Московский государственный технический университет имени Н. Э. Баумана.
Факультет “Информатика и системы управления”.
Кафедра “Системы обработки информации и управления”.

Утверждаю:

Галкин В.А.
"__"_____ 2020 г.

Курсовая работа
по курсу
Сетевые технологии в АСОИУ
«Программа пересылки файлов»

Описание программы
(вид документа)

бумага А4
(вид носителя)

(количество листов)
28
Вариант 13

ИСПОЛНИТЕЛИ:

студенты группы
ИУ5-64/62

Тарасов В.Ю. _____

Морозенков О.Н. _____

"__"_____2020 г.

Москва - 2020

1. Введение

Программный продукт написан с использованием технологии Svelte, Electron, Nodejs на языке программирования Javascript.

Для создания графического интерфейса и взаимодействия с COM-портом использовались стандартные библиотеки и элементы управления. Дополнительные функции, не относящиеся к стандартным, приведены ниже.

2. Компонент App

```
<script>
    import Settings from '../Settings/Settings.svelte';
    import ContextProvider from '../ContextProvider/
ContextProvider.svelte';
    import Route from '../Route/Route.svelte';
    import Welcome from '../Welcome/Welcome.svelte';
    import Chat from '../Chat/Chat.svelte';

    import {onMount} from 'svelte';
    import {router} from '../../modules/router/store';
    import Modal from '../Modal/Modal.svelte';

    export let defaultRoute = 'welcome';

    onMount(() => {
        router.nav(defaultRoute);
    });
</script>

<ContextProvider {...$$props} >
    <Modal />
    <Route name="welcome">
        <Welcome/>
    </Route>
    <Route name="settings">
        <Settings/>
    </Route>
    <Route name="chat">
        <Chat/>
    </Route>
</ContextProvider>
```

3. Компонент Welcome

```
<script>
    import Page from '../Page/Page.svelte';
    import { Button } from 'svelma';
    import { router } from '../../modules/router/store';

    function onClick() {
```

```
      router.nav('settings');
    }
</script>
<Page title="Welcome">
    <h1>Welcome</h1>
    <div class="ico port"></div>
    <h4 class="help">
        Before start sending file,
        need to tune your COM port
    </h4>

    <Button on:click={onClick} type="is-primary">Go To
Settings</Button>
</Page>


<style>
    .ico {
        width: 256px;
        height: 256px;
        margin: auto auto 12px;
    }

    .help {
        margin-bottom: 52px;
    }

</style>
```

4. Компонент Route

```
<script>
    import { router } from '../../modules/router/store';
    export let name = '';
</script>

{#if name === $router.name }
    <slot></slot>
{/if}
```

5. Компонент ContextProvider

```
<script>
    import {setContext} from 'svelte';

    export let context = {};

    Object.entries(context).forEach(([key, value]) => {
      setContext(key, value);
```

```
    });
</script>

<slot></slot>
```

6. Компонент Settings

```
export const type = (input) => (error) => {
  if (error && error.field === input) {
    return 'is-danger';
  }

  return void 0;
};

export const text = (input) => (error) => {
  if (error && error.field === input) {
    return error.error;
  }

  return void 0;
};


<script>
    import {Input, Field, Button} from 'svelma'
    import {settings, apiError, error} from '../../modules/
settings/store';
    import {text, type} from './Settings.helpers';
    import {getContext} from 'svelte';
    import {ContextKeys} from '../../modules/context';
    import {router} from '../../modules/router/store';
    import Page from '../Page/Page.svelte';

    const api = getContext(ContextKeys.API);

    async function onSubmit(e) {
        e.preventDefault();

        const {error} = await api.connect($settings);
        console.log('onSumbit', error);
        if (error) {
            apiError.set(error);
            return;
        }

        router.nav('chat');
    }

    $: speedText = text('speed')($error);
```

```
    $: speedType = type('speed')($error);
    $: comportType = type('comport')($error);
    $: comportText = text('comport')($error);
    $: apiErrorText = text('none')($error);

</script>


<Page title="Settings">
    <h1>Settings</h1>
    {#if apiErrorText }
        <p class="help is-danger">{apiErrorText}</p>
    {/if}
    <form action="#" on:submit={onSubmit} class="form">
        <Field label="COM-port" type={comportType}
message={comportText}>
            <Input
                    type="text"
                    bind:value={$settings.comport}
                    placeholder="COM1, COM2 and etc"
                    class={comportType}
            />
        </Field>

        <Field label="Speed" type={speedType}
message={speedText}>
            <Input
                    type="text"
                    bind:value={$settings.speed}
                    placeholder="1-2000"
                    class={speedType}
            />
        </Field>

        <Button type="is-primary" nativeType="submit"
disabled={!!$error && !apiErrorText}>Submit</Button>
    </form>
</Page>

<style>
    .form {
        max-width: 240px;
        margin: auto;
    }
</style>
```

7. Компонент Modal

```
<script>
```

```
    import { modal, modalResponse } from '../../modules/modal/
store';
    import { Notification, Dialog } from 'svelma'

    const types = {
        alert: openAlert,
        success: openSuccess,
        prompt: openPrompt
    };

    $: types[$modal.type] && types[$modal.type]($modal.text);

    function openAlert(text) {
        Notification.create({ message: text, type: 'is-danger',
position: 'is-top', duration: 5000 })
    }

    function openSuccess(text) {
        Notification.create({ message: text, type: 'is-success',
position: 'is-top', duration: 5000 })
    }

    function openPrompt(text) {
        Dialog.prompt({
            message: text,
        }).then(prompt => modalResponse.set(prompt))
    }
</script>
```

8. Компонент Page

```
<script>
    export let title = 'Telegram 2.0';
</script>
<svelte:head>
    <title>{title}</title>
</svelte:head>

<main class="content">
    <slot></slot>
</main>


<style>
    main {
        text-align: center;
        padding: 24px;
        margin: 0 auto;
        height: 100%;
    }
```

```
</style>
```

9. Компонент Chat

```
<script>
    import Page from '../Page/Page.svelte';
    import {Button} from 'svelma';
    import {router} from '../../modules/router/store';
    import {modal, modalResponse} from '../../modules/modal/
store';
    import {getContext, onMount} from 'svelte';
    import {ContextKeys} from '../../modules/context';
    import {waitForNotNull} from '../../helpers/store';

    const api = getContext(ContextKeys.API);

    function onClick() {
        router.nav('settings');
    }

    onMount(() =>
        api.listen('file-get', async ({name}) => {
            modal.prompt(`Someone send you file ${name}. Write
path where to save it`);

            const value = await waitForNotNull(modalResponse);

            const {error} = await api.save({path: value});

            if (error) {
              modal.alert(`Error while save file ${error}`);
              return;
            }

            modal.success('File was saved to ' + value);
        })
    );

    async function onChange(e) {
        if (!e || !e.target || !e.target.files || !
e.target.files[0]) {
            modal.alert('Error while read file');
            return;
        }

        const {error} = await api.send({file:
e.target.files[0]});

        if (error) {
            modal.alert(`Error while load file ${error}`);
```

```
            return;
        }

        modal.success('File was sent');
    }

</script>

<Page title="chat">
    <h1>Send file</h1>
    <div class="main">
        <div class="file is-boxed">
            <label class="file-label">
                <input class="file-input" type="file"
name="resume" on:change={onChange}>
                <span class="file-cta">
          <span class="file-icon">
            <i class="ico upload"></i>
          </span>
          <span class="file-label">
            Choose a file to upload…
          </span>
        </span>
            </label>
        </div>

        <Button on:click={onClick} type="is-primary">Go To
Settings</Button>
    </div>
</Page>


<style>
    .ico.upload {
        width: 24px;
        height: 24px;
        background-repeat: no-repeat;
    }

    .file.is-boxed {
        margin: auto;
    }

    .main {
        height: 80%;
        display: flex;
        flex-direction: column;
    }
</style>
```

10.Модуль Context

```
export const ContextKeys = {
  API: 'api'
};
```

11. Модуль Modal

```
import { writable, derived } from 'svelte/store';

function createModal() {
  const { subscribe, set, update } = writable({
    type: '',
    text: ''
  });

  return {
    subscribe,
    prompt: (text) => set({
      type: 'prompt',
      text
    }),
    alert: (text) => set({
      type: 'alert',
      text
    }),
    success: (text) => set({
      type: 'success',
      text
    }),
    reset: () => set({})
  };
}

export const modal = createModal();
export const modalResponse = writable('');
```

12. Модуль Router

```
import { writable, derived } from 'svelte/store';

function createRouter() {
  const { subscribe, set, update } = writable({
      name: ''
  });

  return {
    subscribe,
    nav: (name) => set({name}),
```

```
    reset: () => set({})
  };
}

export const router = createRouter();
```

13. Модуль Settings

```
import { writable, derived } from 'svelte/store';

function createSettings() {
  const { subscribe, set, update } = writable({
    comport: 'COM1',
    speed: 10
  });

  return {
    subscribe,
    set: (newSettings) => set(newSettings),
    reset: () => set({})
  };
}

export const settings = createSettings();

export const apiError = writable('');

const isValidCom = (value) => value && !!value.match(/COM[0-9]
[0-9]?/);
const isValidSpeed = (value) => !Number.isNaN(parseFloat(value))
&& value > 0;

export const error = derived(
  [settings, apiError],
  ([$settings, $apiError]) => {
      // if (!isValidCom($settings.comport)) {
      //   return {
      //     field: 'comport',
      //     error: 'Invalid comport format. True format is
COM[0-9][0-9]?'
      //   }
      // }

      if (!isValidSpeed($settings.speed)) {
        return {
          field: 'speed',
          error: 'Invalid speed'
        }
      }
```

```
      if ($apiError) {
        return {
          field: 'none',
          error: $apiError
        }
      }

      return false;
  }
);
```

14. Сервис Api

```
export class ApiServiceDummy {
  async connect(settings) {
    console.log(settings);
    return Promise.resolve({ error: null });
  }

  async send(params) {
    console.log(params);
    return Promise.resolve({ error: null });
  }
}

export class ApiServiceElectron {
  constructor(ipc) {
    this.ipc = ipc;
  }

  async connect(settings) {
    return new Promise((resolve) => {
      this.ipc.on('connect-ok', () => {
        console.log('connect-ok');
        resolve();
      });
      this.ipc.on('connect-error', (event, error) => {
        console.log('connect-error');
        resolve({error});
      });

      console.log('connect');

      this.ipc.send('connect', {
        settings
      });
    });
  }
```

```
  async send(params) {
    return new Promise((resolve) => {
      this.ipc.on('send-ok', resolve);
      this.ipc.on('send-error', (event, error) => {
        resolve({error});
      });

      const file = params.file;
      this.ipc.send('send', {
        file: {
          name: file.name,
          path: file.path,
          size: file.size,
          type: file.type,
          lastModified: file.lastModified,
        }
      });
    });
  }

  listen(event, clb) {
    const handle = (_, args) => {
      clb(args);
    };

    this.ipc.on(event, handle);

    return () => {
      this.ipc.removeListener(event, handle);
    }
  }

  async save(params) {
    return new Promise((resolve) => {
      this.ipc.on('save-ok', resolve);
      this.ipc.on('save-error', (event, error) => {
        resolve({error});
      });

      this.ipc.send('save', {
        path: params.path
      });
    });
  }
}
```

15. Модуль CRC

```
function xorify(startValue, del) {
  let current = startValue;
```

```
  while (current.toString(2).length >= del.toString(2).length) {
    const bitsDiff = current.toString(2).length -
del.toString(2).length;
    const xorFirst = current >> bitsDiff;
    const xorResult = xorFirst ^ del;

    current = (xorResult << bitsDiff) | (~(0b1 << (bitsDiff +
1)) & current);

    const firstPart = xorResult << bitsDiff;
    const secondPart = parseInt('1'.repeat(bitsDiff), 2) &
current;

    current = firstPart | secondPart;
  }

  return current;
}

function encode(toEncode, polynome, n, k) {
  let mx = toEncode << (n - k);
  let px = xorify(mx, polynome);

  return mx ^ px;
}

function decode(toDecode, polynome) {
  return xorify(toDecode, polynome);
}
```

16. Модуль AppConnection

```
const {DataConnection} = require("./data");

const PACKET_SIZE_TYPE     = 8;
const PACKET_SIZE_FILENAME = 128;

const PACKET_OFFS_TYPE     = 0;
const PACKET_OFFS_FILENAME = PACKET_OFFS_TYPE +
PACKET_SIZE_TYPE;
const PACKET_OFFS_FILEDATA = PACKET_OFFS_FILENAME +
PACKET_SIZE_FILENAME;

const TYPE_FILE = 1;
const TYPES_MAX = 1;

class PacketError extends Error {
  constructor(message) {
```

```
    super(message);
    this.name = this.constructor.name;
  }
}

function strToBuf(str, cap) {
  const buf = new ArrayBuffer(cap);
  const bufView = new Uint16Array(buf);
  for (let i = 0; i < Math.max(str.length, cap/2-1); i++) {
    bufView[i] = str.charCodeAt(i);
  }
  return buf;
}

function bufToStr(buf) {
  const len = buf.indexOf(0);
  const bufCut = buf.slice(0, len);
  return String.fromCharCode.apply(null, bufCut);
}

function packetMakeFile(filename, filedata) {
  const packetLen = PACKET_SIZE_TYPE + PACKET_SIZE_FILENAME +
bufLen;
  const packetBuf = new ArrayBuffer(packetLen);
  const packetBufView = new DataView(packetBuf);
  const packetBufUint8View = new Uint8Array(packetBuf);

  packetBufView.setUint8(PACKET_OFFS_TYPE, TYPE_FILE);
  const filenameBuf = strToBuf(filename, PACKET_SIZE_FILENAME);
  packetBufUint8View.set(new Uint8Array(filenameBuf),
PACKET_OFFS_FILENAME/8);
  packetBufUint8View.set(filedata, PACKET_OFFS_FILEDATA/8);

  return packetBuf;
}

function packetParseFile(packetBuf) {
  const packetBufView = new DataView(packetBuf);

  if (!(packetBuf.byteLength > PACKET_SIZE_TYPE +
PACKET_SIZE_FILENAME))
    throw new PacketError('bad header size');

  const type = packetBufView.getUint8(PACKET_OFFS_TYPE);
  if (!(0 <= type && type <= TYPES_MAX))
    throw new PacketError('bad type');

  const filename =
bufToStr(packetBuf.slice(PACKET_OFFS_FILENAME,
PACKET_OFFS_FILENAME + PACKET_SIZE_FILENAME));
  const filedata = packetBuf.slice(PACKET_OFFS_FILEDATA);
```

```
  return {filename, filedata};
}

class AppConnection {
  constructor() {
    this._data = null;
  }

  async accept(path) {
    console.log('accept', path, this._data);
    if (this._data)
      return;

    this._data = new DataConnection();
    try {
      await this._data.accept(path);
    } catch (e) {
      this._data = null;
      throw e;
    }
  }

  async connect() {
    if (!this._data)
      return;

    await this._data.connect();
  }

  async close() {
    if (!this._data)
      return;

    try {
      await this._data.close();
    } finally {
      this._data = null;
    }
  }

  async sendFile(filename, filedata) {
    console.log('send', this._data, this._data.isConnected());
    if (!this._data || !this._data.isConnected())
      return;

    await this._data.write(packetMakeFile(filename, filedata))
  }

  async recvFile() {
    if (!this._data || !this._data.isConnected())
```

```
      return;

    return packetParseFile(await this._data.read());
  }
}


module.exports = {
  AppConnection
};
```

17. Модуль Data

```
const {PhysicalConnection} = require("./physical");
const {TimeoutError} = require("promise-timeout");

// consts
================================================================
==============================================

const PACKET_SIZE_CRC  = 32;
const PACKET_SIZE_TYPE = 8;
const PACKET_SIZE_LEN  = 32;
const HEADER_SIZE = PACKET_SIZE_CRC + PACKET_SIZE_TYPE +
PACKET_SIZE_LEN;

const PACKET_OFFS_CRC  = 0;
const PACKET_OFFS_TYPE = PACKET_OFFS_CRC  + PACKET_SIZE_CRC;
const PACKET_OFFS_LEN  = PACKET_OFFS_TYPE + PACKET_SIZE_TYPE;
const PACKET_OFFS_DATA = PACKET_OFFS_LEN  + PACKET_SIZE_LEN;

const TYPE_CONNECT = 1;
const TYPE_FINISH  = 2;
const TYPE_DATA    = 3;
const TYPE_ACK     = 4;
const TYPE_RETRY   = 5;
const TYPES_MAX = 5;

const ENDIANNESS_LITTLE = true;

const STATE_NONE       = 1;
const STATE_ACCEPTING  = 2;
const STATE_CONNECTING = 3;
const STATE_CONNECTED  = 4;
const STATE_FINISHING  = 5;

// consts
^================================================================
==============================================
```

```
// errors
================================================================
================================================

class PacketError extends Error {
  constructor(message) {
    super(message);
    this.name = this.constructor.name;
  }
}

class ConnectError extends Error {
  constructor(message) {
    super(message);
    this.name = this.constructor.name;
  }
}

class CloseError extends Error {
  constructor(message) {
    super(message);
    this.name = this.constructor.name;
  }
}

class WriteError extends Error {
  constructor(message) {
    super(message);
    this.name = this.constructor.name;
  }
}

class ReadError extends Error {
  constructor(message) {
    super(message);
    this.name = this.constructor.name;
  }
}

// errors
^================================================================
================================================

// packet
================================================================
================================================

function packetMake(type, buf) {
  const bufLen = (buf) ? buf.byteLength : 0;
  const packetLen = HEADER_SIZE + bufLen;
  const packetBuf = new ArrayBuffer(packetLen);
```

```
  const packetBufView = new DataView(packetBuf);
  const packetBufUint8View = new Uint8Array(packetBuf);

  packetBufView.setUint8(PACKET_OFFS_TYPE, type);
  packetBufView.setUint32(PACKET_OFFS_LEN, bufLen,
ENDIANNESS_LITTLE);
  if (bufLen)
    packetBufUint8View.set(buf, PACKET_OFFS_DATA/8);

  const crc = checksumCRC(packetBuf.slice(PACKET_OFFS_CRC +
PACKET_SIZE_CRC));
  packetBufView.setUint32(PACKET_OFFS_CRC, crc,
ENDIANNESS_LITTLE);

  return packetBuf;
}

function packetParse(packetBuf) {
  const packetBufView = new DataView(packetBuf);

  if (!(packetBuf.byteLength >= HEADER_SIZE))
    throw new PacketError('bad header size');

  const crc = packetBufView.getUint32(PACKET_OFFS_CRC,
ENDIANNESS_LITTLE);
  const crcClient = checksumCRC(packetBuf.slice(PACKET_OFFS_CRC
+ PACKET_SIZE_CRC));
  if (crc !== crcClient)
    throw new PacketError('bad crc');

  const type = packetBufView.getUint8(PACKET_OFFS_TYPE);
  if (!(0 <= type && type <= TYPES_MAX))
    throw new PacketError('bad type');

  const len = packetBufView.getUint32(PACKET_OFFS_LEN,
ENDIANNESS_LITTLE);
  if (len !== (packetBuf.byteLength - HEADER_SIZE))
    throw new PacketError('bad len');

  let buf = null;
  if (len)
    buf = packetBuf.slice(HEADER_SIZE);

  return {type, buf};
}

// packet
^================================================================
==============================================
```

```
// conn
================================================================
================================================

class DataConnection {
  constructor() {
    this._phys = null;
    this._state = STATE_NONE;
    this._txDataQueue = [];
    this._rxDataQueue = [];
  }

  async accept(path) {
    console.log('data accept');
    if (this._state !== STATE_NONE)
      return;

    this._phys = new PhysicalConnection();
    try {
      await this._phys.connect(path);
    } catch (e) {
      await this._close();
      throw e;
    }
    this._state = STATE_ACCEPTING;
  }

  async connect() {
    if (this._state !== STATE_ACCEPTING)
      return;

    console.log('data: connect');
    this._state = STATE_CONNECTING;
    await this._write(TYPE_CONNECT, null);

    while (this._state === STATE_CONNECTING)
      await this.loop();
    if (this._state !== STATE_CONNECTED)
      throw new ConnectError('failed connect');
  }

  async close() {
    if (this._state === STATE_NONE)
      return;

    if (this._state === STATE_CONNECTED) {
      this._state = STATE_FINISHING;
      this._txDataQueue = [];
      this._rxDataQueue = [];
      await this._write(TYPE_FINISH, null);
```

```
      while (this._state === STATE_FINISHING)
        await this.loop();
    }

    this._state = STATE_NONE;
    await this._close();
  }

  async isAccepting() {
    return this._state !== STATE_NONE;
  }

  async isConnected() {
    return this._state === STATE_CONNECTED;
  }

  async write(buf) {
    console.log(this._state);
    if (this._state !== STATE_CONNECTED)
      return;

    console.log('write');

    await this._write(TYPE_DATA, buf);
    this._txDataQueue.push(buf);

    while (this._txDataQueue.length)
      await this.loop();
    if (this._state !== STATE_CONNECTED)
      throw new WriteError('closed connection');
  }

  async read() {
    if (this._state !== STATE_CONNECTED)
      throw new ReadError('not connected');

    while (!this._rxDataQueue.length && this._state ===
STATE_CONNECTED)
      await this.loop();
    if (this._state !== STATE_CONNECTED)
      throw new ReadError('closed connection');
    return this._rxDataQueue.shift();
  }

  async loop() {
    if (this._state === STATE_NONE) {
      return;
    }

    if (this._state === STATE_ACCEPTING) {
      const {ok, type} = await this._read();
```

```
  if (!ok)
    return;

  if (type === TYPE_CONNECT) {
    this._state = STATE_CONNECTED;
    await this._write(TYPE_ACK, null);
  }
}

if (this._state === STATE_CONNECTING) {
  const {ok, type} = await this._read();
  if (!ok)
    return;

  if (type === TYPE_ACK) {
    this._state = STATE_CONNECTED;
  } else {
    this._state = STATE_ACCEPTING;
  }
}

if (this._state === STATE_CONNECTED) {
  const {ok, type, buf} = await this._read();
  if (!ok)
    return;

  if (type === null) {
    await this._write(TYPE_RETRY, null);
  }

  if (type === TYPE_CONNECT) {
    this._txDataQueue = [];
    this._rxDataQueue = [];
    await this._write(TYPE_ACK, null);
  }

  if (type === TYPE_FINISH) {
    this._txDataQueue = [];
    this._rxDataQueue = [];
    this._state = STATE_ACCEPTING;
    await this._write(TYPE_ACK, null);
  }

  if (type === TYPE_DATA) {
    this._rxDataQueue.push(buf);
    await this._write(TYPE_ACK, null);
  }

  if (type === TYPE_ACK) {
    this._txDataQueue.splice(0, 1);
  }
```

```
    if (type === TYPE_RETRY) {
      if (this._txDataQueue.length) {
        await this._write(TYPE_DATA, this._txDataQueue[0]);
      }
    }
  }

  if (this._state === STATE_FINISHING) {
    const {ok} = await this._read();
    if (!ok)
      return;

    this._state = STATE_ACCEPTING;
  }
}

async _read() {
  let packetBuf = null;
  try {
    packetBuf = await this._phys.read();
  } catch (e) {
    if (e instanceof TimeoutError)
      return {ok: false, type: null, buf: null};
    throw e;
  }

  let packet = null;
  try {
    packet = packetParse(packetBuf);
  } catch (e) {
    if (e instanceof PacketError)
      return {ok: true, type: null, buf: null};
    throw e;
  }

  return {ok: true, type: packet.type, buf: packet.type};
}

async _write(type, buf) {
  const packetBuf = packetMake(type, buf);
  this._phys.write(packetBuf);
}

async _close() {
  try {
    await this._phys.close();
  } finally {
    this._phys = null;
  }
}
```

```
}

module.exports = {
  DataConnection
};

// conn
^==========================================================
==============================================
```

18. Модуль Physical

```
const SerialPort = require("serialport");
const {promisify} = require("util");
const delay = require("delay");
const {timeout} = require("promise-timeout");

const TIMEOUT = 1000;
const FLAG_DELAY = 100;

async function waitPortFlags(port, {dsr = false, dcd = false,
cts = false} = {}) {
  const portGet = promisify(port.get.bind(port));

  while (true) {
    const flags = await portGet();
    let ok = true;
    if (dsr && !flags.dsr) ok = false;
    if (dcd && !flags.dcd) ok = false;
    if (cts && !flags.cts) ok = false;
    if (ok)
      break;
    await delay(FLAG_DELAY);
  }
  return true;
}

class PhysicalConnection {
  constructor() {
    this._port = null;
  }

  async connect(path, {tout = TIMEOUT} = {}) {
    console.log('physical connect', path);
    return timeout(this._connect(path), tout);
  }

  async _connect(path) {
    if (this._port)
      return;
```

```
    const portOpts = {
      autoOpen: false,
      baudRate: 9600,
      dataBits: 8,
      parity: 'none',
      rtscts: false,
      stopBits: 1,
    };
    console.log(path);
    this._port = new SerialPort(path, portOpts);
    const portOpen =
promisify(this._port.open.bind(this._port));
    const portSet = promisify(this._port.set.bind(this._port));

    await portOpen();
    console.log('before prot set');
    await portSet();
    console.log('after prot set');
    // this._port.set({dtr: true}, console.log);
    await waitPortFlags(this._port, {dsr: true, dcd: true});
  }

  async close({tout = TIMEOUT} = {}) {
    try {
      await timeout(this._close(), tout);
    } finally {
      if (this._port) {
        try {
          this._port.close();
        } finally {
          this._port = null;
        }
      }
    }
  }

  async _close() {
    if (!this._port)
      return;
    const portClose =
promisify(this._port.close.bind(this._port));
    const portSet = promisify(this._port.set.bind(this._port));

    await portSet({dtr: false, rts: false});
    await portClose();
    this._port = null;
  }

  async write(buf, {tout = TIMEOUT} = {}) {
    return timeout(this._write(buf), tout);
```

```
  }

  async _write(buf) {
    if (!this._port)
      return;
    const portDrain =
promisify(this._port.drain.bind(this._port));
    const portSet = promisify(this._port.set.bind(this._port));

    await portSet({rts: true});
    await waitPortFlags(this._port, {cts: true});
    this._port.write(buf);
    await portDrain(buf);
  }

  async read({tout = TIMEOUT} = {}) {
    return timeout(this._read(), tout);
  }

  async _read() {
    if (!this._port)
      return;
    const portSet = promisify(this._port.set.bind(this._port));

    await portSet({cts: true});
    await waitPortFlags(this._port, {cts: true});

    return this._port.read();
  }
}


module.exports = {
  PhysicalConnection
};
```

19. Входная точка сетевого кода

```
const { app, BrowserWindow, ipcMain, ipcRenderer } =
require('electron');
const {AppConnection} = require("./network/app");
const path = require('path');
const fs = require('fs');

// Live Reload
require('electron-reload')(__dirname, {
  electron: path.join(__dirname, '../node_modules', '.bin',
'electron'),
  awaitWriteFinish: true
});
```

```
// Handle creating/removing shortcuts on Windows when
installing/uninstalling.
if (require('electron-squirrel-startup')) {
  // eslint-disable-line global-require
  app.quit();
}

let window = null;

const createWindow = () => {
  // Create the browser window.
  window = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true
    }
  });

  window.loadFile(path.join(__dirname, '../public/index.html'));
};

app.on('ready', createWindow);

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit();
  }
});

app.on('activate', () => {
  if (BrowserWindow.getAllWindows().length === 0) {
    createWindow();
  }
});

const conn = new AppConnection();

// подключение
ipcMain.on('connect', (event, {settings}) => {
  console.log(settings);
  conn.accept(settings.comport).then(() => {
    return conn.connect();
  }).then(() => {
    console.log('connect-ok');
    event.reply('connect-ok');
    subscribeRecvFile();
  }).catch((e) => {
    event.reply('connect-error', e);
  });
```

```
});

// прием
function subscribeRecvFile() {
  conn.recvFile().then((filename, buf) => {
    window.webContents.send('file-get', { name: filename });

    fs.writeFile(filename, buf, (err) => {
      window.webContents.send('save-ok');
    })
  });
}

function toArrayBuffer(buf) {
  const ab = new ArrayBuffer(buf.length);
  const view = new Uint8Array(ab);
  for (let i = 0; i < buf.length; ++i) {
    view[i] = buf[i];
  }
  return ab;
}

// отправка
ipcMain.on('send', (event, {file}) => {

  fs.readFile(file.path, (err, data) => {
    if (err) {
      event.reply('send-error');
      console.log(err);
      return;
    }

    try {
      const buf = toArrayBuffer(data);
      console.log('before send');
      conn.sendFile(file.name, buf)
        .then(() => {
          console.log('send ok');
          event.reply('send-ok');
        }).catch(() => {
          event.reply('send-error');
        });

    } catch (e) {
      console.log(err);
      event.reply('send-error');
    }
  });
});
```

20. Входная точка клиентского кода

```
import './styles.scss';
import App from './components/App/App.svelte';
import {ContextKeys} from './modules/context';
import {ApiServiceElectron} from './services/api';
const {ipcRenderer} = require('electron');

const api = new ApiServiceElectron(ipcRenderer);

const app = new App({
  target: document.body,
  props: {
    context: {
      [ContextKeys.API]: api
    },
    defaultRoute: 'welcome'
  }
});


export default app;
```

21. Функции помощники

```
export const waitForNotNull = (store) => {
  let unsub = null;

  return new Promise(resolve => {
    unsub = store.subscribe((value) => {
      if (value) {
        resolve(value);
        store.set(null);
      }
    });
  }).finally(unsub);
};
```