KOCAELI UNIVERSITY

Faculty of Engineering
Department of Software Engineering

Software Development I

Project Report

Task Management Web Application

Name & Surname:	Sena Nur Dülger	**Student ID:**	220229053
Name & Surname:	Irmak Yılmaz	**Student ID:**	230229038

Date: November 11, 2025

Task Management Web Application

Abstract—**This report presents the design and implementation of a web-based Task Management application built with the MERN stack (MongoDB, Express.js, React, Node.js). The system enables registered users to create, update, categorize, and track tasks by status and due date, while enforcing secure authentication and robust data handling. The report focuses only on the required evidence items.**

Index Terms—**task management, MERN stack, React, Node.js, Express.js, MongoDB, REST API, JSON Web Token, JWT, bcrypt, authentication, password hashing, data visualization, frontend backend integration, database connectivity, error handling, validation**

I. Introduction

Task management and productivity-focused web applications have become one of the fastest-evolving areas of full-stack development. Addressing the everyday needs of businesses and individuals—planning, team coordination, and tracking deadlines—this project delivers a secure and scalable Task Management application using the MERN architecture. The app follows contemporary practices: a React UI consumes a REST API built with Node.js/Express; data persist in MongoDB; authentication is token-based with JWT; and user passwords are stored as salted bcrypt hashes. This report covers the following sections:

1) *Introduction:* Summary of the topic and scope of the report.
2) *Technologies Used:* Chosen stack and brief rationale.
   a) Frontend (UI)
   b) Backend (API)
   c) Database
   d) Security and Encryption
   e) Additional Tools
3) *Implementation Results:* Evidence of a working integration.
   a) Frontend + Backend Integration
   b) Database Connection
4) *Interface Draft:* Initial appearance of the main screens.
5) *Encrypted Password:* Example bcrypt hash and brief explanation.
6) *Invalid Login Outcome:* 401 response and user-facing message.
7) *Conclusion:* Short evaluation and next steps.
8) *References*

II. Technologies Used

This section briefly summarizes the technologies used in the project and the reasons for their selection.

Technology	**Selection**	**Reasons**
Frontend (UI)	React + Vite	React was selected for its component-based architecture and vibrant ecosystem, enabling fast development and maintainable UIs. Combined with Vite and React Router, it provides instant hot-module refresh, efficient routing, and route-level code splitting for improved performance. Styling is handled using standard CSS modules for a clean and lightweight design.
Backend (API)	Node.js + Express	Node.js with Express was selected for its lightweight, event-driven runtime and mature middleware ecosystem. It makes building RESTful endpoints straightforward, adding JWT/httpOnly-cookie authentication, input validation, CORS, and centralized error handling, while integrating cleanly with a MongoDB data layer for a simple end-to-end JavaScript stack.
Database	MongoDB	MongoDB was selected for its flexible, JSON/BSON-based document model. It integrates seamlessly with the MERN stack and is ideal for storing variable data structures like user tasks.
Security and Encryption	Bcrypt + JWT	Bcrypt was chosen to securely hash user passwords. It incorporates salting to protect against rainbow table attacks and stores hashes irreversibly in the database. JWT was selected to create secure, stateless authentication tokens, allowing the server to verify API requests without storing session data.

TABLE I
Technology Selections and Reasons (Part I)

Technology	Selection	Reasons
Additional Tools	Chart.js, Axios, Nodemon	Chart.js, Axios, and Nodemon were selected to streamline visualization, networking, and developer ergonomics. Chart.js delivers responsive charts with minimal setup; Axios provides concise HTTP calls with built-in auth and error handling; and Nodemon speeds backend iteration by auto-restarting on file changes, improving productivity and maintaining a smooth development workflow.

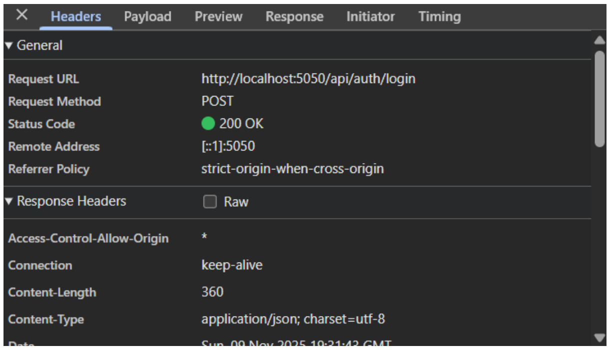
TABLE II
TECHNOLOGY SELECTIONS AND REASONS (PART II)

III. IMPLEMENTATION RESULTS

Evidence of a working integration, including frontend + backend integration and an active database connection.

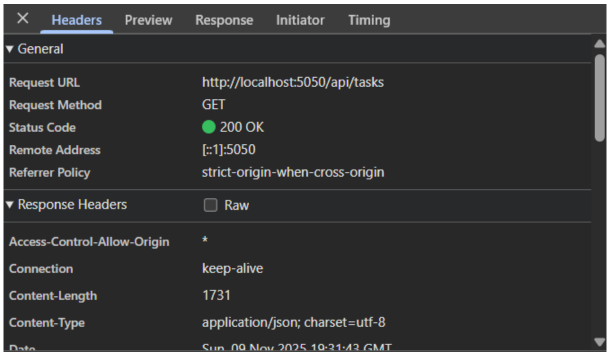
A. Frontend + Backend Integration

The first figure shows the browser's network tab capturing a successful POST request from the sign-in interface to `/api/users/login`, which returned 200 OK, confirming the connection and successful authentication. The second figure shows a subsequent GET request to `/api/tasks` returning 200 OK with JSON data.



```
Request URL                    http://localhost:5050/api/auth/login
Request Method                 POST
Status Code                    ● 200 OK
Remote Address                 [::1]:5050
Referrer Policy                strict-origin-when-cross-origin

Access-Control-Allow-Origin    *
Connection                     keep-alive
Content-Length                 360
Content-Type                   application/json; charset=utf-8
Date                           Sun, 09 Nov 2025 19:31:43 GMT
```

Fig. 1. POST `/api/users/login` — 200 OK

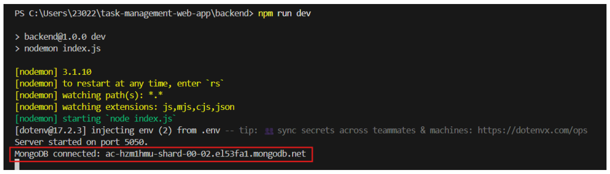


```
Request URL                    http://localhost:5050/api/tasks
Request Method                 GET
Status Code                    ● 200 OK
Remote Address                 [::1]:5050
Referrer Policy                strict-origin-when-cross-origin

Access-Control-Allow-Origin    *
Connection                     keep-alive
Content-Length                 1731
Content-Type                   application/json; charset=utf-8
Date                           Sun, 09 Nov 2025 19:31:43 GMT
```

Fig. 2. GET `/api/tasks` — 200 OK

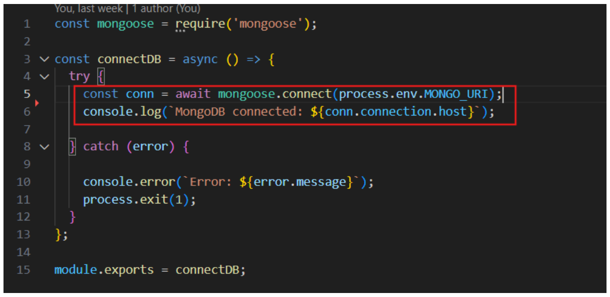
B. Database Connection

The backend establishes a persistent database session by initializing Mongoose with the environment variable `MONGO_URI`. On success, it logs the connected host (e.g., *MongoDB Connected: <host>*), confirming that queries will persist and read data from the live database during runtime.



```
PS C:\Users\23022\task-management-web-app\backend> npm run dev

> backend@1.0.0 dev
> nodemon index.js

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
[dotenv@17.2.3] injecting env (2) from .env -- tip: 👥 sync secrets across teammates & machines: https://dotenvx.com/ops
Server started on port 5050.
MongoDB connected: ac-hzm1hmu-shard-00-02.el53fa1.mongodb.net
```

Fig. 3. Terminal: successful connection log



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

const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI);
    console.log(`MongoDB connected: ${conn.connection.host}`);

  } catch (error) {

    console.error(`Error: ${error.message}`);
    process.exit(1);
  }
};

module.exports = connectDB;
```

Fig. 4. Code: Mongoose connect using `MONGO_URI`

IV. INTERFACE DRAFT

This section presents the initial interface design of the *TaskMint* application.

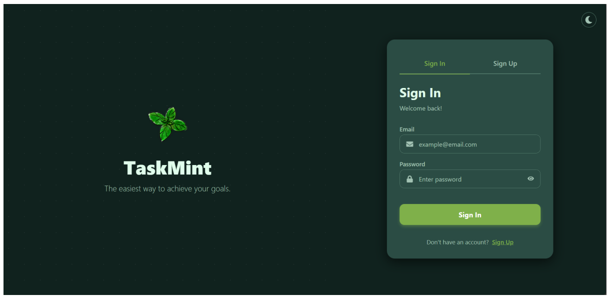


Fig. 5. Sign In

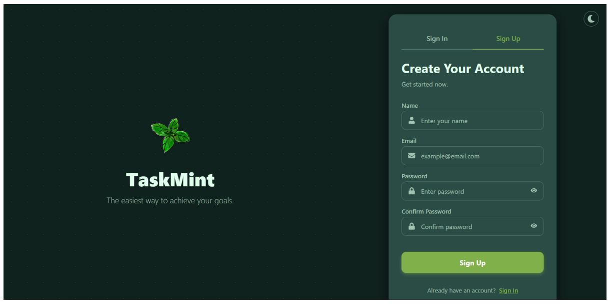
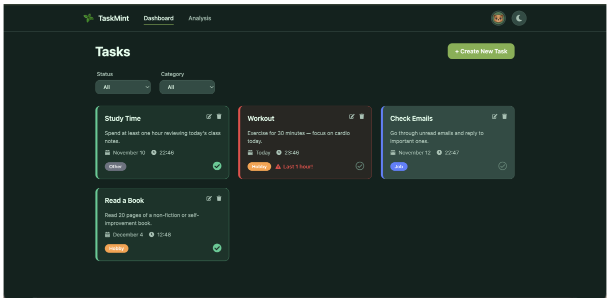


Fig. 6. Sign Up



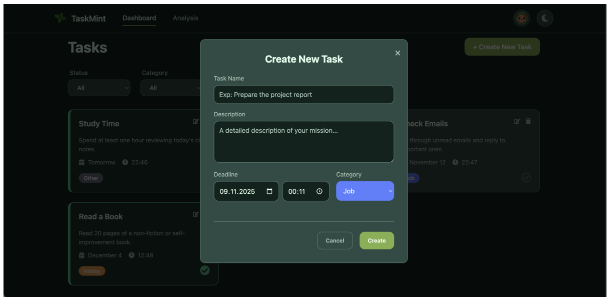
Study Time
Spend at least one hour reviewing today's class notes.
November 10 22:46
Other

Workout
Exercise for 30 minutes — focus on cardio today.
Today 23:46
Hobby Last 1 hour!

Check Emails
Go through unread emails and reply to important ones.
November 12 22:47
Job

Read a Book
Read 20 pages of a non-fiction or self-improvement book.
December 4 12:48
Hobby

Fig. 7. Dashboard



Task Name: Exp: Prepare the project report
Description: A detailed description of your mission...
Deadline: 09.11.2025 00:11
Category: Job

Fig. 8. Create New Task

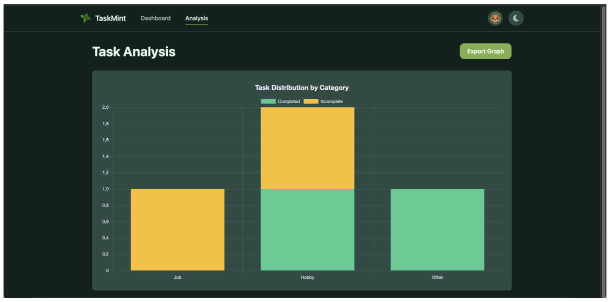
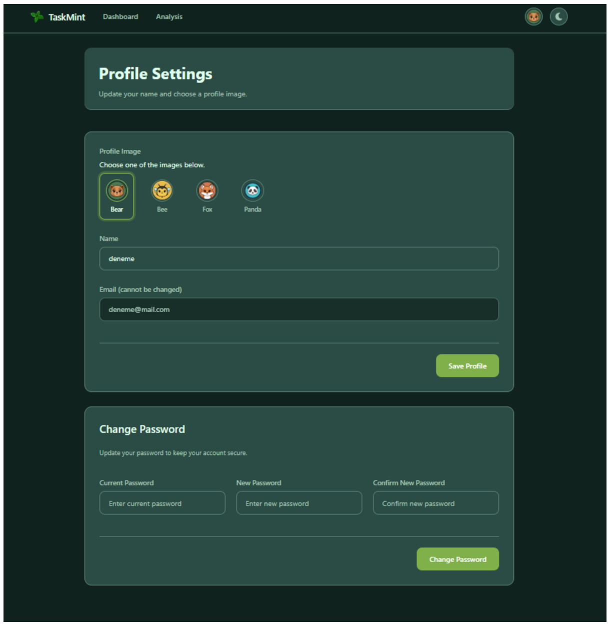


Fig. 9. Analysis



Profile Settings
Update your name and choose a profile image.

Profile Image
Choose one of the images below.

Name: deneme
Email (cannot be changed): deneme@mail.com

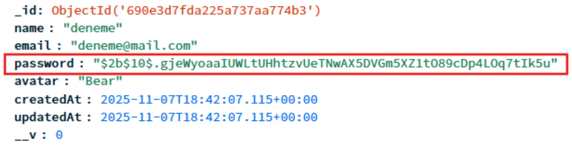
Change Password
Update your password to keep your account secure.

Current Password: Enter current password
New Password: Enter new password
Confirm New Password: Confirm new password

Fig. 10. Profile Settings

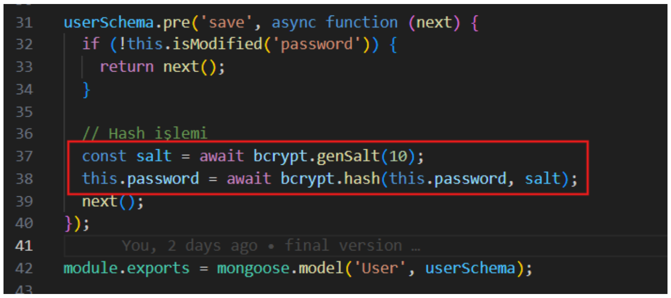
The design adopts a cohesive dark theme with mint accents. Figs. 5. - 6. show the entry flow with tabbed authentication, validated inputs, and a single primary action. Fig. 7. presents a card-based dashboard where each task exposes title, details, due date/time, status, and category chips with inline actions. Fig. 8. illustrates the creation modal that captures name, description, deadline (date & time) and category with clear cancel/confirm affordances. Fig. 9. summarizes the completion by category in a readable bar chart with export support, and Fig. 10. provides profile editing with avatar selection and a password-change panel. The Edit Task view reuses the same modal layout as Fig.8.; all fields are pre-filled with the selected task's data and the primary action becomes Save. The general spacing, hierarchy, and typography are tuned for clarity and responsive behavior.

V. Encrypted Password



```
_id: ObjectId('690e3d7fda225a737aa774b3')
name : "deneme"
email : "deneme@mail.com"
password : "$2b$10$.gjeWyoaaIUWLtUHhtzvUeTNwAX5DVGm5XZ1tO89cDp4LOq7tIk5u"
avatar : "Bear"
createdAt : 2025-11-07T18:42:07.115+00:00
updatedAt : 2025-11-07T18:42:07.115+00:00
__v : 0
```

Fig. 11. Hashed value in DB



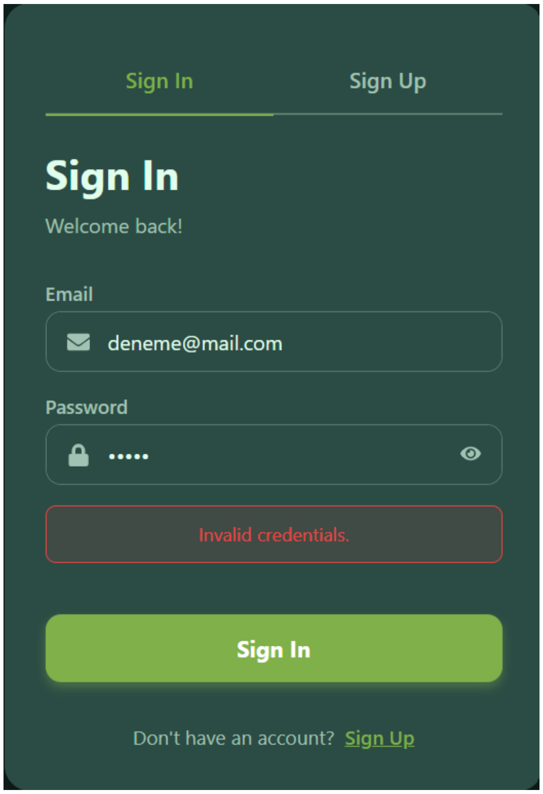
```
userSchema.pre('save', async function (next) {
  if (!this.isModified('password')) {
    return next();
  }

  // Hash işlemi
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});
        You, 2 days ago • final version …
module.exports = mongoose.model('User', userSchema);
```

Fig. 12. Hashing in code

Passwords are not stored in plain text. As shown in Fig. 11 and Fig. 12, the system generates a random salt using `bcrypt.genSalt(10)` and then hashes the password with `bcrypt.hash()`. Thanks to the salt and the cost factor (10 rounds), the same password produces different hashes for different users and is computationally hard to reverse. Only the salted hash is saved to the database, which prevents exposure of the original password even if the database is leaked.

VI. Invalid Login Outcome

When an incorrect email or password is submitted, the system rejects the request with an HTTP 401 Unauthorized response and a generic message. The UI surfaces "Invalid credentials.", no session or token is issued, and the flow cleanly handles both success and error paths without leaking sensitive details.

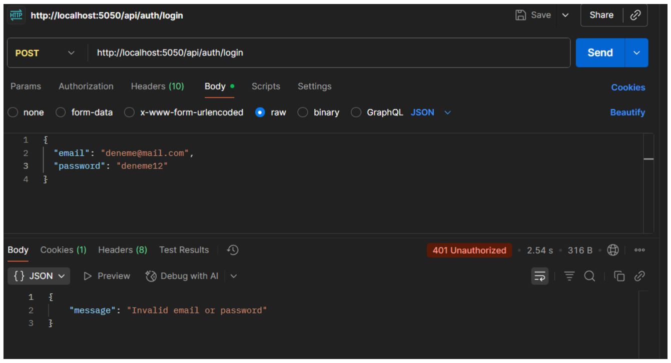


Email: deneme@mail.com

Password: •••••

Invalid credentials.

Fig. 13. UI: "Invalid credentials."



http://localhost:5050/api/auth/login

POST http://localhost:5050/api/auth/login

○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

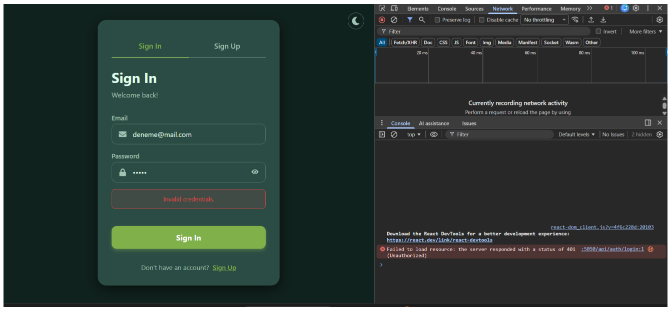
```
{
  "email": "deneme@mail.com",
  "password": "deneme12"
}
```

Body — 401 Unauthorized • 2.54 s • 316 B

{} JSON

```
{
    "message": "Invalid email or password"
}
```

Fig. 14. API client: 401 Unauthorized (Postman)



Email: deneme@mail.com

Password: •••••

Invalid credentials.

```
                                       react-dom_client.js?v=4f6c228d:20103
Download the React DevTools for a better development experience:
https://react.dev/link/react-devtools
Failed to load resource: the server responded with a status of 401  :5050/api/auth/login:1
(Unauthorized)
```

Fig. 15. Combined view (UI + console)

VII. Conclusion

All midterm requirements for the Task Management Web Application are complete. The core MERN stack is operational, including an active database connection, successful frontend-backend integration, and secure JWT/Bcrypt authentication. The initial UI draft is also finished. Future work will now focus on implementing full task (CRUD) operations, automated backend testing, and the final "Analysis" dashboard.

References

[1] "Build a Full-Stack MERN Task Manager — React, Node.js, MongoDB, Express — MERN Project," YouTube, [Online Video]. Accessed: Nov. 9, 2025. [Online]. Available: https://www.youtube.com/watch?v=fZK57PxKC-0&list=PLLSbWBaXAMvKZw4F1qwYPuDxwYF_gq7i4

[2] "Build a Full-Stack MERN Project Task Manager Tool — React.js + Node.js — MERN (2025)," YouTube, [Online Video]. Accessed: Nov. 9, 2025. [Online]. Available: https://www.youtube.com/watch?v=VAKDr1lsix0&list=PLLSbWBaXAMvKZw4F1qwYPuDxwYF_gq7i4&index=2

[3] "MERN Stack Project — Fullstack Tutorial," YouTube, [Online Video]. Accessed: Nov. 9, 2025. [Online]. Available: https://www.youtube.com/watch?v=H-9l-gTq-C4&list=PL0Zuz27SZ-6P4dQUsoDatjEGpmBpcOW8V

[4] "Connect frontend and backend — React JS, Node JS, Express — Send data from backend to frontend — API," YouTube, [Online Video]. Accessed: Nov. 9, 2025. [Online]. Available: https://www.youtube.com/watch?v=OgYKAI38JvY&list=PLLSbWBaXAMvKZw4F1qwYPuDxwYF_gq7i4&index=10

[5] "Node.js Full Course for Beginners — Complete All-in-One Tutorial — 7 Hours," YouTube, [Online Video]. Accessed: Nov. 9, 2025. [Online]. Available: https://www.youtube.com/watch?v=f2EqECiTBL8&list=PLLSbWBaXAMvKZw4F1qwYPuDxwYF_gq7i4&index=11

[6] "Understand MERN Stack," GeeksforGeeks, [Online]. Accessed: Nov. 9, 2025. [Online]. Available: https://www.geeksforgeeks.org/mern/understand-mern-stack/

[7] OpenAI, "ChatGPT," [Online]. Accessed: Nov. 9, 2025. [Online]. Available: https://openai.com/chatgpt

[8] Google, "Gemini," [Online]. Accessed: Nov. 9, 2025. [Online]. Available: https://gemini.google.com

[9] Google, "Stitch," [Online]. Accessed: Nov. 9, 2025. [Online]. Available: https://stitch.withgoogle.com/