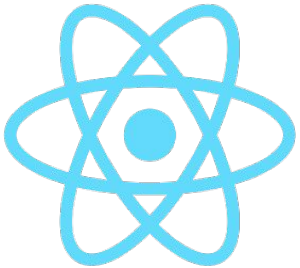
Messenger



Андреев Сергей, 444

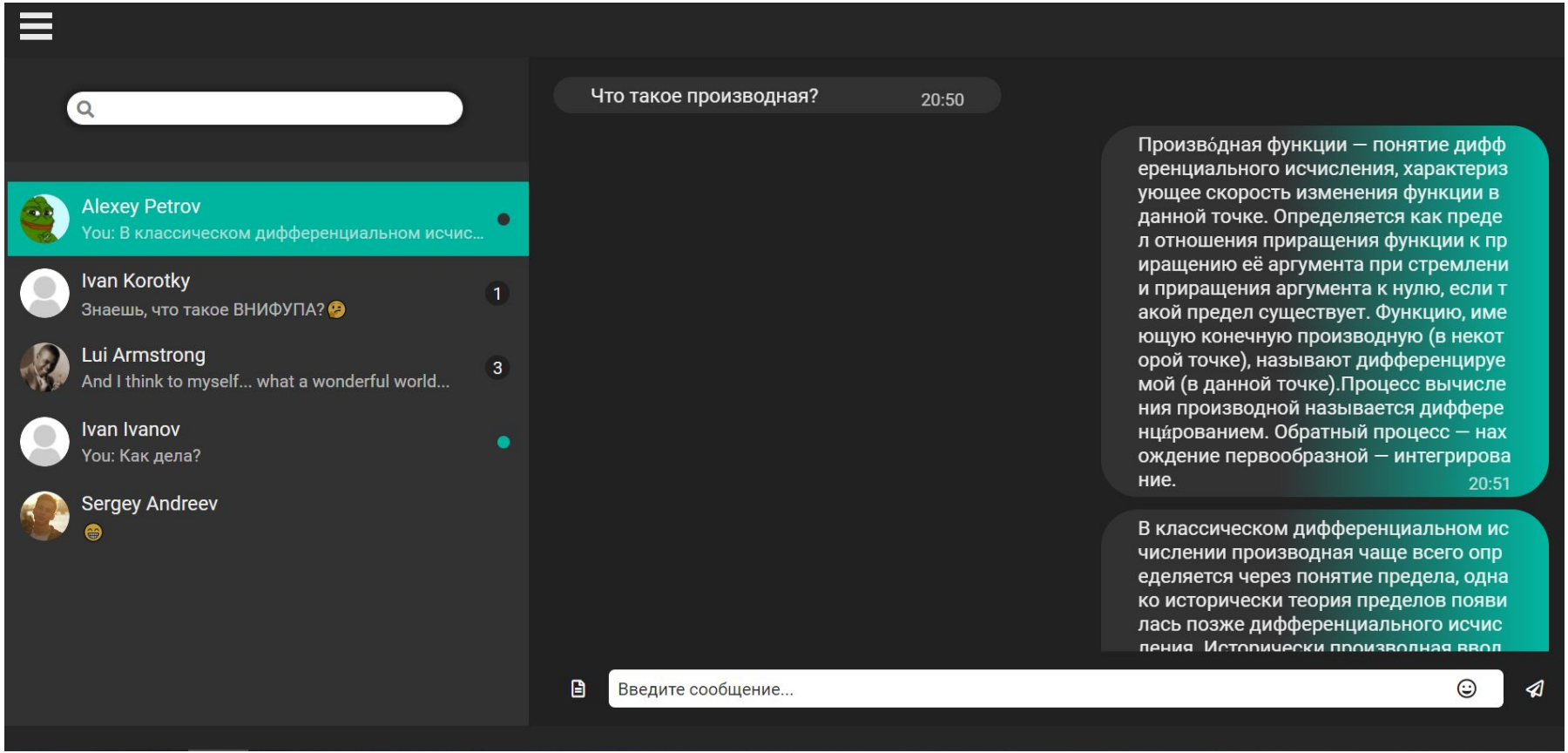
Требования

- Возможность создавать и изменять аккаунт, добавлять аватар
- Возможность находить другие аккаунты по их тегу и отправлять им сообщения
- Отображение списка диалогов и ленты сообщений с выбранным пользователем
- Отображение статуса сообщений (прочитано/не прочитано)
- Отображение статуса пользователей (онлайн, оффлайн)
- Изменение состояния (списка диалогов, сообщений) в реальном времени

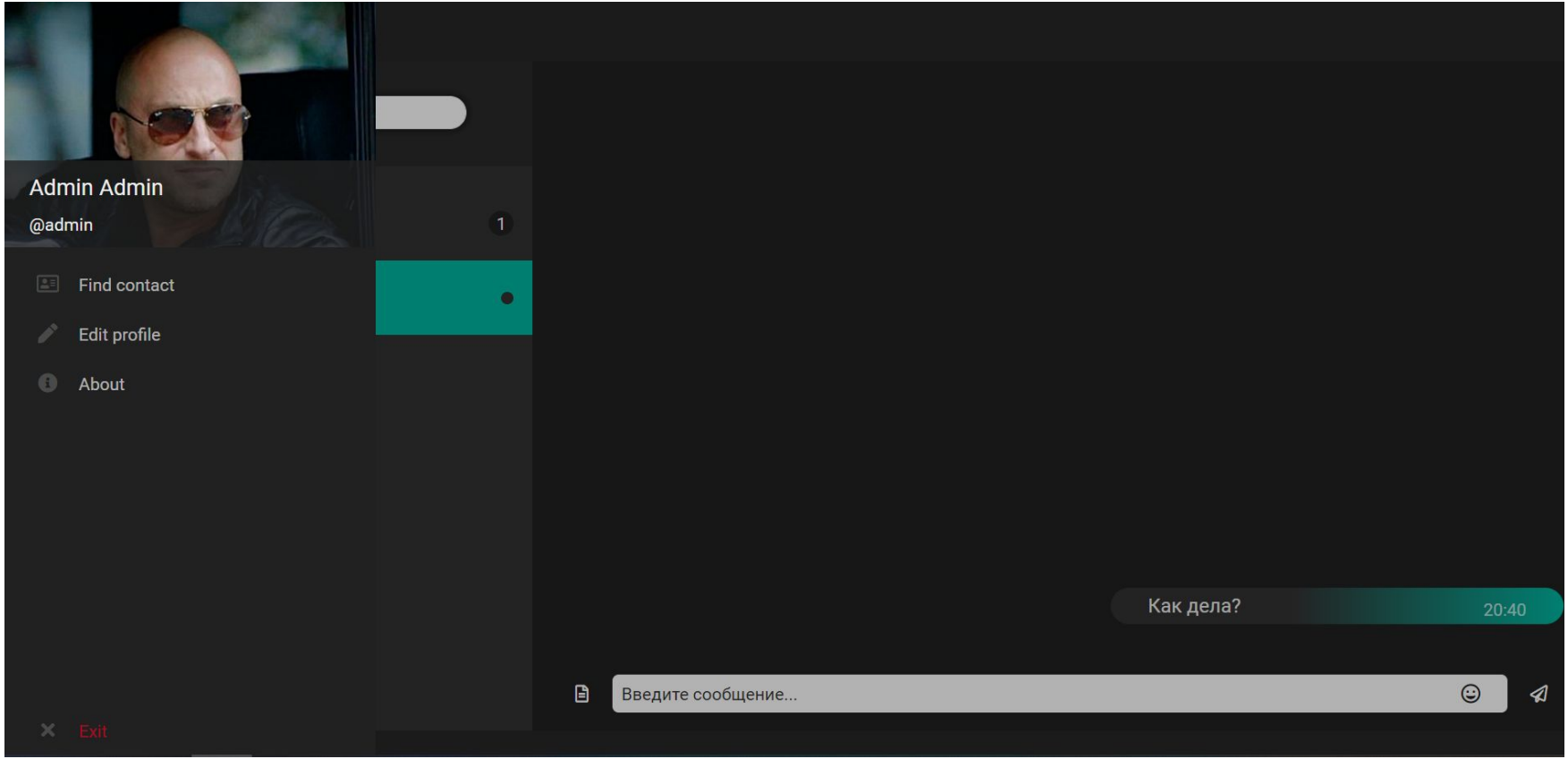
Демо

http://**188.166.100.185/**

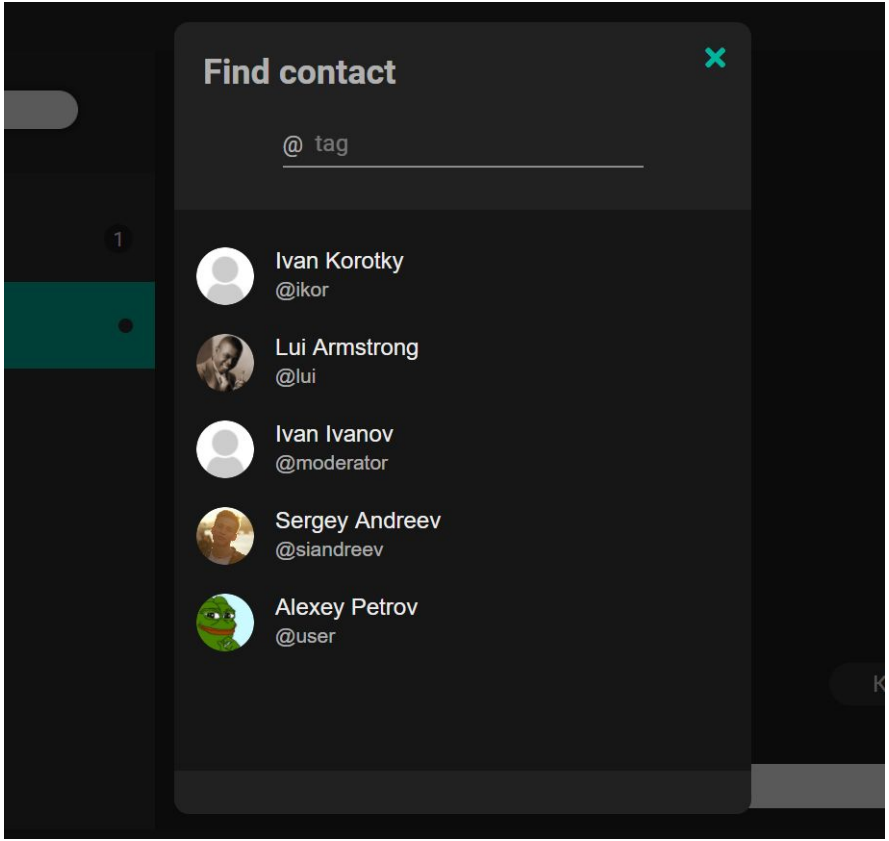
Результат: основное окно

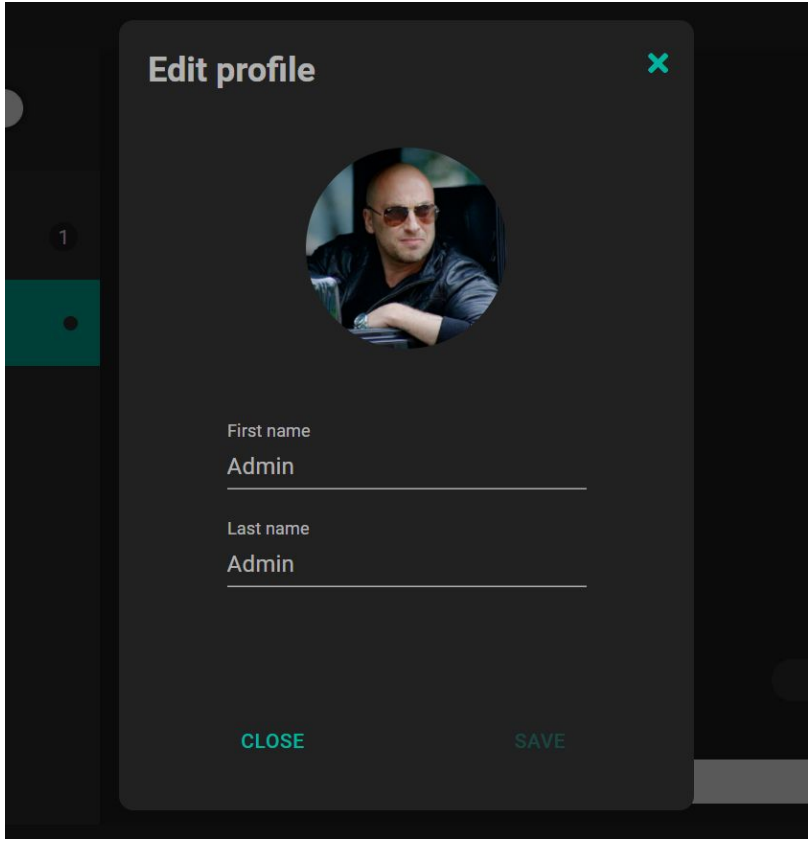


Результат: боковое меню

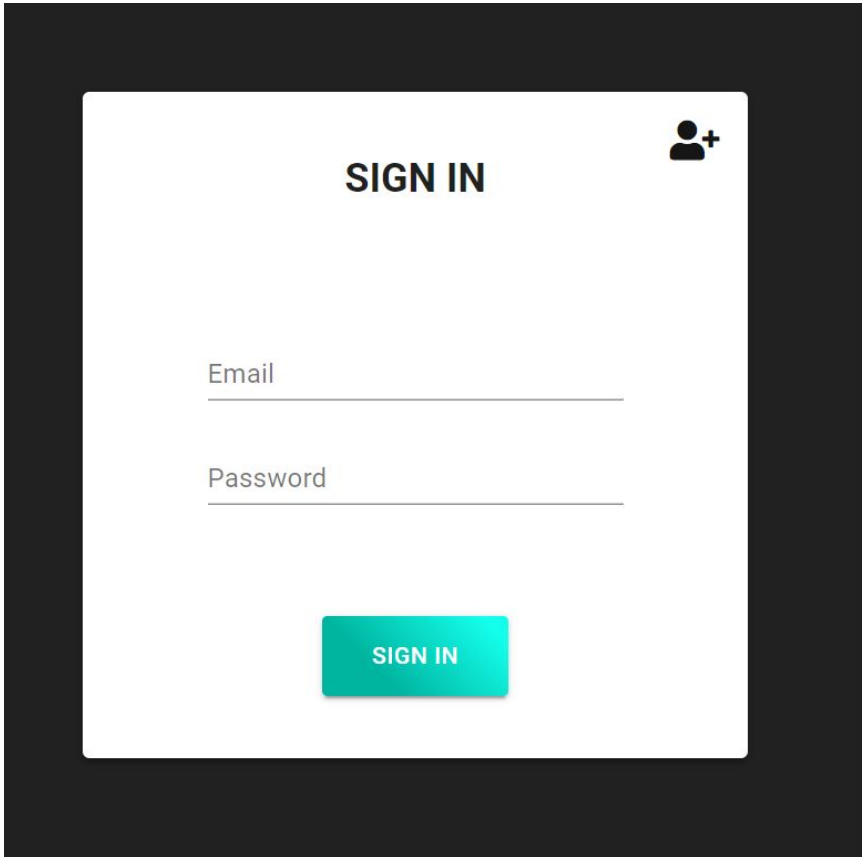


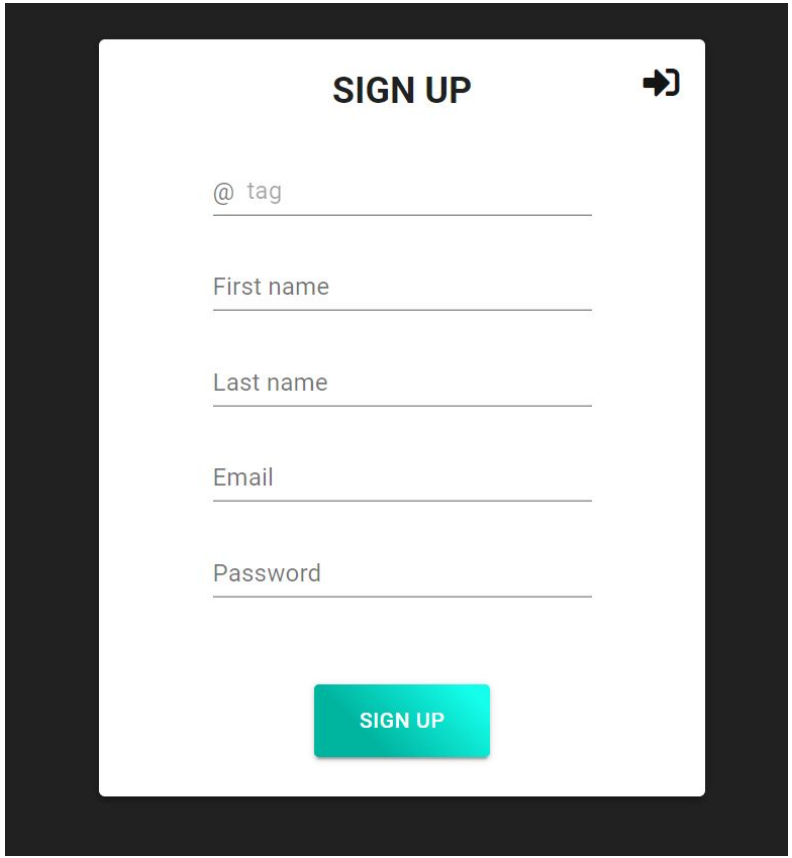
Результат: модальные окна



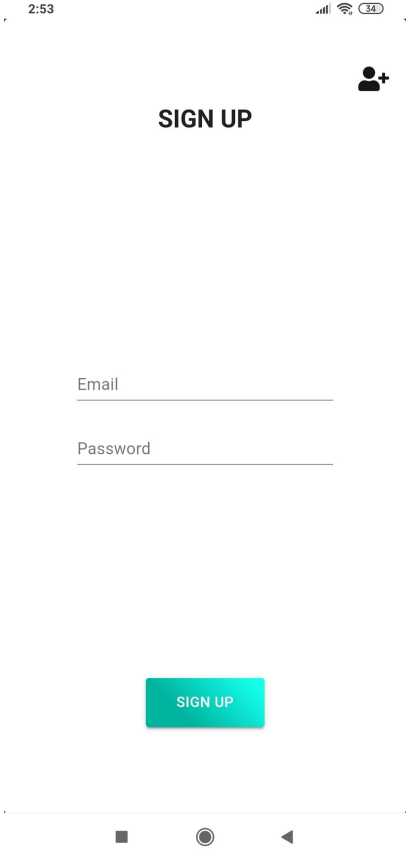


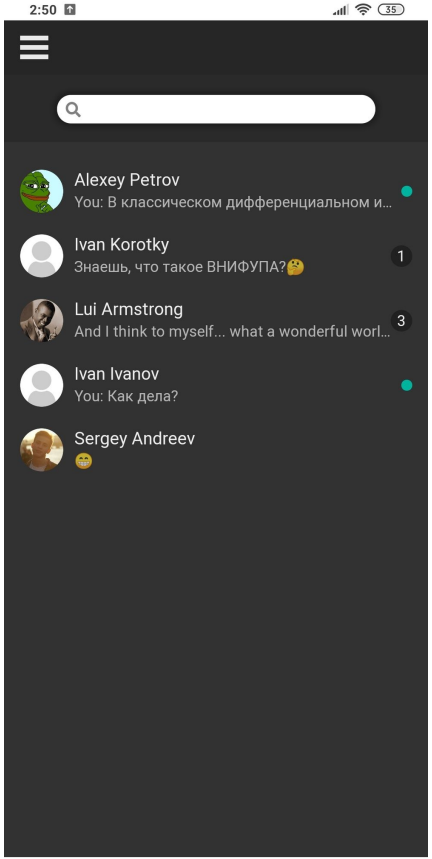
Результат: вход и регистрация

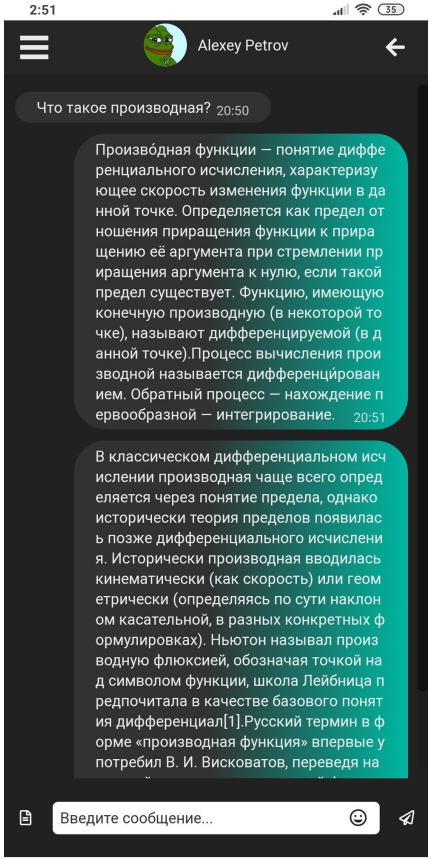


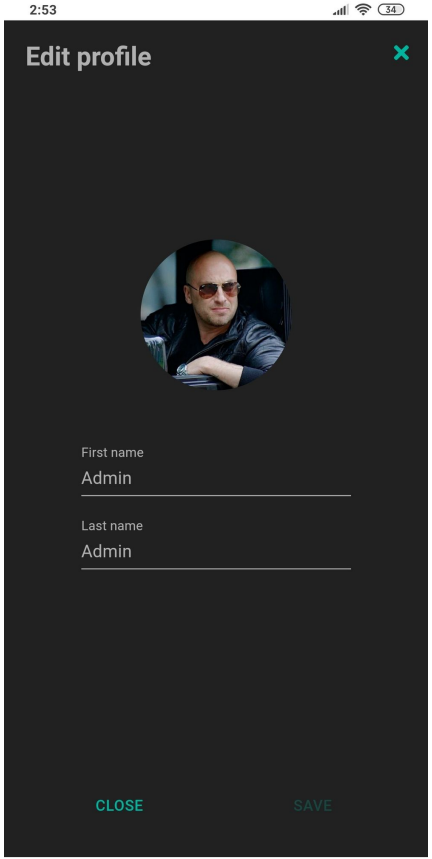


Результат: мобильная версия



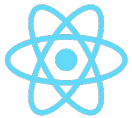






Стек технологий

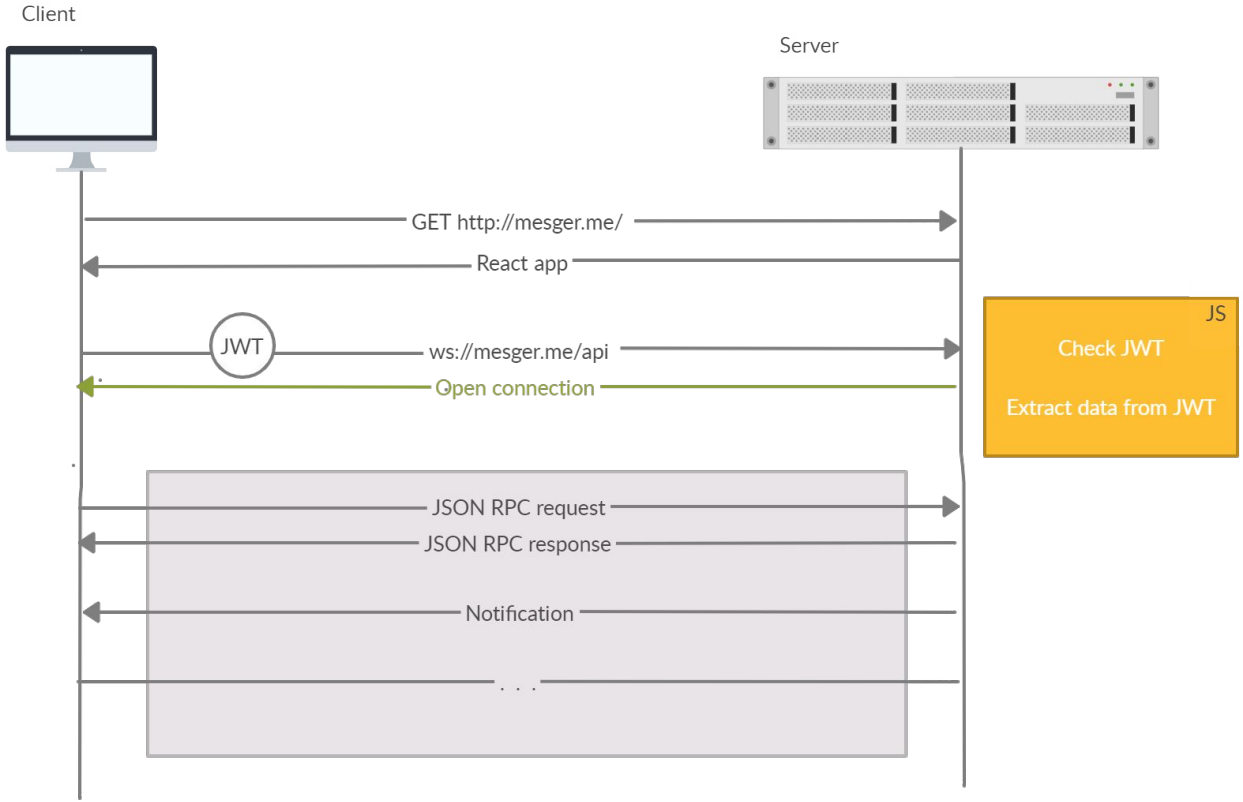
Клиент:

- React 
- Redux 
- Material UI 
- SCSS 

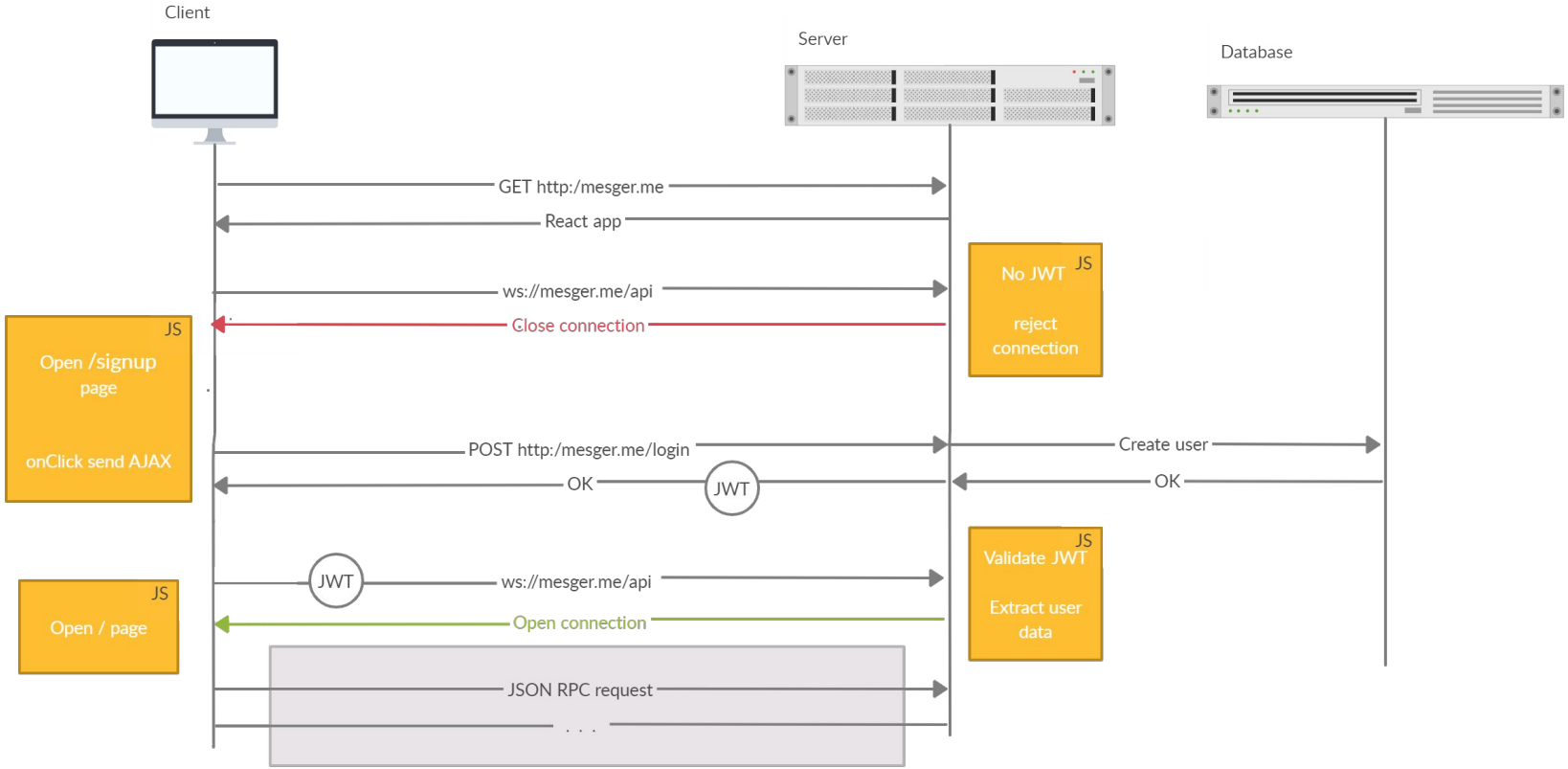
Сервер:

- Node js 
- Express 
- MongoDB 
- WebSocket
- JWT

Алгоритм сетевого взаимодействия: пользователь зарегистрирован



Алгоритм сетевого взаимодействия: пользователь не зарегистрирован



Node js

Node js -- среда исполнения JS на основе Chrome V8

- Быстрое получение MVP
- Хорошая масштабируемость
- Нет потоков => пропускная способность в сотни раз выше
- Единый язык на клиенте и на сервере

Express

- Горизонтальное масштабирование (легко добавлять новые пути)
- Вертикальное масштабирование (легко добавлять middleware к существующим путям)
- Множество модулей для работы с куки, телом post запроса, вебсокетами

```
import Express from "express";

const app = new Express();

app.get("/app/path" , (req : Request<P, ResBody, ReqBody, ReqQuery> , res : Response<ResBody> ) => {
    res.send( body: "Hello world!")
})

app.use('/app', (req, res, next) => {
    console.log('Request Type:', req.method);
    next();
});

app.listen( port: 8080);
```

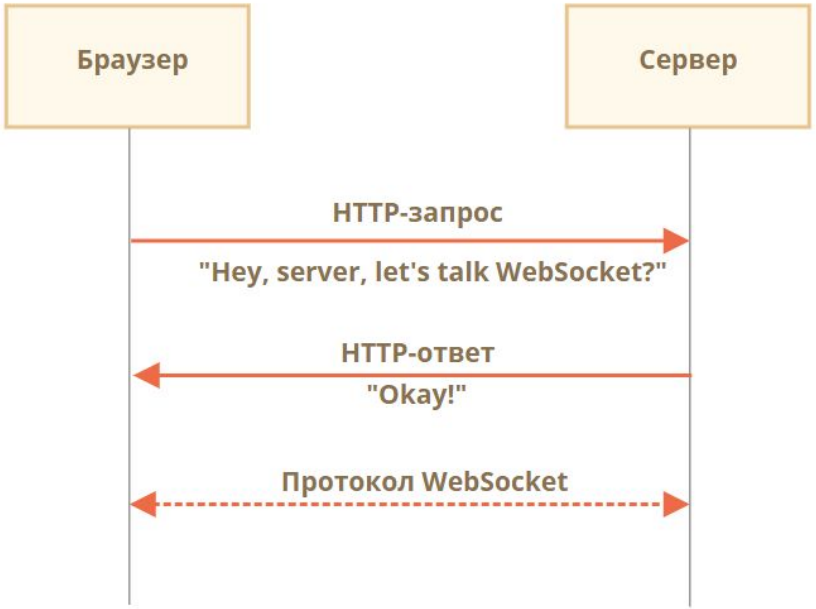
JWT

Json web token -- способ пользователя подтверждать свою личность в каждом запросе к api


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e
yJfaWQiOiI1ZjdiMTA2YmJlZDI4ZjI0YzA4NGM
0NDAiLCJ0YWciOiJAYWRtaW4iLCJlbWFpbCI6I
mFkbWluQG1lc2dlci5tZSJ9.l6MFulCbzi_wFw
cTOMIrQMJxPXXGH6CbObBGGNMFLAs
```

- Сервер получает логин и пароль, сверяет хеш пароля с данными в бд
  - При успешной авторизации создает header и payload JWT
  - Кодирует их с base64
  - Подписывает JWT хешем от закодированных данных и приватного ключа
- При api запросе сервер берет хеш от пришедших header, payload и своего приватного ключа. Если хеш совпадает с подписью, JWT подлинный

WebSocket



HTTP-заголовок запроса

```
GET /api
Host: mesger.me
Origin: http://mesger.me
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Key: Iv8io/9s+lYFgZWcXczP8Q==
Sec-WebSocket-Version: 13
```

HTTP-заголовок ответа

```
101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: hsBlbuDTkk24srzEOTBUlZAlC2g=
```

WebSocket: client

```
let clientWS = new WebSocket( url: "ws://mesger.me/api");

clientWS.onopen = function(e : Event ) {
    clientWS.send( data: "Hi server!");
};

clientWS.onmessage = function(event : MessageEvent ) {
    console.log(`[message] ${event.data}`);
};

clientWS.onclose = function(event : CloseEvent ) {
    if (event.wasClean) {
        alert(`[close] Connection closed clearly, code=${event.code} reason=${event.reason}`);
    } else {
        alert('[close] Connection refused');
    }
};
```

WebSocket: server

```
import Express from "express";
import EnableWebSocket from 'express-ws';

const app = new Express();
EnableWebSocket(app);

app.ws( route: '/api', middlewares: function(ws : WebSocket , req : Request ) {
    console.log("socket opened");
    ws.on( event: 'message', listener: function(msg : Data ) {
        ws.send( data: "hello from server!");
    });
});
```

React

- Иной подход к созданию веб-приложений, основанный на AJAX и рендеренге на стороне клиента
  - (Возможен серверный рендеринг, например с Next.js)
- Ускорение рендеринга (виртуальный DOM)
- Разделение кода не по технологиям, а по отдельным компонентам, инкапсулирующим в себе свою разметку, логику и стили
- Возможность повторного использования компонентов
- Обеспечение стабильного и чистого кода
- Простота отладки и хорошая поддерживаемость
- Легкая масштабируемость

React: props, state, жизненный цикл

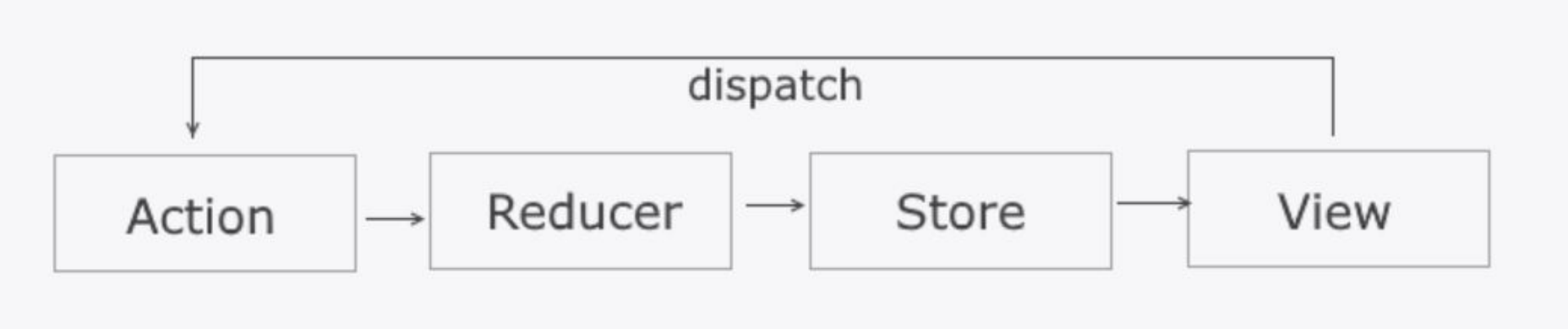
```
import React from "react";
import ReactDOM from "react-dom"

class Timer extends React.Component {
    constructor(props) {
        super(props);
        this.state = { seconds: 0 };
    }
    tick() {
        this.setState( state: state => ({
            seconds: state.seconds + 1
        }));
    }
    componentDidMount() {
        this.interval = setInterval( handler: () => this.tick(),  timeout: 1000);
    }
    componentWillUnmount() {
        clearInterval(this.interval);
    }
    render() {
        return (<div>Секунды: {this.state.seconds}</div>
        );
    }
}
```

```
ReactDOM.render(
    <Timer />,
    document.getElementById( elementId: 'timer-example')
);
```

Redux

- Нескольким компонентам требуется одна и та же часть состояния приложения.
- Слишком много свойств передается через несколько иерархий компонентов
- Управление состоянием с помощью setState раздувает компонент



Material UI

```
import React from 'react';
import ReactDOM from 'react-dom';
import Button from '@material-ui/core/Button';

function App() {
  return (
    <Button variant="contained" color="primary">
      Hello World
    </Button>
  );
}

ReactDOM.render(<App />, document.querySelector('#app'));
```

Material UI: JSS

```
import React from 'react';
import { styled } from '@material-ui/core/styles';
import Button from '@material-ui/core/Button';

const MyButton = styled(Button)({
  background: 'linear-gradient(45deg, #FE6B8B 30%, #FF8E53 90%)',
  border: 0,
  borderRadius: 3,
  boxShadow: '0 3px 5px 2px rgba(255, 105, 135, .3)',
  color: 'white',
  height: 48,
  padding: '0 30px',
});

export default function StyledComponents() {
  return <MyButton>Styled Components</MyButton>;
}
```

SCSS

SCSS -- препроцессор для CSS, диалект языка SASS

- переменные
- импорты
- вложенность
- математические выражения
- сложение строк, поддержка вставки в строку
- функции, условия, циклы

SCSS: пример

```scss
/* variables.scss */

$primary: #217a4e;
$secondary: blue;
$height: 120px;
```

```scss
/* app.scss */

@import "./variables";

$local-height: 40px;

.block {
  height: ($height / 2) + $local-height;
  background-color: $secondary;

  &__element {
      color: $primary;

    &:hover {
      color: black;
    }
  }
}
```