ПРИЛОЖЕНИЕ А
(обязательное)
Исходный текст клиентского приложения

Файл index.js

```
import 'babel-polyfill';

import ReactDOM from 'react-dom';
import Router from './router';
import CameraService from './services/CameraService';
import SpinnerService from './services/SpinnerService';

import './index.css';

CameraService.init();
SpinnerService.init();
ReactDOM.render(
  Router.initializeRoute(),
  document.getElementById('root')
);
```

Файл router.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import { Router, Route, IndexRoute, browserHistory } from 'react-
router';
import configureStore from './store/configureStore';
const store = configureStore();

import App from './components/containers/App/App';
import Login from './components/containers/Login/Login';

import Home from './components/containers/Home/Home';

import NotFound from './components/containers/Misc/NotFound';
import RegistrationStep1 from
'./components/containers/Registration/RegistrationStep1';
import RegistrationStep2 from
'./components/containers/Registration/RegistrationStep2';

import AddMedicalCard from
'./components/containers/AddMedicalCard/AddMedicalCard';
import EditMedicalCard from
'./components/containers/EditMedicalCard/EditMedicalCard';
import UploadPhoto from
'./components/containers/UploadPhoto/UploadPhoto';

import MedicalCard from
'./components/containers/MedicalCard/MedicalCard';
import ExportList from './components/containers/ExportList/ExportList';

import Card from './components/containers/Card/Card';
import AddChild from './components/containers/AddChild/AddChild';

const RouterSetting = {
```

```
        initializeRoute () {
            return (
                <Provider store={store}>
                    <Router history={browserHistory}>
                        <Route path="/" component={App}>
                            <IndexRoute component={Home}/>
                            <Route path="/login" component={Login}/>
                            <Route                    path="/register-step1"
component={RegistrationStep1}/>
                            <Route                    path="/register-step2"
component={RegistrationStep2}/>
                            <Route                  path="/add-medical-card"
component={AddMedicalCard}/>
                            <Route                 path="/edit-medical-card"
component={EditMedicalCard}/>
                            <Route                      path="/medical-card"
component={MedicalCard}/>
                            <Route                       path="/export-list"
component={ExportList}/>
                            <Route path="/card" component={Card}/>
                            <Route path="/photo" component={UploadPhoto}/>

                            <Route path="/add-child" component={AddChild}/>
                            <Route path="*" component={Login}/>
                        </Route>
                    </Router>
                </Provider>
            );
        }
    };

    export default RouterSetting;
```

Файл configureStore.js

```
import { createStore, applyMiddleware, combineReducers } from 'redux';
import thunkMiddleware from 'redux-thunk';
import createLogger from 'redux-logger';
import reducer from '../reducers/index';

const logger = createLogger();

const createStoreWithMiddleware = applyMiddleware(
  thunkMiddleware,
  logger
)(createStore);

export default function configureStore(initialState) {
  return createStoreWithMiddleware(reducer, initialState);
}
```

Файл CameraService.js

```
import WebCameraDriver from '../drivers/WebCameraDriver';
import CordovaCameraDriver from '../drivers/CordovaCameraDriver';
import BaseService from './BaseService';

class CameraService extends BaseService{

    static getPhotoByCamera () {
```

```
            const cameraOptions = {
                destinationType:
CameraService.driver.destinationType().FILE_URI,
                sourceType: CameraService.driver.pictureSourceType().CAMERA,
                allowEdit: false,
                quality: CameraService.QUALITY_IMAGE,
                targetWidth: CameraService.SQUARE_SIZE,
                targetHeight: CameraService.SQUARE_SIZE,
                correctOrientation: true
            };

            return       new       Promise((resolve,       reject)       =>
CameraService.driver.getPicture(photoURI   =>   resolve(photoURI),   error   =>
reject(error), cameraOptions));
        }

        static getPhotoFromAlbum () {
            const cameraOptions = {
                destinationType:
CameraService.driver.destinationType().FILE_URI,
                sourceType:
CameraService.driver.pictureSourceType().PHOTOLIBRARY,
                allowEdit: false,
                quality: CameraService.QUALITY_IMAGE,
                targetWidth: CameraService.SQUARE_SIZE,
                targetHeight: CameraService.SQUARE_SIZE,
                correctOrientation: true
            };

            return       new       Promise((resolve,       reject)       =>
CameraService.driver.getPicture(photoURI   =>   resolve(photoURI),   error   =>
reject(error), cameraOptions));
        }

        static getResizeImageByURI (uri, width, height) {
            return new Promise(resolve => {
                const img = new window.Image();

                img.crossOrigin = 'Anonymous';
                img.onload = function () {
                    resolve(CameraService.getResizeImage(this,        width,
height).toDataURL());
                };

                img.src = uri;
            });
        }

        static getResizeImage (img, width, height) {
            let image = img;
            let sW = image.width;
            let sH = image.height;
            const resizeStep = 1.4;
            const getCanvas = (newWidth, newHeight) => {
                const canvas = window.document.createElement('canvas');

                canvas.width = newWidth;
                canvas.height = newHeight;
                canvas.getContext('2d').drawImage(image,   0,   0,   newWidth,
newHeight);

                return canvas;
            };
```

```
        image = getCanvas(sW, sH);

        while (sW > width) {
            sW = Math.round(sW / resizeStep);
            sH = Math.round(sH / resizeStep);
            if (sW < width) {
                sW = width;
                sH = height;
            }

            image = getCanvas(sW, sH);
        }
        return image;
    }

    static getDriver (cordovaDriver, webDriver) {
        if (window.isDevice) {
            return cordovaDriver;
        }

        return webDriver;
    }

    static init () {
        CameraService.driver = CameraService.getDriver(
            CordovaCameraDriver,
            WebCameraDriver
        );

        CameraService.driver.init();
    }
}

CameraService.driver = null;

CameraService.SQUARE_SIZE = 1700;
CameraService.QUALITY_IMAGE = 100;

export default CameraService;
```

Файл NavigationService.js

```
import { browserHistory } from 'react-router';
import BaseService from './BaseService';

class NavigationService extends BaseService {

    static navigateTo (url) {
        browserHistory.push(url);
    }
}

export default NavigationService;
```

Файл SpinnerService.js

```
import WebSpinnerDriver from '../drivers/WebSpinnerDriver';
```

```
import CordovaSpinnerDriver from '../drivers/CordovaSpinnerDriver';
import BaseService from './BaseService';

class SpinnerService extends BaseService {
    static show (title, message, callback) {
        SpinnerService.driver.show(title, message, callback);
    }

    static hide () {
        SpinnerService.driver.hide();
    }

    static getDriver (cordovaDriver, webDriver) {
        if (window.isDevice) {
            return cordovaDriver;
        }

        return webDriver;
    }

    static init () {
        SpinnerService.driver = SpinnerService.getDriver(
            CordovaSpinnerDriver,
            WebSpinnerDriver
        );

        SpinnerService.driver.init();
    }
}

SpinnerService.driver = null;

export default SpinnerService;
```

Файл TextFormatorService.js

```
import is from 'is_js';
import BaseService from './BaseService';

class TextFormatorService extends BaseService {

    /**
     * @private
     *
     * @return {String}
     */
    static getWordForMedicalCard (rule,  arrayLines) {

        const    currentLine    =    arrayLines.filter(item    =>    (new
RegExp(rule)).test(item.text));

        if (is.not.null(currentLine) && currentLine.length !== 0) {
            return currentLine[0].words[1].text;
        }

        return null;
    }

    /**
     * @public
     *
     * @return {Object}
```

```
         */
        static getFormatedMedicalCardObject (listData) {
            const ruleDoctor =  /^doctor/i;
            const ruleSpecialization =  /^specialization/i;
            const ruleSummary =  /^summary/i;
            const ruleRecommendations =  /^recommendations/i;
            const ruleDateVisit = /^date visit/i;

            const ruleMedication = /^medication/i;
            const ruleStep = /^step/i;
            const ruleQuantity = /^quantity/i;
            const ruleCountDays = /^count days/i;

            return {
                doctor: {
                    lastNameDoctor:
TextFormatorService.getWordForMedicalCard(ruleDoctor, listData),
                    specialization:
TextFormatorService.getWordForMedicalCard(ruleSpecialization, listData)
                },
                medications: {
                    countDays:
TextFormatorService.getWordForMedicalCard(ruleCountDays, listData),
                    nameOfMedication:
TextFormatorService.getWordForMedicalCard(ruleMedication, listData),
                    quantity:
TextFormatorService.getWordForMedicalCard(ruleQuantity, listData),
                    stepTreatment:
TextFormatorService.getWordForMedicalCard(ruleStep, listData)
                },
                recommendations:
TextFormatorService.getWordForMedicalCard(ruleRecommendations, listData),
                summary:
TextFormatorService.getWordForMedicalCard(ruleSummary, listData),
                dateOfVisit:
TextFormatorService.getWordForMedicalCard(ruleDateVisit, listData)
            }
        }
    }

    export default TextFormatorService;
```

Файл authReducer.js

```
    'use strict';
    import * as userType from '../constants/userTypes';
    import { loadUser, loadUserProfile } from '../actions/utils';
    import LocalStorageService from '../services/LocalStorageService';
    import Immutable from 'immutable';

    const initialState = {
        user: null,
        profile: null,
        passcode: null,
        userRole: null,
        loggingIn: false,
        loggingOut: false,
        loginError: null
    };

    function initializeState(){
```

```
        const user = loadUser(); //TODO refactoring user and userProfile.
Now copy-paste =(
        const userProfile = loadUserProfile();
        return Object.assign({}, initialState, user, userProfile);
    }

    function successRegistration (state, action) {
        const newState = Object.assign({}, state, { loggingIn: false, user:
action.user, role: action.role });
        LocalStorageService.set('auth', 'authData', newState.toString());

        return newState;
    }

    export default function authReducer(state = initializeState(), action =
{}) {
        switch (action.type) {
            case userType.REGISTER_REQUEST:
                return Object.assign({}, state, {loggingIn: true});
            case userType.REGISTER_SUCCESS:
                return Object.assign({}, state, { loggingIn: false, user:
action.user, role: action.role });
            case userType.REGISTER_FAILURE:
                return {
                    ...state,
                    loggingIn: false,
                    user: null,
                    role: null,
                    loginError: action.error
                };
            case userType.REGISTER_PROFILE_REQUEST:
                return Object.assign({}, state, {loggingIn: true});
            case userType.REGISTER_PROFILE_SUCCESS:
                return Object.assign({}, state, {
                    loggingIn: false,
                    firstName: action.firstName,
                    lastName: action.lastName,
                    familyName: action.familyName,
                    idcard: action.idcard,
                    gender: action.gender,
                    dateOfBirth: action.dateOfBirth
                });
            case userType.REGISTER_PROFILE_FAILURE:
                return {
                    ...state,
                    loggingIn: false,
                    user: null,
                    role: null,
                    loginError: action.error
                };
            case userType.LOGIN_REQUEST:
                return Object.assign({}, state, {loggingIn: true});
            case userType.LOGIN_SUCCESS:
                return Object.assign({}, state, {
                    loggingIn: false, profile: action.profile });
            case userType.LOGIN_FAILURE:
                return {
                    ...state,
                    loggingIn: false,
                    profile: null,
                    loginError: action.error
                };
            case userType.LOGOUT_REQUEST:
```

```
                return {
                    ...state,
                    loggingOut: true
                };
            case userType.LOGOUT_SUCCESS:
                return {
                    ...state,
                    loggingOut: false,
                    user: null,
                    userRole: null,
                    loginError: null
                };
            case userType.LOGOUT_FAILURE:
                return {
                    ...state,
                    loggingOut: false,
                    logoutError: action.error
                };
            case userType.CHECK_PASSCODE_REQUEST:
                return Object.assign({}, state);
            case userType.CHECK_PASSCODE_SUCCESS:
                {
                    console.log('State: ', state);
                    console.log('action: ', action.passcode);
                    return Object.assign({}, state, {
                        passcode: action.passcode });
                }

            case userType.CHECK_PASSCODE_FAILURE:
                return {
                    ...state,
                    passcode: null,
                    loginError: action.error
                };
            default:
                return state;
        }
    }
```

Файл medicalCardReducer.js

```
    'use strict';
    import * as medCardType from '../constants/medicalCardTypes';

    const initialState = {
        medicationCard: [{
            doctor: null,
            medication: null,
            payment: null,
            medicalCard: null
        }],
        cards: [],
        isAddedIn: false,
        messageError: null
    };

    function initializeState(){
        return Object.assign({}, initialState, {});
    }

    export default function medicalCardReducer(state = initializeState(),
action = {}) {
```

```
        switch (action.type) {
            case medCardType.ADD_REQUEST:
                return Object.assign({}, state, {isAddedIn: true});
            case medCardType.ADD_SUCCESS:
                return Object.assign({}, state, {
                    isAddedIn:            false,            medicationCard:
action.medicationCard});
            case medCardType.ADD_FAILURE:
                return {
                    ...state,
                    isAddedIn: false,
                    messageError: action.error
                };

            case medCardType.EDIT_REQUEST:
                return Object.assign({}, state, {isAddedIn: true});
            case medCardType.EDIT_SUCCESS:
                return Object.assign({}, state, {
                    isAddedIn:            false,            medicationCard:
action.medicationCard});
            case medCardType.EDIT_FAILURE:
                return {
                    ...state,
                    isAddedIn: false,
                    messageError: action.error
                };

            case medCardType.DELETE_REQUEST:
                return Object.assign({}, state, {isAddedIn: true});
            case medCardType.DELETE_SUCCESS:
                return Object.assign({}, state, {
                    isAddedIn:            false,            medicationCard:
action.medicationCard});
            case medCardType.DELETE_FAILURE:
                return {
                    ...state,
                    isAddedIn: false,
                    messageError: action.error
                };

            case medCardType.GET_ALL_REQUEST:
                return Object.assign({}, state, {isAddedIn: true});
            case medCardType.GET_ALL_SUCCESS:
                return Object.assign({}, state, {
                    isAddedIn: false, cards: action.cards});
            case medCardType.GET_ALL_FAILURE:
                return {
                    ...state,
                    isAddedIn: false,
                    messageError: action.error
                };
            default:
                return state;
        }
    }
```

Файл index.js

```
import { combineReducers } from 'redux';
import authReducer from './authReducer';
import medicalCardReducer from './medicalCardReducer';
import userReducer from './userReducer';
```

```
export default combineReducers(
    {
        authReducer,
        medicalCardReducer,
        userReducer
    }
);
```

Файл CordovaCameraDriver.js

```
class CordovaCameraDriver {

    static getPicture (cameraSuccess, cameraError, cameraOptions) {
        window.navigator.camera.getPicture(cameraSuccess,   cameraError,
cameraOptions);
    }

    static destinationType () {
        return window.navigator.camera.DestinationType;
    }

    static pictureSourceType () {
        return window.navigator.camera.PictureSourceType;
    }

    static init () {
        const isPluginAvailable = window
            && window.navigator
            && window.navigator.camera;

        if (!isPluginAvailable) {
            throw new Error('Cordova plugin window.navigator.camera is
not found');
        }
    }
}

export default CordovaCameraDriver;
```

Файл WebCameraDriver.js

```
import mockImage from '../../rus.png';

class WebCameraDriver {

    static getPicture (getPhotoSuccess) {
        return getPhotoSuccess(mockImage);
    }

    static destinationType () {
        return { FILE_URI: null };
    }

    static pictureSourceType () {
        return { PHOTOLIBRARY: null };
    }

    static init () {
```

```
            return true;
        }
    }

    export default WebCameraDriver;
```

Файл AddMedicalCard.js

```
    import React, { Component, PropTypes } from 'react';
    import { connect } from 'react-redux';
    import { addMedCard } from '../../../actions/medicalCard';
    import BaseInput from '../../Inputs/BaseInput';
    import BaseComponent from '../../BaseComponent/BaseComponent';
    import Form from 'muicss/lib/react/form';
    import Textarea from 'muicss/lib/react/textarea';
    import Header from '../../Header/Header';
    import BaseButton from '../../buttons/BaseButton/BaseButton';
    import './AddMedicalCard.css';

    class AddMedicalCard extends BaseComponent {
        constructor(props) {
            super(props);

            this.handleSaveMedicalCard                                    =
this.handleSaveMedicalCard.bind(this);
        }

        handleSaveMedicalCard(event) {
            event.preventDefault();
            const { userId } = this.props;

            const doctor = {
                firstNameDoctor:
this.getNode(this.refs.firstNameDoctor).childNodes[0].value,
                lastNameDoctor:
this.getNode(this.refs.lastNameDoctor).childNodes[0].value,
                familyNameDoctor:
this.getNode(this.refs.familyNameDoctor).childNodes[0].value,
                specialization:
this.getNode(this.refs.specialization).childNodes[0].value
            };

            const medication = {
                nameOfMedication:
this.getNode(this.refs.nameOfMedication).childNodes[0].value,
                dateStarted:
this.getNode(this.refs.dateStarted).childNodes[0].value,
                stepTreatment:
this.getNode(this.refs.stepTreatment).childNodes[0].value,
                quantity:
this.getNode(this.refs.quantity).childNodes[0].value,
                countDays:
this.getNode(this.refs.countDays).childNodes[0].value
            };

            const payment = {
                cost: this.getNode(this.refs.cost).childNodes[0].value,
                payedUser:
this.getNode(this.refs.payedUser).childNodes[0].value,
                payedMedicalComp:
this.getNode(this.refs.payedMedicalComp).childNodes[0].value
            };
```

```
            const medicalCard = {
                dateOfVisit:
this.getNode(this.refs.dateOfVisit).childNodes[0].value,
                summary:
this.getNode(this.refs.summary).childNodes[1].value,
                recomendations:
this.getNode(this.refs.recomendations).childNodes[1].value,
                comments:
this.getNode(this.refs.comments).childNodes[1].value
            };

            this.props.dispatch(addMedCard(doctor,   medication,   payment,
medicalCard, userId));
        }

        render() {
            return (
                <div className="AddMedicalCard">
                    <Header   location={this.props.location}   title={'Add
Medical Card'} isMainPage={false}
                            handleLogout={   ()   =>   console.log('Doing
something') }
                    />
                    <Form className="AddMedicalCard__form">
                        <legend>Doctor Info</legend>
                        <BaseInput    type="text"    ref="firstNameDoctor"
hint="First Name Doctor" />
                        <BaseInput    type="text"    ref="lastNameDoctor"
hint="Last Name Doctor" />
                        <BaseInput   type="text"   ref="familyNameDoctor"
hint="Family Name Doctor" />
                        <BaseInput    type="text"    ref="specialization"
hint="Specialization" />

                        <legend>Medication Info</legend>
                        <BaseInput   type="text"   ref="nameOfMedication"
hint="Name Of Medication" />
                        <BaseInput      type="date"      ref="dateStarted"
placeholder="date Started" />
                        <BaseInput    type="text"    ref="stepTreatment"
hint="Step Treatment" />
                        <BaseInput       type="text"       ref="quantity"
hint="Quantity" />
                        <BaseInput  type="text"  ref="countDays"  hint="Count
Days" />


                        <legend>Payment Info</legend>
                        <BaseInput type="text" ref="cost" hint="Cost" />
                        <BaseInput  type="text"  ref="payedUser"  hint="Payed
User" />
                        <BaseInput   type="text"   ref="payedMedicalComp"
hint="Payed Medical Comp" />

                        <legend>MedicalCard Other Info</legend>
                        <BaseInput      type="date"      ref="dateOfVisit"
placeholder="Password (hint: password)" />

                        <Textarea ref="summary" hint="Summary" />
                        <Textarea ref="recomendations" hint="Recomendations"
/>
                        <Textarea ref="comments" hint="Comments" />
```

```
                    </Form>
                    <BaseButton         handler={this.handleSaveMedicalCard}
text={'Save Medical Card'}/>

                </div>
            );
        }
    }

    AddMedicalCard.contextTypes = {
        router: PropTypes.object.isRequired,
        store: PropTypes.object.isRequired
    };

    AddMedicalCard.propTypes = {
        user: PropTypes.string,
        loginError: PropTypes.object,
        dispatch: PropTypes.func.isRequired
    };

    function mapStateToProps(state) {
        const { authReducer } = state;
        if (authReducer.profile !== null) {
            return { userId: authReducer.profile._id};
        }
        else if (authReducer.user !== null) {
            return { userId: authReducer.user._id};
        }

        return { userId: null };
    }

    export default connect(mapStateToProps)(AddMedicalCard);
```

Файл home.js

```
    import React, { Component, PropTypes } from 'react';
    import { connect } from 'react-redux';
    import { logout } from '../../../actions/auth';
    import { getAllMedCard } from '../../../actions/medicalCard';
    import HorizontalBar from '../../HorizontalBar/HorizontalBar';
    import Header from '../../Header/Header';
    import NavigationService from '../../../services/NavigationService';
    import CameraService from '../../../services/CameraService';
    import { loadUserProfile }  from '../../../actions/utils';
    import Passcode from '../Passcode/Passcode';
    import './Home.css';

    class Home extends Component {
        constructor(props) {
            super(props);

            this.state = {
                showPassLock: true
            };
            this.handleGetAllMedicalCard                                    =
this.handleGetAllMedicalCard.bind(this);
            this.handleExportMedicalCard                                    =
this.handleExportMedicalCard.bind(this);
            this.getPhotoByCamera = this.getPhotoByCamera.bind(this);
```

```
        this.getPhotoFromAlbum = this.getPhotoFromAlbum.bind(this);
    }

    handleLogout () {
        const { user } = this.props;
        this.props.dispatch(logout(user));
        this.context.router.push('/login');
    }

    handleAddMedicalCard () {
        NavigationService.navigateTo('/add-medical-card');
    }

    handlePhoto () {
        NavigationService.navigateTo('/photo');
    }

    handleGetAllMedicalCard (event) {
        event.preventDefault();
        const { id } = this.props;

        this.props.dispatch(getAllMedCard(id, true));
    }

    handleExportMedicalCard (event) {
        event.preventDefault();
        const { id } = this.props;

        this.props.dispatch(getAllMedCard(id, false));
    }

    handleDone () {
        this.setState({ showPassLock: false });
    }

    componentWillMount () {
        const { user, passcode } = this.props;

        if (passcode !== null) {
            this.setState({ showPassLock: false });
        }

        if (user === null) {
            NavigationService.navigateTo('/login');
        }
    }

    getPhotoByCamera () {
        CameraService.getPhotoByCamera()
            .then(photoURI => {
                NavigationService.navigateTo(`/photo?src=${photoURI}`);
            });
    }

    getPhotoFromAlbum () {
        CameraService.getPhotoFromAlbum()
            .then(photoURI => {
                NavigationService.navigateTo(`/photo?src=${photoURI}`);
            });
    }

    render() {
        const { firstName, lastName, email} = this.props;
```

```
            return (
                <div className="Home">
                    <Header location={this.props.location} title={firstName}
isMainPage={true}
                            handleLogout={ () => this.handleLogout() }
                    />
                    <div className="Home__content">
                        <a  className="item-container  BaseRoundList__item-
container--1"    onClick={this.handleGetAllMedicalCard}>    <p>    Get    All
Card</p></a>

                        <a  className="item-container  BaseRoundList__item-
container--2" onClick={this.getPhotoByCamera}> <p> Photo import</p></a>

                        <a  className="item-container  BaseRoundList__item-
container--3"    onClick={this.handleExportMedicalCard}>    <p>Convert    to
PDF</p></a>

                        <a  className="item-container  BaseRoundList__item-
container--4" onClick={this.getPhotoFromAlbum}> <p> Image import</p></a>
                    </div>
                </div>
            );
        }
    }

    Home.propTypes = {
        // {
        //     this.state.showPassLock
        //         ? <Passcode handleDone={this.handleDone.bind(this)}/>
        //         : null
        // }
        user: PropTypes.string,
        dispatch: PropTypes.func.isRequired,
        location: PropTypes.object.isRequired
    };

    Home.contextTypes = {
        router: PropTypes.object.isRequired,
        store: PropTypes.object.isRequired
    };

    const mapStateToProps = (state) => {
        const { authReducer } = state;
        let passCode = null;
        if (authReducer.passcode !== null) {
            passCode =  authReducer.passcode;
        }
        if (authReducer.profile !== null) { //after Login
            return {
                firstName: authReducer.profile.firstName,
                email: authReducer.profile.email,
                lastName: authReducer.profile.lastName,
                familyName: authReducer.profile.familyName,
                gender: authReducer.profile.gender,
                dateOfBirth: authReducer.profile.dateOfBirth,
                id: authReducer.profile._id,
                passcode: passCode
            }
        }
        else if (authReducer.user !== null) { //after Reg
            return {
```

```
                firstName: authReducer.user.firstName,
                email: authReducer.user.email,
                lastName: authReducer.user.lastName,
                familyName: authReducer.user.familyName,
                gender: authReducer.user.gender,
                dateOfBirth: authReducer.user.dateOfBirth,
                id: authReducer.user._id,
                passcode: passCode
            }
        }
        else {
            return {
                user: null
            }
        }
    };

    export default connect(
        mapStateToProps
    )(Home);
```

Файл Card.js

```
    import React, { Component, PropTypes } from 'react';
    import { connect } from 'react-redux';
    import BaseComponent from '../../BaseComponent/BaseComponent';
    import Header from '../../Header/Header';
    import NavigationService from '../../../services/NavigationService';
    import BaseButton from '../../buttons/BaseButton/BaseButton';
    import { deleteMedCard } from '../../../actions/medicalCard';
    import Button from 'muicss/lib/react/button';
    import './Card.css';

    class Card extends BaseComponent {
        constructor(props) {
            super(props);
            this.itemNotParse = props.location.query.item;
            this.item = JSON.parse(this.itemNotParse) || null;

            this.onTapEditHandler = this.onTapEditHandler.bind(this);
            this.onTapDeleteHandler = this.onTapDeleteHandler.bind(this);
            this.handleExport = this.handleExport.bind(this);
        }

        onTapDeleteHandler () {
            this.props.dispatch(deleteMedCard(this.item._id));
            NavigationService.navigateTo('/');
        }

        onTapEditHandler () {
            NavigationService.navigateTo(`/edit-medical-
card?item=${this.itemNotParse}`);
        }

        handleExport () {
            const doc = new jsPDF();

            doc.setFontSize(20);
```

```
            doc.text(5,                    25,                    `Doctor:
${this.item.doctor.lastNameDoctor},${this.item.doctor.firstNameDoctor.charAt(
0)}.${this.item.doctor.familyNameDoctor.charAt(0)}.`);
            doc.text(5,                35,                `Specialization:
${this.item.doctor.specialization}`);
            doc.text(5, 45, `Date Visit: ${this.item.dateOfVisit}`);
            doc.text(5, 55, `Summary: ${this.item.summary}`);
            doc.text(5, 65, `Recommendations: ${this.item.recomendations}`);
            doc.text(5, 75, `-----------------------------------------------
------`);
            doc.text(5,                  105,                  `Medication:
${this.item.medications.nameOfMedication},                          step:
${this.item.medications.stepTreatment},                        quantity:
${this.item.medications.quantity},              count              days:
${this.item.medications.countDays}.`);

            doc.save('MedicalCard.pdf');
        }

        render() {
            return (
                <div className="Card">
                    <Header  location={this.props.location}  title={'Medical
Card Info'} isMainPage={false}
                            handleLogout={    ()    =>    console.log('Doing
something') }
                    />
                    <div className="Card__form-info">
                        <div className="Card__info-item">
                            <span       className="Card__item-title">Doctor:
</span>
                            <span                    className="Card__item-
value">{`${this.item.doctor.lastNameDoctor},${this.item.doctor.firstNameDocto
r.charAt(0)}.${this.item.doctor.familyNameDoctor.charAt(0)}.`}</span>
                        </div>

                        <div className="Card__info-item">
                            <span                    className="Card__item-
title">Specialization: </span>
                            <span                    className="Card__item-
value">{this.item.doctor.specialization}</span>
                        </div>

                        <div className="Card__info-item">
                            <span  className="Card__item-title">Date  Visit:
</span>
                            <span                    className="Card__item-
value">{this.item.dateOfVisit}</span>
                        </div>

                        <div className="Card__info-item">
                            <span      className="Card__item-title">Summary:
</span>
                            <span                    className="Card__item-
value">{this.item.summary}</span>
                        </div>

                        <div className="Card__info-item">
                            <span                    className="Card__item-
title">Recommendations: </span>
                            <span                    className="Card__item-
value">{this.item.recomendations}</span>
                        </div>
```

```
                        <div className="Card__info-item">
                            <span   className="Card__item-title">Medication:
</span>
                            <span                       className="Card__item-
value">{this.item.medications.nameOfMedication},</span>
                            <span classNAme="Card__item-title">date: </span>
                            <span                       className="Card__item-
value">{this.item.medications.dateStarted};</span>
                            <span classNAme="Card__item-title">step: </span>
                            <span                       className="Card__item-
value">{this.item.medications.stepTreatment},</span>
                            <span     className="Card__item-title">quantity:
</span>
                            <span                       className="Card__item-
value">{this.item.medications.quantity},</span>
                            <span  className="Card__item-title">count  days:
</span>
                            <span                       className="Card__item-
value">{this.item.medications.countDays}.</span>
                        </div>

                        <div className="Card__info-item">
                            <span     className="Card__item-title">Comments:
</span>
                            <span                       className="Card__item-
value">{this.item.comments}</span>
                        </div>
                    </div>


                    <Button   className="Card__delete-button"   variant="fab"
color="accent" onClick={this.onTapDeleteHandler}>D</Button>
                    <Button   className="Card__edit-button"   variant="fab"
color="primary" onClick={this.onTapEditHandler}>E</Button>

                    <BaseButton handler={this.handleExport} text={'Export to
file'}/>
                </div>
            );
        }
    }

    Card.contextTypes = {
        router: PropTypes.object.isRequired,
        store: PropTypes.object.isRequired
    };

    Card.propTypes = {
        user: PropTypes.string,
        loginError: PropTypes.object,
        dispatch: PropTypes.func.isRequired
    };

    function mapStateToProps(state) {
        const { authReducer } = state;
        if (authReducer.profile !== null) {
            return { userId: authReducer.profile._id};
        }
        else if (authReducer.user !== null) {
            return { userId: authReducer.user._id};
        }
```

```
    return { userId: null };
}

export default connect(mapStateToProps)(Card);
```

Файл UploadPhoto.js

```
import React, { Component, PropTypes }  from 'react';
import BaseComponent from '../../BaseComponent/BaseComponent';
import BaseButton from '../../buttons/BaseButton/BaseButton';
import Header from '../../Header/Header';
import ImageItem from '../../ImageItem/ImageItem';
import NavigationService from '../../../services/NavigationService';
import TextFormatorService from '../../../services/TextFormatorService';
import SpinnerService from '../../../services/SpinnerService';

import './UploadPhoto.css';

class UploadPhoto extends BaseComponent {
    constructor (props) {
        super(props);

        this.photoSrc = props.location.query.src;

        this.state = {
            messageInfo: ''
        };

        this.takeImport = this.takeImport.bind(this);
    }

    takeImport () {
        const self = this;
        const baseImage = new window.Image();
        baseImage.src = this.photoSrc;
        SpinnerService.show();
        baseImage.onload = function () {
            Tesseract.recognize(baseImage, { lang: 'eng' } )
                .then(data => {
                    const                medicalCard                =
TextFormatorService.getFormatedMedicalCardObject(data.lines);
                    NavigationService.navigateTo(`/edit-medical-
card?item=${JSON.stringify(medicalCard)}`);
                })
                .catch(() => self.setState({ messageInfo: 'Sorry. Not
parsed upload photo. Try again!' }))
                .then(SpinnerService.hide);
        }
    }

    render () {
        return (
            <div className='UploadPhoto'>
                <Header location={this.props.location} title={'Medical
Card Info'} isMainPage={false}
                        handleLogout={ () => console.log('Doing
something') }
                />
```

```
                    <ImageItem height="350" width="350" src={this.photoSrc}
ref="ImageItem" />

                    <p>{this.state.messageInfo}</p>

                    <BaseButton handler={this.takeImport} text={'Import to
MedCard'}/>

                </div>
            );
        }
    }

    UploadPhoto.contextTypes = {
        router: PropTypes.object.isRequired,
        store: PropTypes.object.isRequired
    };


    export default UploadPhoto;
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Исходный текст серверного приложения

Файл router.js

```
var express = require('express');
var router = express.Router();

var UserController = require('./server/controllers/userController');
var MedicineCardController = require('./server/controllers/MedicineCardController');

//Add Enable CORS
router.use(function(req, res, next) {
    res.header('Access-Control-Allow-Origin', '*');
    res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept');
    next();
});

router.post('/api/signup', UserController.register);
router.post('/api/login', UserController.login);
router.post('/api/profile', UserController.addProfile);
router.post('/api/logout', function(req, res) {
    res.status(200).json({ 'message': 'User logged out' });
});
router.post('/api/addchild', UserController.addChild);
router.post('/api/pass-check', UserController.checkPasscode);

router.post('/api/medcard', MedicineCardController.addNewCard);
router.post('/api/edit-medcard', MedicineCardController.editMedCard);
router.post('/api/delete-medcard', MedicineCardController.deleteMedCard);

router.post('/api/medicat', MedicineCardController.addMedication);
router.post('/api/diagnos', MedicineCardController.addDiagnostic);
router.post('/api/getucard', MedicineCardController.getAllMedCard);

module.exports = router;
```

Файл medicineCardController.js

```
var MedicalService = require('../services/medicalService');

var medicineCardController = {

    addNewCard: function(req,res) {
        MedicalService.addMedicalCard(req.body, function (err, data) {
            if (err || data.status === 500) {
                res.json({success: false, status: 500})
            }
            else {
                res.status(200).json(data);
            }
        });
    },

    editMedCard: function(req,res) {
```

```
            MedicalService.editMedicalCard(req.body, function (err, data) {
                if (err || data.status === 500) {
                    res.json({success: false, status: 500})
                }
                else {
                    res.status(200).json(data);
                }
            });
        },

        deleteMedCard: function(req,res) {
            var id = req.body.idMedCard;
            MedicalService.deleteMedicalCard(id, function (err, data) {
                if (err || data.status === 500) {
                    res.json({success: false, status: 500})
                }
                else {
                    res.status(200).json(data);
                }
            });
        },

        addMedication: function(req,res) {
            MedicalService.addNewMedication(req.body, function (err, data) {
                if (err || data.status === 500) {
                    res.json({success: false, status: 500})
                }
                else {
                    res.status(200).json(data);
                }
            });
        },

        addDiagnostic: function(req,res) {
            MedicalService.addDiagnostic(req.body, function (err, data) {
                if (err || data.status === 500) {
                    res.json({success: false, status: 500})
                }
                else {
                    res.status(200).json(data);
                }
            });
        },

        getAllMedCard: function(req,res) {
            MedicalService.getAllMedicalCard(req.body, function (err, data)
{
                if (err || data.status === 500) {
                    res.json({success: false, status: 500})
                }
                else {
                    res.status(200).json(data);
                }
            });
        }
    };

    module.exports = medicineCardController;
```

Файл userController.js

```
    var UserService = require('../services/userService');
```

```
var ValidationHelper = require('../helpers/validationHelper');

var userController = {

    register: function(req,res) {
        console.log(req.body);
        if (ValidationHelper.checkRegistrationData(req.body)) {
            UserService.registration(req.body, function (err, data) {
                if (err || data.status === 500) {
                    res.json({success: false, status: 500})
                }
                else {
                    res.status(200).json(data);
                }
            });
        }
        else {
            res.json({ success: false, status: 500, message: 'need more
info' })
        }
    },

    login: function(req, res) {
        if (ValidationHelper.checkLoginData(req.body)) {
            UserService.login(req.body, function (err, data) {
               if (err || data.status === 500) {
                   res.status(401).json({'message'      :      'Invalid
user/password'});
               }
               else {
                   res.status(200).json(data);
               }
            });
        }
        else {
            res.status(401).json({'message' : 'Invalid user/password'});
        }
    },

    addProfile: function(req, res) {
        if (ValidationHelper.checkProfileData(req.body.profile)) {
            UserService.addProfile(req.body.profile,   function   (err,
data) {
                if (err || data.status === 500) {
                    res.status(401).json({'message'      :      'Invalid
user/password'});
                }
                else {
                    res.status(200).json(data);
                }
            });
        }
        else {
            res.status(401).json({'message' : 'Invalid user/password'});
        }
    },

    addChild: function(req, res) {
        if (ValidationHelper.checkProfileData(req.body.profile)) {
            UserService.addChild(req.body.profile, function (err, data)
{
                if (err || data.status === 500) {
```

```
                        res.status(401).json({'message'      :      'Invalid
user/password'});
                    }
                    else {
                        res.status(200).json(data);
                    }
                });
            }
            else {
                res.status(401).json({'message' : 'Invalid user/password'});
            }
        },

        checkPasscode: function (req, res) {
            UserService.checkPasscode(req.body, function (err, data) {

                console.log(data);

                if (err || data.status === 500) {
                    res.status(401).json({'message'        :        'Invalid
user/password'});
                }
                else {
                    res.status(200).json(data);
                }
            });
        }
    };

    module.exports = userController;
```

Файл cryptoHelper.js

```
    var crypto = require('crypto');
    var config = require('../config/config');

    var cryptoHelper = {};

    cryptoHelper.encrypt = function(text) {

        var cipher = crypto.createCipher('aes-256-cbc', config.secretKey);
        var crypted = cipher.update(text, 'utf8', 'hex');
        crypted += cipher.final('hex');
        return crypted;

    };

    cryptoHelper.decrypt = function(text) {

        var      decipher      =      crypto.createDecipher('aes-256-cbc',
config.secretKey);
        var dec = decipher.update(text, 'hex' ,'utf8');
        dec += decipher.final('utf8');
        return dec;

    };

    cryptoHelper.createHash = function(text, salt) {

        salt = salt ? salt : '';
        return crypto.createHash('md5').update(text + salt).digest('hex');
```

```
};

module.exports = cryptoHelper;
```

Файл validationHelper.js

```
var validationHelper = {

    checkRegistrationData: function(regData) {

        if(!regData.email         ||         !regData.password         ||
!regData.confirmPassword) {
            return false;
        }

        if(regData.password === regData.confirmPassword) {
            return true;
        }

    },

    checkLoginData: function(reqData) {
        if (!reqData.email || !reqData.password) {
            return false;
        }

        return true;
    },

    checkProfileData: function(reqData) {
        if      (!reqData.firstName      ||      !reqData.lastName      ||
!reqData.familyName || !reqData.gender || !reqData.dateOfBirth) {
            return false;
        }

        return true;
    }
};

module.exports = validationHelper;
```

Файл doctorModel.js

```
var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var DoctorSchema = new Schema({
    _id: Schema.Types.ObjectId,
    firstNameDoctor: String,
    lastNameDoctor: String,
    familyNameDoctor: String,
    specialization: String
});

module.exports = mongoose.model('Doctor', DoctorSchema);
```

Файл MedicalCardModel.js

```
var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var MedicalCardSchema = new Schema({
    userId: String,
    dateOfVisit: Date,
    summary: String,
    recommendations: String,
    comments: String,
    doctor: { type: Number, ref: 'Doctor' },
    medications: { type: Number, ref: 'Medication' },
    payment: { type: Number, ref: 'Payment' },
});


module.exports = mongoose.model('MedicalCard', MedicalCardSchema);
```

Файл userModel.js

```
var UserModel = require('../models/user');
var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var UserSchema = new Schema({
    email: String,
    passwordHash: String,
    passwordSalt: String
});

module.exports = mongoose.model('User', UserSchema);
```

Файл profileModel.js

```
var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var ProfileSchema = new Schema({
    _id: Schema.Types.ObjectId,
    email: String,
    firstName: String,
    lastName: String,
    familyName: String,
    gender: String,
    dateOfBirth: Date,
    iDMedCard: String,
    isChild: Boolean,
    parent: Schema.Types.ObjectId
});

module.exports = mongoose.model('Profile', ProfileSchema);
```