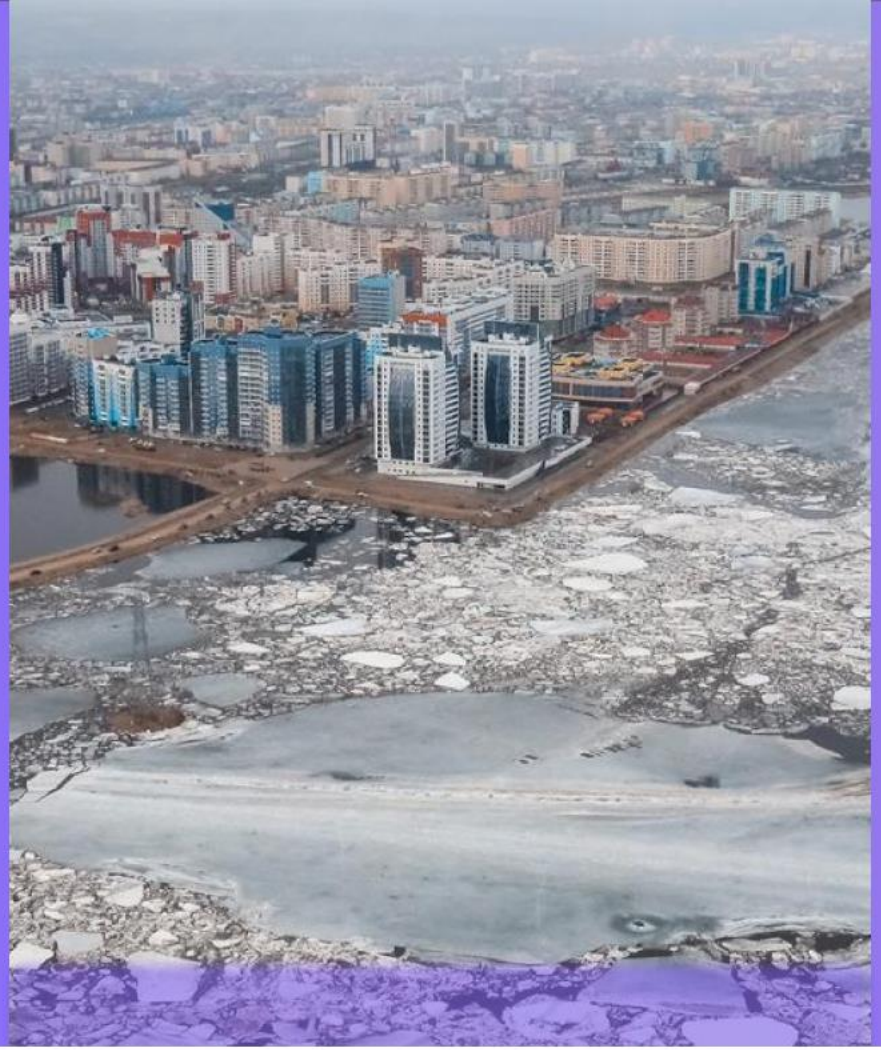
# Разработка скрипта обработки данных и прогнозирования паводковых ситуаций

**Выполнили:**
Гоголев Валерий Геннадиевич, 11 класс СУНЦ

**Научиный руководитель:**
Захаров Дьулустаан Семенович
Аспирант ИМИ, сотрудник ДНК

# Актуальность

Одной из главных проблем Якутии являются паводки. Ежегодно от весеннего и летнего паводков страдают несколько районов республики. Зачастую это связано с отсутствием своевременного оповещения о возможных происшествиях, что позволило бы организовать эвакуацию населения и материальных ценностей и существенно уменьшило бы ущерб от паводка.

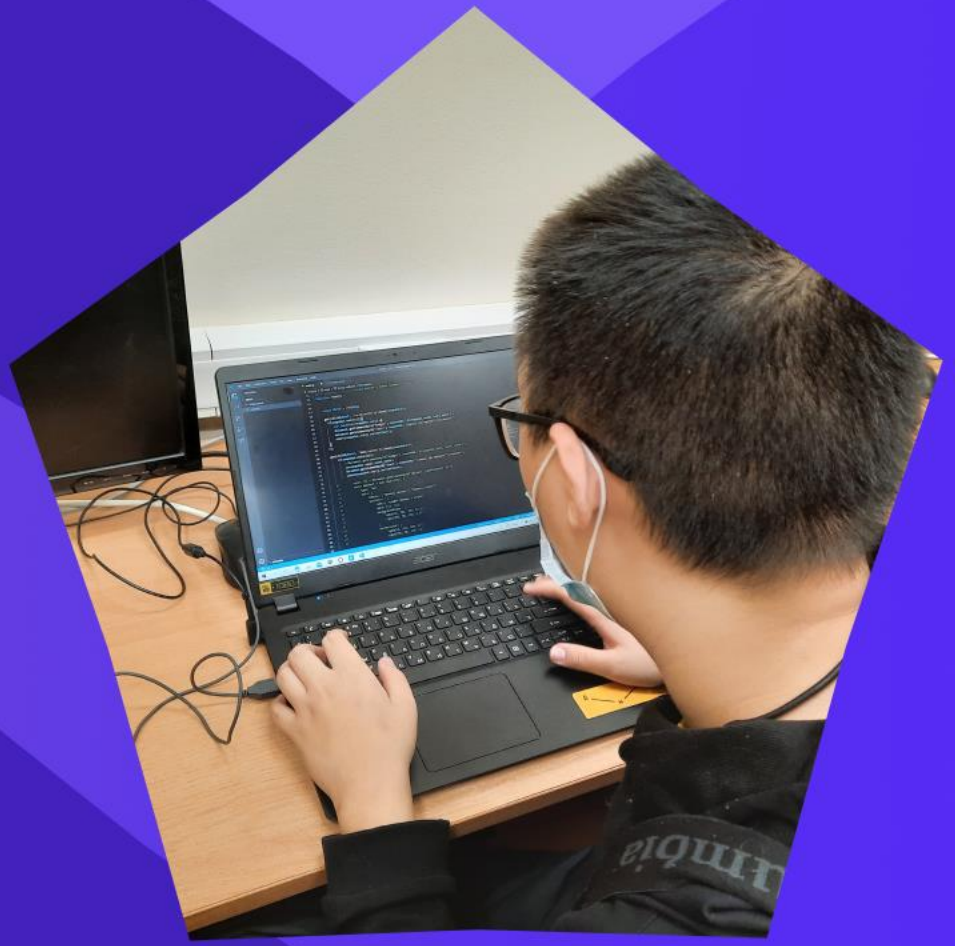


**1.186 млрд**

рублей ущебра из-за павдоков за 2018 год в Якутии

**990 млн**

рублей ущебра из-за павдоков за 2020 год в Якутии



# Цели

Разработать скрипт веб платформы для обработки данных и прогнозирования паводковых ситуаций.

# Задачи

| | | |
|---|---|---|
| 1.Рассмотреть методы прогнозирования | 2. Изучить облачную базу Google Firebase | 3. Изучить фреймворк Express, Nodejs |
| 4. Разработать функцию чтения данных из базы и вывода статистических данных | 5. Разработать функцию чтения данных с Firestore и алгоритм прогнозирования достижения критического уровня воды | 6. Реализовать уведомление по почте при достижении критического уровня |

# Способы прогнозированний

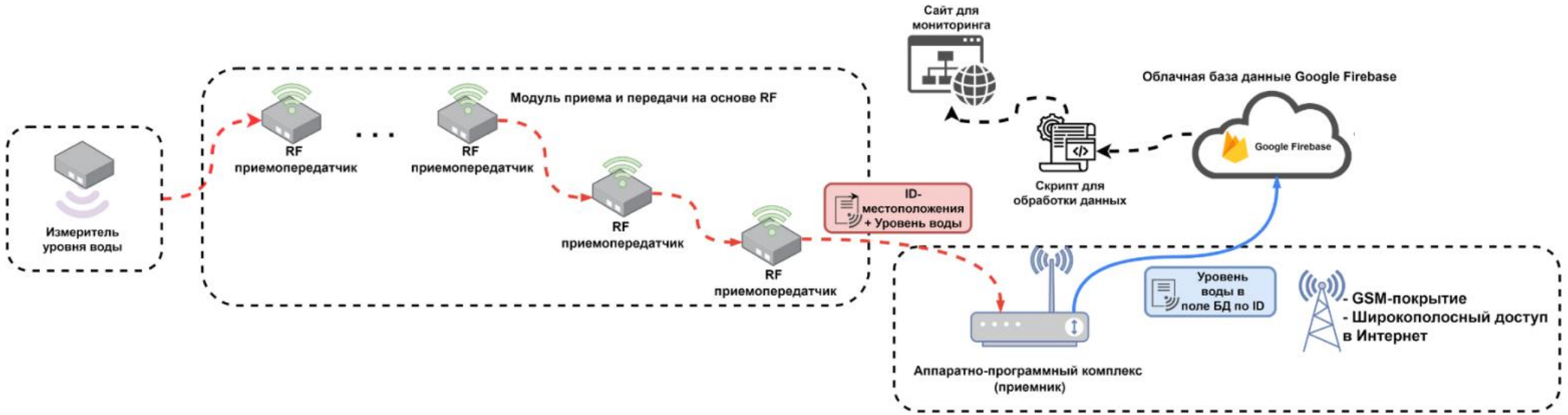
1.**Гидрологические модели**, на основе данных о погоде, о топографии и о географии для оценки объема воды

2.**Статистические модели -** исторические данные о паводках, чтобы предсказать вероятность возникновения паводка в будущем

3. **Моделирование с помощью искусственного интеллекта -** анализ больших данных и обучение на этой основе информации для создания прогнозов

4. **Комбинированные модели -** объединяют данные из различных источников, включая гидрологические и метеорологические данные, для улучшения точности прогнозирования паводков.

# Структурная схема



# Чтение из Realtime

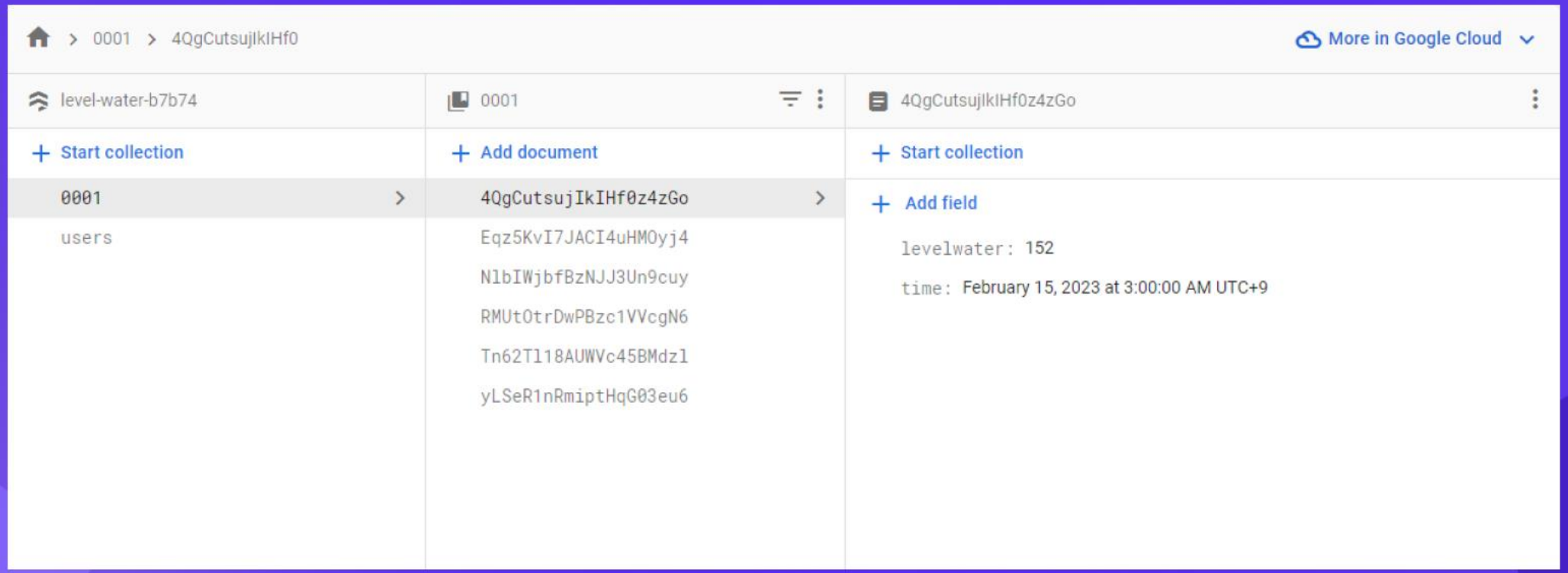
```
const locations = [
    {
        name: title001 && title001.val(),
        location: {
            lat: lat001 && lat001.val(),
            lng: lon001 && lon001.val()
        },
        levelwater: levelwater001 && levelwater001.val(),
        levelcrit: levelcrit001 && levelcrit001.val(),
    },
```

Объявляются соответствующие переменные, переменным присваивается значение из базы данных, и далее информация отображается в веб-платформе

```
const [levelwater002] = useObject(ref(db, 'id/002/levelwater'));
const [levelcrit002] = useObject(ref(db, 'id/002/levelcrit'));
const [lat002] = useObject(ref(db, 'id/002/lat'));
const [lon002] = useObject(ref(db, 'id/002/lng'));
const [title002] = useObject(ref(db, 'id/002/title'));
```

# Firestore

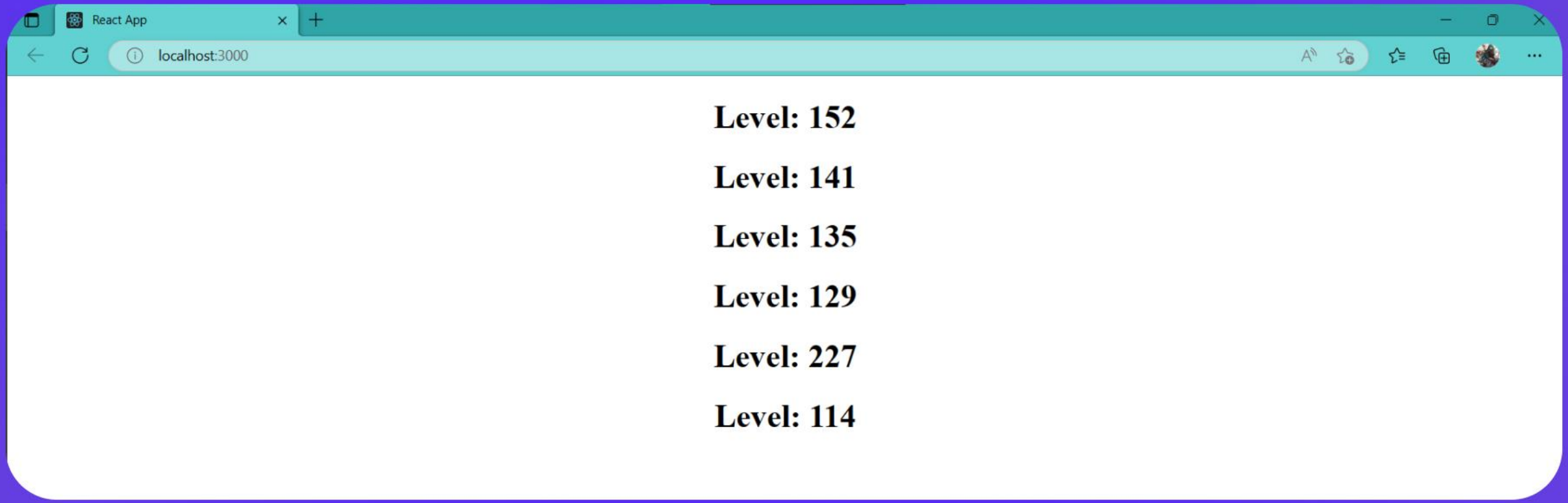
Firestore - db, состоит из ответвлений



# Чтение из данных

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getDatabase } from "firebase/database"


// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is opt
const firebaseConfig = {
    apiKey: "AkezP_5fH6hpRGXOGkJmZ8",
    authDomain: "level-water-b7b74.firebaseapp.com",
    databaseURL: "https://level-water-b7b74-default-rtdb.firel
    projectId: "level-water-b7b74",
    storageBucket: "level-water-b7b74.appspot.com",
    messagingSenderId: "437701271434",
    appId: "1:437701271434:web:ebbeeab5b2d577bb557273",
    measurementId: "G-BEXN3CZFCQ"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
export const db = getDatabase(app);
```

```
function App() {
  const [levelwaters, setLevels] = useState([])
  const levelsCollectionRef = collection(db, "0001")
  var arr = []

  useEffect(()=>{
    const getLevels = async () => {
      const data = await getDocs(levelsCollectionRef);
      setLevels(data.docs.map((doc) => ({ ...doc.data(), id: doc.id})));
    }
    getLevels()
  }, [])

  return (
    <div className="App">
      {
        levelwaters.map((level)=>{
          arr.push(level.levelwater)
          console.log(arr)
          return <div>
            <h1> Level: {level.levelwater} </h1>
          </div>
        })
      }
    </div>
  )
}
```




# Наглядный пример выведения данных

Расчет каждые 2 часа (утро, день, вечер)


# Алгоритм прогнозирования изменения уровня воды

# Уведомление

```
const nodemailer = require('nodemailer')

const transporter = nodemailer.createTransport(
    {
        host: 'smtp.gmail.com',
        port: 587,
        secure: false,
        // port 465 true
        auth: {
            user: 'levelwatermeter@gmail.com',
            pass: 'XPgD6LrPLd4G-3'
        }
    },
    {
        from: 'Mailer Test <levelwatermeter@gmail.com>',
    }
)

const mailer = message => {
    transporter.sendMail(message, (err, info)  => {
        if(err) return console.log(err)
        console.log('Email sent: ', info)
    })
}

module.exports = mailer
```

При достижении критического уровня воды будет отправлена информация спец. службам, а для удобного и быстрого оповещения населения было разработано мобильное приложение и также оповещение по почте. Также для мониторинга уровня воды разработан сайт.

# Реализация уведомления

Модуль NodeMailer

```
const express = require('express')
const bodyParser = require('body-parser')
const mailer = require('./nodemailer')

const app = express()

const PORT = 3001
let user = undefined

app.use('/css', express.static(__dirname + '/node_modules/bootstrap/dist/css'))
//
app.use(bodyParser.urlencoded({ extended: false }))
app.post('/registration', (req, res) => {
    if(!req.body.email || !req.body.pass) return res.sendStatus(400)
    const message = {
        to: req.body.mail,
        subject: 'Congratulations! You are successfully registered on our site',
        text: `С успешной регистрацией!
        Вашие данные для учетной записи:
        Логин: ${req.body.email}
        Пароль: ${req.body.pass}

        Данное письмо не требует ответа
        `
    }
    mailer(message)
    user = req.body
    res.redirect('/registration')
})
app.get('/registration', (req, res) => {
    if(typeof user !== 'object') return res.sendFile(__dirname + '/registration.html')
    res.send('Регистрация прошла успешно! Данные учетной записи отправлена на email: ${user.mail}')
    user = undefined
})

app.listen(PORT, () => console.log('server listening at https://localhost:${PORT}/registration'))
```

# Заключение

1.Рассмотрены методы прогнозирования, для наилучшего и оптимального оповещения.

2.Изучена облачная база Google Firebase.

3.Изучены фреймворки Express Node.js, npm модули, такие как NodeMailer, Firebase и внутренние модули React.

4.Разработана функция чтения данных из базы Realtime и вывода статистических данных на веб-платформу.

5.Разработана функция чтения данных с Firestore и выведен алгоритм прогнозирования достижения критического уровня воды.

6.Реализовано уведомление по почте при достижении критического уровня.

# Планы на будущее

1. В будущем, планируется тестирование Искусственного интеллекта для прогнозирования паводковых ситуаций, и будут оптимизированы и улучшены расчеты данных.
2. Будут добавлены датчики термометра, влажности.
3. Рассмотрение осадков, выпадающих в реальном времени, скорость изменения речной стадии, знания о типах шторма, о характеристики дренажного бассейна реки, таких как почвенно-влажностные условия, температура грунта, снежный покров, топография, растительный покров и непроницаемая площадь суши, которые могут помочь предсказать, насколько обширным и разрушительным может стать наводнение.