RELATÓRIO DE AVALIAÇÃO DA QUALIDADE DO SOFTWARE DO PROJETO

Análise das Funcionalidades e Qualidade Técnica

Síntese

Este relatório avalia a qualidade do software desenvolvido para a gestão de horários no Iscte. O projeto possui funcionalidades sólidas e testes automatizados, mas carece de visualizações gráficas completas e melhorias na interface do utilizador.

Edgar Costa
Manuel Cunha
João Costa
Tiago Martins
Rafael Pardal
Leonardo Costa

Introdução

Este relatório avalia a qualidade do software do projeto de visualização de horários de aulas e ocupação de salas. A análise é baseada nos seguintes ficheiros fornecidos:

▪ **script.js**
▪ **ClassScheduler.js**
▪ **ClassScheduler.test.js**
▪ **DateConverter.js**
▪ **DateConverter.test.js**

Critérios de Avaliação

A avaliação será feita com base nos seguintes critérios:

⇒ **Funcionalidade**
⇒ **Usabilidade**
⇒ **Eficiência**
⇒ **Manutenibilidade**
⇒ **Testabilidade**

1. Funcionalidade

Cobertura de Requisitos:

Carregamento de Horários a partir de CSV
Status: Implementado
Descrição: A aplicação permite carregar horários a partir de arquivos CSV locais ou remotos, como no GitHub.
Detalhes Técnicos: Utilização da biblioteca Papa.parse para leitura de arquivos CSV. A função handleFileSelect em script.js gere o carregamento dos dados, que são então processados e exibidos na tabela.

Exibição e Navegação do Horário em Tabela
Status: Implementado
Descrição: O horário é exibido em tabela com todas as colunas necessárias. A tabela permite esconder colunas, ordenar por qualquer campo e aplicar filtros com operadores E e OU. As colunas "Semana do ano" e "Semana do semestre" são adicionadas corretamente.
Detalhes Técnicos: Utilização da biblioteca Tabulator para a exibição e manipulação da tabela. As funções initializeTable e applyCustomFilters em script.js gerem a criação e filtragem da tabela.

Exibição e Navegação do Cadastro de Salas
Status: Implementado
Descrição: O cadastro de salas é exibido em tabela e permite filtrar por características, disponibilidade e combinação de filtros com operadores E e OU.
Detalhes Técnicos: Similar à exibição do horário, a biblioteca Tabulator é utilizada para a tabela de salas. A função initializeTable2 em script.js gere a criação da tabela.

Gravação de Horários em CSV e JSON
Status: Implementado
Descrição: A aplicação permite gravar o horário alterado pelo utilizador em formatos CSV e JSON.
Detalhes Técnicos: As funções exportToCSV e exportToJSON em script.js gerem a exportação dos dados da tabela para os formatos CSV e JSON.

Sugestão de Slots para Aulas de Substituição
Status: Implementado
Descrição: O software sugere slots de substituição baseando-se em regras definidas pelo utilizador, como períodos excluídos e preferências de salas.
Detalhes Técnicos: A função findSubstitutionSlots em ClassScheduler.js processa os dados para sugerir slots de substituição. A interface para entrada dos filtros e visualização dos slots está parcialmente implementada em script.js.

Sugestão de Slots para Aulas de UC
Status: Implementado
Descrição: O software sugere slots para aulas de UC com base em regras definidas pelo utilizador, incluindo número de aulas, períodos excluídos e preferências de salas.
Detalhes Técnicos: A função findUCAllocationSlots em ClassScheduler.js processa os dados para sugerir slots para aulas de UC. A interface para entrada dos filtros e visualização dos slots está parcialmente implementada em script.js.

Visualização de Sugestões de Slots
Status: Parcialmente Implementado
Descrição: As sugestões são exibidas em tabela, mas a interface para seleção, eliminação e adição manual de alternativas pode ser melhorada.
Detalhes Técnicos: As funções showSubstitutionTable e showUCAllocationTable em script.js gerem a exibição das sugestões em tabelas. A interação com o utilizador para selecionar ou eliminar sugestões necessita de melhorias.

Visualização Gráfica de Conflitos entre Aulas
Status: Parcialmente Implementado
Descrição: As funções para identificar conflitos foram implementadas, mas a visualização gráfica utilizando diagramas de rede ou chord diagrams não está completa.
Detalhes Técnicos: A função findConflicts em ClassScheduler.js identifica os conflitos entre aulas. A função drawConflictGraph em script.js é responsável pela renderização gráfica, que ainda precisa de ser completada.

Visualização Gráfica do Mapa de Ocupação das Salas
Status: Parcialmente Implementado
Descrição: As funções para processar dados de ocupação foram implementadas, mas a visualização gráfica utilizando heatmaps não está completa.
Detalhes Técnicos: A função processRoomData em script.js processa os dados de ocupação das salas. A função drawHeatmap é responsável pela renderização gráfica, que ainda precisa de ser completada.

2. Usabilidade

Interface de Utilizador:
Status: Implementado, mas com melhorias necessárias
Descrição: A interface para carregar arquivos e aplicar filtros está bem organizada. A utilização de bibliotecas como Tabulator e D3.js é adequada para a manipulação de dados e gráficos.
Detalhes Técnicos: As interações com o utilizador são geridas por eventos JavaScript, como visto em script.js. Contudo, a falta de visualizações gráficas completas limita a usabilidade final.

Facilidade de Navegação:
Status: Implementado, mas com melhorias necessárias
Descrição: A navegação entre diferentes tabelas e modos de exibição é clara, mas poderia ser melhorada com uma interface mais amigável e menos dependente de prompts e alertas do navegador.
Detalhes Técnicos: Funções como showSubstitutionTable e showUCAllocationTable em script.js controlam a navegação entre diferentes tabelas.

3. Eficiência

Desempenho:
Status: Implementado
Descrição: O processamento de dados é feito de maneira síncrona, o que pode ser ineficiente para grandes volumes de dados. As operações de filtro e procura de slots são feitas de forma iterativa e podem ser otimizadas.
Detalhes Técnicos: Funções como findOpenSlots e processRoomData em ClassScheduler.js e script.js realizam operações iterativas que podem ser melhoradas em termos de desempenho.

Tempo de Resposta:
Status: Implementado
Descrição: As operações básicas de carregamento e filtragem de dados são rápidas, mas a renderização dos gráficos deve ser avaliada para garantir tempos de resposta aceitáveis uma vez implementada.
Detalhes Técnicos: A função drawConflictGraph e drawHeatmap em script.js são responsáveis pela renderização gráfica, cujo desempenho precisa ser avaliado.

4. Manutenibilidade

Estrutura do Código:
Status: Implementado
Descrição: O código está bem modularizado com classes específicas para diferentes funcionalidades (ClassScheduler, DateConverter). O uso de ES6 e módulos JavaScript ajuda na organização, mas a ausência de documentação detalhada em algumas funções dificulta a compreensão e manutenção.
Detalhes Técnicos: Funções e classes estão bem definidas e separadas em ClassScheduler.js e DateConverter.js.

Comentários e Documentação:
Status: Implementado
Descrição: Há comentários explicativos em DateConverter.js e ClassScheduler.js, mas a documentação poderia ser mais completa e incluir exemplos de uso. script.js carece de comentários detalhados, o que pode dificultar a manutenção e evolução do código.
Detalhes Técnicos: Documentação adicional e comentários são necessários para todas as funções e classes.

5. Testabilidade

Testes Automatizados:
Status: Implementado
Descrição: Existem testes automatizados para DateConverter.js e ClassScheduler.js, o que é positivo e contribui para a qualidade do software. A cobertura de testes é abrangente,

validando as principais funcionalidades, como conversão de datas, adição de semanas, detecção de conflitos e alocação de slots.
Detalhes Técnicos: Arquivos de teste como DateConverter.test.js e ClassScheduler.test.js cobrem diversas funcionalidades críticas.

Facilidade de Teste:
Status: Implementado
Descrição: A modularização do código facilita a criação de testes unitários. No entanto, a dependência de bibliotecas externas e a interação direta com o DOM (em script.js) pode complicar a criação de testes automatizados.
Detalhes Técnicos: Funções e classes são testáveis devido à sua modularização.

```
error  Static method 'findSubstitutionSlots' has a complexity of 3. Maximum allowed is 2  complexity
error  Static method 'findUCAllocationSlots' has a complexity of 5. Maximum allowed is 2  complexity
error  Static method 'findOpenSlots' has a complexity of 4. Maximum allowed is 2          complexity
error  Arrow function has a complexity of 3. Maximum allowed is 2                         complexity
error  Arrow function has a complexity of 4. Maximum allowed is 2                         complexity
error  Static method 'findConflicts' has a complexity of 4. Maximum allowed is 2          complexity
error  Static method 'isConflicting' has a complexity of 3. Maximum allowed is 2          complexity
```

```
25:25  error  Arrow function has a complexity of 3. Maximum allowed is 2                     complexity
62:33  error  Static method 'getSemesterWeekNumber' has a complexity of 5. Maximum allowed is 2  complexity
```

Nas imagens acima podemos verificar a complexidade ciclomática de algumas funções que falham nos testes. Concluímos que alguns métodos não têm a complexidade que aparece nos resultados devido à biblioteca jslint usada.

```
PASS src/ClassScheduler.test.js
-------------------|---------|----------|---------|---------|-------------------
File               | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------|---------|----------|---------|---------|-------------------
All files          |   75.67 |       72 |   72.72 |   75.47 |
 ClassScheduler.js |   66.66 |    61.76 |    64.7 |   65.78 | 45,62-97,132-135
 DateConverter.js  |     100 |    93.75 |     100 |     100 | 51
-------------------|---------|----------|---------|---------|-------------------

Test Suites: 2 passed, 2 total
Tests:       22 passed, 22 total
```

```
import DateConverter from "./DateConverter";

describe('convertToDate', () => {
    test('deve retornar uma data correta para uma string de data válida', () => {
      const str = '01/12/2022';
      const result = DateConverter.convertToDate(str);
      expect(result).toEqual(new Date(2022, 11, 1));
    });

    test('deve retornar null para uma string de data inválida', () => {
      const str = '';
      const result = DateConverter.convertToDate(str);
      expect(result).toBeNull();
    });
  });


  describe('addWeeksToData', () => {
    test('deve adicionar semanas do ano e do semestre corretamente', () => {
      const data = [
        { 'Data da aula': '01/01/2024' },
        { 'Data da aula': '15/02/2024' },
      ];

      const result = DateConverter.addWeeksToData(data);

      expect(result).toEqual([
        { 'Data da aula': '01/01/2024', 'Semana do ano': 1, 'Semana do semestre': 'Fora do semestre' },
        { 'Data da aula': '15/02/2024', 'Semana do ano': 7, 'Semana do semestre': 3 },
      ]);
    });


  describe('getWeekNumber', () => {
    test('deve retornar o número da semana corretamente para uma data específica', () => {
      const date = new Date('2024-03-19');
      const result = DateConverter.getWeekNumber(date);
      expect(result).toBe(12);
    });
  });


describe('getSemesterWeekNumber', () => {
  test('deve retornar o número da semana do semestre corretamente para o primeiro semestre', () => {
    const result = DateConverter.getSemesterWeekNumber(10);
    expect(result).toBe(6);
  });

  test('deve retornar o número da semana do semestre corretamente para o segundo semestre', () => {
    const result = DateConverter.getSemesterWeekNumber(40);
    expect(result).toBe(6);
  });

  test('deve retornar "Fora do semestre" para semanas fora dos semestres', () => {
    const result = DateConverter.getSemesterWeekNumber(2);
    expect(result).toBe('Fora do semestre');
  });
});
```

```
describe('getSemesterWeekNumber', () => {
    test('Deve retornar o número da semana correta para o primeiro semestre', () => {
      expect(DateConverter.getSemesterWeekNumber(5)).toBe(1);
      expect(DateConverter.getSemesterWeekNumber(10)).toBe(6);
      expect(DateConverter.getSemesterWeekNumber(19)).toBe(15);
    });

    test('Deve retornar o número da semana correta para o segundo semestre', () => {
      expect(DateConverter.getSemesterWeekNumber(35)).toBe(1);
      expect(DateConverter.getSemesterWeekNumber(40)).toBe(6);
      expect(DateConverter.getSemesterWeekNumber(49)).toBe(15);
    });

    test('Deve retornar "Fora do semestre" para semanas fora dos semestres', () => {
      expect(DateConverter.getSemesterWeekNumber(4)).toBe('Fora do semestre');
      expect(DateConverter.getSemesterWeekNumber(20)).toBe('Fora do semestre');
      expect(DateConverter.getSemesterWeekNumber(34)).toBe('Fora do semestre');
      expect(DateConverter.getSemesterWeekNumber(50)).toBe('Fora do semestre');
    });
  });
```

```
describe('ClassScheduler', () => {
    describe('findSubstitutionSlots', () => {
        it('should return an alert if horario or salas are missing', () => {
            global.alert = jest.fn();  // Mock alert function
            const result = ClassScheduler.findSubstitutionSlots({}, null, null);
            expect(global.alert).toHaveBeenCalledWith('Por favor, carregue os ficheiros de horários e salas primeiro.');
            expect(result).toEqual([]);
        });

        it('should return the correct slots', () => {
            const filters = { DataIni: new Date('2024-05-14'), DataFim: new Date('2024-05-15') };
            const horario = [
                { 'Data da aula': '14/05/2024', 'Sala atribuída à aula': 'A101', 'Hora início da aula': '08:00', 'Hora fim da aula': '10:00
                { 'Data da aula': '14/05/2024', 'Sala atribuída à aula': 'B202', 'Hora início da aula': '11:00', 'Hora fim da aula': '13:00
                { 'Data da aula': '15/05/2024', 'Sala atribuída à aula': 'A101', 'Hora início da aula': '09:00', 'Hora fim da aula': '11:00
            ];
            const salas = [
                { 'Nome sala': 'A101' },
                { 'Nome sala': 'B202' }
            ];
            const expectedSlots = [
                { data: '2024-05-14', sala: 'A101', HoraIni: '10:00', HoraFim: '22:30' },
                { data: '2024-05-14', sala: 'B202', HoraIni: '08:00', HoraFim: '11:00' },
                { data: '2024-05-14', sala: 'B202', HoraIni: '13:00', HoraFim: '22:30' },
                { data: '2024-05-15', sala: 'A101', HoraIni: '11:00', HoraFim: '22:30' }
            ];

            jest.spyOn(ClassScheduler, 'findOpenSlots').mockReturnValue(expectedSlots);

            const result = ClassScheduler.findSubstitutionSlots(filters, horario, salas);
            expect(result).toEqual(expectedSlots);
        });
    });
```

```
describe('findUCAllocationSlots', () => {
    it('should return an alert if horario or salas are missing', () => {
        global.alert = jest.fn();
        const result = ClassScheduler.findUCAllocationSlots({}, null, null);
        expect(global.alert).toHaveBeenCalledWith('Por favor, carregue os ficheiros de horários e salas primeiro.');
        expect(result).toEqual([]);
    });

    it('should allocate the correct number of classes', () => {
        const filters = { UCName: 'Test', NumberOfClasses: 2 };
        const horario = [
            { 'Data da aula': '14/05/2024', 'Sala atribuída à aula': 'A101', 'Hora início da aula': '08:00', 'Hora fim da aula': '10:00' },
            { 'Data da aula': '15/05/2024', 'Sala atribuída à aula': 'A101', 'Hora início da aula': '09:00', 'Hora fim da aula': '11:00' }
        ];
        const salas = [
            { 'Nome sala': 'A101' },
            { 'Nome sala': 'B202' }
        ];
        const slots = [
            { data: '2024-05-14', sala: 'A101', HoraIni: '10:00', HoraFim: '22:30' },
            { data: '2024-05-15', sala: 'A101', HoraIni: '11:00', HoraFim: '22:30' }
        ];

        jest.spyOn(ClassScheduler, 'findOpenSlots').mockReturnValue(slots);

        const result = ClassScheduler.findUCAllocationSlots(filters, horario, salas);
        expect(result).toEqual([
            { data: '2024-05-14', sala: 'A101', HoraIni: '10:00', HoraFim: '22:30', UC: 'Test' },
            { data: '2024-05-15', sala: 'A101', HoraIni: '11:00', HoraFim: '22:30', UC: 'Test' }
        ]);
    });
});
```

```
describe('findOpenSlots', () => {
    it('should return empty array if dates are not defined', () => {
        const result = ClassScheduler.findOpenSlots([], [], {});
        expect(result).toEqual([]);
    });


});

describe('listSalas', () => {
    it('should return the list of salas sorted', () => {
        const salas = [{ 'Nome sala': 'B102' }, { 'Nome sala': 'A101' }];
        const result = ClassScheduler.listSalas(salas);
        expect(result).toEqual(['A101', 'B102']);
    });
});



describe('findConflicts', () => {
    it('should find conflicting classes', () => {
        const data = [
            { 'Data da aula': '14/05/2024', 'Hora início da aula': '08:00', 'Hora fim da aula': '10:00' },
            { 'Data da aula': '14/05/2024', 'Hora início da aula': '09:00', 'Hora fim da aula': '11:00' }
        ];
        jest.spyOn(DateConverter, 'convertToDate').mockImplementation((date) => new Date(date.split('/').reverse().join('-')));

        const result = ClassScheduler.findConflicts(data);
        expect(result).toEqual([{ source: data[0], target: data[1] }]);
    });
});
```

```
describe('isConflicting', () => {
    it('should return true if classes are conflicting', () => {
        const aula1 = { 'Data da aula': '14/05/2024', 'Hora início da aula': '08:00', 'Hora fim da aula': '10:00' };
        const aula2 = { 'Data da aula': '14/05/2024', 'Hora início da aula': '09:00', 'Hora fim da aula': '11:00' };
        jest.spyOn(DateConverter, 'convertToDate').mockImplementation((date) => new Date(date.split('/').reverse().join('-')));

        const result = ClassScheduler.isConflicting(aula1, aula2);
        expect(result).toBe(true);
    });

    it('should return false if classes are not conflicting', () => {
        const aula1 = { 'Data da aula': '14/05/2024', 'Hora início da aula': '08:00', 'Hora fim da aula': '10:00' };
        const aula2 = { 'Data da aula': '15/05/2024', 'Hora início da aula': '09:00', 'Hora fim da aula': '11:00' };
        jest.spyOn(DateConverter, 'convertToDate').mockImplementation((date) => new Date(date.split('/').reverse().join('-')));

        const result = ClassScheduler.isConflicting(aula1, aula2);
        expect(result).toBe(false);
    });
});


describe('ClassScheduler - getDisponibilidadeSala', () => {

    test('should return available time slots for a given classroom schedule', () => {

        const horario = [
            { 'Sala atribuída à aula': 'A101', 'Hora início da aula': '08:00:00', 'Hora fim da aula': '10:00:00' },
            { 'Sala atribuída à aula': 'A101', 'Hora início da aula': '13:00:00', 'Hora fim da aula': '15:00:00' },

        ];


        const result = ClassScheduler.getDisponibilidadeSala(horario, 'A101');

        const expected = [
            { 'HoraIni': '10:00:00', 'HoraFim': '13:00:00' },
            { 'HoraIni': '15:00:00', 'HoraFim': '22:30:00' }

        ];


        expect(result).toEqual(expected);
    });


});
```

Conclusão

O projeto apresenta uma base sólida com a maioria das funcionalidades principais implementadas e validadas por testes automatizados. No entanto, faltam visualizações gráficas completas e melhorias na interface do utilizador. A documentação e a otimização do código também podem ser melhoradas. Além disso, a identificação de code smells como funções longas, repetição de código e manipulação direta do DOM destaca a necessidade de refatorações para melhorar a qualidade e manutenibilidade do software.