μ MicroManage

Brooke Stevens, Chad Wireman, Chris Beggs, Ian Sexton, Jacob Carlson, & Rahul Depa

Team Name: Sweet Victory

What is MicroManage? Who is our user base?

MicroManage (μ) is a scheduling service which combines an intuitive and minimalist user interface to a database, allowing users to plan out their days, weeks, months, and years.

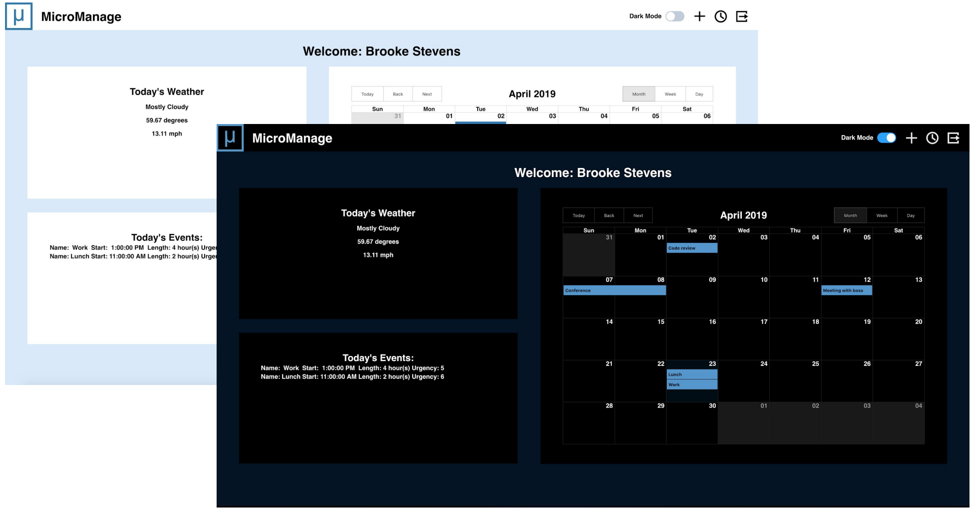
Features at a Glance:

- Add, edit, and delete events dynamically
- Current weather reports
- Light/Dark Mode

Design & Creativity

Focus:

- Clean
- Friendly & Accessible
- Professional
- Unique



Welcome: Brooke Stevens

Today's Weather
Mostly Cloudy
59.67 degrees
13.11 mph

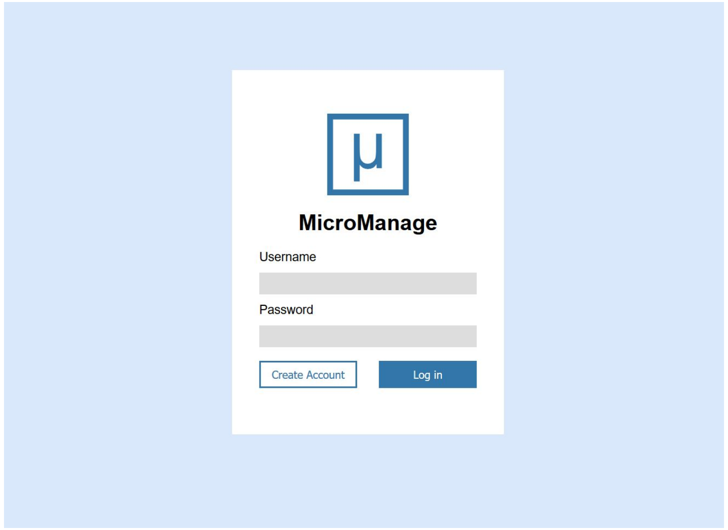
Today's Events:
Name: Work Start: 1:00:00 PM Length: 4 hour(s) Urgency: 5
Name: Lunch Start: 11:00:00 AM Length: 2 hour(s) Urgency: 6

April 2019

Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	01	02 Code review	03	04	05	06
07 Conference	08	09	10	11	12 Meeting with boss	13
14	15	16	17	18	19	20
21	22	23 Lunch Work	24	25	26	27
28	29	30	01	02	03	04

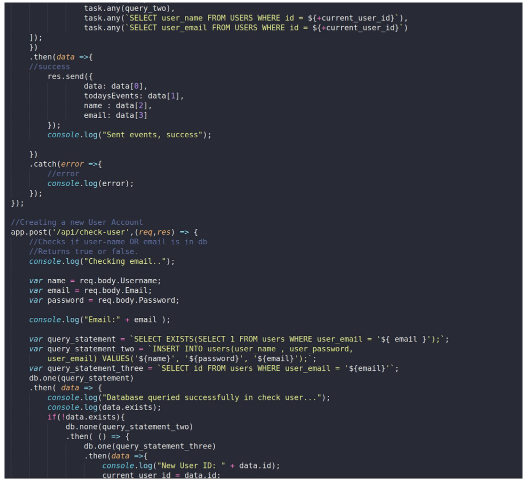
Front-End Features

- Add, edit, and delete events
- Login and registration page
- Email reminders
- Dark Sky API Panel
- Dark Mode



Integration Features

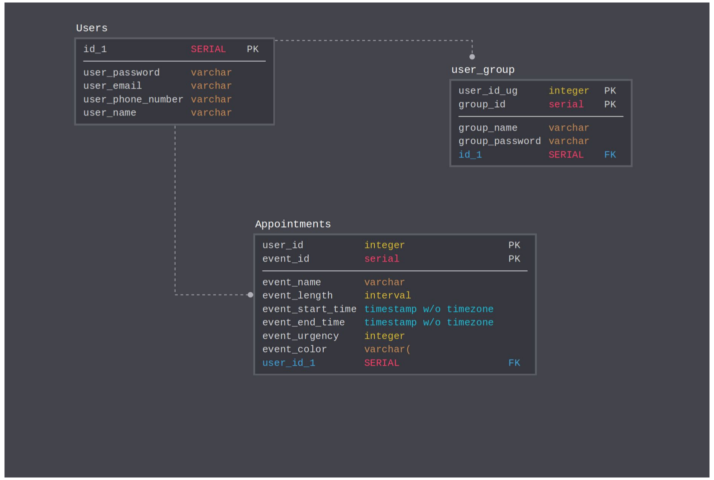
- Node.js for server side JS
- Express framework can create
  - Routes
  - REST API
- API’s
  - written and executed with Pg-Promise
- Data transfer
  - JSON between API and React.js



```
                task.any(query_two),
                task.any(`SELECT user_name FROM USERS WHERE id = ${+current_user_id}`),
                task.any(`SELECT user_email FROM USERS WHERE id = ${+current_user_id}`)
    ]);
    })
    .then(data =>{
    //success
        res.send({
                data: data[0],
                todaysEvents: data[1],
                name : data[2],
                email: data[3]
        });
        console.log("Sent events, success");

    })
    .catch(error =>{
        //error
        console.log(error);
    });
});

//Creating a new User Account
app.post('/api/check-user',(req,res) => {
    //Checks if user-name OR email is in db
    //Returns true or false.
    console.log("Checking email..");

    var name = req.body.Username;
    var email = req.body.Email;
    var password = req.body.Password;

    console.log("Email:" + email );

    var query_statement = `SELECT EXISTS(SELECT 1 FROM users WHERE user_email = '${ email }');`;
    var query_statement_two = `INSERT INTO users(user_name , user_password,
        user_email) VALUES('${name}', '${password}', '${email}');`;
    var query_statement_three = `SELECT id FROM users WHERE user_email = '${email}'`;
    db.one(query_statement)
    .then( data => {
        console.log("Database queried successfully in check user...");
        console.log(data.exists);
        if(!data.exists){
            db.none(query_statement_two)
            .then( () => {
                db.one(query_statement_three)
                .then(data =>{
                    console.log("New User ID: " + data.id);
                    current_user_id = data.id;
```

Back-end Features

- Linked via primary keys and foreign keys
  - All user appointments in a single table
  - Can uniquely access with their ID numbers
- Unique ID
- Appointments table
  - contains every users events
  - each row contains the unique user ID
  - Similar to user group table.



Users

id_1	SERIAL	PK
user_password	varchar	
user_email	varchar	
user_phone_number	varchar	
user_name	varchar	

user_group

user_id_ug	integer	PK
group_id	serial	PK
group_name	varchar	
group_password	varchar	
id_1	SERIAL	FK

Appointments

user_id	integer	PK
event_id	serial	PK
event_name	varchar	
event_length	interval	
event_start_time	timestamp w/o timezone	
event_end_time	timestamp w/o timezone	
event_urgency	integer	
event_color	varchar(	
user_id_1	SERIAL	FK

Challenges (front-end)

1. Complications with progress bar
2. Express and React both used the same port

Solutions:

1. Replaced progress bar with weather component
2. Proxy requests from 3000 to 3001

Challenges (integration & back-end)

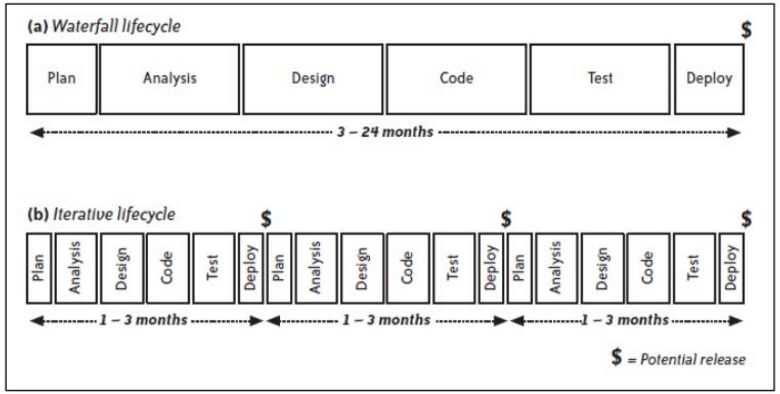
1. Formatting data passing between database, Express API’s, and front end
2. With vanilla Node/Express, changing and saving the js files means you must restart the npm instance

Solutions:

1. Including functions in the Express Js file that changed the date format so the Js on the front end could read it
2. Used Nodemon, a Node.js package that automatically restarts the server when you save the Js file

Project Management

- Meetings took place regularly
  - Tuesday and Thursday at 5:00 pm
- Began with waterfall, switched to AGILE
- Communication
  - Slack
  - Trello
  - GitHub



Software/Tools Used

Front-End

- **HTML/CSS:** 3/5 - unnecessary work required for simple tasks
- **JavaScript:** 5/5 - easy to use
- **React.js:** 4/5 - good for dynamic sites bad for multi-page sites
- **Express.js:** 4/5 - easy to learn and works well with other Node packages

Back-End

- **PSQL:** 5/5 - Easy to learn and understand, great for the purposes of our app
- **Pg-Promise:** 4/5 - easy syntax, great documentation
- **Dark Sky API:** 4/5 - easy to use, not accurate
- **Mailjet:** 4/5 - easy to use, promise based

Live Demo