FACULDADE UNINASSAU PARNAÍBA

CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS(ADS)

Grupo	Danilo, Kueren e Augusto
Disciplina	Desen. De Aplic. p/ Internet e Front-End Frameworks
Professor	Luiz Lins
Período	2º (Segundo)

- **Termo de Abertura do Projeto**

1.1 Título do Projeto: "Beach Time".

1.2 Justificativa do Projeto: O gerenciamento que a arena de esportes utiliza depende muito de tarefas manuais, como agendamento via telefone, mensagens por aplicativos (WhatsApp) e marcação de horário presencialmente. Isso resulta em conflitos nas informações recebida, entre eles:

- **Duplicidade de agendamentos e erros humanos.**
- **Dificuldade no controle de pagamentos e falta de comparecimento.**
- **Falta de um canal direcionado para comunicação e pagamento com os clientes**
- **Falta de dados centralizados que permitem analisar o desempenho das tarefas e uma decisão estratégica de como resolver.**

O projeto "Beach Time" visa automatizar e integrar a gestão da arena, proporcionado um ambiente prático e de melhor experiência para os clientes e aprimorando a administração do negócio pelos responsáveis.

1.3 Objetivos do Projeto:

- Implementar um sistema web para agendamento online das quadras, permitindo que os clientes confiram a disponibilidade de horários de forma prática e rápida.
- Melhorar a comunicação entre clientes e administradores, evitando erros e uma proporcionando uma experiência digital moderna e intuitiva.
- Integrar pagamentos online de aluguel para facilitar o acesso aos clientes.
- O sistema visa a modernização, deixando de lado o que tinha de manual no gerenciamento dos agendamentos, assim promovendo a escalabilidade do negócio e influenciando a adoção desse tipo de sistema por outros interessados por causa de seus resultados positivos.
- O projeto deverá ser concluído e implantado em um prazo de 3 meses.

1.4 Requisitos de Alto Nível:

- Plataforma de agendamento de quadras com visualização em tempo real.
- Sistema de pagamento online (cartão de crédito, PIX).
- Cadastro e perfil de usuário para clientes.
- Painel administrativo para gestão de horários, preços e usuários.
- Sistema de notificações automáticas (confirmação, lembretes).

1.6 Partes Interessadas Iniciais (Stakeholders):

- Proprietários/Sócios da Arena
- Gerente Administrativo
- Funcionários da Recepção
- Clientes (Atletas amadores e profissionais, grupos de amigos)
- Equipe de Desenvolvimento de Software

2.0 Escopo do Produto: O "Beach Time" é uma solução web acessível por computadores e celulares. Ele permitirá que os clientes visualizem a disponibilidade das quadras, realizem agendamentos e pagamentos de forma autônoma. O sistema oferecerá funções para gerenciar todos as tarefas administrativas, edição de preços de alocação da quadra, horários que podem ser disponibilizados, avisos sobre a indisponibilidade da localidade, seja por motivos diversos.

2.1 Critérios de Aceitação:

- O cliente deve ser capaz de se cadastrar, agendar uma quadra e realizar o pagamento em menos de rapidamente.
- Deve exibir em tempo real todas as quadras ocupadas e disponíveis.
- O sistema deve processar os pagamentos sempre com sucesso.
- O sistema deve enviar e-mails e/ou notificações de confirmação, assim como lembretes para os agendamentos confirmados.

2.2 Entregas do Projeto (Deliverables):

- Aplicação Web para Clientes (Front-end).
- Painel Administrativo (Front-end e Back-end).
- API de serviços (Back-end).
- Banco de Dados modelado e implementado.
- Documentação técnica e manual do usuário.

- Sessões de treinamento para a equipe da arena.

2.3 Exclusões do Projeto (O que NÃO será feito):

- Desenvolvimento de um aplicativo nativo (iOS/Android) nesta fase.
- Integração com sistemas de controle de acesso físico (catracas).
- Módulo de venda de produtos (bebidas, materiais esportivos).
- Funcionalidades de rede social (ex: encontrar parceiros de jogo).
- Programa de fidelidade complexo.

3.0 Público-Alvo e Funcionalidades Detalhadas

3.1 Público-Alvo: O sistema é focado para os gerentes que trabalham com quadras de areia, para esportes como: beach tennis, vôlei e futevôlei de praia, no geral esportes de areia, que precisam de uma forma mais prática, ágil e fácil de se utilizar. Além disso, irá agregar os clientes/atletas que utilizarão essa plataforma para agendar as suas atividades esportivas, treinos, jogos amistosos etc.

3.2 Funcionalidades Detalhadas:

Agendamento de Quadras:

- Visualização de horários por dia/semana/quadra.
- Status de horários (disponível, reservado, confirmado, bloqueado).
- Seleção de horário, duração e quadra.
- Reserva de horários recorrentes
- Bloqueio de horários para manutenção ou eventos pela administração.

Gestão de Clientes (CRM):

- Cadastro de clientes com nome, e-mail, telefone.
- Login social (Google, Facebook) para facilitar o acesso.
- Área do cliente para visualizar histórico de agendamentos e status de pagamentos.
- Possibilidade de cancelar agendamentos (com regras de negócio pré-definidas).

Financeiro:

- Integração de forma de pagamentos para processá-los.
- Geração de faturas ou recibos.
- Controle de pagamentos (pendente, pago, estornado).
- Visualização de entradas.

Administração:

- Painel central com a ocupação, receita do dia, próximos agendamentos.
- Gestão completa de usuários, quadras, preços e regras de negócio.
- Ferramenta para comunicação com clientes.
- Visualização e exportação de relatórios.

4.0 Riscos Iniciais Identificados: Como todo projeto de tecnologia, o "Beach Time" possui riscos que precisam ser monitorados desde o início para garantir seu sucesso. Alguns riscos identificados são:

- Resistência à mudança por parte da administração atual e dos clientes acostumados com métodos tradicionais.
- Possíveis falhas na integração de pagamentos online, que podem afetar diretamente a confiança e a funcionalidade do sistema.
- Problemas com conectividade e acesso em regiões com infraestrutura limitada, o que pode prejudicar a adoção do sistema.

5.0 Arquitetura do Sistema e Especificações Técnicas

5.1 Visão Geral da Arquitetura: O sistema será projetado e baseado em nuvem, garantindo escalabilidade. A comunicação entre o front-end e o back-end será realizada através de uma API segura.

5.2 Arquitetura Proposta:

- **Front-end:** Uma Single Page desenvolvida para ser responsiva, garantindo uma experiência de usuário em desktops, tablets e smartphones, além de integrar a utilização do Vue.js.
- **Back-end:** Uma API que centraliza todas as regras de negócio, processamento de dados e comunicação com serviços externos, feito com apoio do node.js.
- **Banco de Dados:** Um banco de dados relacional para garantir a integridade e a consistência dos dados transacionais (agendamentos, pagamentos, usuários).

5.3 Requisitos Não-Funcionais:

- **Performance:** O tempo de carregamento da página de agendamento não deve exceder 3 segundos. As respostas da API devem ter uma latência média inferior a 200ms.
- **Segurança:** Utilização de HTTPS em todas as comunicações. Proteção contra ataques comuns (SQL Injection, XSS). Conformidade com a LGPD (Lei Geral de Proteção de Dados).
- **Escalabilidade:** A arquitetura deve suportar um aumento de 300% no número de usuários simultâneos sem degradação da performance.

- **Disponibilidade:** O sistema deve ter um uptime de 99.8%.

6.0 Gerenciamento das Partes Interessadas (Stakeholders)

Parte Interessada	Papel no Projeto	Principais Interesses	Influência	Estratégia de Engajamento
Proprietários da Arena	Patrocinadores (Sponsors)	Retorno sobre o investimento (ROI), eficiência, crescimento.	Alta	Manter informado através de relatórios e reuniões mensais. Envolver em decisões estratégicas.
Gerente Administrativo	Usuário Chave / Especialista	Facilidade de uso, otimização do tempo, relatórios precisos.	Alta	Envolver ativamente na definição de requisitos e nos testes de aceitação. Coletar feedback contínuo.
Funcionários da Recepção	Usuários Finais	Simplicidade da interface, rapidez nas operações diárias.	Média	Realizar treinamentos práticos. Criar manuais de uso rápido.
Clientes (Atletas)	Usuários Finais	Agendamento fácil, pagamento seguro, boa experiência.	Baixa	Coletar feedback através de pesquisas de satisfação. Comunicar novidades e benefícios do sistema.
Equipe de Desenvolvimento	Executores do Projeto	Clareza nos requisitos, autonomia técnica, prazos realistas.	Média	Manter engajado através de reuniões diárias (daily scrums) e planejamento de sprints.

Plano de Comunicação:

- **Reuniões Semanais de Status (Equipe do Projeto):** Alinhamento de progresso, desafios e próximos passos.
- **Relatórios Quinzenais de Progresso (Gerente e Proprietários):** Sumário executivo do avanço, riscos e status do orçamento.
- **Reuniões de Demonstração (Stakeholders Chave):** Apresentação das funcionalidades desenvolvidas ao final de cada ciclo (sprint).

7.0 Critérios de Sucesso: O sucesso do projeto "Beach Time" será medido não apenas pela entrega do software e pelo impacto positivo que ele gera no negócio. Os critérios são:

Todas as funcionalidades descritas no projeto forem implementadas e aprovadas pelos clientes.

Feedback positivo dos clientes, medido por uma pesquisa de satisfação com nota média de 4/5

Adoção do sistema pela equipe interna, que deve conseguir operar de forma autônoma após o treinamento.

7.1 Processo de Encerramento: O projeto será encerrado quando:

O UAT for assinado pelo patrocinador (proprietário da arena), confirmando que todas as entregas estão de acordo com os critérios de aceitação.

O sistema estiver estável em ambiente de produção por pelo menos 30 dias (período de operação assistida).

Toda a documentação final for entregue e arquivada.

Uma reunião de "Lições Aprendidas" for conduzida com a equipe do projeto para documentar os sucessos e as áreas de melhoria para projetos futuros.

A equipe do projeto for desmobilizada e os recursos liberados.

📋 Documentação CRUD - Beach Time

🎯 Visão Geral do Projeto

Beach Time é uma aplicação web para gerenciamento de quadras esportivas de areia, desenvolvida com:

Frontend: Vue.js 3 + Vite

Backend: Node.js + Express

Banco de Dados: MySQL

Autenticação: JWT (JSON Web Tokens)

📁 Estrutura do Projeto

beach-time/

├── frontend/ # Aplicação Vue.js

│ ├── public/

```
|   |   └── images/
|   |       ├── quadra1.jpg
|   |       ├── quadra2.jpg
|   |       ├── quadra3.jpg
|   |       └── quadra4.jpg
|   ├── src/
|   |   ├── views/
|   |   |   ├── AuthView.vue     # Login e Registro
|   |   |   └── HomeView.vue     # Página Principal
|   |   ├── router/
|   |   |   └── index.js         # Configuração de Rotas
|   |   ├── App.vue
|   |   └── main.js
|   └── package.json
|
└── backend/                  # API Node.js
    ├── config/
    |   └── database.js         # Configuração MySQL
    ├── controllers/
    |   ├── authController.js    # Lógica de Autenticação
    |   └── avisoController.js   # Lógica de Avisos
    ├── middleware/
    |   └── authMiddleware.js    # Verificação JWT
    ├── routes/
    |   ├── authRoutes.js        # Rotas de Auth
    |   └── avisoRoutes.js       # Rotas de Avisos
    ├── .env                    # Variáveis de Ambiente
```

```
   ├── server.js             # Servidor Principal
   └── package.json
```

🗄️ Estrutura do Banco de Dados

Tabela: users

```
sqlCREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  password VARCHAR(255) NOT NULL,
  isAdmin BOOLEAN DEFAULT FALSE,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Tabela: avisos

```
sqlCREATE TABLE avisos (
  id INT AUTO_INCREMENT PRIMARY KEY,
  titulo VARCHAR(255) NOT NULL,
  descricao TEXT NOT NULL,
  adminId INT NOT NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (adminId) REFERENCES users(id)
);
```

🔐 Sistema de Autenticação

1. Registro de Usuário (CREATE)

Endpoint: POST /api/auth/register

Arquivo: controllers/authController.js

Request Body:

json{

"name": "João Silva",

"email": "joao@example.com",

"password": "senha123"

}

Response (201 Created):

json{

"message": "Usuário cadastrado com sucesso!",

"userId": 1

}

Funcionalidade:

✓ Valida se o email já existe

✓ Criptografa a senha com bcrypt

✓ Cria usuário no banco (isAdmin = false por padrão)

2. Login de Usuário (READ)

Endpoint: POST /api/auth/login

Arquivo: controllers/authController.js

Request Body:

json{

"email": "joao@example.com",

"password": "senha123"

}

Response (200 OK):

json{

"message": "Login realizado com sucesso!",

```
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": 1,
    "name": "João Silva",
    "email": "joao@example.com",
    "isAdmin": false
  }
}
```

Funcionalidade:
- ✅ Valida credenciais
- ✅ Compara senha com hash do banco
- ✅ Gera token JWT válido por 24h
- ✅ Retorna dados do usuário

---

3. **Verificar Token (READ)**

Endpoint: `GET /api/auth/verify`

Arquivo: `controllers/authController.js`

Headers:
```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```
Response (200 OK):

json{

  "user": {

    "id": 1,

    "name": "João Silva",

    "email": "joao@example.com",

    "isAdmin": false

  }

}

```

**Funcionalidade**:

- ✅ Valida token JWT

- ✅ Busca usuário no banco

- ✅ Retorna dados atualizados

---

## 📢 CRUD de Avisos

### 1. **Criar Aviso (CREATE)**

**Endpoint**: `POST /api/avisos`

**Arquivo**: `controllers/avisoController.js`

**Headers**:

```

Authorization: Bearer {token}

Content-Type: application/json

Request Body:

json{

"titulo": "Manutenção Programada",

"descricao": "A quadra Premium estará em manutenção no dia 25/11"

}

Response (201 Created):

json{

"message": "Aviso criado com sucesso!",

"avisoId": 1

}

Funcionalidade:

✓ Verifica se usuário é admin

✓ Cria aviso vinculado ao adminId

✓ Retorna ID do aviso criado

Restrições:

✗ Apenas admins podem criar avisos

2. Listar Avisos (READ)

Endpoint: GET /api/avisos

Arquivo: controllers/avisoController.js

Response (200 OK):

json[

{

```
  "id": 1,
  "titulo": "Manutenção Programada",
  "descricao": "A quadra Premium estará em manutenção no dia 25/11",
  "adminId": 1,
  "adminName": "Admin Beach",
  "createdAt": "2024-11-20T10:30:00.000Z",
  "updatedAt": "2024-11-20T10:30:00.000Z"
 },
 {
  "id": 2,
  "titulo": "Novo Horário de Funcionamento",
  "descricao": "A partir de dezembro, funcionaremos até 23h",
  "adminId": 1,
  "adminName": "Admin Beach",
  "createdAt": "2024-11-21T14:20:00.000Z",
  "updatedAt": "2024-11-21T14:20:00.000Z"
 }
]
```


Funcionalidade:

✓ Retorna todos os avisos

✓ JOIN com tabela users para pegar nome do admin

✓ Ordenado por data de criação (mais recente primeiro)

✓ Rota pública (não requer autenticação)

3. Buscar Aviso por ID (READ)

Endpoint: GET /api/avisos/:id

Arquivo: controllers/avisoController.js

Exemplo: GET /api/avisos/1

Response (200 OK):

json{

  "id": 1,

  "titulo": "Manutenção Programada",

  "descricao": "A quadra Premium estará em manutenção no dia 25/11",

  "adminId": 1,

  "adminName": "Admin Beach",

  "createdAt": "2024-11-20T10:30:00.000Z",

  "updatedAt": "2024-11-20T10:30:00.000Z"

}

Response (404 Not Found):

json{

  "message": "Aviso não encontrado"

}

```

---

### 4. **Atualizar Aviso (UPDATE)**

**Endpoint**: `PUT /api/avisos/:id`

**Arquivo**: `controllers/avisoController.js`

**Headers**:

```

Authorization: Bearer {token}

Content-Type: application/json

Request Body:

json{

"titulo": "Manutenção Programada - ATUALIZADO",

"descricao": "A manutenção foi adiada para o dia 26/11"

}

Response (200 OK):

json{

"message": "Aviso atualizado com sucesso!"

}

```

**Funcionalidade**:

- ✅ Verifica se usuário é admin

- ✅ Atualiza titulo e/ou descricao

- ✅ Atualiza campo `updatedAt` automaticamente

**Restrições**:

- ❌ Apenas admins podem atualizar

---

### 5. **Deletar Aviso (DELETE)**

**Endpoint**: `DELETE /api/avisos/:id`

**Arquivo**: `controllers/avisoController.js`
```

Headers:

```
Authorization: Bearer {token}
```

Exemplo: DELETE /api/avisos/1

Response (200 OK):

```
json{
  "message": "Aviso deletado com sucesso!"
}
```

Response (404 Not Found):

```
json{
  "message": "Aviso não encontrado"
}
```

Funcionalidade:

✅ Verifica se usuário é admin

✅ Deleta aviso do banco

✅ Verifica se aviso existe antes de deletar

Restrições:

❌ Apenas admins podem deletar

🔒 Middleware de Autenticação

Arquivo: middleware/authMiddleware.js

Função: Proteger rotas que exigem autenticação

Como funciona:

Recebe token JWT do header Authorization: Bearer {token}

Verifica validade do token com jwt.verify()

Decodifica o token e extrai userId

Busca usuário no banco de dados

Adiciona usuário ao req.user

Permite acesso à rota

Rotas Protegidas:

POST /api/avisos - Criar aviso (admin only)

PUT /api/avisos/:id - Atualizar aviso (admin only)

DELETE /api/avisos/:id - Deletar aviso (admin only)

GET /api/auth/verify - Verificar token

🧭 Frontend - Integração com API

AuthView.vue - Login e Registro

Registro:

```
javascriptasync handleSignUp() {
  const response = await fetch('http://localhost:5000/api/auth/register', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      name: this.signUpName,
      email: this.signUpEmail,
      password: this.signUpPassword
    })
  })
```

```
  const data = await response.json()

  if (response.ok) {
    // Sucesso - muda para tela de login
    this.isSignUp = false
  }
}
```

Login:

```
javascriptasync handleSignIn() {
  const response = await fetch('http://localhost:5000/api/auth/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      email: this.signInEmail,
      password: this.signInPassword
    })
  })

  const data = await response.json()

  if (response.ok) {
    localStorage.setItem('token', data.token)
    localStorage.setItem('user', JSON.stringify(data.user))
    this.$router.push('/home')
  }
}
```

HomeView.vue - Gerenciamento de Avisos

Listar Avisos:

javascriptasync fetchAvisos() {

```
  const response = await fetch('http://localhost:5000/api/avisos')
  const data = await response.json()
  this.avisos = data
}
```

Criar Aviso (Admin):

javascriptasync createAviso() {

```
  const token = localStorage.getItem('token')

  const response = await fetch('http://localhost:5000/api/avisos', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${token}`
    },
    body: JSON.stringify({
      titulo: this.novoAviso.titulo,
      descricao: this.novoAviso.descricao
    })
  })

  if (response.ok) {
    await this.fetchAvisos() // Recarrega lista
    this.novoAviso = { titulo: '', descricao: '' }
  }
}
```

Deletar Aviso (Admin):

javascriptasync deleteAviso(avisoId) {

```
  const token = localStorage.getItem('token')
```

```
 const response = await fetch(`http://localhost:5000/api/avisos/${avisoId}`, {
   method: 'DELETE',
   headers: {
     'Authorization': `Bearer ${token}`
   }
 })

 if (response.ok) {
   await this.fetchAvisos() // Recarrega lista
 }
}
```

🚀 Como Rodar o Projeto

Backend

```
bashcd beach-time-backend
npm install
npm run dev
```

Servidor roda em: http://localhost:5000

Frontend

```
bashcd vueboilerplate2
npm install
npm run dev
```

Aplicação roda em: http://localhost:5173

🔑 Variáveis de Ambiente (.env)

```
envDB_HOST=localhost
DB_USER=root
DB_PASSWORD=sua_senha_mysql
```

DB_NAME=beach_time

JWT_SECRET=beach_time_super_secret_key_2024

PORT=5000

```

---

## 📊 Fluxo de Dados

### Registro → Login → Home

```
1. Usuário preenche formulário de registro
  ↓
2. Frontend envia POST /api/auth/register
  ↓
3. Backend cria usuário no MySQL
  ↓
4. Usuário faz login
  ↓
5. Backend gera token JWT
  ↓
6. Frontend salva token no localStorage
  ↓
7. Frontend redireciona para /home
  ↓
8. HomeView verifica token com GET /api/auth/verify
  ↓
9. Se válido, carrega avisos com GET /api/avisos
```
```

Criar Aviso (Admin)

```
1. Admin logado clica em "Criar Aviso"
  ↓
2. Preenche título e descrição
  ↓
3. Frontend envia POST /api/avisos com token JWT
  ↓
4. Middleware verifica se é admin
  ↓
5. Backend cria aviso vinculado ao adminId
  ↓
6. Frontend recarrega lista de avisos
```

🛡️ Segurança

Senhas

- ✅ Criptografadas com **bcrypt** (10 rounds)
- ✅ Nunca armazenadas em texto puro
- ✅ Hash não pode ser revertido

Tokens JWT

- ✅ Assinados com chave secreta (JWT_SECRET)
- ✅ Expiração de 24 horas

- ✅ Incluem apenas dados necessários (userId, isAdmin)

Rotas Protegidas

- ✅ Middleware verifica token em todas as rotas sensíveis
- ✅ Verificação de permissão admin para criar/editar/deletar avisos

CORS

- ✅ Configurado para aceitar requisições do frontend

📝 Recursos Futuros

Agendamento de Quadras (CRUD)

- [ ] CREATE: Reservar horário em uma quadra
- [ ] READ: Ver reservas do usuário
- [ ] UPDATE: Remarcar reserva
- [ ] DELETE: Cancelar reserva

Sistema de Pagamentos

- [ ] Integração com gateway de pagamento
- [ ] Histórico de pagamentos

Notificações

- [ ] Email de confirmação de reserva
- [ ] WhatsApp para lembretes

□ Testando as APIs (Postman/Insomnia)

1. Registrar Usuário

```
POST http://localhost:5000/api/auth/register
Body: {
  "name": "Teste User",
  "email": "teste@test.com",
  "password": "123456"
}
```

2. Fazer Login

```
POST http://localhost:5000/api/auth/login
Body: {
  "email": "teste@test.com",
  "password": "123456"
}
```

3. Listar Avisos

```
GET http://localhost:5000/api/avisos
```

4. Criar Aviso (com token)

```
```

POST http://localhost:5000/api/avisos

Headers: Authorization: Bearer {seu_token}

Body: {

"titulo": "Teste Aviso",

"descricao": "Descrição do teste"

- }