; c++) {
        server.SendMessage(e.Client.Server.GetConnectedClient(c),
                        new JavaScriptSerializer().Serialize(manager.GetAllStudents()));
        } };
```

По този начин актуализираната информация достига до всички заинтересовани клиенти в момент, в който настъпи промяната. Всеки клиент получава независимо обновлението на данните.

Пример за използване на библиотеката в реализирането за невронна мрежа. Целта на този пример е да покаже как библиотеката може да се използва за създаване на приложения с различни архитектури.

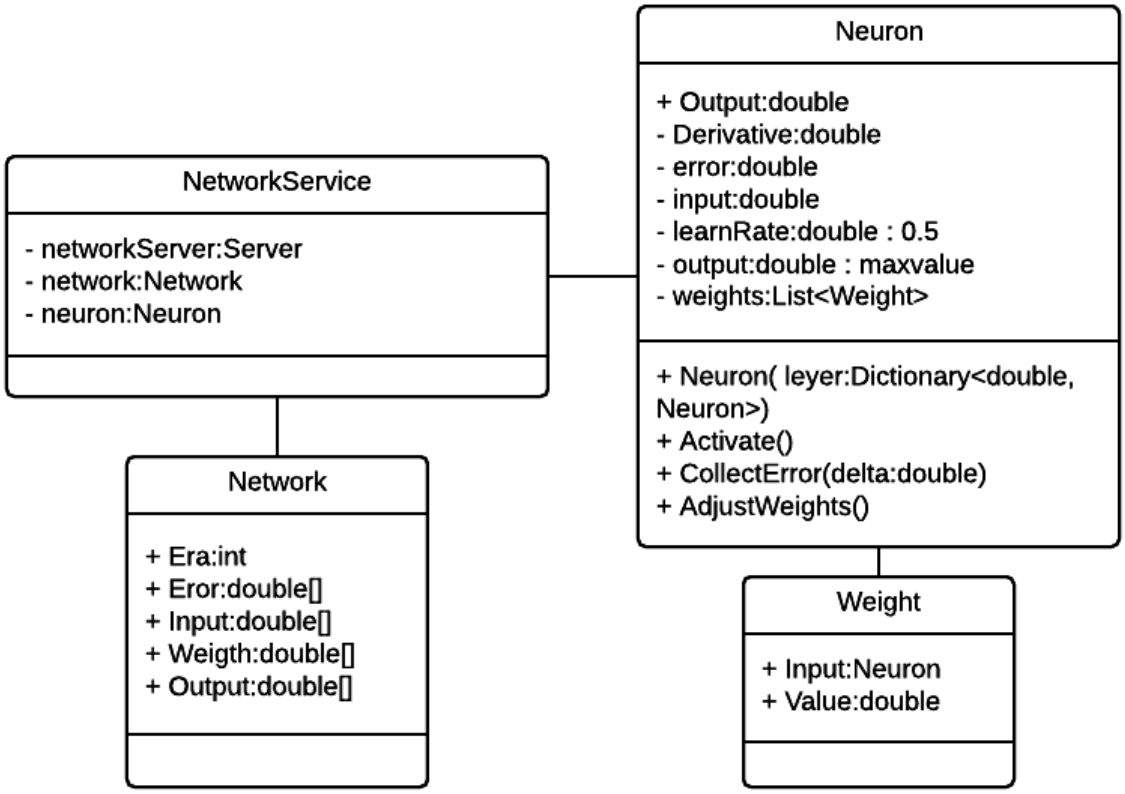


Фигура 8: Архитектура на приложение за невронна мрежа

При невронната мрежа първия неврон получава съобщение за обработка на входен сигнал. Когато е готов с обработката подава сигнал към следващия слои в мрежата. В момента в които невроните обработят входните сигнали от съобщението те го предават на изходния неврон. Работата на всеки неврон завършва по различно време което създава нуждата изходния неврон да изчака получаването на всички входни съобщения преди да го обработи. Изходния неврон след като получи резултата го предава към първия по обратната връзка. Ако резултатната грешка граничи с минималната грешка мрежата е обучена в противен случаи се коригират теглата на връзките между невроните в мрежата и обучението продължава.

Реализацията на схемата от фигура 6 включва:

- Сървър имплементиращ услуга за активиране на неврон и изчисляване тегловните коефициенти на връзките между невроните описани в точки 4.2;
- Клиент, който служи за първоначално инициализиране на невронната мрежа и проследяване прогреса на обучение.



Фигура 9: Диаграма на класовете на приложение за невронна мрежа

Класът Network дефинира ерата на обучение, входа към следващия слой с неврони, теглата на дъгите свързващи невроните, еталонния изход и грешката. Network е транспортен клас за пренасяне на данните за мрежата между отделните модули.

Класът Weight съхранява в свойството Value теглото на дъгата и неврона в свойството Input с които е свързан един неврон от класа Neuron.

Класът Neuron имплементира логиката за обучение. Свойство Output връща изхода на неврона. Свойството error връща грешката при обучение на текущия неврон, а полето weights съхранява свързаните неврони. Метода Activate изчислява входната сума. Метода CollectError натрупва грешката от свързаните неврони. Метода AdjustWeights коригира стойностите на теглата на дъгите между невроните.

NetworkService имплементира обекта от библиотеката Server. Полето network дефинира невронната мрежа слоевете и еталонните данни. Полето neuron съхранява неврона, които се обучава.

Събитието на сървъра за свързване на нов клиент поражда изпращане на данните за мрежата към клиента предизвикал събитието.

```
server.OnClientConnected += (object sender, OnClientConnectedHandler e) =>
{
        server.SendMessage(e.Client, new JavaScriptSerializer().Serialize(network));
};
```

Клиентът пробужда обучението на мрежата, при изпращане на съобщение със структурата на мрежата която се съхранява в JSON обект обновяван след всяка промяна при обучението на мрежата.

```
        var timer = new WebSocketClient({
           url: "ws://localhost:8186/",
           wsInitSelector: document.getElementById("demo"),
           wsSwndSelector: document.getElementById("send"),
           OnSend: function () {
              return JSON.stringify(network);
           },
           OnOpen: function () { console.log('WebSocket connection Open
NeuronNetwork 1'); },
           OnMessage: function (e) {
              //clear("test");
              appendChild("test", e);
              network = JSON.parse(e);
           },
        });
```

Всеки цикъл на обучение поражда изпращане на актуалните данни за обучението към клиентите.

```
server.OnMessageReceived += (object sender, OnMessageReceivedHandler e) =>
{
//Стъпка 1
Network network = new JavaScriptSerializer().Deserialize<Network>(e.Message);

           Dictionary<double, Neuron> leyer = new Dictionary<double, Neuron>();

           Neuron neuron0 = new Neuron();
           neuron0.Output = wetwork.Input[0];

           leyer.Add(wetwork.Weigth[0], neuron0);
           leyer.Add(wetwork.Weigth[1], neuron1);
           Neuron neuron = new Neuron(leyer);
//Стъпка 2
           neuron.Activate();

              neuron2.CollectError(network.Error[2]);
              neuron3.CollectError(network.Error[3]);
```

```
            neuron2.AdjustWeights();
            neuron3.AdjustWeights();
//Стъпка 3
                network.Input = new double[] { neuron2.Output, neuron3.Output };
                        network.Error[0] = neuron2.error;
                network.Error[1] = neuron3.error;
                neuron2.weights[0].Value, neuron2.weights[1].Value,
                        neuron2.weights[2].Value, neuron2.weights[3].Value };
                        network.Weigth[0] = neuron2.weights[0].Value;
                network.Weigth[1] = neuron2.weights[1].Value;
                network.Weigth[2] = neuron3.weights[0].Value;
                network.Weigth[3] = neuron3.weights[1].Value;

server.SendMessage(e.Client, new JavaScriptSerializer().Serialize(network));
};
```

Първата стъпка при получаване на съобщение за обучение е попълване на network обект със описанието на мрежата, от който се създава неврона (Neuron) за обучение. Втората стъпка е обучението на неврона, изчислението на грешката и коригирането на теглата на дъгите. Като трета стъпка се обновява network обекта и се изпраща към следващата стъпка на обучението.

В примера всеки слои с неврони е отделен процес със собствена производствена мощ. По този начин се постига независимост на изчисленията, за да се постигне по добра изчислителна скорост по същия начин невроните могат да се разделят в различни процеси. Използването на библиотеката за обработка на данни в реално време позволява актуализираната информация да достига до всички заинтересовани клиенти в момент, в който настъпи промяната. Вместо всеки да изпраща заявки със запитвания дали данните са готови за използване. Всеки клиент получава независимо обновлението на данните.