

Веб-решение для автоматизации учета рабочего времени и координации задач на предприятиях

Н.А. Епрынцева

*Воронежский филиал «Российский экономический университет
имени Г.В. Плеханова», г. Воронеж*

Аннотация: В статье рассматриваются актуальные вопросы автоматизации управления бизнес-процессами посредством разработки специализированного веб-приложения для планирования задач и учета рабочего времени. Исследование направлено на решение проблемы оптимизации организационных процессов в условиях цифровой трансформации современного предприятия. Автором предложена архитектура информационной системы, основанная на использовании современных веб-технологий: клиентская часть реализована с применением фреймворка React, серверная - на платформе Node.js с использованием Express.js, для хранения данных выбрана реляционная СУБД MySQL. Особое внимание уделено реализации механизма управления состоянием приложения с использованием библиотеки Effector, что обеспечивает существенное повышение производительности за счет минимизации избыточных ререндеров интерфейса.

Ключевые слова: автоматизация бизнес-процессов, управление задачами, учет рабочего времени, веб-приложение, React, Node.js, Effector, производительность информационных систем.

Введение. В условиях цифровой трансформации эффективное управление бизнес-процессами и контроль выполнения задач становятся ключевыми факторами успеха любого предприятия. Автоматизация планирования задач и учета рабочего времени сотрудников позволяет не только повысить производительность труда, но и улучшить координацию между подразделениями, обеспечивая прозрачность процессов. Современные информационные системы, такие как веб-приложения, предоставляют инструменты для автоматизации распределения задач, мониторинга их выполнения, анализа рабочего времени и оптимизации ресурсов. В настоящее время, предприятия используют информационные системы для оптимизации планирования задач и учета рабочего времени. Такие системы позволяют:

1. Эффективно распределять задачи – автоматизация планирования обеспечивает равномерную загрузку сотрудников и своевременное выполнение рабочих операций.

2. Повышать контроль и прозрачность - возможность отслеживать выполнение задач и фиксировать рабочее время помогает минимизировать ошибки и контролировать производственные процессы.
3. Оптимизировать использование ресурсов – учет рабочего времени позволяет анализировать производительность сотрудников и принимать обоснованные управленческие решения.
4. Повышать гибкость управления – информационные системы позволяют адаптировать планирование к изменяющимся условиям и требованиям бизнеса.

Таким образом, разработка информационной системы планирования задач и учета рабочего времени становится важным инструментом повышения эффективности работы предприятия и обеспечения его конкурентоспособности на рынке.

На сегодняшний день существует множество программных продуктов, однако наибольшую популярность приобретают веб-приложения. Многие крупные организации, такие как СберБанк, Газпромнефть, ВТБ и другие активно разрабатывают приложения с целью оптимизации своих бизнес-процессов [1-3].

Система планирования задач и учёта рабочего времени на предприятии представляет собой современное React-приложение (JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов), которое выполняет важные функции организации работы сотрудников и учёта времени. В качестве основы клиентской части приложения используется популярный фреймворк React, что позволяет создавать динамичные и отзывчивые интерфейсы, обеспечивая удобное взаимодействие с пользователем. Для обеспечения асинхронного взаимодействия с сервером и отправки HTTP-запросов используется модуль Axios, который значительно упрощает работу с API.

Серверная часть приложения реализована с использованием Express.js - мощного и гибкого фреймворка для Node.js, который предоставляет удобный интерфейс для разработки веб-приложений. Express.js используется для обработки запросов от клиента, выполнения логики приложения и взаимодействия с базой данных. В качестве хранилища данных используется реляционная база данных MySQL, которая обеспечивает надёжность и высокую скорость обработки данных, а также поддержку сложных запросов для работы с различной информацией.

Логическая структура приложения представлена на диаграмме (рис. 1), где видно, как различные компоненты взаимодействуют друг с другом. Подключение к базе данных осуществляется через файл config.js, где прописываются все необходимые параметры для подключения к MySQL. Эти параметры включают адрес сервера, имя пользователя, пароль и имя базы данных, что позволяет приложению корректно работать с хранилищем данных [4, 5].

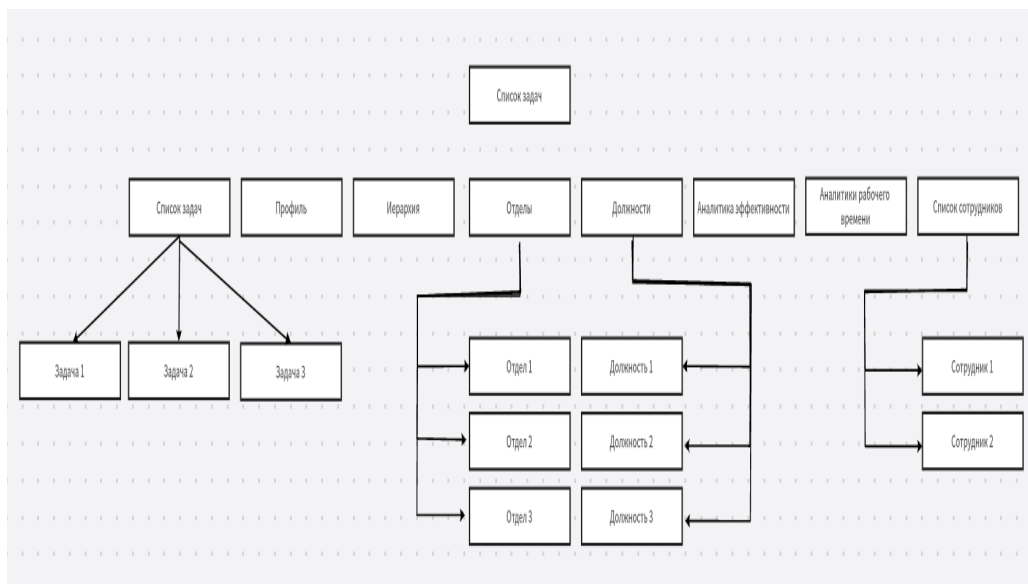


Рис. 1. - Логическая структура приложения

Инициализация React-приложения: Чтобы создать фронтенд-приложение на базе React, необходимо выполнить следующую команду: `prx create-react-app Kanban`

```
PS D:\CursorAi> cd .\newProject\  
PS D:\CursorAi\newProject> npx create-react-app Kanban
```

Рис. 2 - Инициализация React приложения

Где Kanban - это имя создаваемой папки для проекта. После выполнения команды все необходимые файлы и директории будут автоматически сгенерированы. Для перехода в папку проекта необходимо выполнить команду: *cd newProject*.

Чтобы запустить приложение на локальном сервере, используется команда: *npm start*.

По умолчанию приложение будет доступно по адресу <http://localhost:3000>.

Для работы с API необходимо установить библиотеку Axios: *npm install axios*.

Создание серверной части. Для создания серверной части необходимо перейти в корневую директорию проекта и создать отдельную папку командой: *mkdir backend*. Команду *cd backend* используем для перехода в новую папку.

Инициализация Node.js проекта производится командой: *npm init -y*. Это создаст файл *package.json*, который будет использоваться для управления зависимостями сервера.

Установка Express.js и других зависимостей для сервера производится командой: *npm install express cors mysql bcrypt nodemon*:

- *express* - для создания серверной логики.
- *cors* - для разрешения кросс-доменных запросов.
- *mysql* - для работы с базой данных MySQL.
- *bcrypt* – для шифрования пароля (используется при регистрации пользователя).
- *nodemon* – динамический перезапуск сервера при изменении кода.

Для дальнейшего запуска сервера можем скорректировать файл `package.json`, добавив в него следующую запись:

```
"scripts": {  
  "start": "node index.js",  
  "dev": "nodemon index.js"  
},
```

Рис. 3 - Файл `package.json`

Данная запись обозначает, что при запуске команды `npm run dev` будет запущен наш модуль `nodemon`, который будет перезапускать сервер при изменении кода.

Запуск сервера: Необходимо создать файл `index.js` в папке `backend` и добавить основной код для запуска сервера (рис. 4). Запуск сервера происходит с помощью команды: `Npm run dev` – при условии, что это заранее указано в файле конфигурации `package.json`.

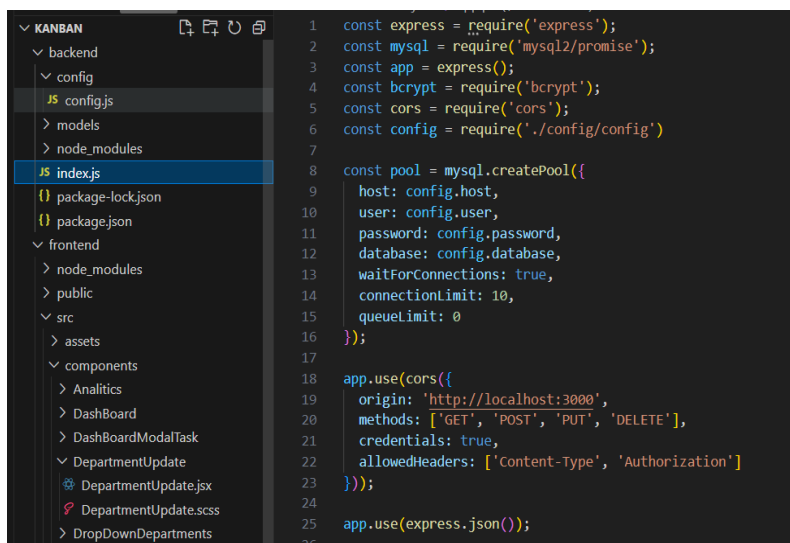


Рис. 4 - Файл `index.js` серверной части приложения

Запуск базы данных MySQL и подключение: Параметры для подключения к БД задаются в файле `config.js`.

Формат подключения следующий:

```
const config = {  
  host: '*.46.*.24*'  
  user: 'dim*****',
```

```
password: '****m*****',  
database: *****,  
}  
  
module.exports = config
```

Рис. 5 - Файл с конфигурацией для подключения к БД

Данный config затем используется в корневом файле index.js.

Через require происходит импорт параметров, хранящихся в нем, и используется для создания подключения – connection:

```
const pool = mysql.createPool({  
  host: config.host,  
  user: config.user,  
  password: config.password,  
  database: config.database,  
  waitForConnections: true,  
  connectionLimit: 10,  
  queueLimit: 0  
});
```

Рис. 6 - Настройка подключения к БД

Теперь наша рабочая область готова для дальнейшего создания приложения учёта рабочего времени и управления задачами на предприятии.

Отправка запросов на сервер. Для отправки запросов используется установленный ранее модуль axios. Он позволяет отправить данные на сервер и получить соответствующий ответ. Пример такого запроса отражен на рисунке 7. Менеджер состояний – это инструмент, который помогает управлять состоянием веб-приложения, обеспечивая эффективную передачу данных между компонентами, улучшая производительность и упрощая отладку и тестирование.

```
const fetchStatus = async () => {
```

```
try {  
    const response = await axios.post('http://localhost:5001/getStatusList',  
    {});  
    if (Array.isArray(response.data.statusList)) {  
        setStatusList(response.data.statusList);  
    } else {  
        console.error(Данные не являются массивом:', response.data);  
    }  
} catch (error) {  
    console.error('Ошибка обработки:', error);  
}  
}
```

Рис.7 - Функция отправки api запроса на сервер

В качестве менеджера состояния в данном приложении принято решение использовать Effector вместо стандартного useState, предоставляемого React, в силу его более широких возможностей и преимуществ для управления состоянием. Основной причиной выбора Effector является его способность централизованно обрабатывать состояние всего приложения. Это обеспечивает более гибкую и структурированную архитектуру, позволяя поддерживать модульность, изоляцию бизнес-логики и лёгкую интеграцию с различными модулями. Кроме того, Effector предлагает оптимизированное обновление интерфейса - перерисовка происходит только в тех компонентах, которые подписаны на изменения конкретных сторов, что существенно снижает количество ненужных рендеров и повышает производительность приложения.

Напротив, useState обладает более узкой специализацией и ограничивается локальным управлением состоянием внутри одного компонента React. Это делает его подходящим лишь для простых сценариев без сложных взаимосвязей. В контексте этого приложения использование Effector позволило достичь большей производительности, упрощения поддержки кода и обеспечения чистоты архитектуры.

После получения результата выполнения запроса, данные заносятся в effector с помощью вызова `setTaskHistoryList` [6 - 8].

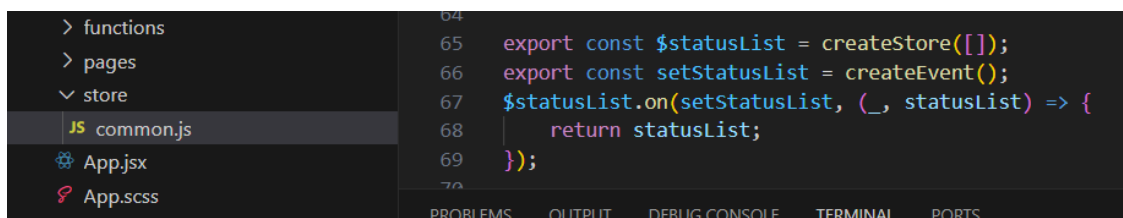


Рис.8 - Пример создания стора в effector

Теперь мы можем использовать `useUnit` в любой части проекта для объявления переменной и автоматической подписки на её обновления. Это означает, что при изменении значения `$statusList`, все компоненты, которые используют его, будут автоматически перерисованы, обеспечивая актуальность отображаемой информации без дополнительных действий со стороны разработчика. Для отображения данных пользователю применяется метод `map` - удобная встроенная функция для перебора элементов массива и их преобразования. В данном случае переменная `taskHistory`, содержащая массив задач, обрабатывается с помощью функции `map`, которая позволяет пройтись по каждому элементу массива и представить его в виде переменной `entry`. Каждый элемент в свою очередь представлен в формате «ключ-значение».

На рисунке 10 представлен пример использования `map`. Здесь каждый элемент массива `taskHistory` визуализируется на странице. Важно отметить, что сама переменная `taskHistory` также управляется с помощью Effector и вызывается с использованием хука `useUnit`, что обеспечивает реактивное обновление данных и синхронизацию с пользовательским интерфейсом [9,10].

```
const taskHistory = useUnit($taskHistoryList);
```

Рис.9 - Использование объявленного стора из Effector

```
<tbody>
  {
    taskHistory.map((entry) => {
      const user = allUsers.find(el => el.id === entry.UserID);

      return (
        <tr key={entry.ID}>
          <th>{entry.ID}</th>
          <th>{user.lastname + ' ' + user.name}</th>
          <th>{new Date(entry.ChangeDate).toLocaleString()}</th>
          <th>{entry.Message}</th>
          <th>{entry.ValueType}</th>
          <th>{entry.PreviousValue}</th>
          <th>{entry.NewValue}</th>
        </tr>
      )
    })
  }
</tbody>
```

Рис. 10 - Пример использования функции map для перебора массива.

Экономическое обоснование проекта разработки информационной системы планирования задач и учета рабочего времени обусловлено необходимостью оптимизации бизнес-процессов предприятия и повышения эффективности работы сотрудников. В условиях цифровизации бизнеса важно иметь инструмент, который снижает издержки, повышает прозрачность процессов и минимизирует простои сотрудников. Внедрение системы позволит оптимизировать бизнес-процессы, повысить производительность труда и минимизировать финансовые потери. В рамках проекта предложен современный подход к разработке корпоративной системы управления задачами и учета рабочего времени. Использование передовых технологий, таких как React и Node.js, обеспечивает высокую производительность, масштабируемость и гибкость системы. Применение Effector для управления состоянием позволяет повысить производительность и обеспечить реактивное обновление интерфейсов.

Далее перечислим аргументы, которые усилят обоснование проекта, подчеркнув его экономическую выгоду и технологические преимущества.

1. Автоматизация учета рабочего времени снизит административную нагрузку на HR-отдел и сократит количество ошибок, связанных с ручным вводом данных.
2. Централизованное управление задачами позволит руководителям оперативно перераспределять ресурсы и контролировать выполнение проектов.
3. Снижение трудозатрат на планирование и отчетность высвободит до 20% рабочего времени сотрудников, что положительно скажется на их продуктивности.
4. Аналитика и отчетность в реальном времени помогут выявлять узкие места в бизнес-процессах и своевременно принимать управленческие решения.
5. Интеграция с другими корпоративными системами (CRM, ERP) обеспечит единое информационное пространство и исключит дублирование данных.
6. Гибкие настройки ролей и прав доступа позволят адаптировать систему под специфику работы разных подразделений компании.
7. Снижение простоев за счет автоматического контроля deadlines и напоминаний о задачах повысит дисциплину сотрудников.
8. Масштабируемость архитектуры позволит расширять функционал системы без значительных затрат на доработку.
9. Использование облачных технологий снизит затраты на развертывание и обслуживание инфраструктуры.
10. Прогнозируемая окупаемость проекта составляет 12–18 месяцев за счет сокращения издержек и роста эффективности труда.

Заключение. Разработанное веб-приложение для планирования задач и учета рабочего времени демонстрирует эффективный подход к автоматизации бизнес-процессов предприятия. Использование современных

технологий, таких как React, Express.js и MySQL, обеспечило высокую производительность, надежность и масштабируемость системы. Внедрение менеджера состояний Effector позволило оптимизировать управление данными, сократив количество ненужных ререндеров и упростив поддержку кода.

Литература

1. Вайс Д. Node.js и Express: создание веб-приложений. – Москва: БХВ-Петербург, 2022. – 416 с. – ISBN 978-5-9775-2342-4. URL: book.ru/nodejs-express.
2. Кац А. Effector: эффективное управление состоянием. – Москва: Альпина, 2021. – 368 с. – ISBN 978-5-9614-0810-8. URL: alpinabook.ru/effector.
3. Кузнецов Д. Д. Разработка веб-приложения для малого предприятия. // Молодой ученый, 2023, № 18 (465). — С. 14-16.
4. Сербиновский Б.Ю., Сербиновская А.А., Белоус М.А. Развитие коммуникаций и сбыта продукции и услуг ресторанной сети с использованием IT-технологий (опыт моделирования и проектирования многофункционального веб-сайта). Часть 2. // Инженерный вестник Дона, 2013, №2. URL: ivdon.ru/ru/magazine/archive/n8y2025/10303.
5. Айдинян А.Р., Легонько О.Л., Мамонов Н.С. Эффективность применения некоторых алгоритмов хеширования для веб-приложения с интерфейсами регистрации и аутентификации пользователей. Инженерный вестник Дона, 2025, №8. URL: ivdon.ru/ru/magazine/archive/n8y2025/10303.
6. Линч Р. JavaScript и Node.js для профессионалов. – Москва: Питер, 2020. – 512 с. – ISBN 978-5-4461-1578-4. URL: piter.com/nodejs-book.

7. Нортон Д. Express.js и Node.js. Разработка серверных приложений. – Москва: Вильямс, 2021. – 384 с. – ISBN 978-5-93286-153-6. URL: williamspublishing.com/express-node.
8. Пейдж Дж. Основы разработки на React. – Москва: БХВ-Петербург, 2023. – 416 с. – ISBN 978-5-9775-1758-3. URL: book.ru/react-basics.
9. Bromberg L. Cryptographic Hash Functions and Some Applications to Information Security // Springer Proceedings in Mathematics and Statistics. – 2017. – pp. 85-97.
10. Kundu R. Cryptographic Hash Functions and Attacks – A Detailed Study // International Journal of Advanced Research in Computer Science. – 2020. – Vol. 11, No. 2. – pp. 37-44.

References

1. Vays D. Node.js i Express: sozдание veb-prilozheniy [Node.js and Express: Web Application Development]. Moskva: BKhV-Peterburg, 2022. 416 p. ISBN 978-5-9775-2342-4. URL: book.ru/nodejs-express.
2. Kats A. Effector: effektivnoe upravlenie sostoyaniem [Effector: Efficient State Management]. Moskva: Al'pina, 2021. 368 p. ISBN 978-5-9614-0810-8. URL: alpinabook.ru/effector.
3. Kuznetsov D. D. Molodoy uchenyy, 2023, № 18 (465). pp. 14-16.
4. Serbinovskiy B.Yu., Serbinovskaya A.A., Belous M.A. Inzhenernyj vestnik Dona, 2013, №2. URL: ivdon.ru/ru/magazine/archive/n8y2025/10303.
5. Aydinyan A.R., Legon'ko O.L., Mamonov N.S. Inzhenernyj vestnik Dona, 2025, №8. URL: ivdon.ru/ru/magazine/archive/n8y2025/10303.
6. Linch R. JavaScript i Node.js dlya professionalov [JavaScript and Node.js for Professionals]. Moskva: Piter, 2020. 512 p. ISBN 978-5-4461-1578-4. URL: piter.com/nodejs-book.



7. Norton D. Express.js i Node.js. Razrabotka servernykh prilozheniy [Express.js and Node.js: Server-Side Application Development]. Moskva: Vil'yams, 2021. 384 p. ISBN 978-5-93286-153-6. URL: williamspublishing.com/express-node.
8. Peydzh Dzh. Osnovy razrabotki na React [React Development Fundamentals]. Moskva: BKhV-Peterburg, 2023. 416 p. ISBN 978-5-9775-1758-3. URL: book.ru/react-basics.
9. Bromberg L. Springer Proceedings in Mathematics and Statistics. 2017. pp. 85-97.
10. Kundu R. International Journal of Advanced Research in Computer Science. 2020. Vol. 11, No. 2. pp. 37-44.

Дата поступления: 17.09.2025

Дата публикации: 26.10.2025