ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

КАФЕДРА ИНФОРМАЦИОННЫХ И РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

СИСТЕМА ДИСТАНЦИОННОГО МОНИТОРИНГА СЕРДЕЧНОЙ ДЕЯТЕЛЬНОСТИ ПАЦИЕНТА

Выпускная квалификационная работа
обучающегося по направлению подготовки
12.03.04 Биотехнические системы и технологии
очной формы обучения, группы 12001514
Безвесельного Евгения Евгеньевича

Научный руководитель
ст. преподаватель
Унковский А.В.

БЕЛГОРОД 2019

РЕФЕРАТ

Система дистанционного мониторинга сердечной деятельности пациента. – Безвесельный Евгений Евгеньевич, выпуская квалификационная работа бакалавра, Белгород, Белгородский государственный национальный исследовательский университет (НИУ «БелГУ»), количество страниц 75, включая приложения 99, количество рисунков 47, количество использованных источников 27, количество таблиц 1.

КЛЮЧЕВЫЕ СЛОВА: удаленная диагностика, мониторинг сердечной деятельности, дистанционная передача сигнала, ЭКГ, Arduino Leonardo, Raspberry Pi.

ОБЪЕКТ ИССЛЕДОВАНИЯ: процессы регистрации сердечного ритма человека и передачи его с помощью сети Интернет.

ПРЕДМЕТ ИССЛЕДОВАНИЯ: методы отправки и приема данных с помощью сети Интернет.

ЦЕЛЬ РАБОТЫ: обеспечение удаленного доступа к физиологическим данным пациента за счет разработки системы дистанционного мониторинга сердечной деятельности

ЗАДАЧИ ИССЛЕДОВАНИЯ: обзор существующих средств диагностики и мониторинга сердечной деятельности; проектирование и разработка аппаратной части системы; проектирование и разработка программного обеспечения для аппаратной части; проектирование и разработка веб-сервиса.

МЕТОДЫ ИССЛЕДОВАНИЯ: отправка данных с использованием HTTP-протокола, обработка сигнала с помощью медианного фильтра.

ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ: спроектирована и разработана система дистанционного мониторинга сердечной деятельности пациента.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4
1 Обзор существующих средств дистанционного мониторинга сердечной деятельности человека 6
1.1 Система дистанционной передачи, централизованного анализа и архивирования 6
1.2 Дистанционная система мониторинга «Самоконтроль ЭКГ» 12
2 Проектирование и разработка аппаратной части системы дистанционного мониторинга сердечной деятельности пациента 17
2.1 Проектирование модульной схемы устройства 17
2.2 Выбор аппаратных составляющих устройства 19
2.3 Проектирование монтажной схемы устройства 25
3 Проектирование и разработка программного обеспечения системы дистанционного мониторинга сердечной деятельности пациента 28
3.1 Разработка программного драйвера микроконтроллера и программного фильтра 28
3.2 Разработка программного обеспечения микрокомпьютера для работы с микроконтроллером 33
3.2.1 Настройка параметров работы микрокомпьютера Raspberry Pi 3 33
3.2.2 Разработка программного драйвера для микроконтроллера 36
3.2.3 Проектирование и разработка пользовательского приложения 40
3.3 Проектирование и разработка веб-сервиса системы 52
3.3.1 Настройка параметров работы веб-сервера 52
3.3.2 Проектирование и реализация базы данных 54
3.3.3 Проектирование и реализация веб-интерфейсов системы 56
3.4 SWOT-анализ системы дистанционного мониторинга сердечной деятельности 72
ЗАКЛЮЧЕНИЕ 73
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 75
ПРИЛОЖЕНИЕ А 78
ПРИЛОЖЕНИЕ Б 80
ПРИЛОЖЕНИЕ В 84
ПРИЛОЖЕНИЕ Г 92

ВВЕДЕНИЕ

Автоматизация – одно из направлений технического и научного прогресса, нацеленного на освобождение человека от участия в каком-либо процессе. В силу развития информационных технологий уровень автоматизации во всех сферах деятельности человека постоянно повышается, это позволяет, в первую очередь, автоматизировать процессы любой сложности; создать универсальные средства автоматизации, что значит, обезопасить процессы от риска вмешательства человеческого фактора. Последнее, в определенных сферах научного познания, играет решающую роль.

Ярким примером, где человеческий фактор может привести к фатальным последствиям, является медицина – сфера деятельности, где предмет изучения и взаимодействия представляет собой человеческое здоровье, а в особенных случаях – жизнь.

Однако, пока что нельзя говорить об абсолютном уровне интеграции информационных технологий и медицины, в силу того, что полная алгоритмизация биохимических процессов явления жизни белкового организма в настоящее временя невозможна из-за того, что данные процессы не изучены полностью.

Но, следует отметить, что внедрение информационных технологий в медицину является перспективной деятельностью, так как этот процесс нацелен на уменьшение риска вмешательства человеческого фактора. И, на данный момент времени, процесс внедрения протекает медленно, но успешно.

Примерами успешного смешения двух областей научного познания могут послужить такие нововведения как: электронные очереди, которые пришли на замену обыкновенным «живым» очередям, где пациентам приходится ожидать своего приема непосредственно в лечебном учреждении; электронные базы данных, которые пришли на замену бумажным картотекам и каталогам; внедрение электронных методов хранения информации (различные

медицинские снимки могут хранится в более детализированном и интерактивном виде: цифровые изображения); цифровые методы передачи медицинских данных – телемедицина; цифровые методы обработки сигналов (обработка рентгенограмм, электрокардиографических данных и т. п.).

Во всех перечисленных случаях, основополагающим мотивом для интеграции информационных технологий в медицину является возможность автоматизирования каких-либо процессов, с целью экономии времени, средств, а самое главное, уменьшения риска вмешательства человеческого фактора.

Исходя из всего вышеописанного можно сделать вывод, что интеграция информационных технологий в медицинской практике с целью автоматизации, является одной из самых перспективных и актуальных отраслей совместного развития двух научных сфер.

Данная выпускная квалификационная работа направлена на разработку системы автоматизированного дистанционного мониторинга сердечной деятельности человека.

Посредством данной системы станет возможным производить дистанционную передачу данных ЭКГ в медицинский центр от пациента, который находится либо на амбулаторном лечении, либо же пребывает на лечении в стационаре.

Цель выпускной квалификационной работы: обеспечение удаленного доступа к физиологическим данным пациента засчет разработки системы дистанционного мониторинга сердечной деятельности.

Для достижения поставленной цели необходимо решить следующие задачи:

а) обзор средств диагностики и мониторинга сердечной деятельности;

б) проектирование и разработка аппаратной части системы;

в) проектирование и разработка программного обеспечения для аппаратной части;

г) проектирование и разработка веб-сервиса дистанционной системы мониторинга сердечной деятельности пациента.

1 Обзор существующих средств дистанционного мониторинга сердечной деятельности человека

1.1 Система дистанционной передачи, централизованного анализа и архивирования

Автоматизация медицинских процедур – перспективная отрасль развития как медицины, так и информационных технологий. На данный момент времени, в странах СНГ, хотя и существуют некоторые разработки, эффективность которых доказана, но уровень их интеграции в медицинские учреждения довольно низок, особенно, если речь идет о малонаселенных или дальних пунктах проживания людей [1].

Несмотря на то, что уровень внедрения таких технологий находится на довольно низкой отметке, сам факт проведения таких работ существует.

Так, например, федеральное государственное бюджетное учреждение «Российский кардиологический научно-производственный комплекс» Министерства здравоохранения и социального развития Российской Федерации имеет опыт с внедрением информационных систем в работу кардиологических исследований [1]. Система называется «Дистанционная передача ЭКГ и системы централизованного анализа и архивирования ЭКГ». Эта разработка заключается в использовании мобильных устройств и компьютеров в качестве регистраторов ЭКГ-данных и их передачи посредством сети Интернет, либо посредством использования телефонной линии [9]. Данные передаются в медицинский центр и хранятся на специальном сервере. Затем полученные данные ЭКГ распределяются по врачам-кардиологам, которые в свою очередь проводят анализ ЭКГ на своих персональных компьютерах.

Им удалось разработать три типа устройств, каждый из которых отличался своим предназначением. Например, для снятия ЭКГ в условиях

экстренных ситуаций, когда медицинский персонал выезжает на вызов скорой помощи, используется так называемый мобильный регистратор [10].

Он представляет собой устройство на базе смартфона или карманного компьютера под управлением операционной системы Windows. Данное устройство подключается к датчикам ЭКГ, предварительно закрепленными на пациенте, снимет показатели, сохраняет их в энергонезависимой памяти смартфона или карманного компьютера. После этого, в зависимости от наличия подключения сети Интернет, данные исследования передаются в медицинский центр на центральный сервер. В случае, если произвести соединение с сетью Интернет не удается, информация передается по зарезервированному каналу телефонной связи [9]. В базовой комплектации, которая была использована разработчиками, мобильный регистратор выглядит, как показано на рисунке 1.1

Недостатком данного регистратора является неудобство и трудоемкость анализа ЭКГ в силу малой диагонали дисплея смартфона. Также, в состав оборудования для экстренного снятия ЭКГ, входит усилитель сигнала, изображенный на рисунке 1.2, встраивающийся между датчиками сердечной активности и самим смартфоном. Цифровой усилитель сигнала используется для усиления и частичной фильтрации данных ЭКГ [1].

Рисунок 1.1 – Мобильный регистратор на базе смартфона

Рисунок 1.2 – Цифровой усилитель ЭКГ

Следует отметить, что такие мобильные регистраторы просты в использовании для среднего медицинского персонала, что экономит средства на дополнительном обучении [1].

Такие же мобильные регистраторы, но немного иной комплектации, например, без усилителя сигнала, используются уже в самих медицинских учреждениях, в которых либо нет кабинета кардиолога, либо пациент не может переместиться в этот кабинет. В этом случае снятие данных производится в палате пациента [10]. Обычно, поводом для использования именно мобильного регистратора служит отсутствие возможности подключения устройства ЭКГ к электросети, ведь мобильный регистратор питается от автономного аккумулятора.

Федеральным государственным бюджетным учреждением «Российский кардиологический научно-производственный комплекс» было разработано стационарное устройство регистрации ЭКГ-показателей, изображенное на рисунке 1.3. Такой прибор имеет в несколько раз большие габариты, но позволяет снять показатели лишь в тех условиях, когда пациент обладает низким уровнем мобильности. Данное устройство оборудовано на базе персональных компьютеров с сенсорным дисплеем, что повышает уровень удобства в работе с данными ЭКГ.

Рисунок 1.3 – Стационарный кардиорегистратор в специализированной стойке

Стационарный кардиорегистратор представляет собой связку нескольких приборов: сенсорный дисплей с клавиатурой, принтер и блок бесперебойного питания, на случай отключения электрической сети [10]. С помощью такого устройства можно проводить снятие ЭКГ в палатах, которые не имеют для этого специального оборудования. С помощью принтера, полученные данные можно сразу распечатать. Данное устройство подключается к локальной сети медицинского учреждения, и после успешного снятия ЭКГ данные передаются по локальной сети на центральный сервер [1].

Как при использовании мобильных регистраторов, так и при использовании стационарных, порядок процедуры на первом этапе остается одинаков: с помощью программного обеспечения, заполняются личные данные о пациенте, такие как фамилия, имя, отчество, дата рождения, идентификационный номер медицинской карты (если такая имеется), адрес регистрации. Все эти данные прикрепляются к данным ЭКГ в виде пакета и после передаются на центральный сервер [9].

Общая логическая схема взаимодействия дистанционных и стационарных регистраторов с центральным сервером и рабочим местом врача имеет вид, представленный на рисунке А.1 (приложение А). Данная схема обладает тремя типами подключений:

а) неопосредованная связь между двумя звеньями обозначена на схеме стрелками со сплошной линией. В конкретном случае, связь реализуется в границах локальной сети учреждения;

б) связь между двумя звеньями, которая реализуется посредством других сетей обозначена стрелками со штриховой линией. В конкретном случае, имеется ввиду связь с сервером через локальную Wi-Fi-сеть;

в) связь между двумя сетями обозначается стрелками с пунктирной линией. Например, передача данных ЭКГ к главному серверу посредством подключения к глобальной сети Интернет или к телефонной линии.

Конечным пунктом получения ЭКГ-сигнала является рабочее место врача [1], которое представляет собой персональный компьютер с широкоформатным

жидкокристаллическим монитором, подключенный к локальной сети лечебного учреждения. Рабочее место врача изображено на рисунке 1.4.

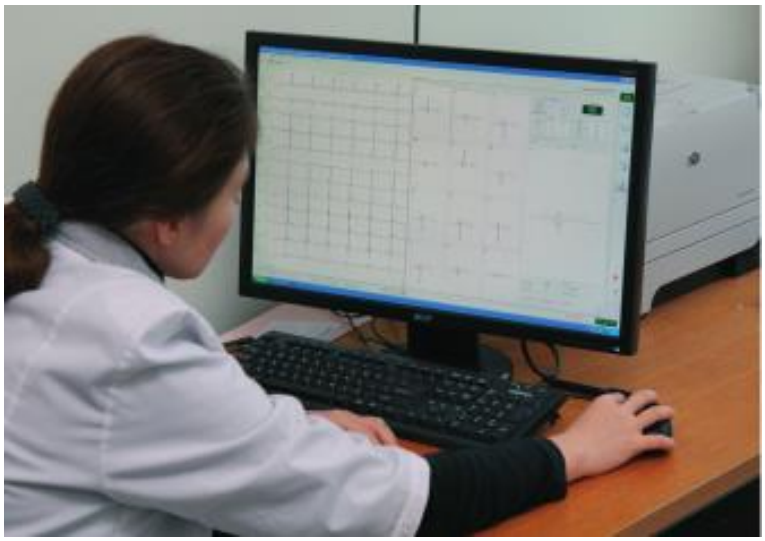


Рисунок 1.4 – Рабочее место врача

На таком персональном компьютере установлено дополнительное специальное программное обеспечение, с помощью которого врач получает доступ к данным ЭКГ пациентов, хранящихся на сервере.

С помощью этого клиент-серверного программного обеспечения реализуется что-то вроде личного кабинета врача, в самом приложении реализована авторизация для доступа к данным пациентов, закрепленным за каждым врачом [10]. Чаще всего распределение пациентов происходит случайным образом, вне зависимости от истории болезни.

С помощью личного кабинета врач может просмотреть недавние данные ЭКГ, которые отображаются в списке всех данных ЭКГ с пометкой. Важная особенность процесса диагностирования заключается в том, что лишь один врач в одно и тоже время может проводить диагностику конкретного пациента. Это исключает одновременное внесение изменений и комментариев двумя врачами [1].

В личном кабинете реализованы основные средства для работы с графиками ЭКГ: увеличение или уменьшение масштаба графика, детектирование различных комплексов сердечного ритма и т.п. [1, 10].

Несмотря на все очевидные преимущества данной разработки, она имеет ряд и недостатков, которые определены техническими и человеческими ограничениями.

Технический аспект непосредственно связан с дистанционной передачей ЭКГ посредством подключения к сети Интернет через стационарное Интернет-соединение, либо через мобильный Интернет [9].

Влияние «человеческого фактора» заключается в необходимости наличия у персонала минимальной компьютерной грамотности.

Но это ограничение со временем исчезает, а в настоящий момент, тенденция роста информационных технологий определяет наличие навыков владения компьютера, и минимальная компьютерная грамотность является довольно востребованным критерием для приема на работу в любое лечебное учреждение [1].

И еще одно относительное ограничение связано с возможной необходимостью печати ЭКГ на месте регистрации. Такой возможностью обладают не все мобильные регистраторы, в силу отсутствия встроенного модуля печати документов.

При наличии функции печати большинство мобильных регистраторов используют только малоинформативную узкую термопечать (40-56 мм) [9].

Таким образом проделанная работа отечественных разработчиков показывает, что внедрение технологии дистанционного мониторинга сердечного ритма в работу различных лечебных учреждений относительно простой и не дорогой процесс (если брать во внимание лишь затраты на саму интеграцию) [17].

К слову говоря, компания «Нейрософт» в своей системы реализовали логику взаимодействия по примеру в приведенной выше схеме.

Свою разработку они назвали «ПОЛИ-СПЕКТР-8G», которая изображена на рисунке 1.5,а и судя по всему, она уже доступна на коммерческом рынке.

а) б)

Рисунок 1.5 – Кардиограф «ПОЛИ-СПЕКТР-8G»: а) внешний вид; б) схема взаимодействия звеньев системы

Данное устройство состоит из модуля усиления сигнала, который подключается к смартфону, и набора датчиков для снятия ЭКГ-показателей. Схема взаимодействия звеньев системы изображена на рисунке 1.5,б [9].

1.2 Дистанционная система мониторинга «Самоконтроль ЭКГ»

АО «МИКАРД-ЛАНА» совместно с ООО «Телемедицинские системы» разработали технологию «Самоконтроль ЭКГ», с помощью которой пользователь может самостоятельность провести ЭКГ-замер себе или своим близким, автоматически отправить ее на удаленный интернет-сервер, получить предварительное автоматическое заключение о состоянии сердечно-сосудистой системы и, при необходимости, обратиться к зарегистрированному на сервере врачу для дистанционной консультации [1, 9]. В качестве технических средств, реализующих эту технологию, используется комплекс «Кардиометр-МТ», мобильный телефон и интернет-сервер [10]. Интернет-сервер является облачным сервисом для хранения ЭКГ-данных, целью которого является прием и автоматическая обработка ЭКГ. Доступ к сервису осуществляется с помощью глобальной сети Интернет [1]. Архив данных об ЭКГ каждого пациента защищен логином и паролем [1]. Пользователю доступны все совершенные им ЭКГ-обследования. Благодаря этому появляется возможность их распечатать на

любом подключенном к сети Интернет компьютере с принтером, а далее предоставить их во время очного визита к врачу.

«Кардиометр-МТ» представляет собой портативный кардиограф, изображенный на рисунке 1.6, с помощью которого происходит снятие ЭКГ пациента, и отправка ее по Bluetooth-соединению либо на компьютер, либо на смартфон под операционной системой «Android» [9].

Далее данные передаются по сети Интернет на сервер, где они поддаются автоматической интерпретации, с помощью различных программных алгоритмов, позволяющих производить те или иные манипуляции с сигналом, а после, данные, сохраняются на дисковом пространстве сервера данного веб-сервиса системы, и они становятся доступны для просмотра пользователем [1].

Рисунок 1.6 – Кардиорегистратор из комплекса «Кардиометр-МТ»

С помощью специального аккаунта пользователь может войти в свой личный кабинет и увидеть данные своей ЭКГ и первичную интерпретацию [1].

В личном кабинете пациента, пользователь может просмотреть полную историю своих обследований, интерфейс изображена на рисунке 1.7, увидеть врачей и медицинские центры, которым дан доступ к его ЭКГ-исследованиям, соответствующий интерфейс изображен на рисунке 1.8, а также, более детально настроить права доступа для конкретного врача, интерфейс изображен на рисунке 1.9.

Также можно и добавить нового врача, а после назначить ему права на просмотр ваших медицинских данных.

Врач может иметь довольно широкий спектр возможностей, который назначается самим пациентом [1].

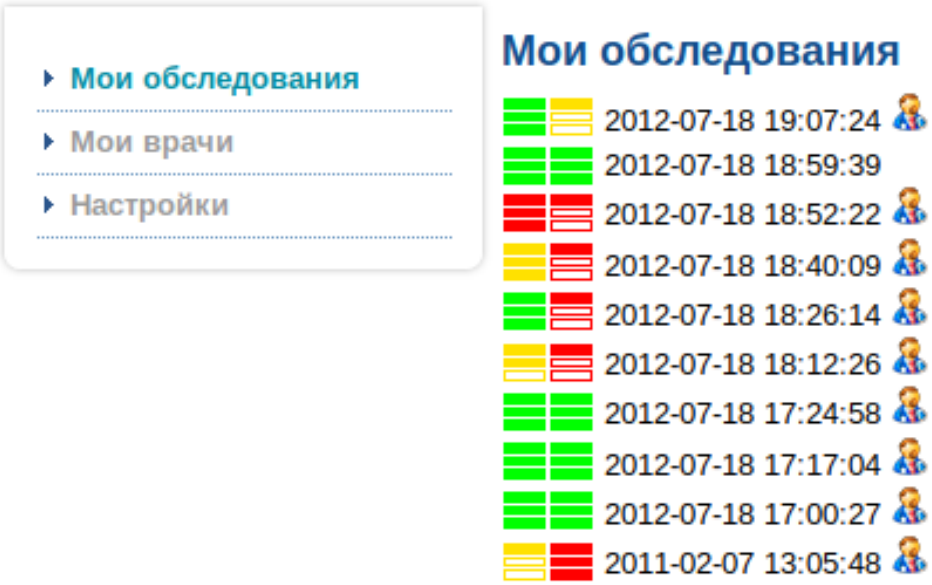


Рисунок 1.7 – Личный кабинет пациента, вкладка «Мои обследования»

Таким образом, данная система в значительной мере упрощает процесс взаимодействия пациента с врачом и позволяет проводить дистанционную диагностику сердечных заболеваний по ЭКГ-данных.

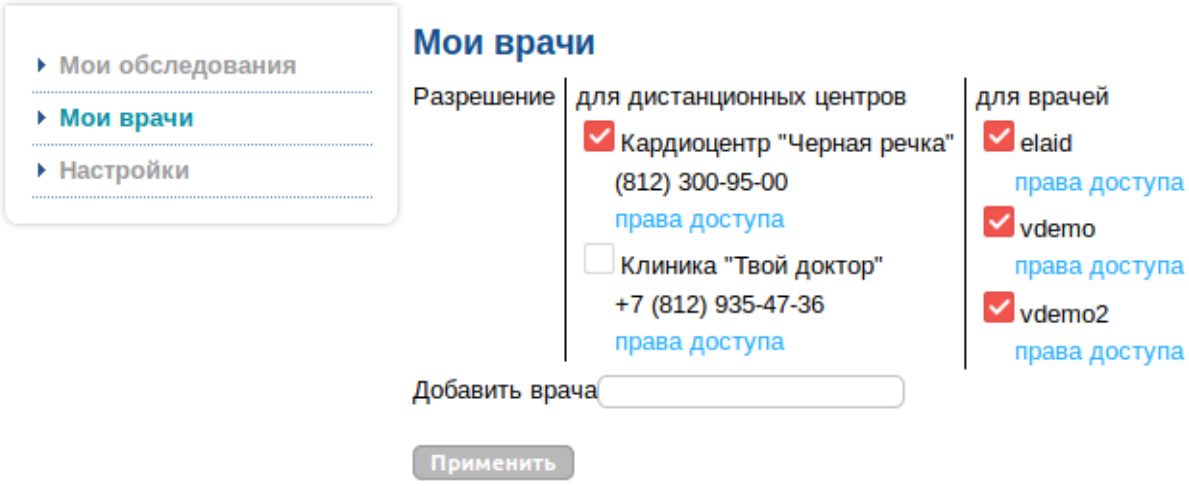


Рисунок 1.8 – Личный кабинет пользователя, вкладка «Мои врачи»

Однако следует отметить, что первичная диагностика реализована с помощью программных алгоритмов, хотя она и не претендует на абсолютную достоверность поставленного заключения, относится к ней следует довольно осторожно [9, 17].

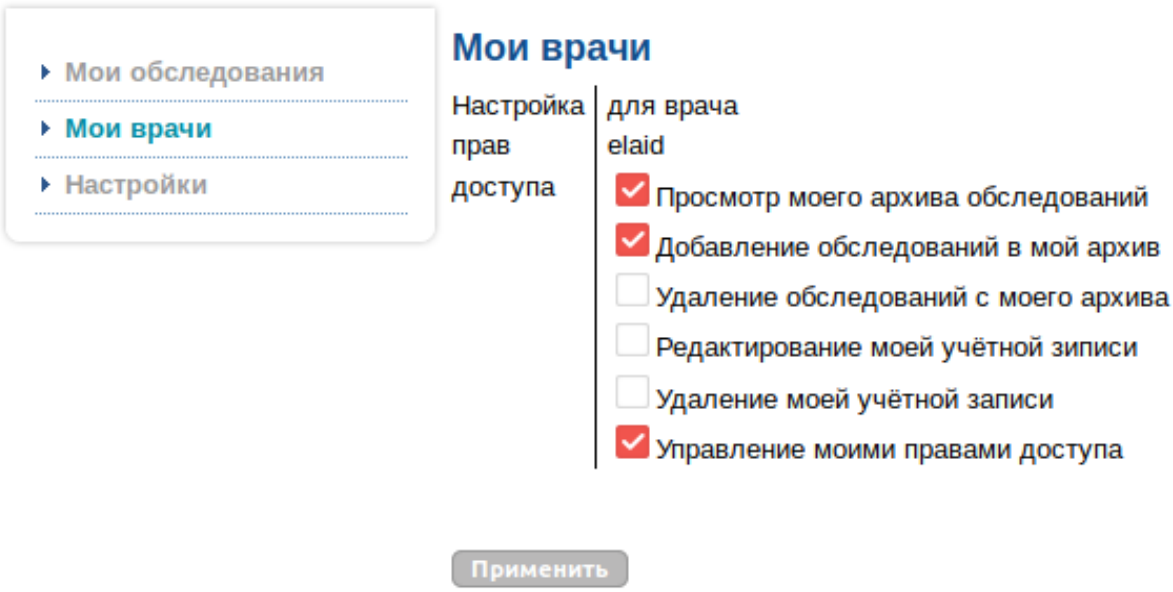


Рисунок 1.9 – Личный кабинет пользователя, вкладка «Мои врачи», интерфейс назначения прав

Также следует отметить и то, что диагностика самим специалистом производится посредством различных частных клиник. Несмотря на то, что все частные клиники в обязательном порядке проходят сертификацию, их ответственность и уровень квалификации может быть под сомнением [1, 17]. И, собственно, цена вопроса в несколько раз выше, в силу того, что частные клиники не предполагают бесплатного обслуживания с помощью полиса медицинского страхования.

Исходя из всего вышесказанного, можно сделать вывод, что на данный момент времени разработан довольно широкий спектр устройств для дистанционного снятия ЭКГ. Причем, некоторые разработки предназначены для использования в рамках лечебных учреждений, а некоторые, сделаны так, что их использование возможно самим пациентом. Однако, уровень

популярности, или уровень интеграции таких устройств в муниципальные структуры довольно низок, вероятнее всего из-за невысокой степени финансирования таких разработок.

Вывод по первому разделу.

Такое финансирование обусловлено тем, что данные разработки хоть и находят успешное применение как в частных клиниках, так и в муниципальных, но это единичные случаи в силу того, что степень ассортимента и разнообразия таких технологий довольно низок, а уровень недоверия пациентов и настороженности по поводу дистанционной диагностики – высок.

Разработка данной системы повлечет за собой планомерное расширение ассортимента выбора таких систем, что позволит увеличить уровень интегрируемости и массовости использования подобных технологий в сфере здравоохранения.

2 Проектирование и разработка аппаратной части системы дистанционного мониторинга сердечной деятельности пациента

2.1 Проектирование модульной схемы устройства

Проектируемое устройство представляет собой прибор, с помощью которого возможно зарегистрировать сигнал сердцебиения человека, обработать сигнал ЭКГ цифровыми фильтрами при необходимости и передать сигнал на сервер [4, 10].

Таким образом, общая модульная схема устройства имеет вид, представленный на рисунке 2.1

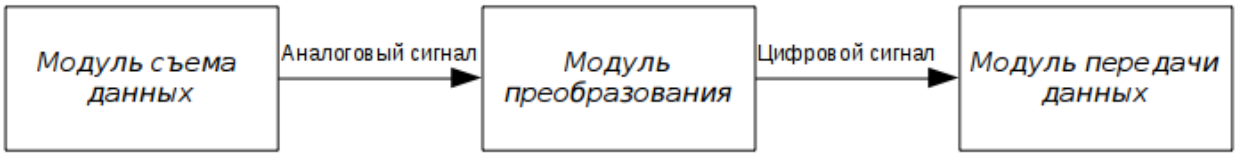


Рисунок 2.1 – Модульная схема проектируемого устройства

Проектируемая система состоит из трех модулей, каждый из которых выполняет определенные последовательные задачи.

Первой задачей является регистрация сердцебиения человека. Она может быть решена путем применения ЭКГ-датчика, с помощью которого снимаются потенциалы с различных участков тела человека. Вычисляя разницу полученных потенциалов, можно получить амплитудный сигнал биения сердца, выраженный в колебаниях электрических потенциалов [2]

Также, для регистрации сердечной активности бывает достаточно регистрировать пульсовую волну, вместо электрокардиограммы. Несмотря на то, что пульсовая волна не обладает таким количеством полезной информации, как ЭКГ-сигнал, с ее помощью можно вычислить такие показатели сердечной активности как количество ударов в минуту, межпиковое расстояние, уровень

давления в кровеносных сосудах. В выпускной квалификационной работе выбран именно такой оптический датчик, о преимуществах которых будет сказано в следующих подразделах.

Таким образом, первым модулем устройства является модуль съема данных (датчик), с помощью которого происходит регистрация сердечной активности человека.

Второй задачей является обработка сигнала, которая включает в себя преобразование аналогового сигнала с датчика в цифровой вид и цифровую фильтрацию с целью удаления помех и артефактов, обусловленные действием внешних электромагнитных факторов и собственных шумов датчика [5, 14, 25].

Таким образом, вторым модулем проектируемого устройства является модуль преобразования, осуществляющий преобразование и фильтрацию сигнала.

Полученный цифровой сигнал следует передать на веб-сервер для излечения информации и дальнейшего просмотра данных врачом-диагностом [14, 18]. Данная функция осуществляется модулем передачи данных. Еще одной функцией данного модуля является авторизация пользователя. Регистрируемый сигнал передается с параметрами, позволяющими однозначно идентифицировать пациента, которому принадлежит сигнал. Доступ к этим данным может быть только у этого пациента и врача, за которым закреплен пациент.

Модульная схема проектируемого устройства имеет вид, представленный на рисунке 2.2.

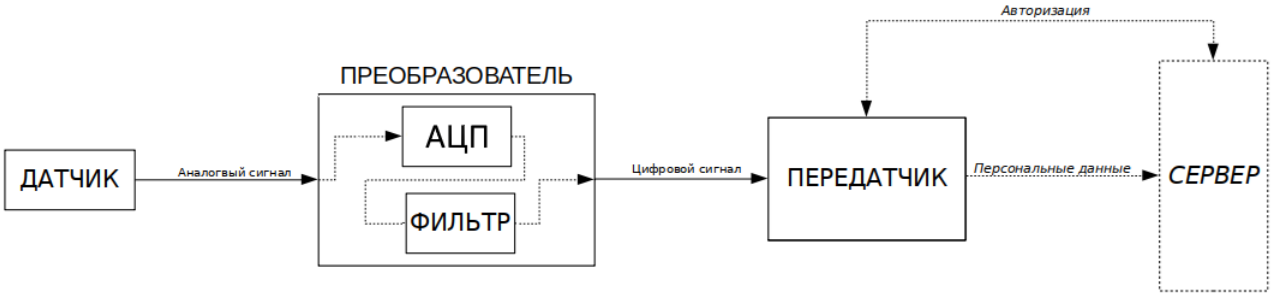


Рисунок 2.2 – Модульная схема проектируемого устройства

Из рисунка видно, что первым, возникающим при работе устройства, информационным потоком является аналоговый сигнал, который поступает в модуль преобразователя, где последовательно с помощью аналого-цифрового преобразователя переводится в дискретный вид, а затем фильтруется. На выходе этого модуля возникает цифровой сигнал, который попадает в модуль передатчика.

Данные передаются только в случае успешной авторизации по запросу модуля передатчика. К передаваемым данным о сердечной деятельности пациента добавляется информация о пациенте. Этот пакет обрабатывается программным обеспечением сервера, перед тем, как происходит сохранение сигнала [14]. Дополнительные данные, которые сервер отправляет после успешной авторизации, позволяют определить чей именно это сигнал, а также, где и как его сохранить в памяти веб-сервера.

2.2 Выбор аппаратных составляющих устройства

В качестве датчика может подойти любое устройство, способное регистрировать показатели сердечной деятельности. Такие устройства могут отличаться принципами своей работы.

Например, ЭКГ-датчик чаще всего представляет собой три электрода, которые накладываются на определенные участки тела человека, которые соответствуют трем основным типам отведения [5].

В датчиках-пульсометрах, также способных регистрировать сердечную активность, используется оптопара, состоящая из светодиода и фоторезистора, который меняет свое сопротивление в зависимости от количества света, попадающего на его фоточувствительный элемент [2]. Принцип работы таких пульсометров заключается в измерении динамики реактивного сопротивления фоторезистора, принимающего на себя отраженный свет, излученный светодиодом – оптопара прикладывается к телу человека в том месте, где плотность тканей имеет наименьший показатель (отсутствие толстых слоев

мышечной ткани), чаще всего, такие датчики фиксируются на указательном пальце человека [18]. В момент удара сердца все капилляры и сосуды наполняются венозной кровью. Кровь, в заполненном капилляре изменяет свойства поглощения и отражения излученного светодиодом света [2]. В течении времени, отраженный от капилляров свет меняет свою интенсивность в соответствии с ударами сердца: заполнение кровью капилляров приводит к повышению коэффициента поглощения, в следствие чего интенсивность отраженного света снижается, соответственно реактивное сопротивление фоторезистора повышается [2, 18]. При оттоке венозной крови интенсивность отраженного света возрастает, в силу чего понижается реактивное сопротивление фоторезистора [2, 10].

Фиксируя изменения показателя сопротивления фоторезистора во времени, получают график зависимости сопротивления от времени, либо напряжения, проходящего через фоторезистор, от времени. Этим графиком называют пульсовую волну [2, 18].

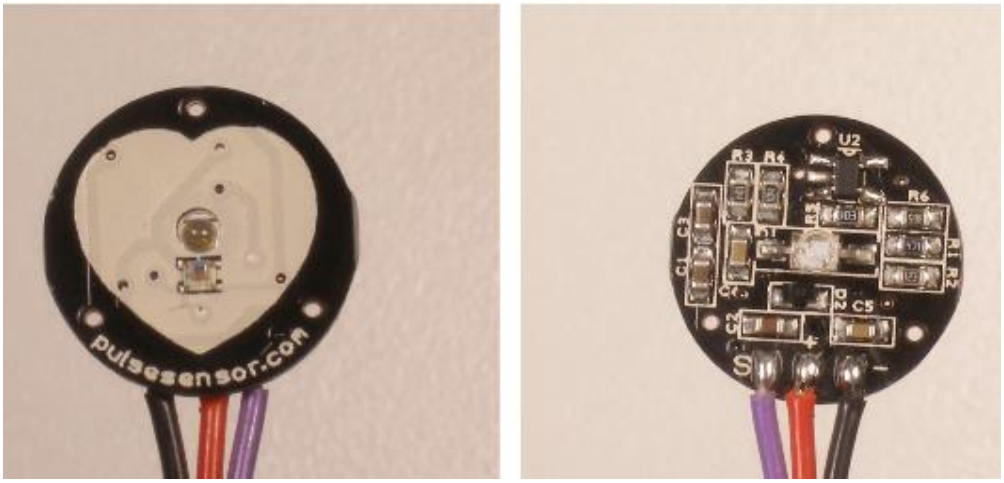
Пульсовая волна существенно отличается от электрокардиограммы и содержит на порядок меньше информации о деятельности сердца: отсутствие выраженных Q и S комплексов, лишь комплекс R, обозначающие сокращение желудочков сердца, и т. п [17].

Однако, пульсометр куда более компактный прибор нежели устройство снятия ЭКГ, и в некоторых случаях, рациональнее всего использовать именно пульсометрию.

К тому же сигнал пульсовой волны получить намного проще и это доступно даже пациенту в домашних условиях.

В выпускной квалификационной работе используется датчик пульсовой волны, однако, в дальнейшем, планируется модернизация прибора путем замены пульсометра на ЭКГ-датчик.

Существует большое разнообразие датчиков пульсовой волны. В работе была выбрана модель PulseSensor от разработчиков из World Famous Electronics llc, изображен на рисунке 2.3.



а) б)

Рисунок 2.3 – Пульсометр PulseSensor: а) вид спереди; б) вид сзади

К преимуществам этого датчика следует отнести его стоимость (на сайте производителя стоимость 24,99 доллара), простоту выполнения и простоту применения [2], наличие в свободном доступе библиотек для работы с различными контроллерами. Датчик имеет три контакта: земля (черный провод), питание (красный провод) и сигнал (фиолетовый провод). На рисунке 2.3,а изображена лицевая сторона пульсометра, которая прикладывается к телу. На ней расположен светодиод и фоторезистор, представляющие собой в совокупности оптопару [2].

Напряжение питания датчика может быть либо 3,3 В, либо 5 В – такая вариативность позволяет использовать данный пульсометр в связке с различными типами микроконтроллеров [2, 18].

При питании в 3,3 В сила тока должна составлять 3 мА, а при питании в 5 В, 4 мА [2].

Рабочий диапазон температур от –40 °С до +85 °С.

Существование свободно распространяемой библиотеки для работы датчика с семейством контроллеров Arduino (наиболее распространенных из-за удобства, простоты использования и возможностей интегрирования [5]) – послужило основной причиной выбора пульсометры PulseSensor в выпускной квалификационной работе.

В качестве компоненты второго модуля системы была выбрана плата Arduino Leonardo, изображена на рисунке 2.4.

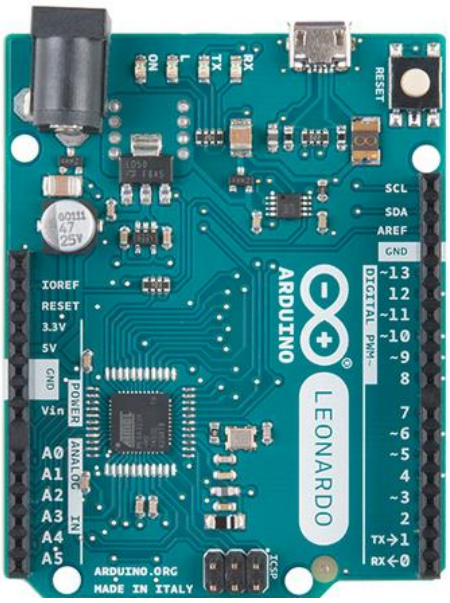


Рисунок 2.4 – Контроллер семейства Arduino серии Leonardo

Как и в случае с пульсометром, основным преимуществом этого контроллера является соотношение цены и качества. Контроллеры семейства Arduino можно запрограммировать под решение огромного количества задач. Причем, язык для написания программ – Си-подобный [5]. Процессор, используемый в Arduino Leonardo, ATMega32U4 [5]. Данный процессор обладает достаточным характеристиками для того, чтобы реализовать фильтрацию сигнала программными средствами в режиме реального времени [5].

При работе с сигналом, он автоматически преобразуется в цифровой вид, благодаря чему появляется возможность использовать цифровой фильтр сигнала, что существенно сокращает затраты на покупку дополнительных радиоэлементов. Стоимость платы Arduino Leonardo на официальном сайте разработчика составляет 2100 рублей [5].

Контроллер обладает большим количеством аналоговых и цифровых выводов и вводов (в зависимости от установленного режима работы) [5].

Полная схема подключения всех портов контроллера изображена на рисунке А.2. На плате расположены три группы контактов: системные, аналоговые и цифровые. К системным контактам относятся порты заземления, порты питания платы, порты выходного напряжения. Ниже расположены аналоговые порты, предназначенные для работы с аналоговым сигналом [5]. В противоположной от них стороне, расположено тринадцать цифровых портов, причем некоторые из них подключены к генератору широтно-импульсных модуляций (обозначение «PWM» на плате).

Большинство портов поддерживают различные протоколы передачи данных, что также можно отнести к важным преимуществам микроконтроллера.

Для работы с датчиком PulseSensor используется только три порта: порт питания (3,3 В или 5 В), порт заземления и порт сигнала [2].

Для решения поставленных в выпускной квалификационной работе задач порт сигнала должен быть настроен в режим ввода, чтобы сигнал с датчика попадал именно на него, где в дальнейшем будет происходить обработка.

Третьим модулем устройства является модуль передатчика. К требованиям этого модуля можно отнести поддержку существующих протоколов передачи данных через сеть Интернет [14]: HTTP, HTTPS, FTP и SSH [23]. Найти узконаправленное устройство, с поддержкой всех этих протоколов весьма сложно, поэтому, в качестве передатчика был выбран миникомпьютер Raspberry Pi 3, рисунок 2.5.

Рисунок 2.5 – Миникомпьютер Rasbperry Pi 3

Поддержка вышеперечисленных сетевых протоколов не является единственной причиной, по которой было выбрано именно это устройство. Так как Rasberry Pi 3 – это полноценный компьютер, со своей операционной системой (Raspbian) на основе дистрибутива Linux Debian, проводить подстройку под определенные нужды этого прибора намного легче [5, 12]. Также, следует отметить, то, что операционная система Linux обладает большим количеством встроенных компиляторов и интерпретаторов, что расширяет возможности разработки драйверного программного обеспечения для датчика [12].

Модуль передатчика, фактически можно назвать терминалом, потому что, сигнал, полученный с датчика останавливается именно на этом этапе, где преобразуется в информационный пакет данных для отправки на сервер [23]. Rasbperry Pi 3 обладает HDMI-видеовыходом, с помощью которого возможно подключить сенсорный дисплей, посредством которого будет происходит взаимодействие пользователя с устройством съема и передачи данных [5]. Благодаря этому, драйвер для датчика и сама программа отправки данных может использовать графический интерфейс, что значительно улучшает условия взаимодействия устройства с конечным пользователем [16]. Выход в сеть в миникомпьютере Rasbperry Pi реализован с помощью Ethernet- и Wi-Fi-модулей. Встроенный Wi-Fi адаптер за счет подключения к беспроводной сети обеспечивает высокую мобильность и удобство в использовании устройства [5].

Как и любой компьютер, Rasbperry Pi обладает набором USB-портов, которые поддерживают режим обмена информации (точно также, как и в Arduino) [5]. Также, как и Arduino, Rasbperry Pi 3 обладает GPIO-интерфейсом [5]. Схема подключения портов (GPIO-интерфейса) Rasbperry Pi изображена на рисунке 2.6. Все порты предназначены исключительно для работы с цифровым сигналом, в зависимости от программной установки поддерживают два режима: ввод или вывод, некоторые поддерживают режим ШИМ. Есть порты питания 3,3 В и 5 В [5, 18].

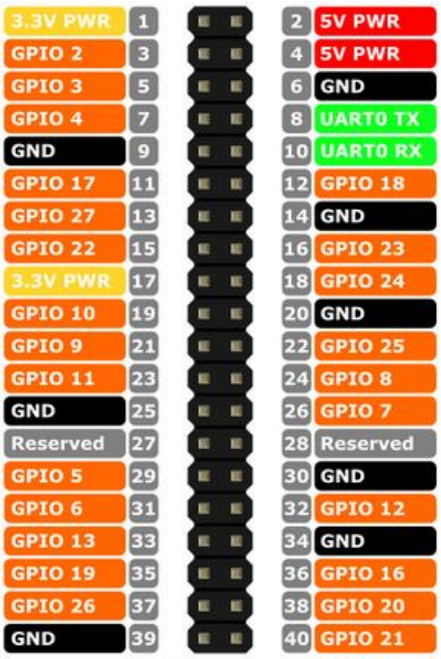


Рисунок 2.6 – Предназначение всех портов интерфейса GPIO

Рабочее напряжение питания миникомпьютера составляет 5 В при силе тока в 2,5 А [18]. Цена на такой миникомпьютер на сайте разработчика составляет 35.5 долларов [5].

Таким образом, готовое устройство будет состоять из трех взаимосвязанных комплектующих: пульсометра PulseSensor, контроллера Arduino Leonardo и миникомпьютера Raspberry Pi 3.

Каждая компонента реализует задачи, описанные в соответствии с модульной схемой устройства, приведенной в предыдущем подразделе.

2.3 Проектирование монтажной схемы устройства

Исходя из модульной схемы проектируемого устройства датчик PulseSensor будет подключен к контроллеру Arduino Leonardo.

Как уже было сказано выше, датчик имеет всего три контакта: питание, заземление и порт аналогового сигнала самого датчика. В свою очередь, контроллер Arduino имеет шесть портов для работы с аналоговым сигналом, также, имеет системные порты питания на 5 В и 3,3 В [5].

Таким образом, сигнал с датчика будет поступать на первый аналоговый порт контроллера Arduino – A0, в качестве питания будет использовано напряжение в 3,3 В; заземление датчика осуществляется с помощью специального порта общего провода на контроллере Arduino [18].

Монтажная схема такого подключения имеет вид, представленный на рисунке 2.7.

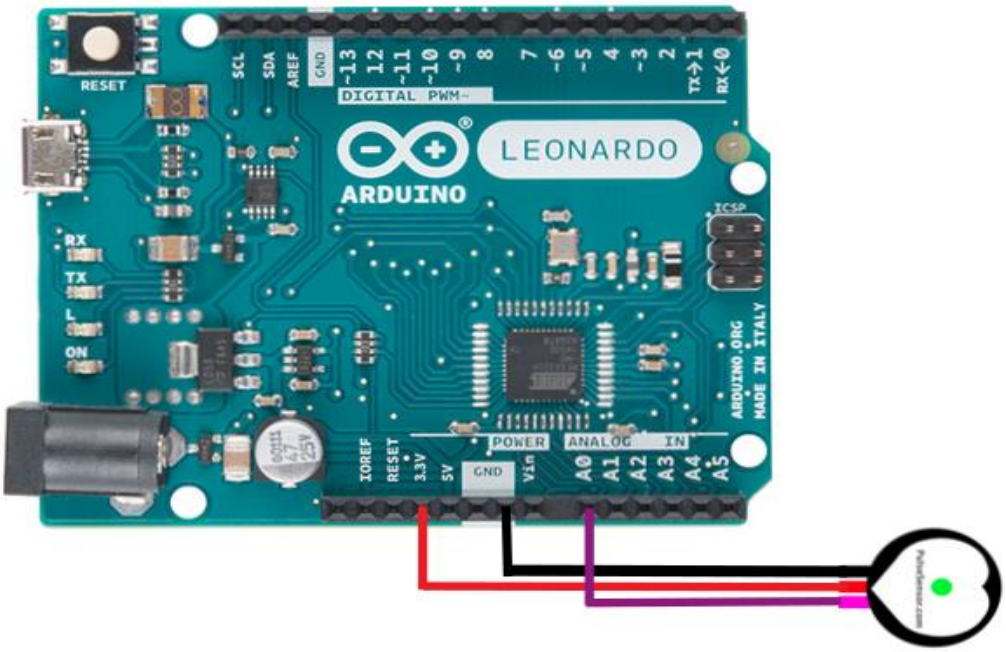


Рисунок 2.7 – Монтажная схема подключения датчика к контроллеру

Монтажная схема подключения «датчик-контроллер» к микрокомпьютеру Raspberry Pi 3 через USB-кабель представлена на рисунке 2.8.

Микрокомпьютер в качестве силовой линии использует кабель USB-C, который подключается в соответствующий разъем.

Напряжение питания составляет 5 В. Подключение связки «контроллер-датчик» к микрокомпьютеру реализован с помощью USB-переходника «USB-B – USB-C».

Следует отметить и то, что данный переходник используется не только для обмена данными, но и для питания самого контроллера с датчиком [5, 18].

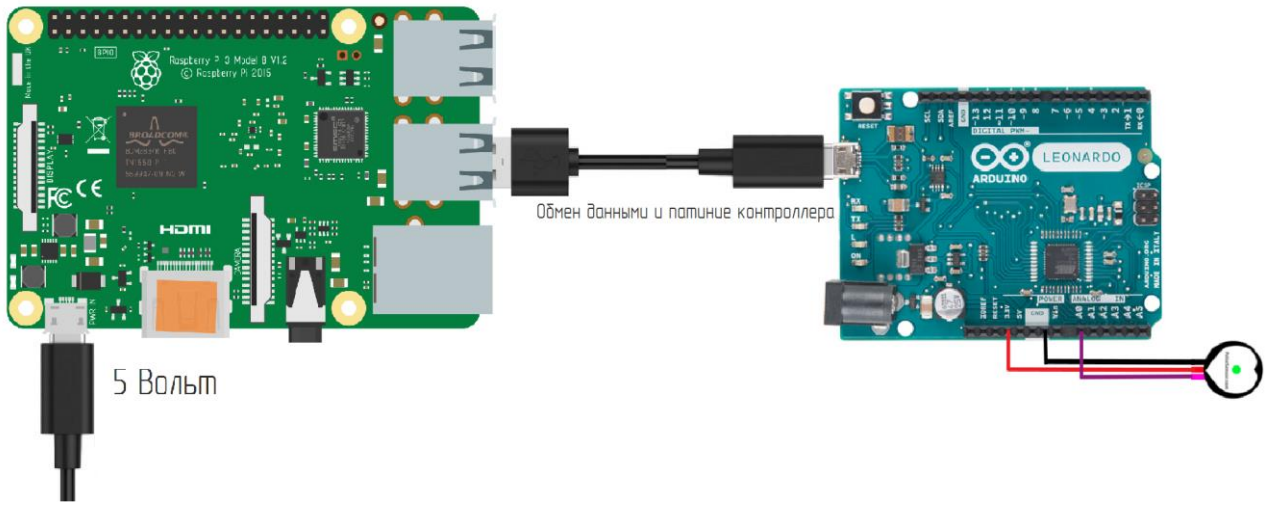


Рисунок 2.8 – Монтажная схема подключение контроллера к микрокомпьютеру Raspberry Pi 3

Такое подключение обусловлено тем, что данный микрокомпьютер может обрабатывать лишь цифровые сигналы, и применение внешнего АЦП проблематично, так как пороговые значения цифрового сигнала у Arduino Leonardo и Raspberry Pi 3 различны (5 В у Arduino и 3,3 В у Raspberry).

Вывод по второму разделу.

Таким образом, во втором разделе выпускной квалификационной работы, в ходе проектирования аппаратной части были выполнены следующие задачи: спроектирована общая модульная схема всего устройства, которая отображает логику подключения и взаимодействия компонентов будущего устройства; подобраны соответствующие аппаратные компоненты для реализации устройства (микроконтроллер Arduino Leonardo, микрокомпьютер Rasbperry Pi 3 и оптический датчик пульсовой волны Pulse Sensor); на основании особенностей выбранных аппаратных частей устройства, была спроектирована монтажная схема подключений всех звеньев разрабатываемого прибора.

3 Проектирование и разработка программного обеспечения системы дистанционного мониторинга сердечной деятельности пациента

3.1 Разработка программного драйвера микроконтроллера и программного фильтра

Для выполнения задачи цифровой фильтрации в модуле преобразования и доступа микрокомпьютера Raspberry Pi 3 к данным с датчика Pulse Sensor [18] необходима программная реализация соответствующих алгоритмов.

Разность пороговых значений цифровых сигналов Arduino и Raspberry Pi не позволяет применить внешний АЦП, однако, перенаправление сигнала в серийный порт решает эту проблему, так как данный микроконтроллер автоматически преобразует аналоговую форму сигнала в цифровую непосредственно перед передачей его на COM-порт [18].

В ходе тестирования оптического датчика было выявлено наличие в полезном сигнале собственных ВЧ-шумов датчика малой амплитуды. Поэтому в качестве цифрового фильтра был выбран медианный фильтр [25], метод работы которого заключается в нахождении n-го количества точек вблизи окрестности i-точки; сортировки их в порядке возрастания; выборе медианного значения X[n / 2] (где X – множество, состоящее из n-го количества точек графика сигнала) [25]. Значение n может подбираться опытным путем, в зависимости от степени зашумленности сигнала. Графическая схема подпрограммы цифрового медианного фильтра (medianFilter) приведена на рисунке Б.1 (приложение Б). Исходный код подпрограммы на языке Си представлен в листинге Б.1.

Данная подпрограмма используется для фильтрации сигнала перед его передачей на серийный порт.

Несмотря на то, что используется Си-подобный язык программирования, привычной функции main, которая присуща всем программам на языках C/C++,

как таковой нет (она добавляется автоматически перед компиляцией программы) [27, 26, 11].

Вместо этого, используется две служебные функции setup и loop, которые продиктованы особенностями процессора Arduino [5, 26].

Функция setup имеет void-тип. Данная функция предназначена для установки начальных параметров микроконтроллера, например, настройки портов на ввод или вывод, установки таймера и т.п. Функция setup выполняется первой и один раз [11, 26]. Функция loop представляет собой набор всех команд и конструкций, выполняющихся во время работы микроконтроллера и работает циклично [5]. Исходный код подпрограммы setup приведен в листинге Б.2. Подпрограмма setup содержит в себе две инструкции: Serial.begin(9600) и Pulse.begin() [5, 11]. Первая команда используется для настройки последовательного порта – с помощью метода begin из класса Serial; метод begin используется для инициализации работы последовательного порта [5, 18]. Вторая команда используется для инициализации работы датчика Pulse Sensor. Структура класса Pulse описана в заголовочном файле iarduino_SensorPulse.h, который подключается в самом начале кода программы с помощью инструкции «#include» [11, 26, 27]. Все константы и объекты представлен в листинге Б.3.

Первая строчка кода из листинга Б.3 отвечает за создание экземпляра класса Pulse, причем аргумент для конструктора класса представляет собой системную константу – A0, которая отвечает за нулевой порт аналоговых контактов. Указание этой константы при создании экземпляра класса связывает датчик с аналоговым портом, к которому подключен датчик.

Вторая строка инициализирует переменную delay_ms, которая отвечает за программную величину задержки (мс) считывания сигнала с датчика [5].

Переменная FILTER_SAMPLES определяет значение окна медианного фильтра. В выпускной квалификационной работе значение окна было подобрано опытным путем и равно 16.

Переменная serial_command инициализирована для хранения команд, полученных из серийного порта от контролирующего устройства. Таким

образом, микроконтроллер не только отправляет данные датчика на последовательный порт, но и считывает данные, которые отправляются микрокомпьютером Raspberry Pi 3.

Для контроля потока данных обмен информацией между двумя устройствами реализован через COM-порт. Так, например, если микрокомпьютер отправляет команду «start» на последовательный порт, микроконтроллер, распознавая ее, начинает считывание и передачу данных с датчика пульса. Если во время передачи данных микрокомпьютер отправляет команду «stop», микроконтроллер останавливает передачу информации, снова ожидая команды «start». С помощью этих двух команд был реализован простой метод сообщения двух основных компонент разрабатываемого устройства.

Изначально при подключении микроконтроллера к последовательному порту, он не отправляет никаких данных, а ожидает команды «start», и получив ее запускает подпрограмму считывания и фильтрации данных.

В ходе выполнения подпрограммы loop происходит постоянное сканирование последовательного порта на наличие запускающей команды. Если такая появляется, происходит выполнение подпрограммы getAnalogData, которая производит считывание, фильтрацию и передачу данных на серийный порт. Исходный код подпрограммы loop представлен в листинге Б.4. Подпрограмма начинается с проверки значения функции Serial.available, котрая возвращает количество байт доступных для считывания с интерфейса последовательного порта [5]. Микроконтроллер определяет наличие каких-либо инструкций от микрокомпьютера в буфере порта. Если значение данной функции отлично от нуля, то в буфер порта поступила какая-то информация.

Для чтения данных из COM-порта в классе Serial предусмотрена функция read, однако, вместо удобной для понимания строки символов, она возвращает лишь байты, которые представляются в виде числа [5].

Для получения строкового массива из последовательного порта была реализована функция str_serial. Исходный код данной функции представлен в листинге Б.5.

При считывании байта информации методом read она удаляется из буфера порта [5, 26]. При наличии непустого (Serial.available() > 0) буфера серийного порта вызывается функция str_serial, в которой происходит цикличный вызов метода read класса Serial, до тех пор, пока значение метода available не будет равным нулю. В ходе получения байт информации из последовательного порта с помощью метода read каждый из них записывается в переменную serial_buffer. Но перед записью байта в эту переменную происходит приведение его к типу char, который отвечает за представление одного символа [11]. Таким образом, после прохождения этого цикла на выходе переменная serial_buffer содержит в себе уже строку информации, а не последовательность байт.

После выполнения функции str_serial, ее значение записывается в глобальную переменную serial_command и далее проверяется на соответствие команде «start».

В случае соответствия происходит вызов функции getAnalogData, иначе функция loop завершается и вызывается снова.

Функция getAnalogData производит считывание и перенаправление потока данных из оптического датчика в последовательный порт до тех пор, пока в буфере серийного порта не появится управляющая инструкция «stop».В случае ее появления функция записывает в буфер серийного порта байтовую последовательность завершения процесса сканирования, в данном случае записывается значение, равное среднему количеству ударов сердца в минуту (расчет производится средствами самого датчика).

Графическая схема подпрограммы имеет вид, представленный на рисуноке Б.2. Подпрограмма getAnalogData не возвращает никаких значений, так как оперирует с буфером последовательного порта с помощью класса Serial. Алгоритм работы подпрограммы заключается в цикличном повторении некоторых инструкций до тех пор, пока в буфере порта не появится управляющая команда «stop». Суть этих инструкций заключается в постоянной проверке состояния датчика: за это отвечает подпрограмма getConnection,

исходный код которой представлен в листинге Б.6. Функция getConnection возвращает логическое значение в зависимости от состояния датчика.

Для проверки состояния используется метод check из класса Pulse, а в качестве аргумента передается константа ISP_VALID, которая определяет контактирует ли оптический датчик с кожей человека [5]. Если контакт установлен, то метод check возвращает константу ISP_CONNECTED, иначе ISP_DISCONNECTED [5]. Принцип проверки заключается в цикличном повторении операции Pulse.check(ISP_VALID) до тех пор, пока она не вернет значение ISP_CONNECTED, что определяет успешный контакт датчика с кожей пользователя [5, 26].

Значение функции getConnection записывается в переменную cnt: если значение не равно true происходит переход в начало цикла, иначе, выполняются три последовательные команды. Первая команда delay является служебной командой языка программирования для прошивки процессоров Arduino, она используется для реализации программной задержки процесса перенаправления отфильтрованного сигнала на последовательный порт. В качестве аргумента принимает глобальную переменную delay_ms, которая содержит в себе значение задержки в миллисекундах [26, 11]. Во второй команде происходит фильтрация сигнала и перенаправление его в буфер серийного порта. Для перенаправления используется метод println из класса Serial, с помощью которого микроконтроллер отправляет в буфер информацию, которая содержится в целочисленном аргументе этой функции [11, 27].

После отправки значения амплитуды сигнала происходит считывание буфера последовательного порта, результат которого заносится в переменную serial_command, а затем проверяется в блоке условия на предмет наличия управляющей конструкции «stop». Если есть совпадение, то происходит выход из цикла, а затем возврат пустого значения функции и перенаправление в подпрограмму loop для ожидания инструкции «start». Исходный код подпрограммы getAnalogData приведен в листинге Б.7. Бесконечный цикл реализуется с помощью цикла while, в условии которого прописана константа

«1» [20, 27]. Выход из цикла осуществляется с помощью оператора break в том случае, когда переменная serial_command содержит в себе инструкцию «stop» [11].

Драйвер устройства реализует обмен данными между микрокомпьютером и микроконтроллером, а также осуществляет цифровую фильтрацию и передачу сигнала в буфер последовательного порта.

3.2 Разработка программного обеспечения микрокомпьютера для работы с микроконтроллером

3.2.1 Настройка параметров работы микрокомпьютера Raspberry Pi 3

Raspberry Pi 3 – это портативный компьютер, использующий Linux-подобную операционную систему Raspbian. Благодаря этому при запуске микрокомпьютера пользователь получает доступ к большому количеству компиляторов и интерпретаторов различных языков программирования, которые установлены по умолчанию в дистрибутив операционной системы [5, 12]. Проблему взаимодействия пользователя с таким устройством, в силу отсутствия каких-либо устройств ввода-вывода информации, можно решить двумя способами: подключение монитора к микрокомпьютеру с помощью HDMI-кабеля либо использование локальной Интернет-сети [5]. В выпускной квалификационной работе был выбран второй способ. Микрокомпьютер подключается к локальной сети Интернет с помощью Ethernet-кабеля [5]. Любой пользователь, также подключенный к данной сети, имеет доступ к микрокомпьютеру по различным каналам связи, например, с помощью SSH-протокола, который является самым популярным в среде Linux [5, 23]. Для доступа к микрокомпьютеру через SSH-туннель следует знать имя и пароль машины, к которой следует подключаться. По умолчанию: имя машины «pi», а пароль «raspberry» [23]. Для подключения к интерфейсу

микрокомпьютера можно использовать приложение Putty, изображено на рисунке В.1, либо же консольную утилиту ssh, изображена на рисунке В.2 [23].

Приложение Putty поддерживает различные типы и протоколы для удаленного подключения (Raw, Telnet, Rlogin, SSH и Serial). В выпускной квалификационной работе использован SSH-протокол [23].

Для подключения, помимо имени и пароля, также следует знать локальный IP-адресс микрокомпьютера, который можно определить с помощью bash-скрипта который сканирует указанный диапазон IP-адресов [23, 12], листинг которого имеет следующий вид «echo 192.168.0.{1..254} | xargs –n1 –P0 ping –c1 | grep “bytes from” –color»

Данный скрипт состоит из трех bash-команд: echo, xargs и grep. Первая команда производит вывод данных в окно терминала, вторая команда производит подбор значений (указанных в фигурной скобке) IP-адреса, а третья команда выводит те адреса, которые отвечают на ping-запрос [12]. Пример выполнения изображен на рисунке 3.1.

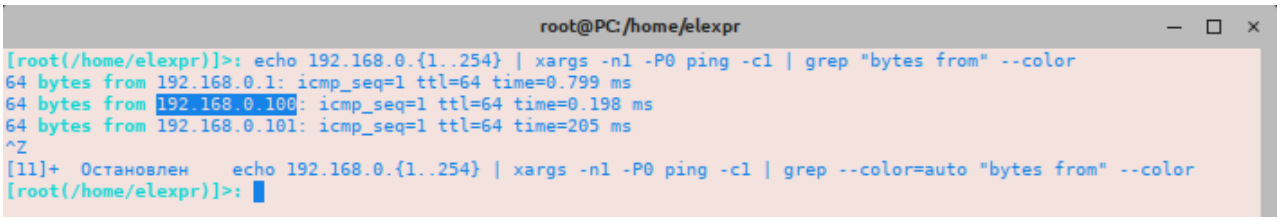


Рисунок 3.1 – Пример поиска активных IP-адресов

После успешного подключения к консольному терминалу Rasbperry Pi 3 следует установить VNC-сервер (Virtual Network Computing) – система удаленного доступа к рабочему столу. Благодаря чему будет получен доступ к графическому окружению операционной системы Raspbian, что в свою очередь позволит проводить отладку разрабатываемого программного обеспечения для работы с микроконтроллером Arduino [5, 23, 12]. Установка VNC-сервера производится командой в консольном терминале: sudo apt-get install realvnc-

vnc-server [12]. Для открытия доступа к графическому окружению Rasbperry Pi 3 следует запустить VNC-сервер, запуск изображен на рисунке 3.2.

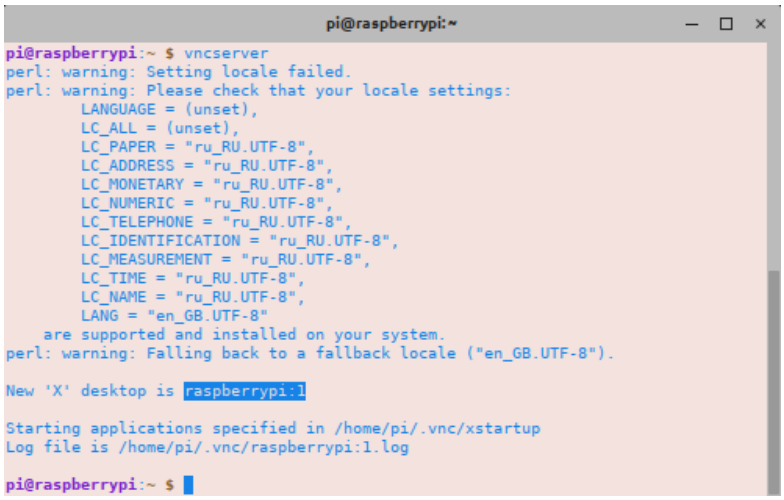


Рисунок 3.2 – Запуск VNC-сервера в консольном терминале микрокомпьютера

После успешного запуска программа сервера вернет номер порта, к которому можно подключится для работы с графическим окружением микрокомпьютера.

К микрокомпьютеру можно подключится с помощью любой программы для работы с VNC-протоколом, например, в среде Linux популярна программа Remmina [12].

Для работы с удаленным устройством следует создать новое подключение, задав необходимые параметры, изображенные на рисунке В.3. Нажав кнопку «Подключится», если не было допущено каких-либо ошибок, окно сменится новым окном с графическим окружением микрокомпьютера Raspberry Pi 3, изображенный на рисунке В.4. В случае необходимости можно сконфигурировать дополнительные настройки подключения к беспроводной сети, например, прописав SSID и пароль от Wi-Fi-точки доступа в специальном файле, задается автоматический поиск данной сети при каждом включении микрокомпьютера [23].

Файл конфигурации беспроводных сетей находится в определенной директории устройства [12], изображен на рисунок 3.3.

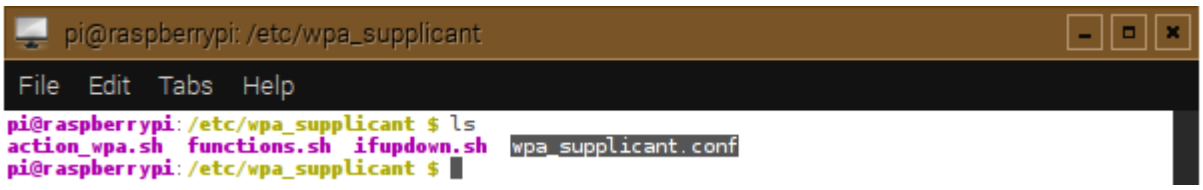


Рисунок 3.3 – Файл /etc/wpa_supplicant/wpa_supplicant.conf

Открыв содержимое, или создав этот файл с таким же именем (если его не существует), следует вписать в него конфигурации, приведенные в листинге В.1. Первые две строчки листинга В.1 содержат настройки и предустановки WPA-интерфейса; далее следует описание параметров точки доступа Wi-Fi: поле ssid – имя точки доступа, psk – пароль точки доступа (в кавычках следует использовать настоящие параметры сети) [12]. Новые конфигурации должны вступить в силу после перезагрузки устройства [12].

Таким образом происходит настройка микрокомпьютера Raspberry Pi 3, которая обеспечивает простой доступ к устройству, удобную отладку приложения и переконфигурирование некоторых параметров, если это потребуется при изменении структуры всей системы.

3.2.2 Разработка программного драйвера для микроконтроллера

Разрабатываемый драйвер устройства должен проверять наличие и параметры подключения устройства к микрокомпьютеру. В выпускной квалификационной работе разработка драйвера осуществляется под Linux-подобные операционные системы [12]. В Linux системах все устройства, подключенные к компьютеру через COM-порт, представляются в виде буферных файлов, которые хранятся в корневой директории dev [12].

При подключении какого-либо устройства автоматически создается буферный файл, название, которого начинается с символов tty. Так, например, при подключении Arduino Leonardo к микрокомпьютеру в директории dev создается файл ttyACM*, где вместо символа звездочки прописывается номер сессии подключения [5, 12]. С помощью утилит ls с ключом –l и утилиты grep в

директории dev определяется файл «ttyACM*» типа «dialout». Листинг скрипта, определяющего буферный файл подключенного микроконтроллера, имеет вид, приведенный в листинге В.2. Если микроконтроллер подключен, то результат выполнения этого скрипта будет сохранен в файле port, напрмер, в нем будет содержаться следующая строка: «crw-rw---- 1 root dialout 166, 0 фев 24 16:42 /dev/ttyACM0».

Программа драйвера первым делом запустит BASH-скрипт, выделит из файла port имя буфера последовательного порта, а затем, запустит подпрограмму считывания сигнала, предварительно отправив инструкцию «start». Главная подпрограмма драйвера, с помощью которой производится доступ к устройству, имеет вид, представленный на рисунке В.5. Функция main принимает три аргумента: smpls – количество данных считанных из порта, path – строка, обозначающая путь к файлу, в который будут записаны показания и ассоциативный массив kw (содержит в себе указатели на различные объекты графического интерфейса разрабатываемой программы, например, компонент «Progress Bar» и т.п.); посредством этого массива драйвер сообщает свое состояние главной программе, которое отображает его с помощью GUI-компонентов.

При запуске данной подпрограммы последовательно выполняются первые три команды: os.system(‘sh ./distant_ecg/tools/check_port.sh’) – запускк BASH-сценария, с помощью которого происходит определение файла буфера последовательного порта; port = open(‘./distant_ecg/tools/port’).readline() – считывание первой строки из файла «port», и запись строки в переменную в одноименную переменную; serial_port = port.split(‘ ‘)[-1][0: -1] – разбиение строки на подстроки, получение последней подстроки из списка и запись конечного результата в переменную serial_port. Последняя совокупность операций предназначена для выделения имени файла буфера порта из строки: «crw-rw---- 1 root dialout 166, 0 фев 24 16:42 /dev/ttyACM0». Если устройство не подключено, то файл port будет полностью пустым, следовательно, вышеописанная последовательность операций вернет null-значение [22].

Если переменная serial_port не пуста, происходит открытие файла буфера порта и передача в него управляющей инструкции ‘start’, затем выполняется подпрограмма readSignal, результат работы которой записывается в массив samples, представляющий собой массив значений амплитуды сигнала датчика пульса. После этого, массив передается функции writeSignal, которая производит запись данных в файл path.

Функция main возвращает последний элемент массива samples, который представляет собой среднее количество ударов сердца, рассчитанное средствами самого датчика.

Для написания драйвера устройства был использован язык программирования Python [21]. Исходный код функции main представлен в листинге В.3. Драйвер использует некоторые встроенные модули: serial – для работы с последовательными портами, os – для работы с командной оболочкой операционной системы [21, 22]. Так, например, с помощью метода system из модуля os, происходит выполнение BASH-сценария [22].

Подпрограмма readSignal выполняет главную функцию драйвера: производит считывание показаний из последовательного порта и помещает их в целочисленный массив в режиме реального времени.

Подпрограмма writeSignal производит запись массива samples в файл FILEPATH, который будет использован пользовательским приложением.

Графическая схема, описывающая ключевые моменты работы алгоритма подпрограммы имеет вид, представленный на рисунке В.6. Происходит считывание поступивших данных из буфера серийного порта ровно столько раз, сколько это задано переменной smpl. После инициализации переменных samples и curr_smp, которые обозначают массив значений сигнала датчика и счетчик цикла соответственно, происходит инициализация переменной буфера последовательного порта с помощью экземпляра класса Serial из модуля serial [21]. В качестве аргументов для конструктора класса выступают два объекта: переменная serial_port – содержащая в себе путь к файлу буфера последовательного порта и константа 9600, обозначающая скорость обмена

данным [5]. В теле цикла выполняются три операции: считывание строки из файла буфера порта, добавление этой строки в массив samples и увеличение счетчика цикла на единицу. После того, как счетчик станет равным переменной smpl, цикл завершится, произойдет закрытие файла буфера порта с помощью метода close из класса Serial, а также, добавится последнее значение массива samples, которое обозначает среднее количество ударов сердца в минуту (рассчитанное с средствами самого датчика Pulse Sensor). Это происходит с помощью подпрограммы getPulse, которая посылает в буфер последовательного порта управляющую инструкцию «stop», в ответ на которую микроконтроллер запрашивает среднее значение пульса у устройства датчика, передает его в буфер порта и переходит в режим ожидания. Прочитав это значение из буфера порта, подпрограмма закрывает файл и завершается возвратом полученной величины.

Исходные коды подпрограмм getPulse и readSignal представлены в листингах В.4 и В.5 соответственно. Подпрограмма, в качестве аргумента, принимает переменную SERIAL_PORT, которая содержит в себе абсолютный путь к файлу буфера порта; передача управляющей инструкции «stop» осуществляется с помощью метода write; в ответ на такую инструкцию микроконтроллер отправляет среднее значение ударов сердца в минуту, которое считывается с помощью метода readline и сохраняется в переменную pulse.

Подпрограмма readSignal принимает еще два аргумента: STATUS_LABEL и PARENT_FRAME, которые нужны для отображения статуса работы датчика в пользовательском графическом интерфейсе. Переменная STATUS_LABEL содержит в себе указатель на компоненту «Label», расположенную на главной форме приложения, а переменная PARENT_FRAME – указатель на родительскую форму (после изменения значения любого компонента формы для отображения этих изменений следует обновить родительскую форму с помощью метода update) [21]. Компонента «Label» используется для отображения процентного соотношения статуса

процесса считывания показаний датчика. Изменение значения этого компонента происходит с помощью метода configure, в котором само значение задается с помощью именованного аргумента text [22].

Драйвер микроконтроллера, с помощью которого происходит обмен данными между устройством и микрокомпьютером состоит из двух условных частей: сценарий командной оболочки Linux, благодаря которому происходит поиск файла буфера порта и Python-модуль, состоящий из подпрограмм, которые реализуют считывание и сохранение полученной информации.

3.2.3 Проектирование и разработка пользовательского приложения

Целью разрабатываемой программы является передача показаний датчика конкретного пользователя в его личный кабинет, который расположен на веб-сервисе.

К задачам относятся следующие пункты:

а) аутентификация пользователя;

б) считывание показаний датчика;

в) формирование пакета персональных данных (показания датчика и личная информация пользователя);

г) отправка пакета POST-запросом на сервер;

д) реализация графического пользовательского интерфейса.

Аутентификация пользователя происходит посредством интернет-запроса к определенному адресу веб-сервиса. Запрос формируется автоматически на основании введенных пользователем данных в форме аутентификации: логина и пароля. Приложение отправляет GET-запрос на веб-сервер, который, в зависимости от полученной хеш-суммы возвращает два статуса: пользователь прошел аутентификацию, либо пользователь с такой хеш-суммой не найден. Ответ сервера формируется в JSON-формате, которые распознается с помощью любого языка программирования [23]. Если ответ сервера положительный, то клиентское приложение производит формирование нового интерфейса, с

помощью которого будет производится съем и отправка данных на сервер. Деаутентификация происходит таким же образов, только с единственным отличием: токен авторизации отправляется на другой адрес веб-сервера, который отвечает за процесс закрытия сессии авторизации.

Процессы аутентификации и деаутентификации реализованы с помощью языка программирования Python [21]. Исходный код подпрограммы аутентификации пользователя представлен в листинге В.6. Подпрограмма sign_in, в качестве входных параметров принимает две переменные: login и password, содержащие логин и пароль соответственно. Переменная self – является служебной переменной, которая представляет собой область имен класса, так как данная подпрограмма является методом класса Auth. Хеш-сумма вычисляется с помощью экземпляра класса md5, который импортируется из одноименного модуля. Метод hexdigest переводит полученную сумму из шестнадцатеричного представления в строковое [22]. GET-запрос к серверу выполняется с помощью метода get из модуля requests. В качестве аргументов, подпрограмма принимает одно значение – адрес веб-ресурса, на который будет производится GET-запрос. Переменная SIGN_IN_ADDR является глобальной в области имен класса и инициализируется в конструкторе класса, исходный код, которого представлен в листинге В.7. В конструкторе также определены такие переменные, как SIGN_IN_ADDR – адрес PHP-сценария аутентификации с GET-параметром auth_token (фигурные скобки используются для форматирования); SIGN_OUT_ADDR – адрес PHP-сценария деаутентификации; UPLOAD_DATA_ADDR – адрес PHP-сценария для загрузки результатов работы датчика; AUTH_TOKEN – глобальная переменная, использующаяся для передачи токена авторизации между методами класса. Переменная HOST определяет корневой хост сайта веб-сервиса.

Адрес сценария аутентификации имеет следующий вид: «/auth/auth.php?auth_token={auth_token}», где {auth_token} это хеш-сумма. Из кода подпрограммы sign_in видно, что подстановка значения происходит с помощью метода format [21]. Метод json из класса, который формируется в

ходе выполнения запроса, возвращает ответ сервера в JSON-представлении. В языке программирования Python, правила оформления ассоциативного массива аналогичны правилам оформления JSON. С помощью метода update происходит добавление ответа сервера в глобальный ассоциативный массив JSON, который объявлен вне всех подпрограмм и классов [21, 22]. Из ассоциативного массива с помощью метода get извлекается значение под ключевым именем «status». Метод sign_in возвращает либо True, либо False, в зависимости от полученного ответа сервера. Для деаутентификации используется подпрограмма sign_out, аналогичная sign_in, однако она оперирует уже с готовым токеном авторизации, приведенным в листинге В.8: выполняется GET-запрос на адрес SIGN_OUT_ADDR с хеш-суммой AUTH_TOKEN, в ответ на который сервер завершает сессию пользования. После выполнения этой подпрограммы закрывается интерфейс отправки данных и инициализируется начальный интерфейс аутентификации.

Помимо вышеперечисленных методов в классе Auth присутствует метод для загрузки данных на сервер. Подпрограмма читает данные из файла с результатами работы датчика. Затем эти данные зашифровываются с помощью алгоритма BASE64. Далее формируется POST-запрос, тело которого изображено на рисунке 3.4.

```
POST = {'user_hash': AUTH_TOKEN,
        'path': PATH,
        'filename': FILENAME,
        'data': BASE64_DATA}
```

Рисунок 3.4 – Тело POST-запроса на сохранение файла

Поле «user_hash» содержит в себе токен авторизации (хеш-сумма) AUTH_TOKEN, в виде строки; «path» – путь к папке пользователя на самом сервере (этот путь сервер возвращает вместе с другими параметрами в виде JSON при успешной аутентификации); «filename» – новое имя файла, в котором

будет хранится результат работы датчика на сервере (имеет формат вида {YEAR}-{MONTH}-{DAY}-{H}-{M}.csv); «data» – зашифрованные данные датчика.

Готовое тело POST-запроса отправляется на обработку PHP-сценария, который расположен по адресу UPLOAD_DATA_ADDR. Сервер обрабатывает этот запрос и сохраняет полученный файл в папке пользователя, при этом возвращая ответ в виде JSON в поле «status» (в случае успеха это код «200», иначе «400»). В зависимости от ответа сервера подпрограмма возвращает значение True (при успешной загрузке файла), либо False (если произошел какой-либо сбой). Исходный код подпрограммы для отправки данных представлен в листинге В.9.

Шифрование данных происходит с помощью метода b64encode из модуля base64. В качестве аргумента этот метод принимает содержимое всего файла в виде строки, которая считывается с помощью метода read [22].

Формирование нового имени файла происходит с помощью замены ключевых маркеров строки (обрамлены в фигурные скобки), каждый из маркеров отображает год, месяц, день, часы и минуты на момент отправки файла. Текущее время и дата вычисляется с помощью метода localtime из модуля time [21].

Отправка POST-запроса происходит с помощью метода post из модуля requests [21]; ответ сервера вносится в переменную server_response, которая содержит метод json, возвращающий ассоциативный массив, из которого с помощью метода get берется значение поля «status», содержащее в себе статус выполнения данной операции.

Пользовательский интерфейс разрабатываемого приложения можно разделить на два взаимосвязанных субинтерфейса: интерфейс аутентификации и рабочий интерфейс съема и отправки данных.

Интерфейс аутентификации инициализируется сразу же после запуска программы. Интерфейс аутентификации имеет вид, представленный на рисунке 3.5.

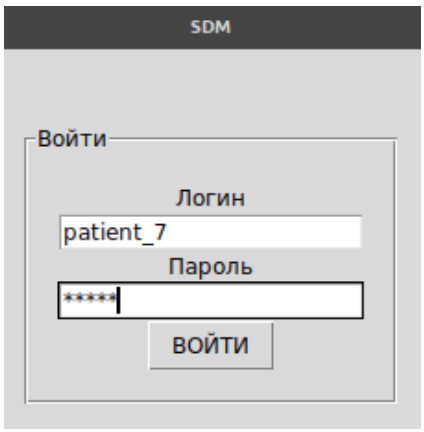
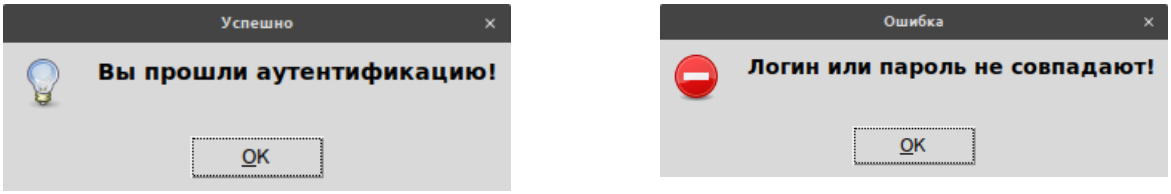


Рисунок 3.5 – Главное окно интерфейса аутентификации

Окно содержит два поля для ввода (логин и пароль) и кнопку «SIGN IN», при нажатии на которую происходит выполнение метода sign_in из класса Auth.

В зависимости от того, какой результат вернет данный метод, интерфейс аутентификации вызывает модальное окно, оповещающее о статусе аутентификации, изображенное на рисунке 3.6.



а) б)

Рисунок 3.6 – Сообщение об аутентификации: а) при успешной аутентификации; б) при ошибке аутентификации

При успешной аутентификации пользователя происходит инициализация интерфейса съема и передачи данных. Если во время аутентификации произошел какой-либо сбой, либо же, пользователь ввел неверные данные, модальное окно принимает вид, приведенный на рисунке 3.6,б.

Весь графический интерфейс приложения спроектирован в виде класса, методы которого отвечают за инициализацию той или иной части всего интерфейса приложения [16]. Исходный код объявления класса и его конструктора представлен в листинге В.10. Класс Client наследует свойства и

методы объекта, который будет передан при инициализации в качестве аргумента Frame. Также, данный класс имеет глобальные переменные: LOGIN, PASSWORD и ECG_DATA_PATH, использующиеся для хранения логина, пароля и абсолютного пути файла, в который будут записаны данные, полученные с датчика; помимо этого, в глобальной области имен класса инициализируется вышеописанный класс Auth.

Конструктор __init__ в качестве аргумента принимает объект, который используется в качестве родительского «фрейма», на котором будет размещаться тот или иной интерфейс. Именно этот «фрейм» и задается при инициализации для его наследования. Строка «Frame.__init__(self, parent, background=”white”)» инициализирует главный корневой «фрейм», являющийся родителем для всех дочерних «фреймов», которые будут создаваться в ходе работы приложения [21, 16]. В переменную child вносится объект Frame из модуля ttk, который наследует свойства объекта parent, являющийся родительским «фреймом». Таким образом создается дочерний «фрейм». С помощью метода pack происходит размещение указанного компонента на родительском «фрейме» [22]. Последним этапом инициализации класса является запуск инициализации интерфейса аутентификации (строка «self.initUI(self.initAuthUI)»).

Исходный код метода initUI представлен в листинге В.11. Метод initUI является общим для любого интерфейса и используется для размещения интерфейса content на дочернем фрейме child, который был определен в глобальной области имен класса. В переменную content передается объект-метод инициализации того или иного интерфейса, причем, каждый такой метод размещает свой интерфейс внутри «фрейма» child, поэтому, первым действием конструктора интерфейсов является очистка дочернего фрейма от уже существующих интерфейсов (выполняется с помощью строки «self.clearFrame(self.child)»). Метод clearFrame в качестве аргумента принимает объект типа Frame и производит поэлементное удаление содержимого этого «фрейма». Такое удаление требуется для того, чтобы старые элементы

интерфейса пропали перед инициализацией новых [21]. Последним действием метода initUI является выполнение метода, указатель которого находится в переменной content. Вызов этого метода производится с аргументом initAuthUI, который является методом для инициализации интерфейса аутентификациит исходный код приведен в листинге В.12. Данный метод, в качестве условного аргумента принимает глобальную область имен класса (self), что позволяет напрямую обращаться к полям и переменным, определенным в любой части класса Client.

Таким образом, этот метод инициализирует свой «фрейм» на родительском (относительно нового «фрейма») «фрейме» child (который был определен при инициализации класса). Это происходит посредством строки «authUI = ttk.Frame(self.child)», затем, происходит размещение этого «фрейма» с помощью метода place, с указанием координат осей X и Y, а также выравниванием (anchor) [22]. Переменная authUI является родительским «фреймом» для всех своих будущих компонент, таких как Label (текстовые подписи), Button (кнопки) и т.п [22].

Так, например, первый компонент подписи размещается аналогично, только вместо метода Frame используется аналогичный метод LabelFrame (добавляющий подпись с рамкой). Как видно из строчки «labelFrame = LabelFrame(authUI, padx = 16, pady = 16, text = «Sign in»)», теперь, в качестве родительского «фрейма» задается authUI вместо child (так как, «фрейм» authUI принадлежит «фрейму» child).

Дополнительные параметры padx, pady и text используются для задания координат размещения подписи и для задания самого текста подписи соответственно [21]. Размещение фрейма происходит с помощью метода pack, который отличается от place способом задания координат [22]. Таким же образом происходит размещение всех остальных надписей Label, полей для ввода Entry и кнопок Button [21].

При добавлении компонента Button, например, кнопки sign_in_button, отвечающей за запуск процесса аутентификации, необходимо привязать ее к

конкретному методу, который отвечает за этот процесс [21]. Привязка происходит с помощью метода bind, из экземпляра класса Button [16, 22]. Данный метод, в качестве первого аргумента принимает тип события (event). В данном случае это строка «<Button-1>», которая обозначает единичное нажатие левой кнопкой мыши. Вторым аргументом выступает функция или метод, который должен выполнится при нажатии на указанную кнопку [21, 22].

В данном случае, указание такой функции происходит неявно (функция является анонимной), с помощью служебной конструкции lambda, которая позволяет описывать анонимные функции с любым набором переменных [21]. Синтаксис lambda-выражений имеет вид: lambda {ARGS}:{EXPRESSION}, где вместо {ARGS} описывается набор переменных, а вместо {EXPRESSION} прописывается выражение, которое следует выполнить, например: lambda x: x*x – функция, возвращающая квадрат значения x [21, 22].

При нажатии на кнопку происходит вызов метода receiv_to_auth со следующим набором именованных аргументов: login = login_entry, password = password_entry. Эти аргументы ссылаются на соответствующие поля для ввода логина и пароля.

После привязки происходит размещение кнопки с помощью метода pack; метод initAuthUI возвращает «фрейм» authUI, содержащий в себе все описанные компоненты интерфейса аутентификации.

Взаимодействие графического интерфейса и методов аутентификации из класса Auth производится с помощью метода receiv_to_auth, исходный которого представлен в листинге В.13. Суть работы этой подпрограммы заключается в вызове метода sign_in из класса Auth. Данный метод в зависимости от статуса аутентификации возвращает либо True, либо False. Это значение вносится в переменную, которая затем проверяется: если аутентификация прошла успешно, то в глобальные переменные LOGIN и PASSWORD записываются значения логина и пароля, после, происходит вызов модального (строка «tkMessageBox.showinfo(“Successful”, “You are sign in now!”)» окна, изображенного на рисунке 3.6,а, а затем, с помощью конструктора интерфейсов

initUI происходит инициализация нового интерфейса съема и отправки данных, за который отвечает метод initLoginUI.

Интерфейс съема и отправки данных имеет вид, представленный на рисунке 3.7

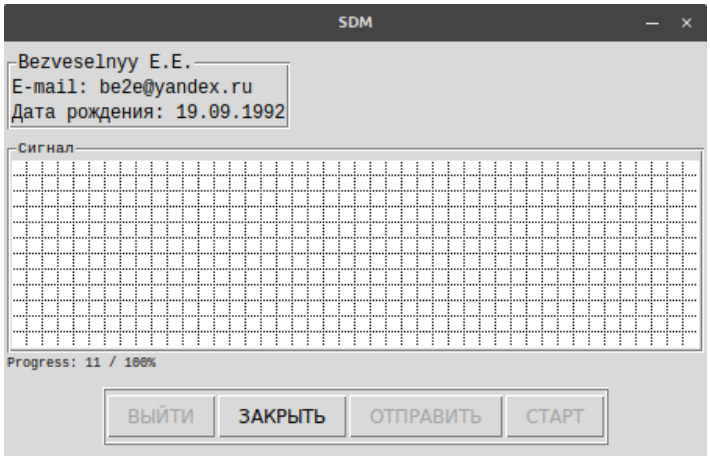


Рисунок 3.7 – Графический интерфейс съема и отправки данных

Окно состоит из трех областей: область информации, в которой выводится информация о пользователе, прошедшем аутентификацию (Ф.И.О., адрес электронной почты и дата рождения), область сигнала, в которой выводится минимизированная копия полученного с датчика сигнала, а также, статус съема, и, панель управления датчиком, содержащая в себе четыре кнопки: «ВЫЙТИ», «ЗАКРЫТЬ», «ОТПРАВИТЬ» и «СТАРТ».

Кнопка «ВЫЙТИ» осуществляет деаутентификацию пользователя и запускают инициализацию интерфейса аутентификации, кнопка «ЗАКРЫТЬ» полностью закрывает все приложение, кнопка «ОТПРАВИТЬ» формирует POST-запрос и посылает его серверу, а кнопка «СТАРТ» начинает процесс съема данных с датчика.

После нажатия кнопки «СТАРТ» происходит считывание данных с датчика, процесс изображен на рисунке 3.8, когда процесс дойдет до отметки в 100% произойдет сохранение сигнала в локальном файле, его считывание и формирование простого графика.

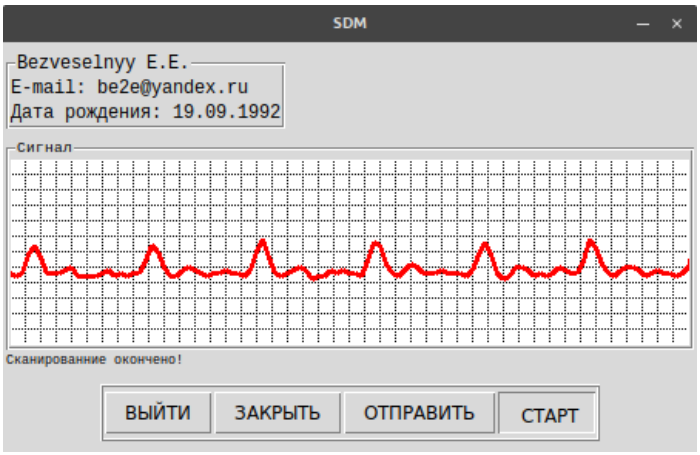


Рисунок 3.8 – Отображение сигнала полученного с помощью датчика

Формирование областей происходит посредством методов: initUserInfoUI – формирование области информации о пользователе, initCardioCanvas – формирование области отображения сигнала, initControlPanel – формирование панели управления, initStatusBar – формирование области строки состояния. Каждый из этих методов в качестве аргумента принимает родительский «фрейм», на котором будет происходить размещение тех или иных элементов интерфейса [21]. Исходный код метода initLoginUI представлен в листинге В.14.

После описания всех методов главный метод initLoginUI создает новый «фрейм» на родительском (относительно самого себя) «фрейме» child, строка: «loginUI = ttk.Frame(self.child)»; размещение происходит с помощью метода pack. После создания нового «фрейма» loginUI, происходит поочередный вызов вышеописанных методов, а в качестве их аргументов, передается «фрейм» loginUI, на котором и будет происходить размещение компонентов графического интерфейса. Механизм и логика работы этого интерфейса полностью сосредоточены в панели управления приложением (initContorlPanel), которая содержит в себе четыре кнопки взаимодействия [16].

Кнопка «ВЫЙТИ» производит деаутентификацию, а событие нажатия этой кнопки привязано к подпрограмме receiv_to_sign_out, которая производит вызов метода sign_out из класса Auth, а затем, производит деконструкцию пользовательского интерфейса с помощью конструктора интерфейсов (строка

«self.initUI(self.initAuthUI)» – таким образом происходит смена рабочего интерфейса на интерфейс аутентификации.

Кнопка «ЗАКРЫТЬ» привязана к подпрограмме quit, которая производит деаутентификацию и полное закрытие приложения, с помощью служебной функции exit.

Кнопка «ОТПРАВИТЬ» по умолчанию неактивна до тех пор, пока не произойдет считывание сигнала. Данная кнопка привязана к подпрограмме receiv_to_sending_data, исходный код которой представлен в листинге В.15. В переменную is_ok вносится результат работы метода send_ecg_data из класса Auth.

Если подпрограмма возвращает False-значение, происходит вызов модального окна с оповещением об ошибке, в противном случае, происходит вызов модального окна с оповещением об успешности операции, окно изображено на рисунке 3.9.

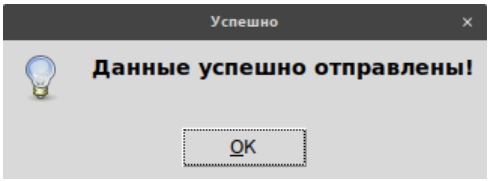


Рисунок 3.9 – Оповещение об успешной отправке данных на сервер

Кнопка «СТАРТ» привязана к методу scan_ecg, который содержит в себе дочерние методы, использующиеся для работы с драйвером устройства.

Суть этого метода заключается в проверке подключения устройства и запуске считывания показаний датчика, а после, прорисовки графика сигнала.

Исходный код подпрограммы scan_ecg представлен в листинге В.16.

Метод scan_ecg содержит в себе три дочерние подпрограммы: make_grid, change_status и preview_plot; например, make_grid прорисовывает сетку графика (т.к. перед прорисовкой самого графика происходит и очистка содержимого компонента Canvas вместе с самой сеткой), подпрограмма change_status используется для активации и деактивации кнопок панели управления во время

съема сигнала, и подпрограмма preview_plot используется для прорисовки части графика сигнала, полученного с датчика.

Подпрограмма деактивирует кнопки панели управления (строка «change_status(DISABLED)»); запускает главную подпрограмму драйвера устройства из пакета distant_ecg модуля read_ecg.

Общий статус работы датчика возвращает главная подпрограмма драйвера. Статус вносится в переменную device_is_ok: значение True в случае корректной работы, False – в случае сбоев. В зависимости от значения этой переменной выводится то или иное модальное окно, сообщающее либо об успехе выполнения, окно изображено на рисунке 3.10, либо о возникновении ошибки.

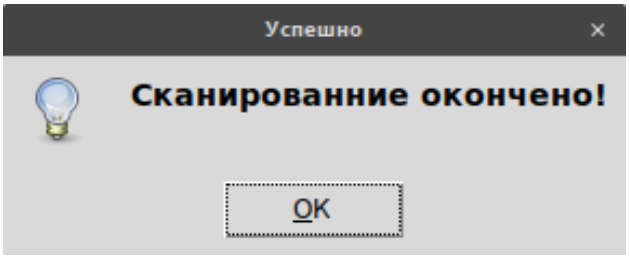


Рисунок 3.10 – Оповещение об успешном сканировании

Далее, происходит активация панели управления, прорисовка графика сигнала, если статус работы датчика положительный, и выход из подпрограммы. Результат работы подпрограммы scan_ecg изображен на рисунке 3.8.

Полная реализация графического интерфейса приложения сосредоточена в классе Client, который наследует свойства объекта Frame, создающийся в главной подпрограмме main.

При создании графического интерфейса использовался модуль Tkinter, который позволяет создавать кроссплатформенные приложения, использующие графический пользовательский интерфейс [21, 16].

Таким образом программная часть программно-аппаратной платформы состоит из трех компонентов:

а) графическое пользовательское приложение для микрокомпьютера;

б) программный драйвер для работы с микроконтроллером;

в) программа прошивки микроконтроллера.

Было разработано специальное пользовательское приложение, способное проводить аутентификацию посредством подключения к базам данных веб-сервера, отправлять пакеты данных, включающие персональную информацию и показания датчика, а также, способное взаимодействовать с датчиком с помощью разработанного драйвера.

Программный драйвер выполняет функцию посредника между микроконтроллером и микрокомпьютером – реализует интерфейс обмена информацией между двумя устройствами через последовательный порт, помимо этого, средствами драйвера можно задавать параметры работы микроконтроллера.

Программное обеспечение микроконтроллера выполняет функцию фильтрации и перенаправления потока данных из датчика в серийный порт, также, помимо этого, с помощью данного программного обеспечения производится анализ управляющих инструкций, отправляемых микрокомпьютером через последовательный порт к микроконтроллеру.

3.3 Проектирование и разработка веб-сервиса системы

3.3.1 Настройка параметров работы веб-сервера

Перед тем, как приступить к проектированию и разработке веб-сайта, следует настроить сервер, на котором будет доступен разрабатываемый сервис [23].

Операционной системой компьютера, который будет использован в качестве сервера, является Linux Ubuntu 18.04, поэтому, в качестве бесплатного серверного программного обеспечения, лучше всего использовать Apache-сервер [12].

Для его установки, в консольном терминале системы следует выполнить команды, приведенные в листинге Г.1. Выполнение этих команд приведет к обновлению всей операционной системы, установке сервера Apache, а также, добавлению команды запуска сервера при включении компьютера.

Для удобной разработки сайта следует прописать некоторые конфигурации в файл .htaccess, который находится в корне сайта (/var/www/site/) [23]. Конфигурации сервера представлены на листинге Г.2.

Первая строка отключает индексацию страниц, благодаря чему пользователь не сможет просматривать дерево структуры сайта [23]. С помощью второй конфигурации устанавливается максимальное количество переменных, которые могут быть переданы с помощью GET или POST-запросов [23]. Последние три строки активируют режим отладки PHP-сценариев [23].

Когда сервер исправно работает, следует установить PHP-интерпретатор, так как веб-сервис будет написан с помощью этого языка программирования. Для этого достаточно выполнить следующую команду в консольном терминале: sudo apt install php [16].

Теперь, следует включить PHP-интерпретацию сценариев в самом приложении Apache, для этого следует выполнить команду «sudo a2enmod php7.0», в данном случае, версия интерпретатора 7.0, и перезапустить сервер: service apache2 restart [16, 23].

Заключительным этапом является установка сервера базы данных, в данном случае, будет использована СУБД MySQL [24]. Для ее установки, в консольном терминале следует выполнить следующую команду: «sudo apt install mysql-server» [16].

В процессе установки, инсталлятор запросит ввести пароль для пользователя базы данных, который будет использоваться при подключении к ней.

Таким образом происходит настройка и конфигурирование компьютера для использования его в качестве сервера, на котором будет размещен

полноценный веб-сервис, взаимодействующий с системой управления базами данных.

3.3.2 Проектирование и реализация базы данных

Целью работы данного веб-сервиса является обеспечение безопасного доступа к результатам дистанционного мониторинга сердечной деятельности.

Пользователь веб-сервиса может выступать в одной из двух ролей:

– врач-диагност, имеющий расширенный доступ к результатам мониторинга и имеющий возможность редактировать данные сердечной деятельности пациента;

– пациент, имеющий возможность просматривать список своих анализов и заключений врача-диагноста, которые закреплены за каждым результатом мониторинга.

Средствами данного веб-сервиса, пользователь в роли врача-диагноста имеет возможность создавать и редактировать заключения о каждом результате дистанционного мониторинга. Помимо этого, пользователь имеет возможность создавать шаблоны, которые могут использоваться всеми врачами-диагностами для постановки заключения. Каждый пользователь в роли пациента имеет возможность проходить аутентификацию с помощью приложения для дальнейшего мониторинга сердечного ритма. Логическая ER-модель будущей базы данных имеет вид, представленный на рисунке 3.11.

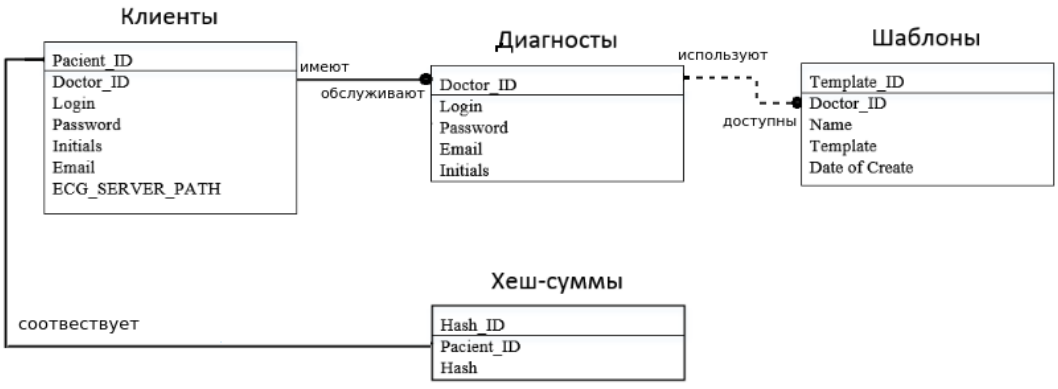


Рисунок 3.11 – Логическая ER-модель базы данных

База данных состоит из четырех взаимосвязанных таблиц: «Клиенты», «Диагносты», «Шаблоны» и «Хеш-суммы».

Таблица «Клиенты» предназначена для хранения информации о пользователях в роли пациентов. Как видно из структуры таблицы, она состоит из семи полей, где поле Pacient_ID – является первичным ключом, поле Doctor_ID – является вторичным ключом [19, 15] (за каждым пациентом закреплен один пользователь-диагност, о чем свидетельствует связь с таблицей «Диагносты»); поля Login, Password, Initials и Email содержат в себе данные о логине, пароле, Ф.И.О. и адресе электронной почты соответственно. Поле ECG_SERVER_PATH содержит информацию о пути к директории сервера, в которой хранятся результаты мониторинга и все заключения.

Таблица «Диагносты» предназначена для хранения информации о пользователях в роли врачей-диагностов, содержит аналогичные поля Login, Password, Initials и Email.

Таблица «Шаблоны» хранит в себе информацию о шаблонах заключений. В качестве первичного ключа используется поле Template_ID. Таблица связана с таблицей «Диагносты» с помощью вторичного ключа Doctor_ID (так как любой врач-диагност может создать шаблон и также использовать любой другой шаблон, присутствующий в базе данных). Поле Name – хранит информацию о кратком имени шаблона, поле Template – сам шаблон, а поле Deate of Create – дату создания этого шаблона.

Таблица «Хеш-суммы» используется для хранения информации о токенах авторизации каждого пользователя из таблицы «Клиенты». При регистрации такого пользователя, в данную таблицу вносится его персональный хеш-код, который в дальнейшем используется для аутентификации пользователя с помощью устройства мониторинга сердечного ритма. Содержит в себе три поля: Hash_ID – первичный ключ, Pacient_ID – идентификационный ключ клиента, которому принадлежит хеш-код в поле Hash.

На основании логической модели была спроектирована физическая модель базы данных, которая изображена на рисунке 3.12.

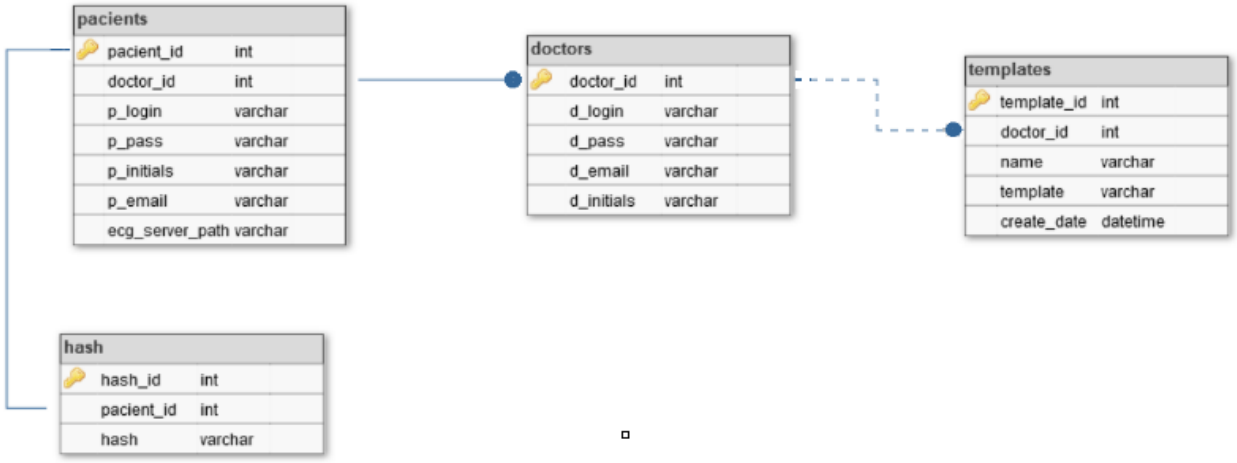


Рисунок 3.12 – Физическая ER-модель базы данных

Все поля, названия которых имеют окончания на «_id» используются в качестве первичных и внешних ключей, а типом представления данных в таких полях является целочисленное значение int [15, 13].

Поля «*_login», «*_pass», «*_email», «*_intials» содержат в себе символьную информацию, поэтому используется тип varchar, позволяющий хранить строки различной длины [24, 15].

Поле create_date, используемое для хранения информации о дате создания шаблона, имеет тип datetime; поля hash и ecg_server_path также хранят информацию в типе varchar [24].

Для реализации базы данных на основе готовой ER-диаграммы, на компьютере требуется установить систему управления базами данных (СУБД). В качестве такой СУБД была выбрана MySQL, в силу простоты в эксплуатации, распространенности и высокого уровня поддержки разработчиками [6, 7].

3.3.3 Проектирование и реализация веб-интерфейсов системы

Цель разрабатываемого веб-сервиса заключается в предоставлении доступа к всем результатам мониторинга, а также, в предоставлении возможности редактирования и формирования заключения по каждому проведенному мониторингу сердечной деятельности.

Для достижения поставленной цели следует выполнить следующие задачи:

а) реализация регистрационной формы и формы авторизации пользователей в системе с разделением на роли: диагност, клиент;

б) реализация возможности загрузки данных, передаваемых устройством;

в) реализация личного кабинета пациента с возможностью просмотра результатов мониторинга и заключений;

г) реализация обратной связи пациента с диагностом;

д) реализация личного кабинета диагноста с возможностью выбора того или иного пациента, просмотра результатов мониторинга пациента, написания заключения и возможностью редактирования сигнала сердечной деятельности.

Для разработки всех интерфейсов будущего веб-сервиса был использован язык разметки гипертекста (HTML), а также, язык описания формы документа CSS [6, 8].

Все интерфейсы будущего сервиса должны быть минималистичны и интуитивно понятны, что устраняет возможность некорректного отображения в различных браузерных приложениях [16].

Для упрощения процесса регистрации и авторизации было решено объединить две формы в одной интернет странице.

Для регистрации пользователю достаточно ввести свой новый логин, пароль и адрес электронной почты, а для авторизации лишь логин и пароль, интерфейс изображен на рисунке 3.13.

Эта форма доступна сразу после перехода на адрес сервера, так как его HTML-код прописан в файле index.php. Исходный код разметки интерфейса приведен в листинге Г.4.

Также, средствами языка программирования JavaScript, был реализован простой механизм валидации содержимого полей формы: проверка на пустые поля и проверка правильности написания адреса электронной почты пользователя системы.

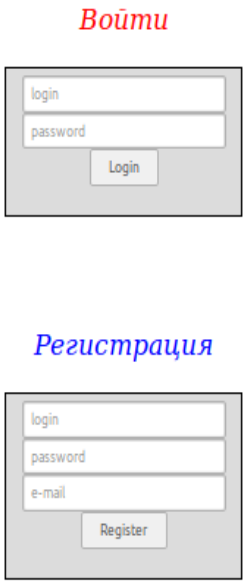


Рисунок 3.13 – Интерфейс регистрации и авторизация пользователя в системе

Главная страница представляет собой две формы: форма авторизация и форма регистрации. Оба процесса происходят посредством POST-запроса на соответствующие PHP-сценарии: login.php и register.php.

Файл login.php предназначен для выдачи сессии авторизации пользователю (с помощью функции make_session), если он верно ввел логин и пароль. Для этого происходит проверка пользователя на существование и на принадлежность его к роли пациента или врача-диагноста (с помощью функции check_user, аргументы которой это логин, пароль и роль, на которую будет происходит проверка).

Исходный код сценария представлен на листинге Г.5

Данный сценарий использует два модуля: connect.php и lr-lib.php, которые предназначены для работы с базой данных и процессами авторизации и регистрации соответственно.

Алгоритм работы сценария register.php заключается в проверке от дублирования введенных пользователем данных.

При дублировании введенных данных происходит перенаправление на страницу с сообщением, что данный пользователь уже существует. При уникальности пользователя происходит создание нового пользователя и

перенаправление в его личный кабинет. Исходный код сценария register.php представлен на листинге Г.6.

В зависимости от того, к какой роли принадлежит пользователь, происходит перенаправление в тот или иной личный онлайн-кабинет. Так, например, пользователь-пациент перенаправляется в личный кабинет пациента, внешний вид которого представлен на рисунке 3.14.

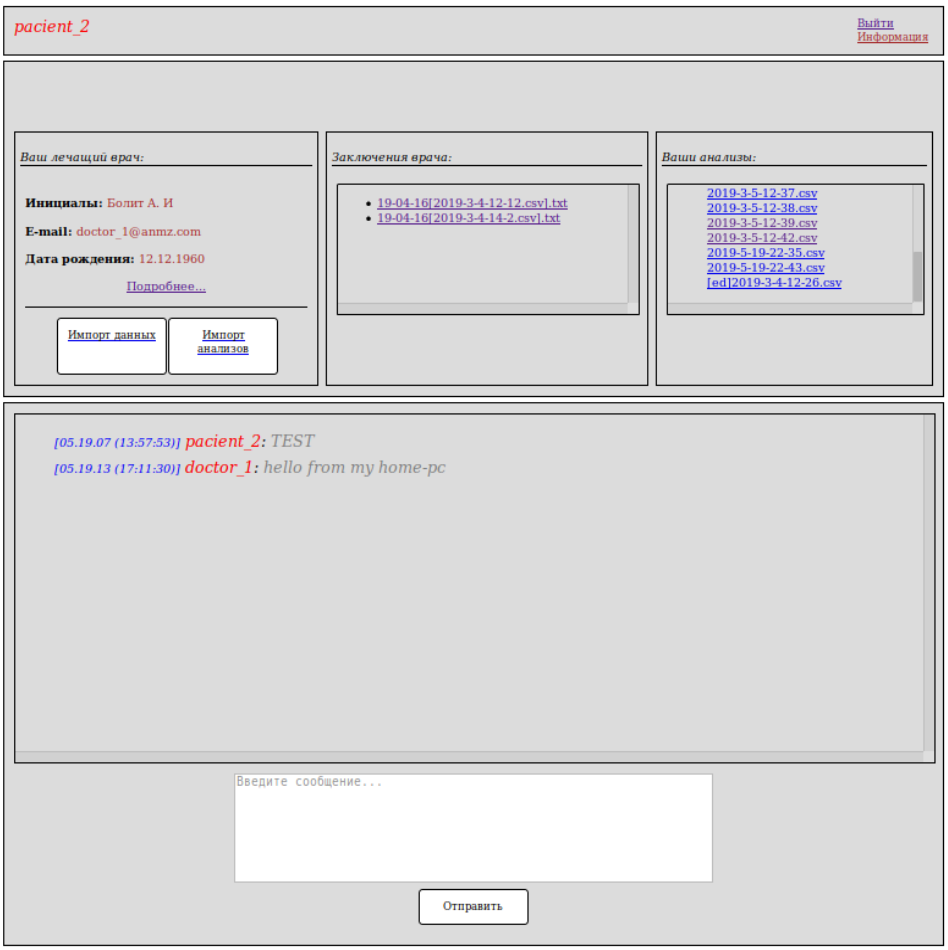


Рисунок 3.14 – Личный онлайн-кабинет пациента

В левом верхнем углу страницы личного кабинета пациента реализован блок, отвечающий за отображение личных данных пользователя, а также, возможность импорта данных в архив, далее следует блок, содержащий в себе заключения диагноста по конкретному результату мониторинга, и блок, отображающий все результаты мониторинга, нажав на конкретный из них, произойдет перенаправление на страницу с просмотром сигнала мониторинга и

заключения по нему. Помимо этого, в личном кабинете пациента реализован чат для обмена мгновенными сообщениями между закрепленным врачом-диагностом и пациентом.

Первый блок реализован с помощью HTML-разметки и PHP-кода, который и формирует главный функционал: вывод данных и возможность импорта информации, исходный код приведен в листинге Г.7.

Вывод данных о закрепленном враче-диагносте происходит с помощью функции get_doctor, которая принимает в качестве аргумента параметр $_SESSION[‘pacients’], обозначающий логин пользователя [8]. Данная функция расположена в модуле user-lib.php, содержащем подпрограммы для формирования различных элементов пользовательского интерфейса. Суть работы функции заключается в выполнении SQL-запроса, с помощью которого происходит поиск диагноста по логину пользователя. С помощью конструкций «echo» происходит вывод информации в виде HTML-разметки [13]. Исходный код подпрограммы приведен в листинге Г.8.

Блоки для вывода заключения и всех результатов мониторинга похожи по своей структуре и механизму работы, исходный код приведен в листинге Г.9.

Отличие от блока с результатами мониторинга заключается лишь в ключевой PHP-подпрограмме: в заключениях используется get_conclusion_list, а в результатах: get_data_list.

Алгоритм работы функции get_conclusion_list заключается в выполнении запроса к таблице с заключениями, откуда берется название файла заключения и его абсолютный путь на сервере. Запрос происходит по двум критериям: код пользователя в системе и код его врача-диагноста. Исходный код подпрограммы представлен в листинге Г.10. Функция возвращает значение переменной html_pattern, которая содержит в себе HTML-код, отображающий список заключений.

Подпрограмма get_data_list используется и в личном кабинете диагноста. Суть этой подпрограммы заключается форматированном выводе ссылок на

результаты мониторинга, причем, ссылка обладает GET-параметром, который является именем файла мониторинга.

В зависимости от режима работы формируются два типа ссылок: на просмотр результата мониторинга от лица пациента, или от лица диагноста. Также, от режима работы зависят зеленые маркеры рядом с ссылкой, отображающие статус мониторинга (написано ли к нему заключение или нет). Исходный код подпрограммы get_data_list приведен в приложении Б.

Чат используется для обмена мгновенными сообщения. Реализован на языке программирования JavaScript [3].

HTML-шаблон, реализующий поле для отправки сообщений и поле для их вывода имеет вид, приведенный в листинге Г.11.

Здесь следует выделить два главных блока: ul (id=”chat”) и div (class=”send-area”). Первый отвечает за вывод сообщений, а второй за их ввод. С этими блоками взаимодействует сам JavaScript с помощью нескольких функций: sendmessage – для совершения XSS-запроса на определенный адрес сайта (осуществляющий прием сообщения и сохранение его в специальном файле), last_mesages – для вывода последних сообщений из файла [3].

Подпрограмма sendmessage принимает два аргумента: код пациента и код диагноста (отправитель и получатель); далее происходит считывание данных из поля ввода (textarea), которому присвоен код «msgtxt»; после считывания данных происходит формирование POST-запроса с текстом сообщения, отправителем и получателем. Запрос отправляется на специальный адрес, где расположен PHP-сценарий, отвечающий за распределение и запись сообщений в файл. Исходный код подпрограммы sendmessage приведен в листинге Г.12.

После отправки POST-запроса происходит вызов подпрограммы last_messages для того, чтобы отправленное сообщение отобразилось в соответствующем блоке (ul). Суть работы данной функции заключается в отправке GET-запроса на специальный адрес, где с помощью PHP-сценария произойдет считывание запрашиваемого файла с сообщениями; сам сценарий вернет считанные значения, и подпрограмма передаст их функции

added_messages_on_panel, суть которой заключается в добавлении сообщений в вышеописанный блок. Исходный код подпрограммы last_messages приведен в листинге Г.13. Данная функция посылает GET-запрос к серверу, на что тот возвращает весь текст из файла сообщений, далее весь текст разбивается на отдельные сообщения и сохраняется в массиве messages, который передается функции added_messages_on_panel. Исходный код подпрограммы представлен в листинге Г.14.

Проверка наличия новых сообщений осуществляется цикличным вызовом подпрограммы last_messages, что прописано в конце документа страницы в теге script, листинг Г.15. Цикличный вызов подпрограммы производится с помощью встроенной функции setInterval, которая принимает в качестве аргументов указатель на саму вызываемую подпрограмму, время задержки (миллисекунды), и аргумент для подпрограммы [3].

Со стороны сервера для приема и выдачи сообщений используются два PHP-сценария: chat.php и chat_messages.php. Первый сценарий принимает и обрабатывает POST-запрос, отправленный подпрограммой sendmessages. Тело данного запроса состоит из четырех параметров: идентификационный код отправителя, получателя, само сообщение и роль отправителя. Далее, происходит определение пути файла с сообщениями, его открытие и запись сообщений. Исходный код сценария представлен в листинге Г.16. Путь к файлу с сообщениями производится с помощью подпрограммы get_path из модуля user-lib.php. Данная подпрограмма лишь совершает поиск пути файла по таблице клиентов с помощью идентификатора клиента в системе.

После того, как файл найден, происходит открытие файла для записи с помощью функции fopen [6, 8]. Здесь, в зависимости от переменной whois, которая определяет роль пользователя, происходит поиск логина пользователя в той или иной таблице (либо в таблице пользователей-пациентов, либо в таблице пользователей-диагностов).

После этого происходит запись логина, текущего времени и самого сообщения в файл. Возврат сообщений происходит посредством GET-запроса к

PHP-сценарию chat_messages.php, исходный код которого представлен в листинге Г.17. Данный сценарий принимает лишь один GET-параметр: идентификационный код пациента (так как файл с сообщениями находится в его личной папке). После, происходит получение пути к файлу с помощью подпрограммы get_path, а затем считывание всего файла и возврат этих данных.

Для более детального просмотра результата мониторинга пользователь может перейти по ссылкам в личном кабинете пациента, которые расположены в соответствующем блоке, эти ссылки ведут на новую страницу, которая позволяет рассмотреть результат мониторинга в виде сигнала, сохранить в изображении или CSV-файле.

Интерфейс данной страницы изображен на рисунке 3.15.

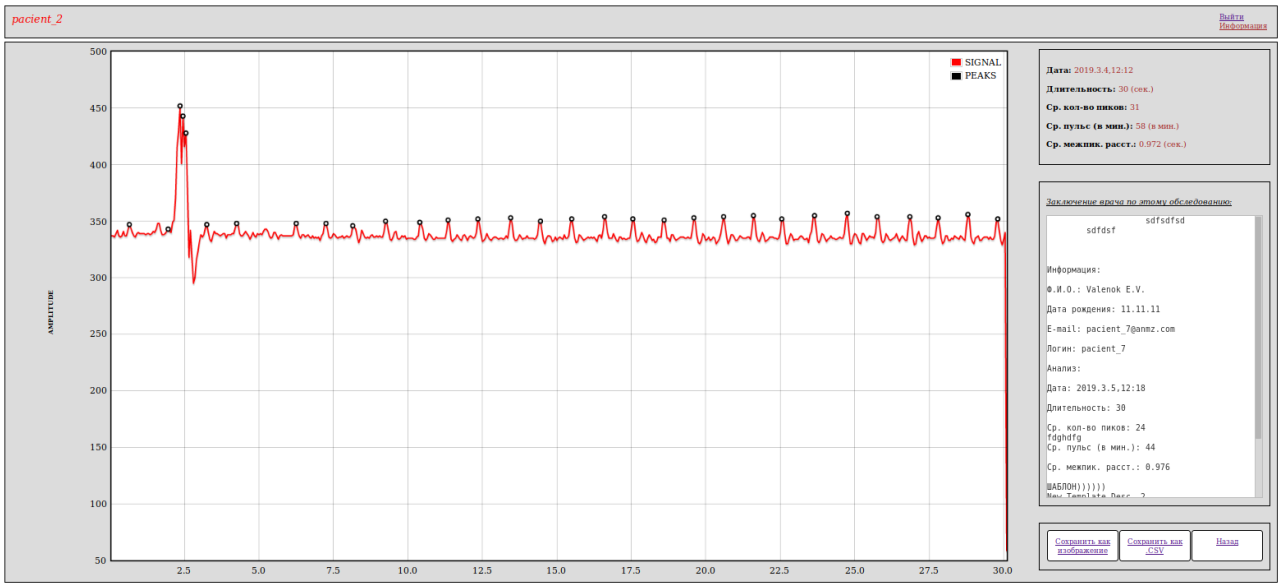


Рисунок 3.15 – Пользовательский интерфейс страницы просмотра результата мониторинга

Окно состоит из трех блоков: блок-просмотра сигнала (здесь, для интерактивного просмотра используется бесплатная JavaScript-библиотека FlotJS), блок данных о результате мониторинга и блок заключения диагноста.

Блок данных формируется из HTML-шаблона, по такому же принципу, как блок личной информации.

Для вычисления средних значений показателей используются PHP-подпрограммы, которые напрямую взаимодействуют с файлом сигнала.

Так, например, количество и расположение пиков сигнала вычисляется с помощью подпрограммы get_peaks, исходный код которой представлен в листинге Г.18. Данная подпрограмма принимает два аргумента: логин пользователя и имя файла мониторинга; далее происходит расчет пиков и их координат с помощью Python-скрипта, который находится на сервере. Данный скрипт возвращает значения в виде массива, который интерпретируется с помощью инструкции eval, и, далее записывается в переменную peaks_index [21]. Для определения пиков был выбран стандартный алгоритм поиска пиковых значений сигнала. Соответственно, для получения среднего значения пиков, достаточно лишь рассчитать размерность полученного массива peaks_index с помощью функции count [7].

Для вычисления частоты сердечных сокращений количество пиков в единицу времени умножается на два, т.к. временной интервал измерения составляет 30 секунд. Вычисление среднего межпикового расстояния производится с помощью подпрограммы get_peaks_length, которая, в качестве аргумента принимает массив peaks_index. Исходный код подпрограммы приведен в листинге Г.19. Данная подпрограмма рассчитывает среднюю разницу между двумя соседними элементами массива data, который содержит в себе индексы пиков. После расчета происходит округление до трех знаков после запятой с помощью функции round [8].

Блок с отображением заключения врача также реализован с помощью HTML и PHP. В тег textarea вносится значение, которое получается при выполнении подпрограммы get_last [16]. Подпрограмма принимает код диагноста и код пациента, затем производит поиск имени файла с заключениями по таблице с заключениями; считывает файл и возвращает считанные данные, а они, в свою очередь помещаются в блок textarea.

Кнопки «Сохранить как изображение» и «Сохранить как .CSV» реализованы с помощью HTML и JavaScript. При нажатии на первую кнопку

происходит вызов подпрограммы download_as_img, которая производит конвертацию графика сигнала в PNG-изображение. Исходный код приведен в листинге Г.20.

При нажатии на вторую кнопку происходит вызов подпрограммы download_as_csv, которая принимает массив значений сигнала dataset и записывает их в виде CSV-файла, вносит его в Blob-буфер [3], а затем инициализирует запрос на загрузку файла из буфера браузера. Исходный код представлен в листинге Г.21.

Для корректного отображения графика сигнала с помощью библиотеки FlotJS следует формировать JSON-массив значений сигнала и индексов пиков. Далее, из этого массива формируется главный массив dataset, который передается функции plot [3].

Данная функция принимает идентификатор тега, в котором будет производится отображение графика, и массив dataset. Скрипт отображения графика сигнала с отмеченными на нем пиками приведен в листинге Г.22.

Таким образом, личный кабинет пациента представляет собой многофункциональный пользовательский интерфейс, позволяющий постоянно держать связь с диагностом, просматривать заключения и результаты мониторинга.

Пользователь системы, имеющий роль диагноста, после успешной авторизации перенаправляется в личный кабинет диагноста, изображенный на рисунке 3.16.

Окно содержит два блока: блок-список пациентов и блок мгновенных сообщений, аналогичный соответствующему блоку в личном кабинете пациента.

Содержимое первого блока представляет собой список всех пациентов, закрепленных за текущим врачом-диагностом.

Слева отображается персональная информация пациента, а справа его текущий статус: в сети ли пользователь, подключен ли он к устройству и список его результатов мониторинга.

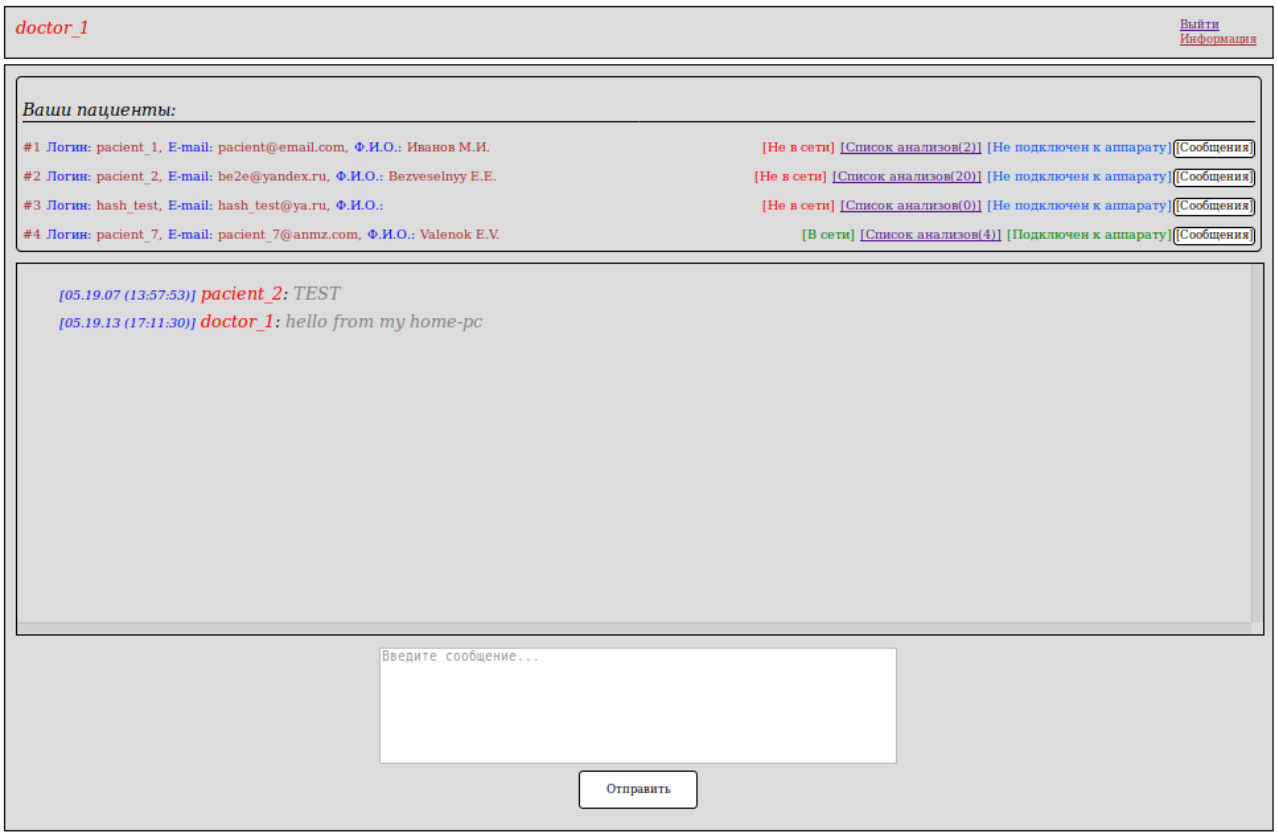


Рисунок 3.16 – Личный кабинет диагноста

При нажатии на кнопку «Сообщения» происходит обновление блока сообщений для отображения чата именно с выбранным пациентом. Структура и механизм работы данного интерфейса аналогичен такому же блоку в личном кабинете пациента, однако, нажав на ссылку «Список анализов», происходит перенаправление пользователя на другую страницу, с помощью которой диагност может просматривать результаты мониторинга и создавать заключение.

Интерфейс просмотра результата мониторинга от лица диагноста имеет вид, представленный на рисунке 3.17.

Окно состоит из четырех функциональных блоков: блок информации о пациенте, результате мониторинга и списка мониторингов; блок для написания заключения; блок для выбора и создания шаблона заключения, и блока для просмотра сигнала.

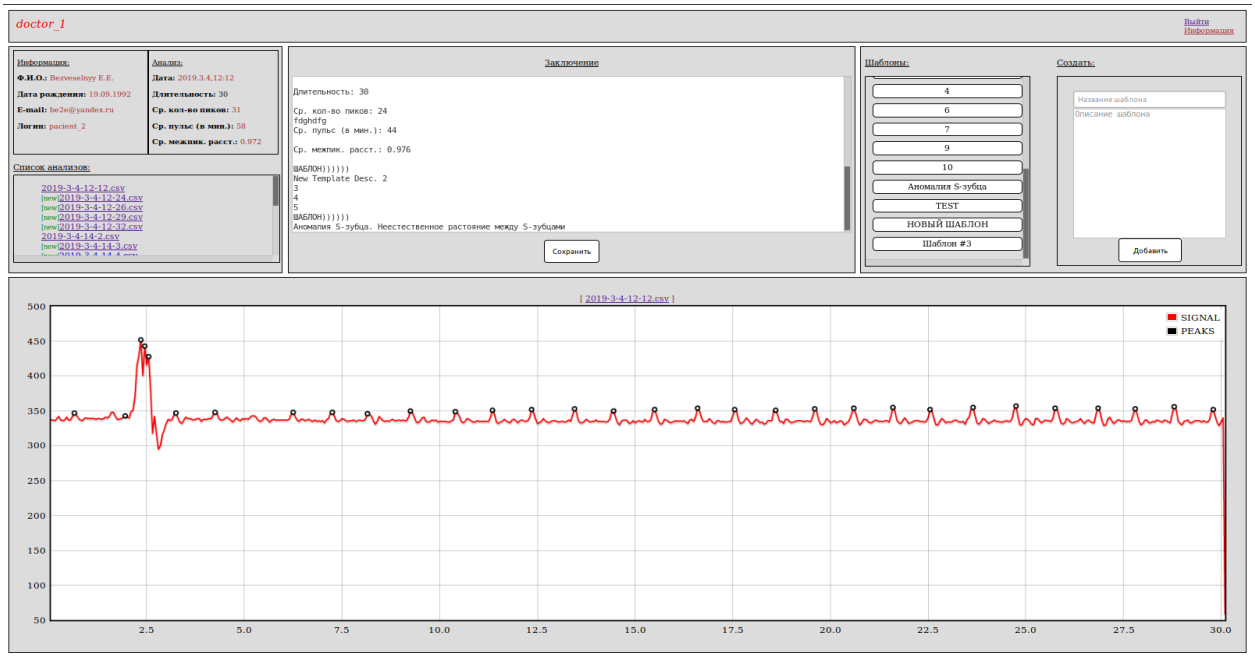


Рисунок 3.17 – Интерфейс просмотра результата мониторинга от лица врача-диагноста

Информация о пациенте, параметры сигнала и список мониторингов формируются таким же образом, как в личном кабинете пациента; отличие лишь в том, что при просмотре результатов мониторинга от лица пациента зеленые отметки ставятся рядом с теми результатами, к которым написано заключение, а в случае просмотра от лица врача-диагноста, процесс происходит полностью наоборот.

Блок для написания заключения состоит из одного текстового поля и кнопки «Сохранить», нажав на которую производится сохранение введенных данных в виде файла, который располагается в папке пациента. Каждый такой файл закреплен за одним файлом результата мониторинга. При нажатии на эту кнопку происходит обработка формы, которая посылает GET-запрос, приведенный листинге Г.23. После нажатия кнопки «Сохранить» происходит валидация формы на пустоту с помощью подпрограммы is_empty, исходный код которой представлен в листинге Г.24. Данная форма принимает три аргумента: DOM-элемент, содержащий в себе тэги, к которым следует применить проверку, сами названия тэгов tags и массив сообщений messages, которые выводятся в зависимости от статуса проверки [3].

Таким образом, если поле пусто, то выводится сообщение «Заполните поле заключения!», а если нет, то выводится модальное окно с подтверждением сохранения «Сохранить заключение?».

Блок выбора и создания шаблона работает подобным образом. В случае создания нового шаблона пользователю необходимо ввести короткое и емкое название для шаблона и его описание; после нажатия кнопки название шаблона появится в списке слева, и оно станет доступным для использования. Список шаблонов формируется посредством запроса в соответствующую таблицу базы данных, откуда берется название и описание шаблона. Нажав на конкретный элемент из списка, описание шаблона копируется в поле для ввода заключения. HTML-код блока со списком шаблонов имеет вид, приведенный в листинге Г.25. Копирование описания шаблона в поле для ввода заключения происходит с помощью подпрограммы add_template, написанной на JavaScript. Исходный код представлен на листинге Г.26.

Прямо над блоком просмотра сигнала располагается название текущего файла мониторинга, нажав на который происходит перенаправление пользователя на новую страницу, с помощью которой можно производить поэлементное редактирование сигнала, что позволяет избавится от некоторых явных помех или артефактов. Интерфейс редактирования сигнала имеет следующий вид, изображенный на рисунке 3.18.

Данный интерфейс состоит из двух блоков: блок, содержащий в себе значения амплитуды сигнала в единицу времени и блок просмотра изменений. Блок с элементами сигнала представляет собой массив тэгов input, значения которых является значением амплитуды сигнала. Каждое значение записано в поле для ввода и доступно для редактирования. После изменения происходит автообновление графика сигнала. Для сохранения изменений используются две кнопки: «Сохранить» и «Сохранить версию». В первом случае происходит сохранение изменений в файл оригинал, во втором, создается копия файла и потом в нее вносятся изменения. HTML-шаблон блока с значениями амплитуды имеет вид, приведенный в листинге Г.27.

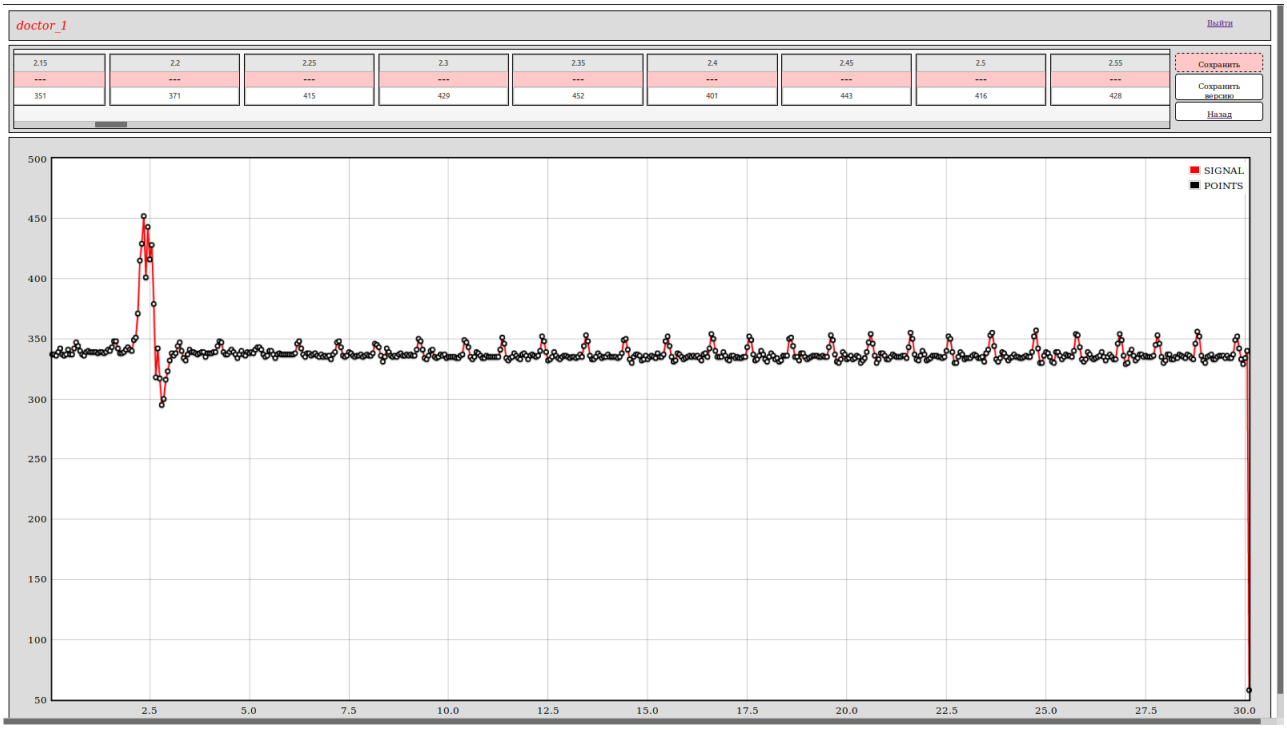


Рисунок 3.18 – Пользовательский интерфейс редактирования сигнала

За формирование списка полей со значениями амплитуды сигнала отвечает PHP-подпрограмма edit_points, которая формирует последовательность div-блоков, внутри которых и находятся поля для ввода input, уже заполненные значениями амплитуды. Исходный код подпрограммы имеет вид, приведенный в листинге Г.28. Подпрограмма формирует готовый HTML-шаблон, состоящий из массива блоков input; в значениях каждого блока содержится соответствующий показатель амплитуды сигнала, таким образом, количество блоков определяется количество точек графика сигнала. Шаблон для форматирования содержит переменная point_tag. Как видно из ее содержимого каждый div-блок, содержащий в себе два блока input (один из них отображает временную отметку), имеет два атрибута: onmouseout и onmouseover, которые отвечают за обработку события наведения курсора и выхода курсора за область блока соответственно [3]. При наведении курсора на блок происходит его выделение цветом и выделение соответствующей точки на графике сигнала. При выходе курсора за область, выделение пропадает [16]. За реализацию этих двух процессов отвечают подпрограммы highlight и

unhighlight соответственно. Они напрямую взаимодействуют с библиотекой FlotJS, компонуя в себе лишь встроенные в нее методы: setupGrid – задание сетки, draw – обновление графика и одноименное с подпрограммами методы включения и выключения подсветки [3]. Таким образом формируется каждый блок, а затем подпрограмма возвращает массив таких блоков и встраивает его форму.

После нажатия кнопки сохранения происходит отправка POST и GET-запроса одновременно, которые обрабатываются PHP-сценарием save_signal.php, исходный код которого представлен в листинге Г.29. POST-запрос содержит в себе значения амплитуды сигнала, а GET содержит лишь информацию о имени файла, и логине пациента, которому принадлежит файл. Сценарий выделяет из GET-запроса три параметра: идентификационный код пациента, которому принадлежит файл мониторинга, имя файла и режим сохранения (запись в оригинал или в новый файл). Если переменная mode равна значению «save_as», то запись производится в новый файл, и к существующему имени файла мониторинга добавляется префикс «[ed]», указывающий на то, что файл является копией с правками диагноста. В цикле из POST-запроса выделяются все значения сигнала, которые после записываются в соответствующий файл. Таким образом реализован пользовательский интерфейс редактирования сигнала, который позволяет исправлять некоторые погрешности результата мониторинга.

Связь между устройством съема и сервером реализуется с помощью трех PHP-сценариев: auth.php, out.php и upload.php, отвечающих за авторизацию, закрытие сессии и загрузку результата мониторинга на сервер. Для авторизации с помощью устройства, пользователь вводит логин и пароль, которые затем преобразуются в хеш-сумму, отправляемую на сервер. Этот запрос обрабатывает файл auth.php, исходный код которого представлен на листинге Г.30. Сценарий выделяет токен авторизации из GET-запроса, а затем производит запрос к таблице хешей, для того, чтобы определить валидность полученного токена. В положительном случае происходит обновление поля

статуса пользователя (is_connect), что отображается в личном кабинете врача-диагноста. Далее происходит вызов процедуры returnUserData, которая производит возврат JSON-массива с данными о пользователе. Таким образом, в случае успешной авторизации, устройство получает массив данных о пользователе, которые и выводят на дисплей устройства.

Файл out.php предназначается для процедуры деаутентификации пользователя. Суть его работы заключается в проверке токена и закрытии сессии авторизации и изменении статуса активности пользователя. Файл upload.php предназначен для сохранения полученного файла на сервер. Устройство отсылает POST-запрос, содержимое которого состоит из токена авторизации и зашифрованных данных результата мониторинга. Исходный код сценарии приведен в листинге Г.31. Скрипт проверяет полученный токен на валидность, и в положительном случае производит декодирование полученных данных с помощью алгоритма BASE64. После чего производит запись в соответствующий файл, имя которого формируется программным обеспечением самого устройства на основании даты и времени мониторинга.

Таким образом, веб-сервис для обеспечения удаленного доступа к результатам мониторинга состоит из трех условных интерфейсов:

– пользовательский интерфейс пациента, с помощью которого он может просматривать свои результаты, заключения, а также, общаться с врачом-диагностом;

– пользовательский интерфейс диагноста, с помощью которого диагност может общаться со своими пациентами, производить оценку результата мониторинга и его редактирование;

– интерфейс обмена данными между сервером и устройство, с помощью которого производится авторизация и отправка файла на сервер.

Вывод по третьему разделу.

Для разработки данного сервиса были использованы следующие инструменты: веб-сервер Apache, СУБД MySQL, язык программирования PHP (версия 7.0), язык программирования JavaScript (реализация различных

функций интерфейса, в том числе просмотр сигнала), язык программирования Python (версия 2.7, реализация расчета параметров сигнала), язык разметки гипертекста HTML и язык описания форм объектов CSS.

Основой всего проекта служит язык программирования PHP, с помощью которого были реализованы основополагающие функции веб-сервиса: регистрация и авторизация пользователей в системе, разделение на роли, реализация функционала пользовательских интерфейсов, хранение и обработка полученных данных, реализация доступа к данным, обмен данным между устройством и сервером.

3.4 SWOT-анализ системы дистанционного мониторинга сердечной деятельности

Был произведен SWOT-анализ системы, в ходе которого были выявлены все сильные, слабые стороны. Результаты представлены на таблице 3.1

Таблица 3.1 – SWOT-анализ

Сильные стороны	Возможности		Угрозы	Итого
	Расширение круга потребителей	Совершенствование разработки	Появление конкурентов	
Низкая стоимость	++	0	-	1
Многофункциональность	++	+	-	2
Простота использования	++	0	0	2
Простота настройки	++	++	0	4
Итого	8	3	-2	18
Слабые стороны				
Низкое финансирование	--	--	+	-3
Итого	-2	-2	1	-6
Общий итог	6	1	-1	12

Были получены результаты, характеризующие данную систему относительно слабых и сильных сторон, а также, относительно всех возможных угроз и потенциальных возможностей данной системы.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была достигнута ее главная цель: обеспечение возможности проведения удаленного мониторинга сердечной деятельности пациента.

Для достижения данной цели были выполнены следующие задачи:

а) обзор средств диагностики и мониторинга сердечной деятельности.

б) проектирование и разработка аппаратной части системы.

в) проектирование и разработка программного обеспечения для аппаратной части системы.

г) проектирование и разработка веб-сервиса системы.

В ходе решения первой задачи был произведен обзор существующих решений, на основании которого были учтены все недостатки и преимущества готовых систем дистанционного мониторинга.

В ходе выполнения второй задачи была спроектирована модульная схема аппаратной части системы, подобраны комплектующие в соответствии с разработанной схемой: датчик PulseSensor, отвечающий за сбор информации о сердечном ритме, микроконтроллер Arduino Leonardo, предназначенный для оцифровки, обработки, фильтрации сигнала, полученного с датчика, и микрокомпьютер Raspberry Pi 3, использующийся в качестве терминала устройства: реализует пользовательский интерфейс для взаимодействия человека с устройством съема, реализует обмен данными между сервером и устройством для авторизации и передачи результата работы датчика.

В ходе проектирования и разработки программного обеспечения на языке программирования C++ был написан драйвер для микроконтроллера Arduino Leonardo, позволяющий проводить фильтрацию, преобразование и обмен данными с микрокомпьютером Raspberry Pi 3.

Также, на языке программирования Python было разработано соответствующее программное обеспечение для микрокомпьютера, с помощью

которого был реализован графический пользовательский интерфейс взаимодействия человека с устройством. Помимо этого, был разработан и реализован программный драйвер, для взаимодействия микрокомпьютера с микроконтроллером. Как и пользовательское приложение, драйвер был написан на языке программирования Python.

В ходе выполнения последней задачи был настроен компьютер, благодаря чему он стал выступать в роли сервера: было установлено серверное программное обеспечение Apache2, PHP-интерпретатор и СУБД MySQL. Была спроектированы логическая и физическая модели база данных. Был разработан веб-сервис, позволяющий производить дистанционный доступ пользователя к результатам мониторинга, составления заключения, обмениваться мгновенными сообщениями между пациентом и врачом-диагностом.

Были разработаны три интерфейса:

– личный кабинет пациента, с помощью которого можно просматривать и скачивать свои результаты мониторинга и заключения к ним, можно производить общение с диагностом;

– личный кабинет диагноста, позволяющий производить оценку результатов мониторинга пациентов, написание заключения, шаблона заключения и редактирование полученного сигнала;

– интерфейс обмена данными, реализующий сообщение между устройством и самим сервером, с помощью чего производится аутентификация пользователя и отправка результата мониторинга.

Таким образом, цель выпускной квалификационной работы была достигнута: разработанная система обеспечивает удаленный доступ к физиологическим данным пациента, преимущества которой заключаются в низкой стоимости и повышенном уровне интегрируемости в существующие сети и системы.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Абросимова, М. А. Информационные технологии в государственном и муниципальном управлении: Учебное пособие / М.А. Абросимова. – М.: КноРУС, 2013. – 248 с.

2. Астапенко, В.А. Фотоэлектроника. Часть 1. Прикладная электроника / В. А. Алехин. – М.: Янус-К, 2010. – 654 с.

3. Атенсио, Л Функциональное программирование на JavaScript: как улучшить код JavaScript-программ / Л Атенсио. – М.: Диалектика, 2018. – 304 с.

4. Барыбин, А.А. Электроника и микроэлектроника. Физико-технологические основы / А.А. Барыбин. – М.: ФИЗМИЛИТ, 2006. – 424 с.

5. Васильев, А. Е. Микроконтроллеры. Разработка встраиваемых приложений / А. Е. Васильев. – СПб.: BHV, 2008. – 304 с.

6. Веллинг, Л. Разработка Web-приложений с помощью PHP и MySQL / Л. Веллинг, Л. Томсон. – М.: Вильямс, 2013. – 848 с.

7. Веллинг, Л. Разработка Web-приложений с помощью PHP и MySQL / Л. Веллинг. – М.: Вильямс, 2006. – 880 с.

8. Заяц, А.М. Проектирование и разработка WEB-приложений. Введение в frontend и backend разработку на JavaScript и node.js: Учебное пособие / А.М. Заяц, Н.П. Васильев. – СПб.: Лань, 2019. – 120 с.

9. Казаков, В.Н. Телемедицина / В.Н. Казаков, В.Г. Климовецкий, А.В. Владзимирский. – Донецк: Типография ООО «Норд», 2002. – 100 с

10. Казаков, В.Н., Климовецкий В.Г., Владзимирский А.В. Телемедицина. – Донецк: Типография ООО «Норд», 2002. – 100 с.

11. Кениг, Э. Эффективное программирование на C++. Практическое программирование на примерах. Т. 2 / Э. Кениг, Б. Э. Му. – М.: Вильямс, 2016. – 368.

12. Колисниченко, Д. Н. Разработка Linux-приложений. / Д. Н. Колисниченко. – СПб.: BHV, 2012. – 432 с.

13. Колисниченко, Д.Н. PHP и MySQL. Разработка Web-приложений / Д.Н. Колисниченко. – СПб.: БХВ-Петербург, 2013. – 560 с.

14. Логвинов, В. В. Схемотехника телекоммуникационных устройств, радиоприемные устройства систем мобил. И стацион. Радиосвязи, теория электрических цепей / В. В. Логвинов и др. – М.: Солон-пресс, 2013. – 656 с.

15. Лукин, В. Н. Введение в проектирование баз данных / В. Н. Лукин. – М.: Вузовская книга, 2015. – 144 с.

16. Мандел Т. Разработка пользовательского интерфейса = The Elements of User Interface Design; учеб.пособие / Т. Мандел ; пер. с англ. – М. : «ДМК Пресс». – 416 с.: ил. (Серия «Для программистов»).

17. Назаренко, Г.И., Гулиев Я.И., Ермаков Д.Е. Медицинские информационные системы. Теория и практика. – Москва: ФИЗМАТЛИТ, 2005. – 302 с.

18. Опадчий Ю. Ф. Аналоговая и цифровая электроника: Полный курс: учебник для вузов / Ю. Ф. Опадчий, А. И. Гуров; ред. О. П. Глудкин. – М.: Горячая линия-Телеком, 2000. – 768 с.

19. Преснякова, Г. В. Проектирование интегрированных реляционных баз данных: Учебное пособие / Г. В. Преснякова. – М.: КДУ, 2007. – 224 с.

20. Программирование на C для начинающих / М. МакГрат. – М.: Эксмо, 2015. – 192 с.

21. Россум, Г. Язык программирования Python = The Python Tutorial / Г. Россум, Ф. Л. Дж. Дрейк, Д. С. Откидач.; Пер. с англ. Д. С. Откидач. – М.: 2001, – 454 с.: ил.

22. Свейгарт Эл. Автоматизация рутинных задач с помощью Python = Automate the Boring Stuff with Python.; Пер. С англ. А. М. Жиллиман. – М.: Вильямс, 2016, – 592 с.: ил.

23. Столлингс, В. Компьютерные сети, протоколы и технологии Интернета / В. Столлингс. СПб.: BHV, 2005. – 832 с.

24. Хоменко А. Д. Базы данных: Учебник для высших учебных заведений: учеб.пособие / А. Д. Хоменко, В. М. Цыганков, Мальцев М. Г. ; под ред. А. Д. Хоменко. – 6-е изд. Доп. – СПб. : «КОРОНА-Век», 2009. – 736 с.

25. Хьюлсман Л. П. Введение в расчет активных фильтров: пер. с англ. / Л. П. Хьюлсман, Ф. Е. Ален; пер. с англ. Н. Н. Слепцова; под ред. А. Е. Знаменского. – М.: Радио и связь, 1984. – 384 с.

26. Шилдт, Герберт. C++ : базовый курс = C++ from the GROUND EDITION, 3-е издание.: учеб.пособие Шилдт Герберт ; пер. с англ. Н. М. Ручко, под ред. Н. М. Ручко. – М. : Издательский дома «Вильямс», 2010. – 624 с.

27. Эккель Б. Философия C++. Введение в стандартный C++. 2-е изд. = Thinking in C++. Second Edition. Volume One: Introduction to Standard C++: учеб. пособие Б. Эккель ; пер. С англ. – СПб. : «Питер», 2004. – 572 с.: ил.

ПРИЛОЖЕНИЕ А

Структурная схема взаимодействия регистраторов. Структура микроконтроллера Arduino Leonardo

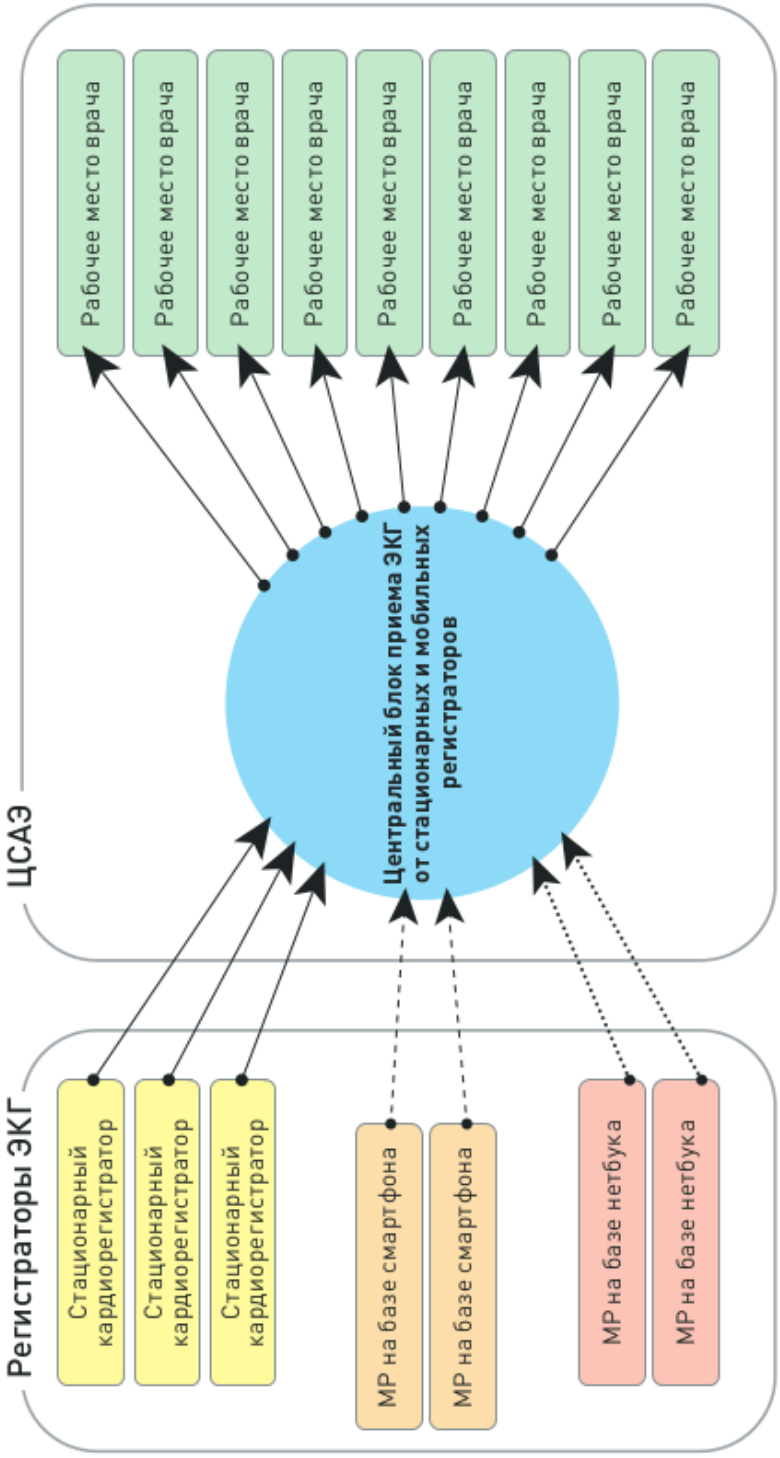


Рисунок А.1 – Логическая схема взаимодействия дистанционных и стационарных регистраторов

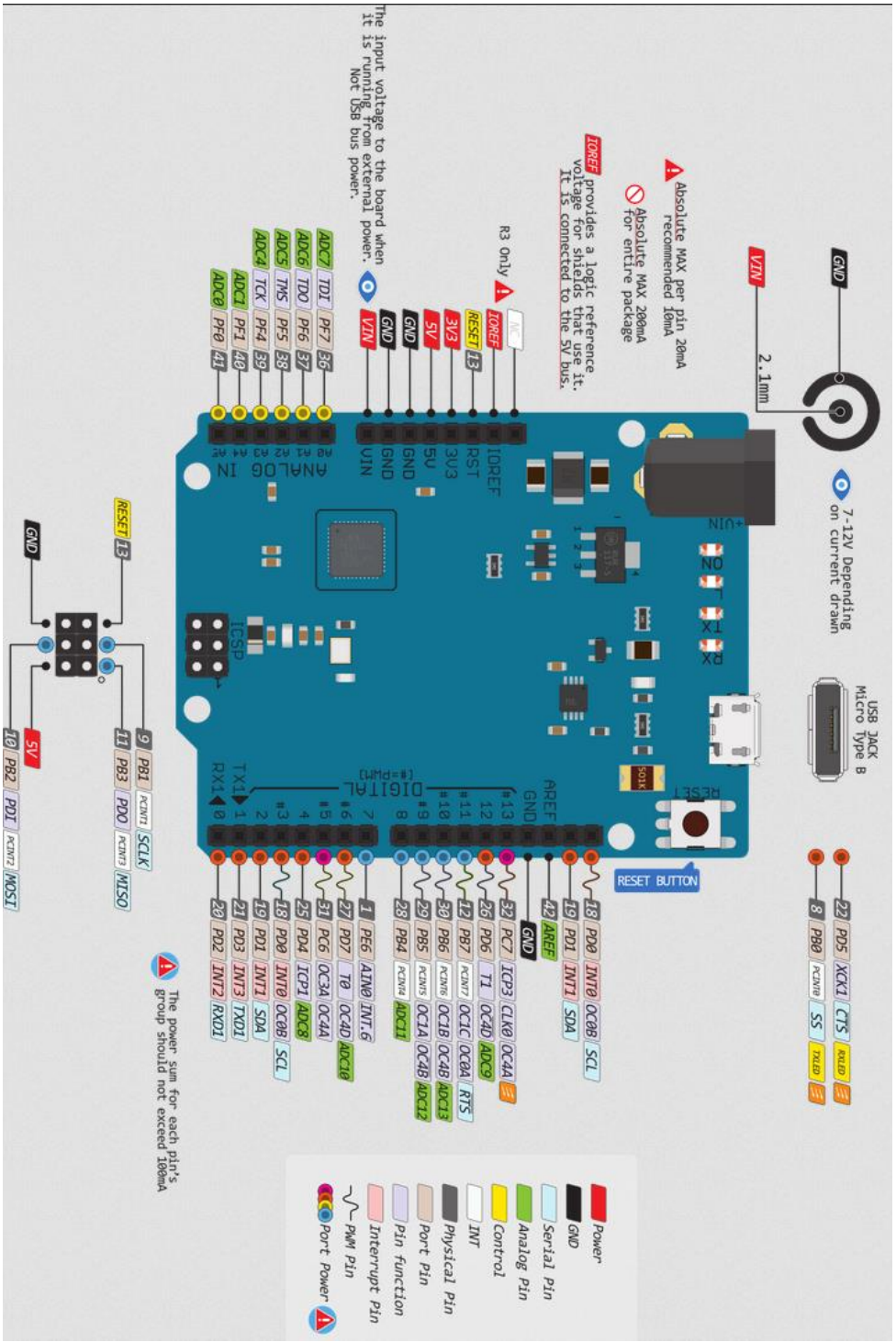


Рисунок А.2 –Назначение портов и контактов контроллера Arduino Leonardo

ПРИЛОЖЕНИЕ Б

Алгоритмы работы программного обеспечения микроконтроллера

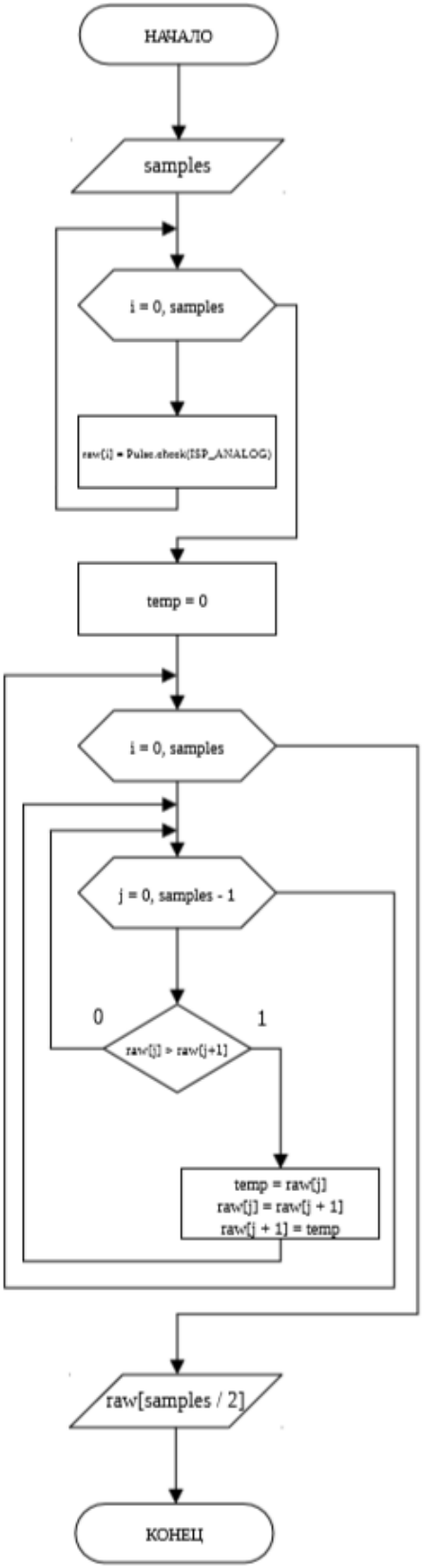


Рисунок Б.1 – Графическая блок-схема подпрограммы медианного фильтра

Листинг Б.1 – Исходный код подпрограммы medianFilter

```
int medianFilter (int samples){
  int raw[samples];
  for (_nti  = 0; i < samples; i++) raw[i] = Pulse.check(ISP_ANALOG);
  int temp = 0;
  for (_nti  = 0; i < samples; i++){
    for (int j = 0; j < samples – 1; j++){
      if (raw[j] > raw[j + 1]){
        temp = raw[j];
        raw[j] = raw[j + 1];
        raw[j + 1] = temp;
      }
    }
  }
  return raw[samples / 2];
}
```

Листинг Б.2 – Исходный код подпрограммы setup

```
void setup() {
  Serial.begin(9600);
  Pulse.begin();
}
```

Листинг Б.3 – Объявление специальных констант и объектов

```
iarduino_SensorPulse Pulse(A0);
int delay_ms = 50;
unsigned FILTER_SAMPLES = 16;
String serial_command;
```

Листинг Б.4 – Исходный код подпрограммы loop

```
void loop() {
  if (Serial.available() > 0) {
    serial_command = str_serial();
    if (serial_command == “start”) getAnalogData();
  }
}
```

Листинг Б.5 – Исходный код подпрограммы str_serial

```
String str_serial() {
  // return the string from serial-port
  String serial_buffer;
  while (Serial.available() > 0) {
    serial_buffer += (char)Serial.read();
  }
  return serial_buffer;
}
```

Листинг Б.6 – Исходный код подпрограммы getConnection

```
bool getConnection() {
  while (Pulse.check(ISP_VALID) != ISP_CONNECTED) {}
  return true;
}
```

Листинг Б.7 – Исходный код подпрограммы getAnalogData

```
int getAnalogData() {
  while (1) {
    if (getConnection()) {
      delay(delay_ms);
      Serial.println(medianFilter(FILTER_SAMPLES));
      serial_command = str_serial();
      if (serial_command == "stop") {
        Serial.println(Pulse.check(ISP_PULSE));
        break;
      }
    }
  }
}
```

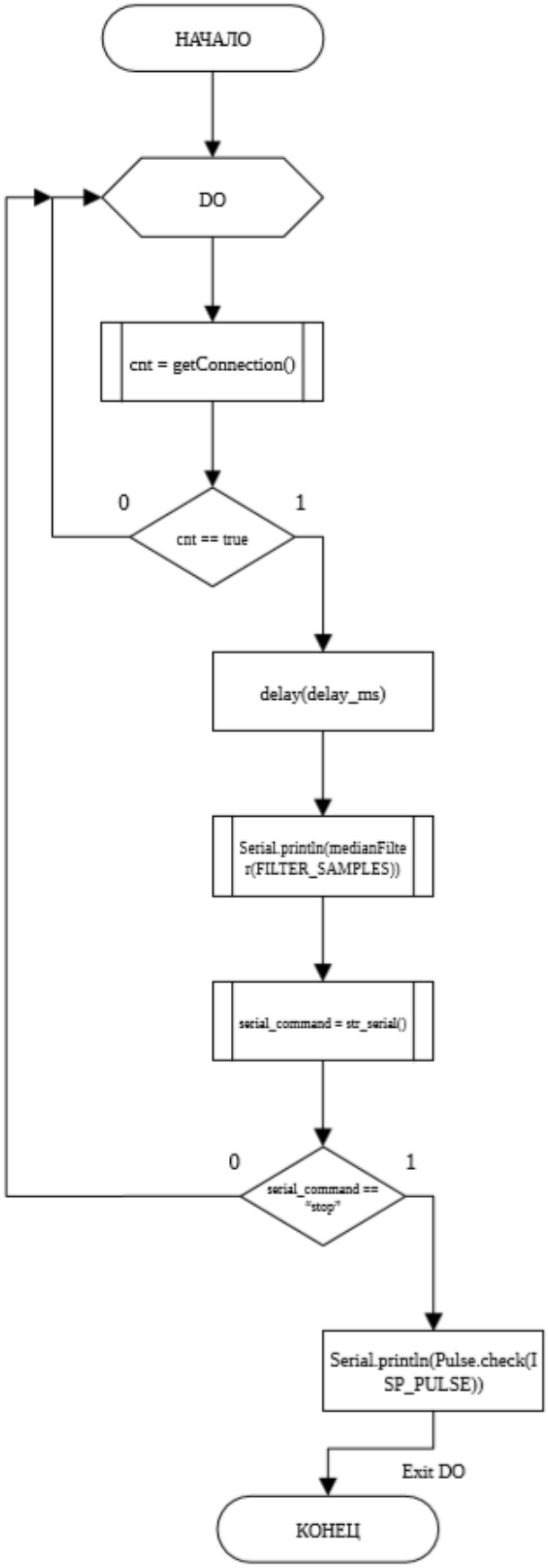


Рисунок Б.2 – Графическая блок-схема алгоритма работы функции getAnalogData

ПРИЛОЖЕНИЕ В

Настройка работы микрокомпьютера. Исходный код программного обеспечения микрокомпьютера

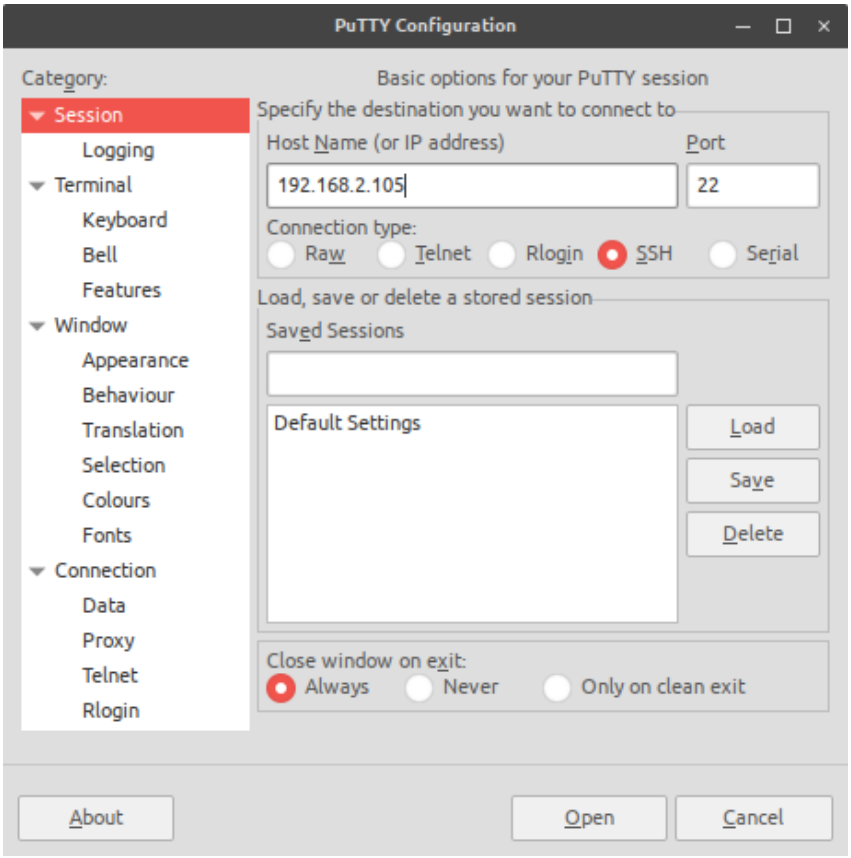


Рисунок В.1 – Подключение к Raspberry Pi 3 с помощью приложения Putty

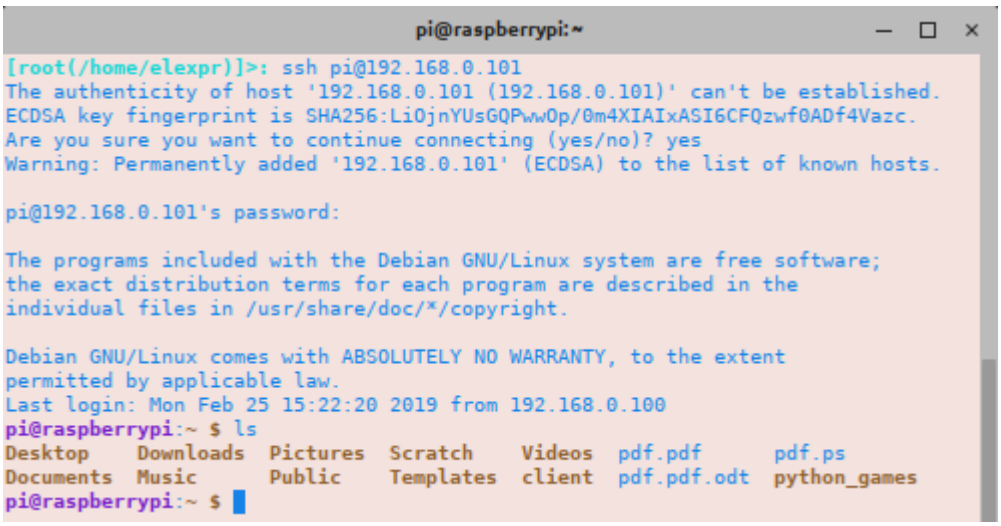


Рисунок В.2 – Подключение с помощью консольной утилиты ssh

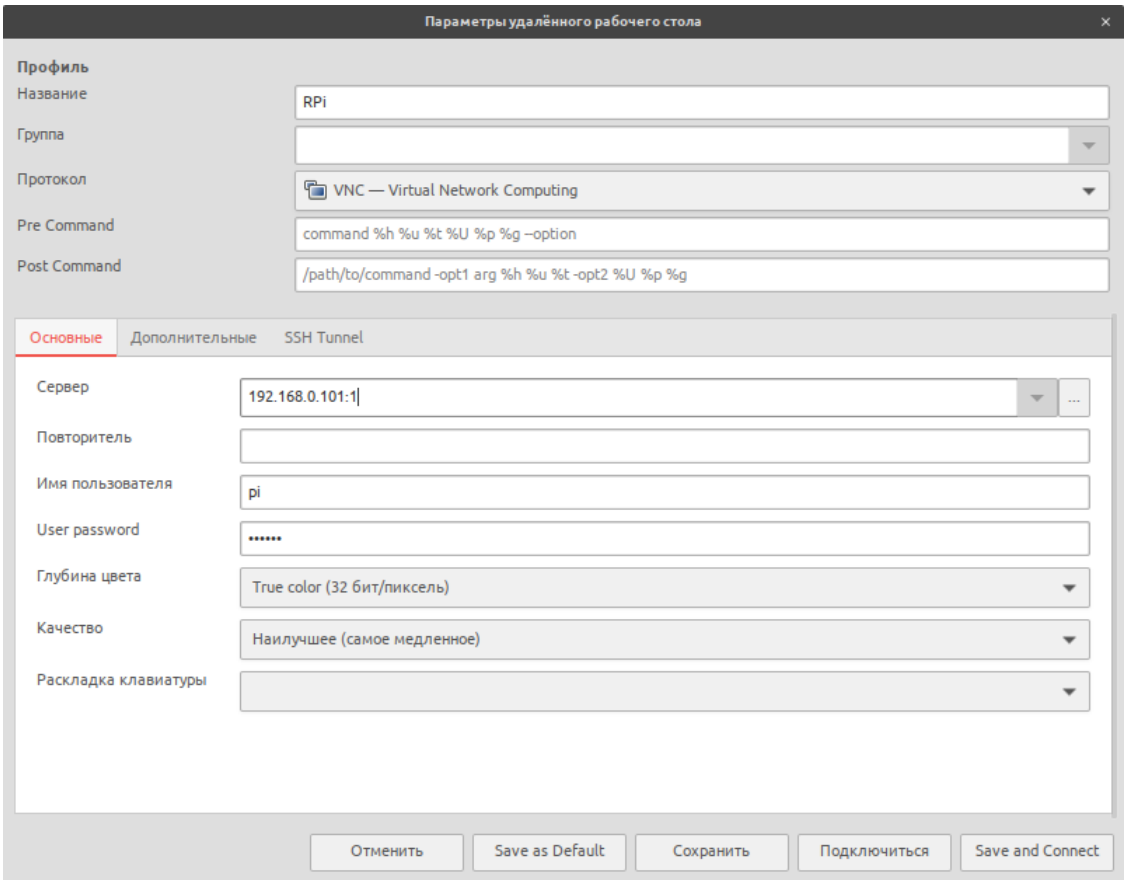


Рисунок В.3 – Настройка подключения к удаленному рабочему столу

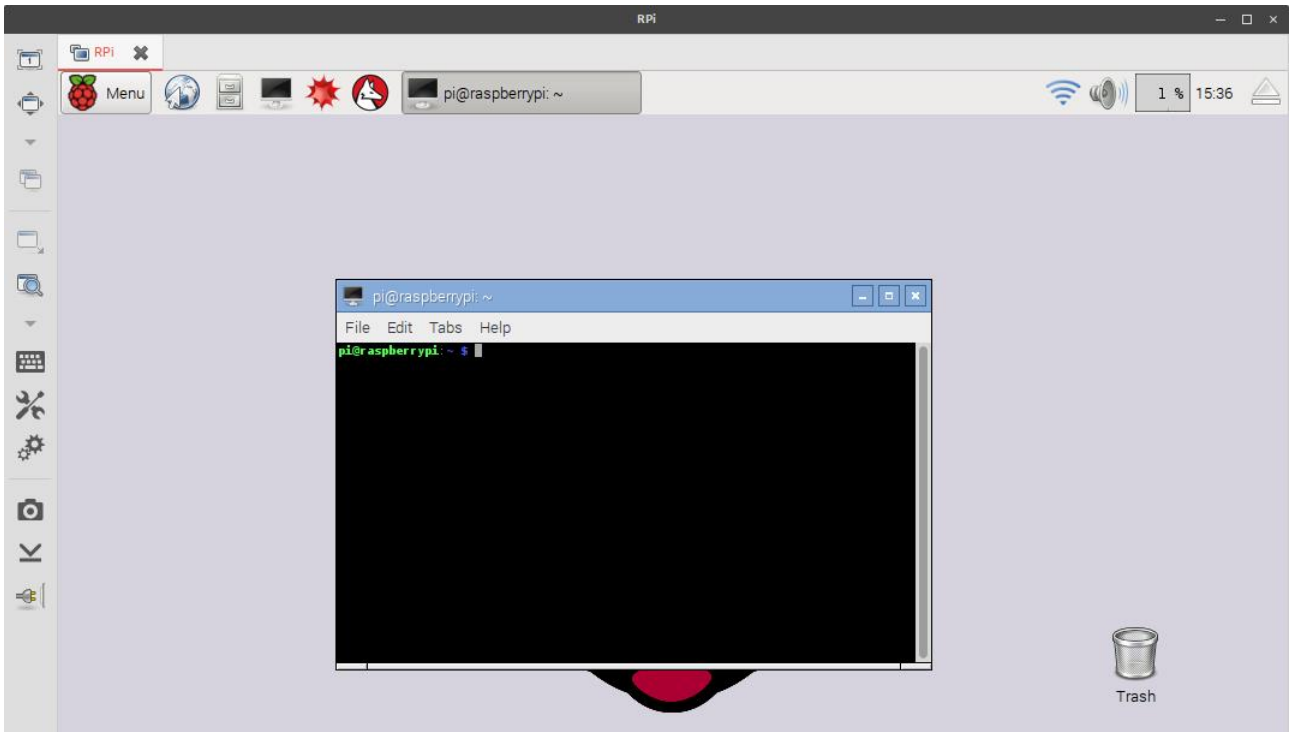


Рисунок В.4 – Графическое окружение рабочего стола Rasbperry Pi 3

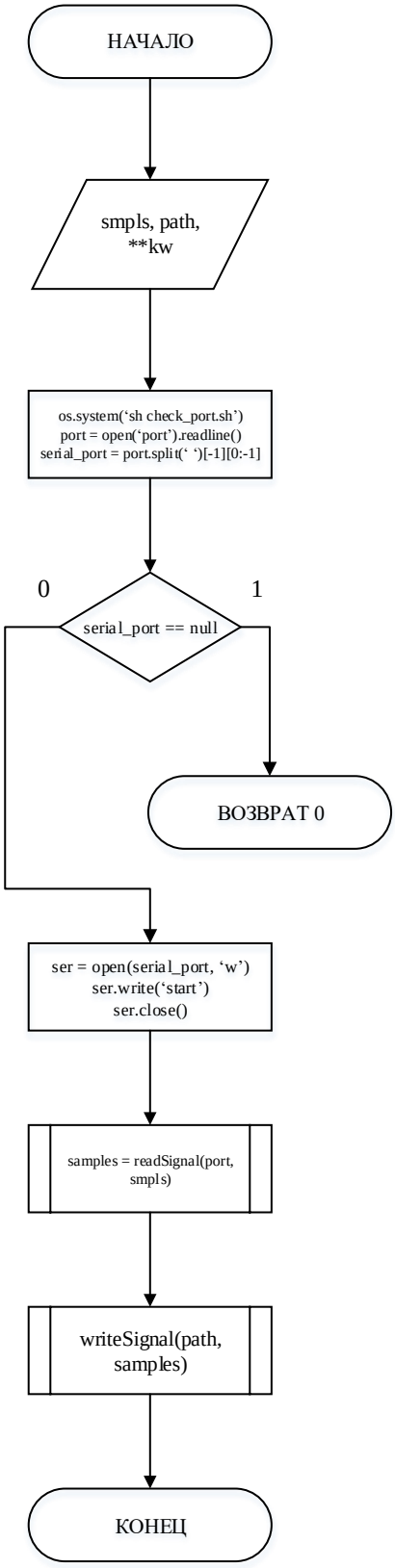


Рисунок В.5 – Графическая блок-схема главной подпрограммы драйвера микроконтроллера

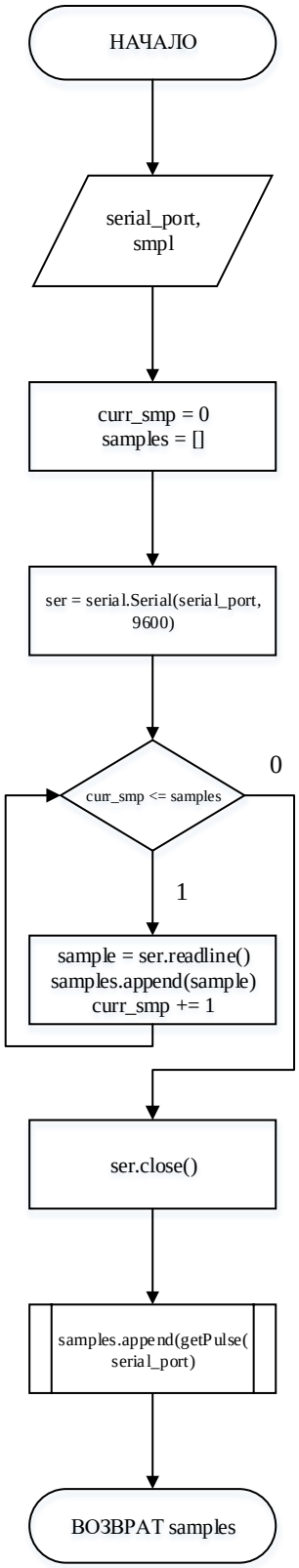


Рисунок В.6 – Графическая блок-схема алгоритма работы подпрограммы readSignal

Листинг В.1 – Конфигурация для автоматического подключения к беспроводной сети

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
ssid="redrum"
psk="********"}
```

Листинг В.2 – Скрипт поиска буферного файла последовательного порта

```
ls –l /dev/ttyACM* | grep dialout > ./distant_ecg/tools/port
```

Листинг В.3 – Исходный код главной подпрограммы драйвера

```
def main(SAMPLES, FILEPATH, **kwargs):
  os.system('sh ./distant_ecg/tools/check_port.sh')
  port = open('./distant_ecg/tools/port').readline()
  SERIAL_PORT = port.split(' ')[-1][0: -1]
  if not SERIAL_PORT: return 0
  ser = open(SERIAL_PORT, 'w')
  ser.write(u'start')
  ser.close()
  samples   =   readSignal(SERIAL_PORT,   SAMPLES,   kwargs.get('status_label'),
  kwargs.get('parent_frame'))
  writeSignal(FILEPATH, samples)
  return samples[-1]
```

Листинг В.4 – Исходный код подпрограммы getPulse

```
def getPulse(SERIAL_PORT):
  ser = serial.Serial(SERIAL_PORT, 9600)
  ser.write('stop')
  ser.readline()
  pulse = ser.readline()
  return pulse
```

Листинг В.5 – Исходный код подпрограммы readSignal

```
def readSignal(SERIAL_PORT, SAMPLES, STATUS_LABEL, PARENT_FRAME):
  '''Read signal and return list of values of signal'''
  current_sample = 0; samples = []; signals = []
  ser = serial.Serial(SERIAL_PORT, 9600)
  while (current_sample <= SAMPLES):
    STATUS_LABEL.configure(text = 'Progress: {PERCENT} / 100%'.format(PERCENT =
int(current_sample / float(SAMPLES) * 100)))
    sample = ser.readline()
    if (sample != '0\n' and sample != '\n'):
      samples.append(sample)
      current_sample += 1
    PARENT_FRAME.update()
  ser.close()
```

Листинг В.6 – Подпрограмма аутентификация пользователя

```
def sign_in(self, login, password):
      MD5 = md5.new(); MD5.update(login + password)
      self.AUTH_TOKEN = MD5.hexdigest()
      server_response=requests.get(self.SIGN_IN_ADDR.format(auth_token=
self.AUTH_TOKEN))
      JSON.update(server_response.json())
      if JSON.get('status') == 200: return True
      else: return False
```

Листинг В.7 – Исходный код конструктора класса Auth

```
def __init__(self):
  self.HOST = 'http://localhost'
  self.SIGN_IN_ADDR = self.HOST + "/auth/auth.php?auth_token={auth_token}"
  self.SIGN_OUT_ADDR = self.HOST + "/auth/out.php?auth_token={auth_token}"
  self.UPLOAD_DATA_ADDR = self.HOST + "/auth/upload.php?auth_token={auth_token}"
  self.AUTH_TOKEN = None
```

Листинг В.8 – Исходный код подпрограммы sign_out

```
def sign_out(self):
      server_response = requests.get(self.SIGN_OUT_ADDR.format(auth_token = self
.AUTH_TOKEN))
```

Листинг В.9 – Исходный код подпрограммы send_ecg_data

```
def send_ecg_data(self, path):
    ecg_data = file(path, 'r')
    b64_ecg = base64.b64encode(ecg_data.read())
    ecg_data.close()
    tuple_time = time.localtime()
    filename='{YEAR}-{MONTH}-{DAY}-{H}-{M}.csv'.format(YEAR                    =
tuple_time.tm_year,
                                                       MONTH                   =
tuple_time.tm_mon,
                                                       DAY = tuple_time.tm_mday,
                                                       H = tuple_time.tm_hour,
                                                       M = tuple_time.tm_min)

    POST = {'user_hash': self.AUTH_TOKEN,
            'path': JSON['path'],
            'filename': filename,
            'data': b64_ecg}

    server_response=requests.post(self.UPLOAD_DATA_ADDR.format(
                                              auth_token= self.AUTH_TOKEN),
                                              data = POST)
    if server_response.json().get('status') == 200: return True
    else: return False
```

Листинг В.10 – Объявление класса Client

```
class Client(Frame):
  '''User-Interface class'''
  LOGIN, PASSWORD = None, None
  ECG_DATA_PATH = './ecg/signal.csv'
  auth = Auth()
```

```
  def __init__(self, parent):
    Frame.__init__(self, parent, background="white")
    self.parent = parent
    self.child = ttk.Frame(self.parent); self.child.pack(fill = BOTH, expand =
1)
    self.initUI(self.initAuthUI)
```

Листинг В.11 – Исходный код метода initUI

```
def initUI(self, content):
    '''Construct User-Interface'''
    self.clearFrame(self.child)
    content()
```

Листинг В.12 – Исходный код подпрограммы initAuthUI

```
def initAuthUI(self):
      '''Construct auth interface'''
      authUI = ttk.Frame(self.child)
      authUI.place(relx = 0.5, rely = 0.45, anchor = CENTER)

      labelFrame = LabelFrame(authUI, padx = 16, pady = 16, text = "Sign in");
      labelFrame.pack()

      login_label = ttk.Label(labelFrame, text = "LOGIN")
      login_entry = Entry(labelFrame)
      login_label.pack()
      login_entry.pack()

      password_label = ttk.Label(labelFrame, text = "PASSWORD")
      password_entry = Entry(labelFrame)
      password_label.pack()
      password_entry.pack()

      sign_in_button = Button(labelFrame, text = "SIGN IN")
      sign_in_button.bind("<Button-1>", lambda event, login=login_entry,
            password = password_entry:
            self.receiv_to_auth(event,login=login,password= password))
      sign_in_button.pack()
      return authUI
```

Листинг В.13 – Исходный код подпрограммы receiv_to_auth

```
  def receiv_to_auth(self, event, **kwargs):
    '''Receiv login and password to sign-method from auth-class'''
    login, password = kwargs.get('login').get(), kwargs.get('password').get()
    is_auth = self.auth.sign_in(login, password)
    if not is_auth:
      tkMessageBox.showerror("Error", "Login and password is not correct!")
    else:
      self.LOGIN, self.PASSWORD = login, password
      tkMessageBox.showinfo("Successfull", "You are sign in now!")
      self.initUI(self.initLoginUI)
```

Листинг В.14 – Исходный код подпрограммы initLoginUI

```
def initLoginUI(self):
  '''Construct Login interface'''
  def initUserInfoUI(parent): <…>

  def initCardioCanvas(parent):
    '''Construct frame with canvas where will be displayed the ECG-data'''
    def make_grid(canvas): <…>
    cardio_frame = LabelFrame(parent, text = "Сигнал", font = 'Courier 8')
    cardio_frame.grid(column=1,sticky = 'ws',padx = 2)
    self.cardio_canvas = Canvas(cardio_frame, width=440, height=120,
                                bg= 'white');
    self.cardio_canvas.pack()
    make_grid(self.cardio_canvas)
    return cardio_frame

  def initControlPanel(parent): <…>

  def initStatusBar(parent): <…>

  loginUI = ttk.Frame(self.child); loginUI.pack(fill = BOTH, expand = 1)
  initUserInfoUI(loginUI); initCardioCanvas(loginUI)
  initStatusBar(loginUI); initControlPanel(loginUI)
  return loginUI
```

Листинг В.15 – Исходный код подпрограммы receiv_to_sending_data

```
def receiv_to_sending_data(self, event):
    is_ok = self.auth.send_ecg_data(self.ECG_DATA_PATH)
    if not is_ok:
      tkMessageBox.showerror('Ошибка', 'Что-то пошло не так…')
    else: tkMessageBox.showinfo("Успешно", "Данные успешно отправлены!")
```

Листинг В.16 – Исходный код подпрограммы scan_ecg

```
def scan_ecg(self, event):
  def make_grid(canvase): <…>

  def change_status(button_status): <…>

  def preview_plot(): <…>
  change_status(DISABLED)
  device_is_ok = distant_ecg.read_ecg.main(600, self.ECG_DATA_PATH,
                                           status_label = self.status,
                                           parent_frame = self.status_bar)
  if not device_is_ok:
    tkMessageBox.showerror('Ошибка', 'Устройство не подключено!')
    self.status.configure(text = 'Устройство не подключено!')
  else:
    tkMessageBox.showinfo('Успешно', 'Сканирование окончено!')
    self.status.configure(text = 'Сканирование окончено!')
  change_status(NORMAL)
  if device_is_ok: preview_plot()
```

ПРИЛОЖЕНИЕ Г

Исходный код веб-сервиса системы

Листинг Г.1 – Установка веб-сервера Apache

```
sudo apt update
sudo apt upgrade
sudo apt install apache2
sudo systemctl enable apache2
```

Листинг Г.2 – Конфигурации сервера в файле.htaccess

```
Options –Indexes
php_value max_input_vars 2000

php_value display_errors 1
php_value display_startup_errors 1
php_value error_reporting E_ALL
```

Листинг Г.3 – Создание таблицы «pacients»

```
CREATE TABLE PACIENTS (
  pacient_id int NOT NULL AUTO_INCREMENT,
  doctor_id int,
  p_login varchar(32),
  p_pass varchar(32),
  p_initials varchar(56),
  p_email varchar(56),
  ecg_server_path varchaer(256),
  PRIMARY KEY (pacient_id));
```

Листинг Г.4 – HTML-код главной страницы

```
<form name=”login_form” method=”POST” action=”./login-register/login.php”>
  <input type=”text” name=”login” placeholder=”login”/>
  <input type=”password” name=”pass” placeholder=”password”/>
  <input type=”submit” value=”Login” name=”submit”/>
</form>
<form name=”register_form” method=”POST” action=”./login-register/register.php”>
  <input type=”text” name=”login” placeholder=”login” />
  <input type=”password” name=”pass” placeholder=”password” />
  <input type=”text” name=”email” placeholder=”e-mail” />
  <input type=”submit” value=”Register” name=”submit” />
</form>
```

Листинг Г.5 – Исходный код сценария login.php

```
include “/var/www/php/db/connect.php”;
include “/var/www/php/lib/lr-lib.php”;
$inp_login = $_POST[‘login’]; $inp_pass = $_POST[‘pass’];
if (check_user($inp_login, $inp_pass, $mysqli, ‘doctors’)) {
  make_session($inp_login, ‘doctors’);
  $mysqli -> close();
  header(‘Location: ../../pages/doctor/doctor.php’);
  exit;
```

```
}
if (check_user($inp_login, $inp_pass, $mysqli, 'pacients')) {
  make_session($inp_login, 'pacients');
  $mysqli -> close();
  header('Location: ../../pages/pacient/pacient.php');
  exit;
}
```

Листинг Г.6 – Исходный код сценария register.php

```
include "/var/www/php/db/connect.php";
include "/var/www/php/lib/lr-lib.php";
$inp_login = $_POST['login'];
$inp_pass = $_POST['pass'];
if (check_user($inp_login, $inp_pass, $mysqli, 'doctors') ||
    check_user($inp_login, $inp_pass, $mysqli, 'pacients')) {
  header('Location: ../../pages/oops/username_is_taken.php');
  exit;
} else {
  new_user($inp_login, $inp_pass, $_POST['email'], $mysqli, 'pacients');
  include "/var/www/php/lib/update_hash.php";
  header('Location: ../../pages/oops/register_done.php');
  exit;
}
```

Листинг Г.7 – Исходный код блока личной информации

```
<div class="item-pacient-info">
  <p class="label">Ваш лечащий врач:</p>
  <div class="your-doctor">
    <?php get_doctor($_SESSION['pacients']) ?>
    <p align="center"><a href="">Подробнее...</a></p>
    <div style="border-top: 1px solid; margin-bottom: 10px"></div>
    <div class="items">
      <a href="./import.php?mode=all-data&auth_token=
        <?php echo get_auth_token($_SESSION['pacients'])?>" target="_blank">
        <div lass="button" style="height:35px;flex-grow:1">
          Импорт данных</div>
      </a>
      <a href="./import.php?mode=diagnoses-only&auth_token=
         <?php echo get_auth_token($_SESSION['pacients']) ?>"target="_blank">
         <div class="button" style="height: 35px; flex-grow: 1">
           Импорт анализов
         </div>
</a></div></div></div>
```

Листинг Г.8 – Исходный код подпрограммы get_doctor

```
function get_doctor($p_l) {
    include "/var/www/php/db/connect.php";
    $d_id = $mysqli -> query("SELECT doctor_id FROM pacients WHERE p_login =
'$p_l'")
                    -> fetch_array()['doctor_id'];
    $doctor_fields = $mysqli -> query("SELECT d_email, d_initials, d_birthday
                                       FROM doctors WHERE doctor_id = $d_id");
    if (!empty($doctor_fields)) {
      $doctor_fields = $doctor_fields -> fetch_array();
      echo '<p><b>Инициалы:</b><span class="brown-color">' .
           $doctor_fields['d_initials'] . "</span></p>";
      echo '<p><b>E-mail:</b> <span class="brown-color">' .
```

```
            $doctor_fields['d_email'] . "</span></p>";
        echo '<p><b>Дата рождения:</b> <span class="brown-color">' .
            $doctor_fields['d_birthday'] . "</span></p>";
      } else {
        echo «<p class='empty-yet'>Вам пока еще не назначен доктор!</p>»;
      }
      $mysqli -> close();
    }
```

Листинг Г.9 – Исходный код блока вывода заключений

```
<div class="item-pacient-info">
  <p class=»label»>Заключения врача: </p>
  <div class="your-doctor">
    <div class="menu-bar">
      <?php echo get_conclusion_list($user_data['doctor_id'],
                                     $user_data['pacient_id']) ?>
    </div>
  </div>
</div>
```

Листинг Г.10 – Исходный код подпрограммы get_conclusion_list

```
function get_conclusion_list($d_id, $p_id) {
  include "/var/www/php/db/connect.php";
  $paths = $mysqli -> query("SELECT path FROM diagnoses
                             WHERE doctor_id = $d_id AND pacient_id = $p_id");
  if ($paths -> num_rows > 0) {
    $html_pattern = "<ul>";
    while ($path = $paths -> fetch_array()) {
      $path = substr(strrchr($path['path'], '/'), 1);
      $html_pattern  =  $html_pattern  .  _nti  _('<li><a  href="">%s</a></li>',
$path);
    }
      return $html_pattern . "</ul>";
  } else {
    return "User id is not defined!";
  }
}
```

Листинг Г.11 – HTML-код блока обмена сообщениями

```
<div class="feedback">
  <ul id="chat" style="list-style: none; line-height: 25px;
                       font-style: oblique;">
  </ul>
</div>
<div class="send-area">
  <textarea id="msgtxt" class="message"
            placeholder="Введите сообщение...">
  </textarea>
  <div style="margin: 5px auto; height: auto"
       class="button"
       onclick="sendmessage(<?php echo $user_data['pacient_id'] ?>,
                            <?php echo $user_data['doctor_id'] ?>,
                            'pacient')»>Отправить
  </div>
</div>
```

Листинг Г.12 – Исходный код функции sendmessage

```
function sendmessage(pid, did, whois) {
  var message = msgtxt.value;
  var request = new XMLHttpRequest();
  if (message == '') {
    alert(«Сообщение не может быть пустым!»);
    return null;
  } else {
    var body = "pid=" + encodeURIComponent(pid) +
               "&did=" + encodeURIComponent(did) +
               "&message=" + encodeURIComponent(message) +
               "&whois=" + encodeURIComponent(whois);
    request.open("POST", "../common/chat.php");
    request.setRequestHeader("Content-Type",
                             "application/x-www-form-urlencoded");
    request.send(body);
  }
  last_messages(pid);
}
```

Листинг Г.13 – Подпрограмма last_messages

```
function last_messages(pid) {
  var message_request = new XMLHttpRequest();
  message_request.open("GET", '../common/chat_messages.php?pid='+pid, false);
  message_request.send();
  console.log(message_request.status);
  var messages = message_request.responseText.split(';\n');
  added_messages_on_panel(messages);
}
```

Листинг Г.14 – Исходный код подпрограммы added_messages_on_panel

```
function added_messages_on_panel(messages) {
  chat.innerHTML = '';
  for (i = 0; i < messages.length - 1; i++) {
    var msg = document.createElement("li");
    spans = messages[i].split('>>');
    msg.innerHTML = '<span style="color: blue; font-size: 9pt">' + spans[0] +
                    '</span>';
    msg.innerHTML += ' <span style="color: red">' + spans[1] + '</span>:';
    msg.innerHTML += ' <span style="color: grey">' + spans[2] + "</span>"
    chat.appendChild(msg);
  }
}
```

Листинг Г.15 – Программный код обновления чата

```
<script type=»text/javascript» async>
  last_messages(<?php echo $user_data['pacient_id'] ?>);
  setInterval(last_messages, 4000, <?php echo $user_data['pacient_id'] ?>)
</script>
```

Листинг Г.16 – Исходный код PHP-сценария приема сообщений

```
include "/var/www/php/lib/user-lib.php";
$whois = $_POST['whois']; $p_id = $_POST['pid'];
$d_id = $_POST['did']; $message = $_POST['message'];
```

```
$path = get_path($p_id);
if ($path == "User ID is not defined!") echo 400;
else {
  $path = substr($path, 0, -3) . 'chat.txt';
  $chatfile = fopen($path, 'a+');
  if ($whois == 'doctor') $login = get_doctor_login($d_id);
  else if ($whois == 'pacient') $login = get_login($p_id);
  $message = _nti _("[%s]>>%s>>%s;\n", date("m.y.d (H:i:s)"), $login, $message);
      fwrite($chatfile, $message); echo 200;
  }
  fclose($chatfile);
```

Листинг Г.17 – Исходный код файла chat_messages.php

```
include "/var/www/php/lib/user-lib.php";
$p_id = $_GET['pid'];
$path = get_path($p_id);
if ($path == "User ID is not defined!") echo 400;
else {
  path = substr($path, 0, -3) . 'chat.txt';
  exit(readfile($path));}
```

Листинг Г.18 – Исходный код подпрограммы get_peaks

```
function get_peaks($p_l, $filename) {
  $path = get_pacient_data($p_l)['ecg'];
  $path = $path . '/' . $filename;
  $output = shell_exec("python /var/www/python/peaks.py " . $path);
  eval("\$peaks_index = array" . $output . ';');
  return $peaks_index;}
```

Листинг Г.19 – Исходный код подпрограммы get_peaks_length

```
function get_peaks_length($data) {
  $avg_length = 0;
  for ($i = 0; $i < count($data) - 1; $i++)
    $avg_length += abs((0.05 + $data[$i] * 0.05) - (0.05 + $data[$i + 1] *
0.05));
  $avg_length /= count($data) - 1;
  return round($avg_length, 3, PHP_ROUND_HALF_EVEN);
}
```

Листинг Г.20 – Исходный подпрограммы download_as_img

```
function download_as_img() {
  var download = document.getElementById("download_canvas");
  url = $("canvas")[0].toDataURL("image/png");
  download.setAttribute(«href», url);
}
```

Листинг Г.21 – Исходный код подпрограммы download_as_csv

```
function download_as_csv(dataset) {
  var text = "timestamp,amplitude\n"; var textFile = null;
  for (i = 0; i < dataset.length; i++) {
    text += dataset[i] + "\n";
  }
```

```
  var download = document.getElementById("download_csv");
  var data = new Blob([text], {type: 'text/plain'});
  if (textFile !== null) {
    window.URL.revokeObjectURL(textFile);
  }
  textFile = window.URL.createObjectURL(data);
  download.setAttribute(«href», textFile);
}
```

Листинг Г.22 – Исходный код скрипта отображения сигнала

```
data = <?php echo $json_data ?>;
  peaks = <?php echo $json_peaks_index ?>;
  dataset = [{label: "SIGNAL", data: data, points: {show: false}},
             {label: "PEAKS", data: peaks, points: {show: true}}];
$.plot($("#flot-placeholder"), dataset, options);
```

Листинг Г.23 – Исходный код формы сохранения заключения

```
<form name="save_diagnos" enctype="multipart/form-data" method="POST"
      action="<?php echo
             _nti _('./save_diagnos.php?doctor=%s&pacient=%s&ecg_name=%s',
             $id, $p_id, $filename) ?>"
      onsubmit="return is_empty(save_diagnos,
                                ['textarea'],
                                ['Сохранить заключение?',
                                 'Заполните поле заключения!'])»>
  <textarea name="diagnos_desc" id="diagnos" class="pacient-diagnos-textarea"
            placeholder=»Напишите заключение и нажмите кнопку сохранить…»
            <?php echo $diagnos_form_disabled ?>>
    <?php echo $last_content; ?>
  </textarea>
  <input class="button" type="submit"
         value = "Сохранить"
         <?php echo $diagnos_form_disabled ?> />
</form>
```

Листинг Г.24 – Исходный код валидатора формы

```
function is_empty(_nti _, tags, messages) {
  inputs = Array(); flag = true;
  for (i = 0; i < tags.length; i++)
    inputs[i] = _nti _.getElementsByTagName(tags[i]);
  for (i = 0; i < inputs.length; i++) {
    for (j = 0; j < inputs[i].length; j++) flag *= Boolean(inputs[i][j].value)
  }
  box_message(flag, messages)
  return Boolean(flag);
}
```

Листинг Г.25 – Исходный код блока списка шаблонов

```
<div class="item-doctor-templates">
  <div style="flex-grow: 1">
    <p><u>Шаблоны:</u></p>
      <div class="template-menu-bar">
        <ul type="none" style="margin-left: -30px">
          <?php echo get_doctor_templates() ?>
        </ul>
```

```
      </div>
  </div>
</div>
```

Листинг Г.26 – Исходный код подпрограммы add_template

```
function add_template(template_id, textarea) {
  template_desc = template_id.attributes['data-title'].value;
  if (!textarea.disabled) textarea.value += '\n' + template_desc;
}
```

Листинг Г.27 – HTML-шаблон блока элементов сигнала

```
<form name="point_editor_form" enctype="multipart/form-data" method="POST"
      action=<?php
                  echo _nti _("./save_signal.php?pacient=%s&filename=%s",
                              $_GET['pacient'], $filename)
              ?>
      onsubmit="return is_empty(point_editor_form,
                                ['input'],
                                ['Сохранить?', 'Заполните пустые поля!'])»>

  <?php edit_points($json_data) ?>
  <input id="save-button" type="submit" name="submit" value="save" />
  <input id="save-as-button" type="submit" name="submit" value="save_as" />
</form>
```

Листинг Г.28 – Исходный код подпрограммы edit_points

```
function edit_points($json_data) {
  $points = json_decode($json_data);
  $html_pattern = '<div class="points-queue">%s</div>';
  $point_tag = '<div class="point"
                       onmouseout= "unhiglight()"
                       onmouseover="highlight(%s)">
                 <input value="%s" readonly>
                 <input value="%s"
                        name="ecg_value_%s"
                        onchange="update_plot()">
               </div> ';
  $tmp = '';
  for ($i = 0; $i < count($points); $i++) {
    $tmp = $tmp . _nti _($point_tag, $i, $points[$i][0], $points[$i][1], $i);
  }
echo _nti _($html_pattern, $tmp);
}
```

Листинг Г.29 – Исходный код PHP-сценария save_signal.php

```
  include "/var/www/php/lib/user-lib.php";
  $id = get_pacient_id($_GET['pacient']);
  $filename = $_GET['filename'];
  $mode = $_POST['submit'];
  if ($mode == 'save_as') $filename = '[ed]' . $filename;
  $path = get_path($id) . '/' . $filename;
  $f = fopen($path, 'w');
  for ($i = 0; $i < count($_POST) - 1; $i++) {
```

```
      $value = $_POST[_nti _('ecg_value_%s', $i)];
      fwrite($f, $value . PHP_EOL);
  }
  fclose($f);
  header(_nti _("Location: ./point_editor.php?
                 filename=%s&pacient=%s",
                 $filename, $_GET['pacient']));
```

Листинг Г.30 – Исходный код сценария авторизации

```
include «/var/www/php/db/connect.php»;
function returnUserData($userid, $mysqli) {
  $user_info=$mysqli->query("SELECT pacient_id,p_email,p_initials,p_birthday,
                                   ecg   FROM   pacients   WHERE   pacient_id   =
                              $userid");
  if ($user_info -> num_rows > 0) {
    $user_info = $user_info -> fetch_array();
    $server_response = array('id' => $user_info['pacient_id'],
                             'initials' => $user_info['p_initials'],
                             'email' => $user_info['p_email'],
                             'birthday' => $user_info['p_birthday'],
                             'path' => $user_info['ecg'],
                             'status' => 200);
  }
  return json_encode($server_response);
}
if (!empty($_GET['auth_token'])) {
  $auth_token = $_GET["auth_token"];
  $auth_user=$mysqli->query("SELECT id
                            FROM users_hash WHERE hash = '$auth_token'");
  $auth_user_id = $auth_user -> fetch_array()['id'];
  if ($auth_user -> num_rows > 0) {
    $mysqli -> query("UPDATE pacients SET is_connect = 1
                     WHERE pacient_id = $auth_user_id");
    echo returnUserData($auth_user_id, $mysqli);
  } else echo json_encode(array('status' => 400));
    $mysqli -> close();
}
```

Листинг Г.31 – Исходный код сценария upload.php

```
include "/var/www/php/db/connect.php";
if (!empty($_GET['auth_token'])) {
  $auth_token = $_GET['auth_token'];
  $auth_user = $mysqli -> query("SELECT id FROM users_hash
                                WHERE hash = '$auth_token'");
  $auth_user_id = $auth_user -> fetch_array()['id'];
  if ($auth_user -> num_rows > 0) {
    $decoded_data = base64_decode($_POST['data']);
    $path = $_POST['path'] . '/' . $_POST['filename'];
    $f = fopen($path, 'w');
    fwrite($f, $decoded_data);
    fclose($f);
    $server_response = array('status' => 200);
    echo json_encode($server_response);
  } else {
    $server_response = array('status' => 400,
                             'error' => "Invalid auth-token");
    echo json_encode($server_response);
  }
  $mysqli -> close();}
```