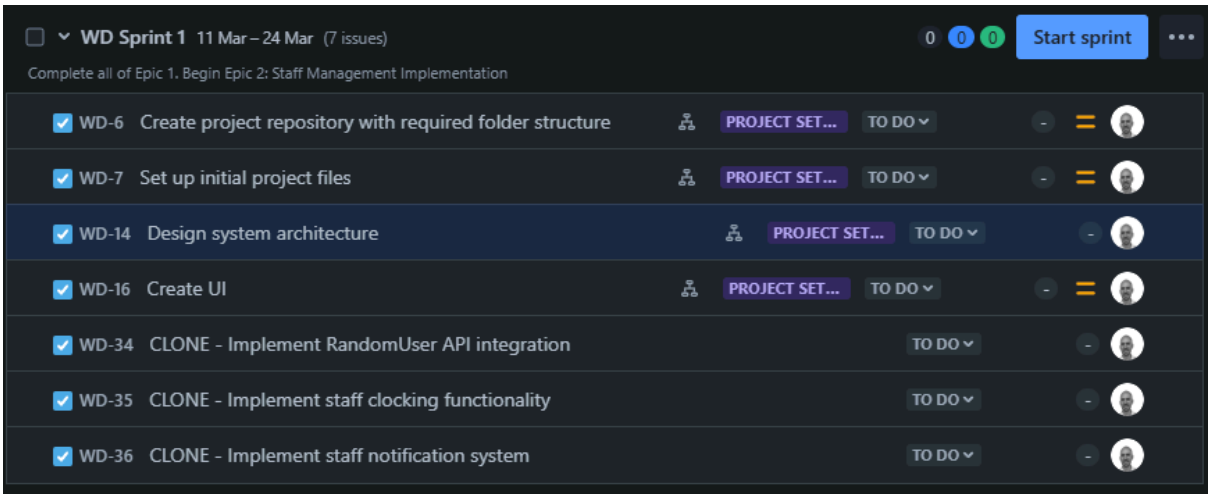
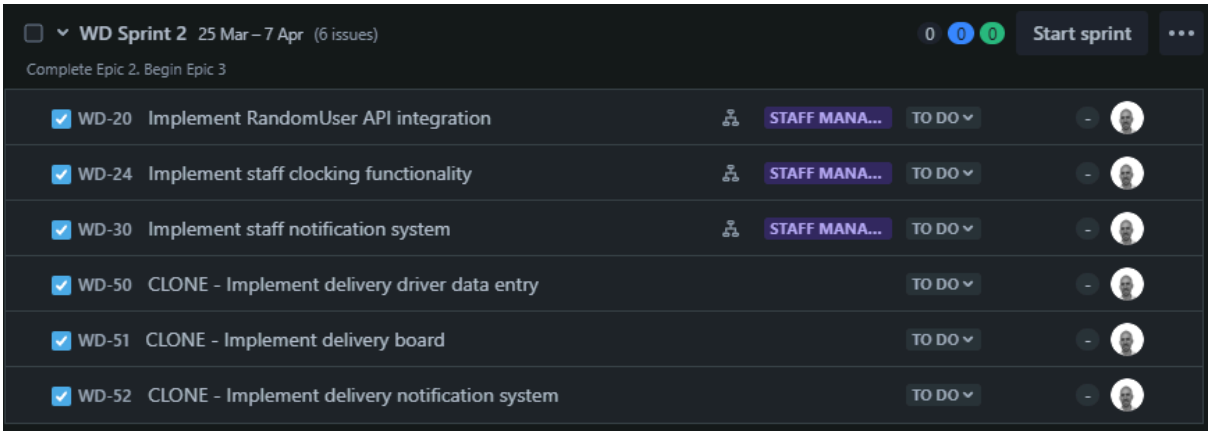
Reflection Report

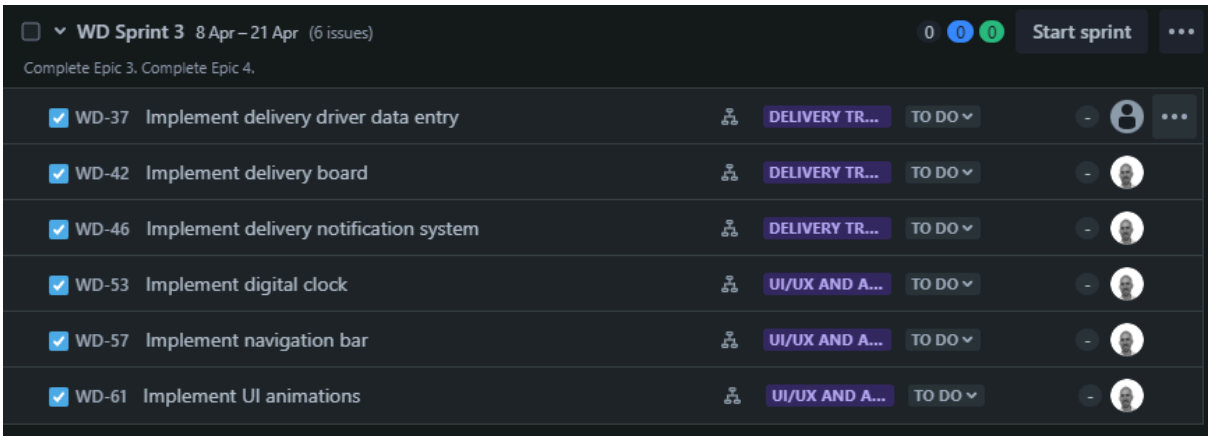
Sprint 1:



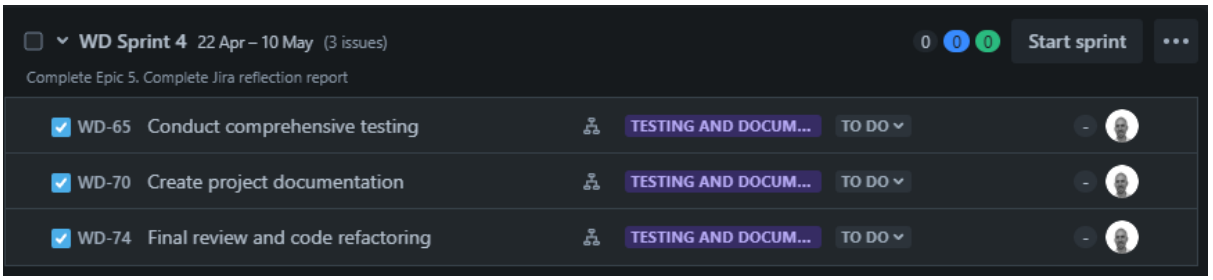
Sprint 2:



Sprint 3:



Sprint 4:



Epic 1 and issues: Project Setup and Planning

WD-1	Project Setup and Planning
WD-6	Create project repository with required folder structure
WD-8	Set up "Web Application" folder
WD-9	Set up "Documentation" folder
WD-10	Create readme.md file
WD-7	Set up initial project files
WD-13	Create initial wdt_app.js file
WD-11	Create HTML structure
WD-12	Create CSS file with company branding
WD-16	Create UI
WD-17	Design Dashboard layout
WD-18	Design Staff table
WD-19	Design Delivery Driver board
WD-14	Design system architecture
WD-15	Define object classes and inheritance structure

This was placed as the first epic to establish a solid foundation.

- **Repository Structure**: Set up folder structure and readme.md requirements.
- **Initial Files**: Established files, including CSS with company branding, before creating functionality.
- **Create UI**: Followed the provided dashboard mockup to guide layout.
- **System Architecture**: Planned the object hierarchy and implemented the Employee, StaffMember, and DeliveryDriver classes with proper inheritance.

Epic 2 and issues: Staff Management Implementation

WD-2	Staff Management Implementation
WD-20	Implement RandomUser API integration
WD-21	Create staffUserGet function
WD-22	Process API response and create staff objects
WD-23	Display staff information in table
WD-24	Implement staff clocking functionality
WD-25	Create Employee class
WD-26	Create StaffMember class with inheritance from Employee
WD-78	Create DeliveryDriver class with inheritance from Employee
WD-27	Implement staffOut function with duration input
WD-28	Implement staffIn function
WD-29	Create time calculation logic
WD-30	Implement staff notification system
WD-31	Create staffMemberIsLate function
WD-32	Implement toast notification for late staff members
WD-33	Add clear notification functionality

- **RandomUser API**: Implemented to meet the requirement for using randomuser.me and the staffUserGet function.
- **Clocking Functionality**: Fulfills the need for class inheritance and "clock in/out" with staffOut and staffIn functions.
- **Staff Notifications**: Implementation of toast notification requirement and use of the staffMemberIsLate function.

Epic 3 and issues: Delivery Tracking Implementation

WD-3	Delivery Tracking Implementation
WD-37	Implement delivery driver data entry
WD-38	Create delivery driver input form
WD-41	Implement validateDelivery function
WD-42	Implement delivery board
WD-43	Implement addDelivery function
WD-44	Implement vehicle type icons
WD-45	Display delivery information in board
WD-46	Implement delivery notification system
WD-47	Create deliveryDriverIsLate function
WD-48	Implement toast notification for late drivers
WD-49	Create confirmation popup for clearing deliveries

- **Delivery Driver Data Entry**: Handles manual delivery input with proper validation.
- **Delivery Board**: Focuses on displaying and managing deliveries, including vehicle icons.
- **Delivery Notifications**: Implements the deliveryDriverIsLate function and confirmation popup for clearing entries.

Epic 4 and issues: UI/UX and Additional Features

	WD-4	UI/UX and Additional Features
	WD-53	Implement digital clock
	WD-54	Create digitalClock function
	WD-55	Format date and time as specified
	WD-56	Update clock every second
	WD-57	Implement navigation bar
	WD-58	Create navbar with specified menu items
	WD-59	Implement dropdown functionality
	WD-60	Style according to brand guideline
	WD-61	Implement UI animations
	WD-62	Add button hover animations
	WD-63	Add toast notification animations
	WD-64	Add additional UX enhancements

These features improve user experience but aren't essential to core functionality, so they were implemented last.

- **Digital Clock**: Added to meet the requirement for a date and time display for the receptionist.
- **Navigation Bar**: Built to fulfill the requirement for a navbar.
- **UI Animations**: Implemented hover effects and UX enhancements as specified.

Epic 5 and issues: Testing and Documentation

	WD-5	Testing and Documentation
	WD-65	Conduct comprehensive testing
	WD-66	Test staff management functionality
	WD-67	Test delivery tracking functionality
	WD-68	Test notification systems
	WD-70	Create project documentation
	WD-71	Update readme.md with installation instructions
	WD-72	Document external libraries used
	WD-73	Create summary report
	WD-74	Final review and code refactoring
	WD-75	Optimize code performance
	WD-76	Ensure compliance with OOP principles
	WD-77	Final review against requirements

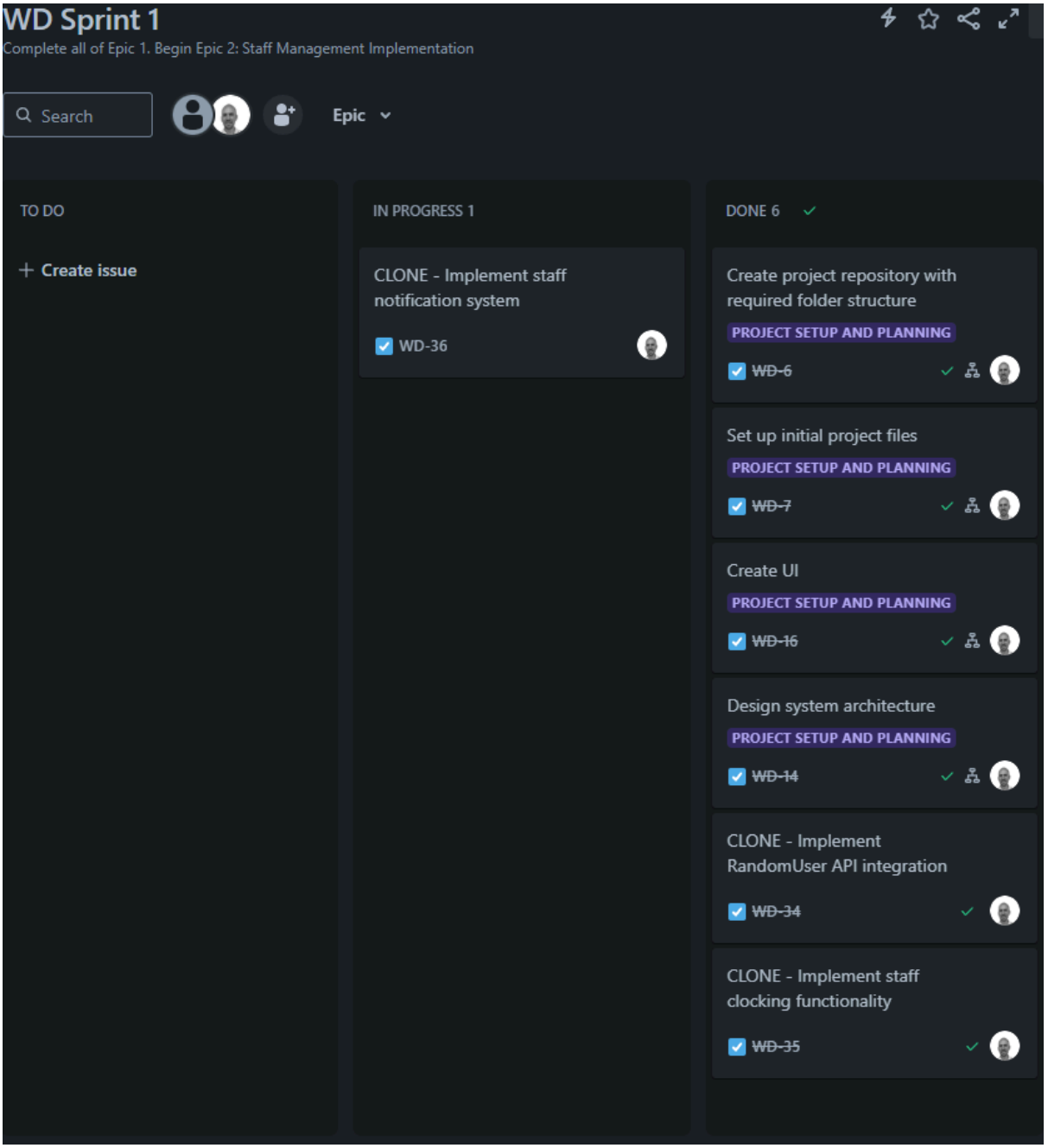
Scheduled near the end of the project to allow for full testing and documentation of completed features.

- **Testing**: Ensured all functionality was thoroughly tested to meet requirements.
- **Documentation**: Covered required deliverables, including readme.md and summary report.
- **Code Review**: Focused on code quality and adherence to object-oriented principles.

Timeline:


FEB
MAR
APR
Sprints
WD Sprint 1
WD Sprint 2
WD Sprint 3
WD Sprint 4
WD-1 Project Setup and Planning
WD-2 Staff Management Implementation
WD-3 Delivery Tracking Implementation
WD-4 UI/UX and Additional Features
WD-5 Testing and Documentation

Board and progress report after Sprint 1:

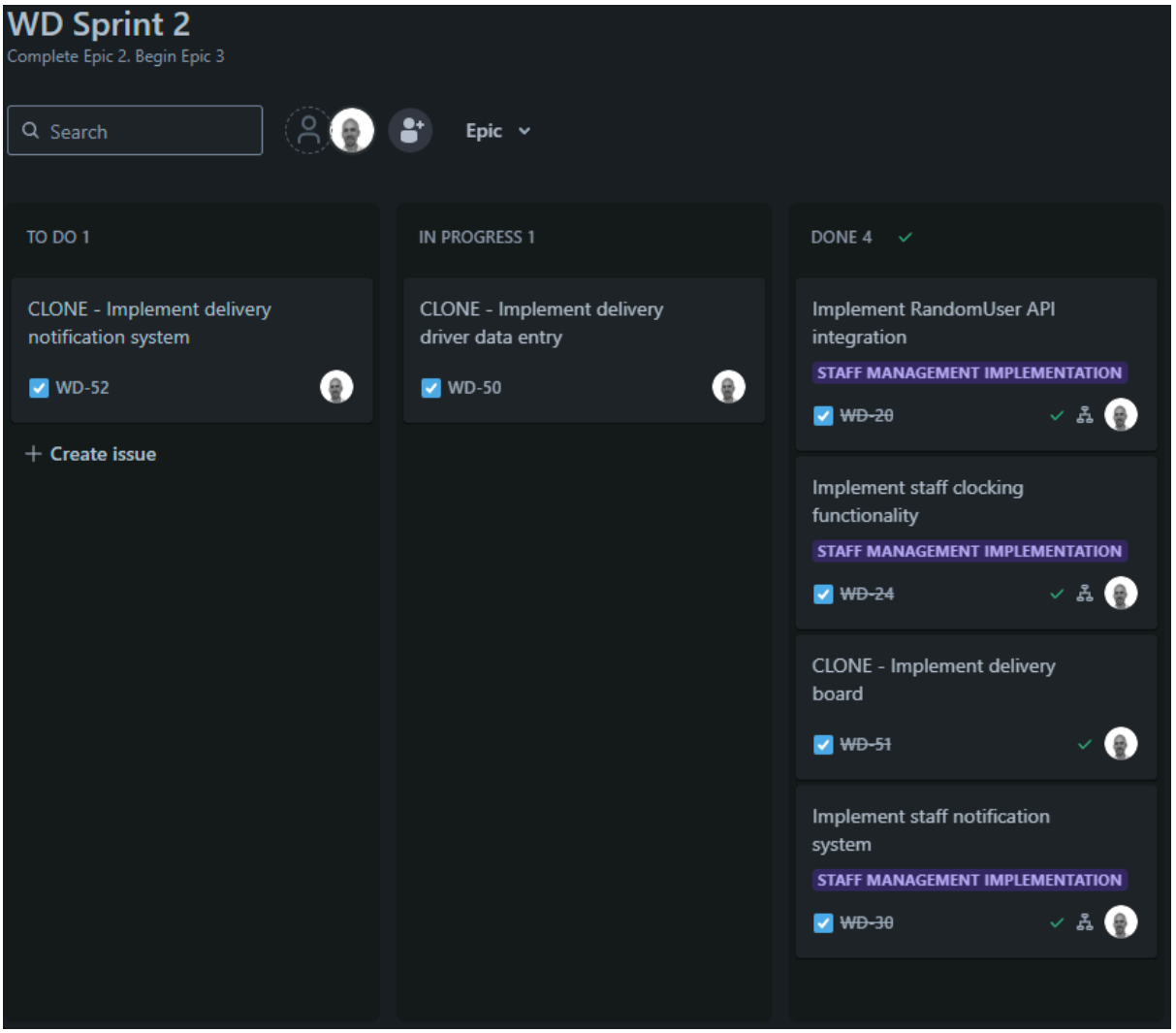


Progress elaboration:

Completed 100% of planned tasks. Key achievements include:

- **API Integration**: Implemented randomuser API to fetch staff.
- **Class Implementation**: Used OOP with StaffMember and DeliveryDriver classes inheriting from Employee.
- **UI Design**: Completed dashboard UI with nav bar, staff table, delivery form, and delivery board.

Board and progress report after Sprint 2:

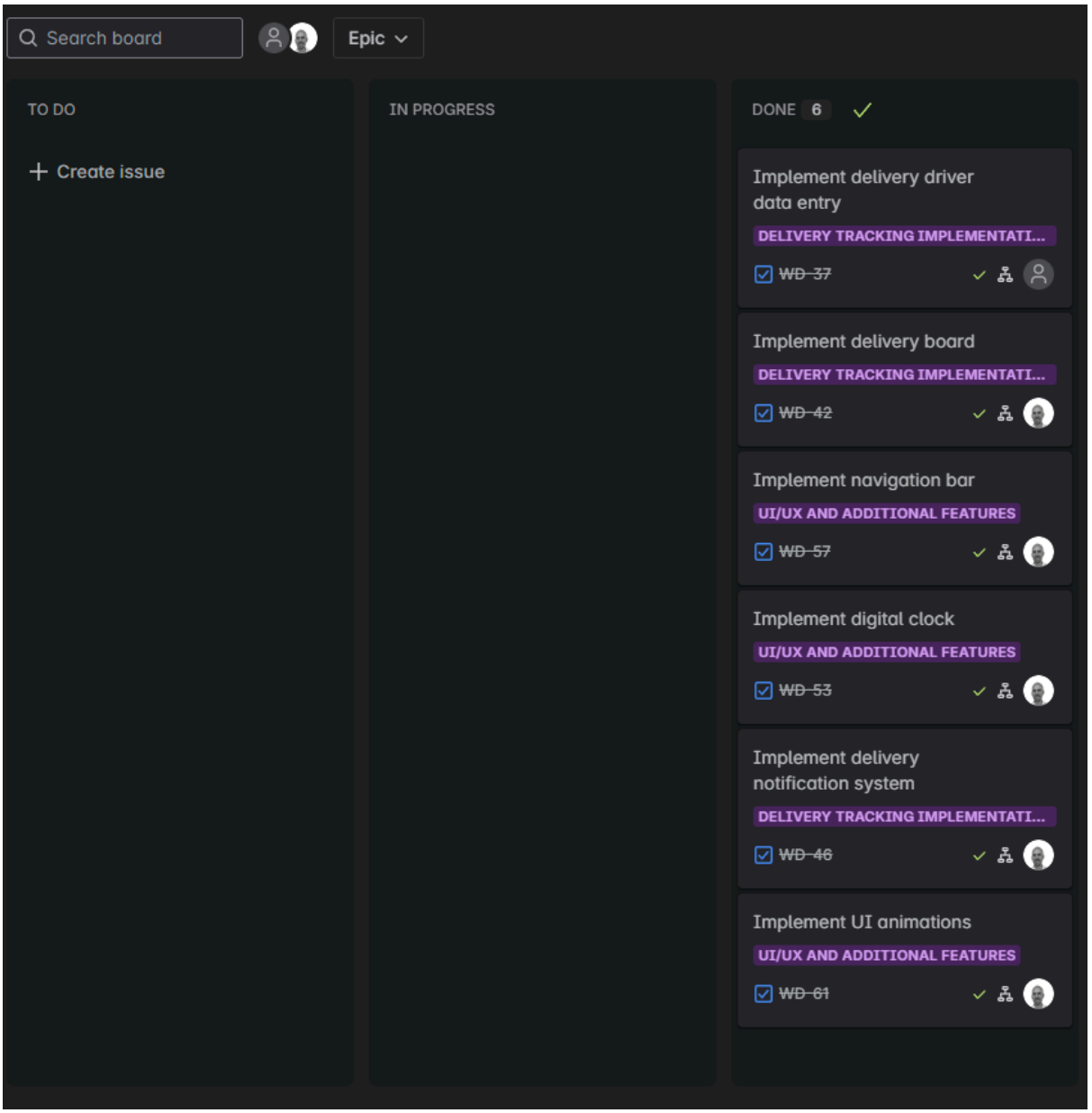


Progress elaboration:

Made steady progress despite having less time due to personal stuff.. Key achievements:

- **Delivery Driver Input**: Built input form and started validation, with some fields still pending.
- **Form Functionality**: Added delivery input display and vehicle icons.
- **Notification System**: Fixed toast notifications for the Staff table and added stacking for multiple late notifications.

Board and progress report after Sprint 3:

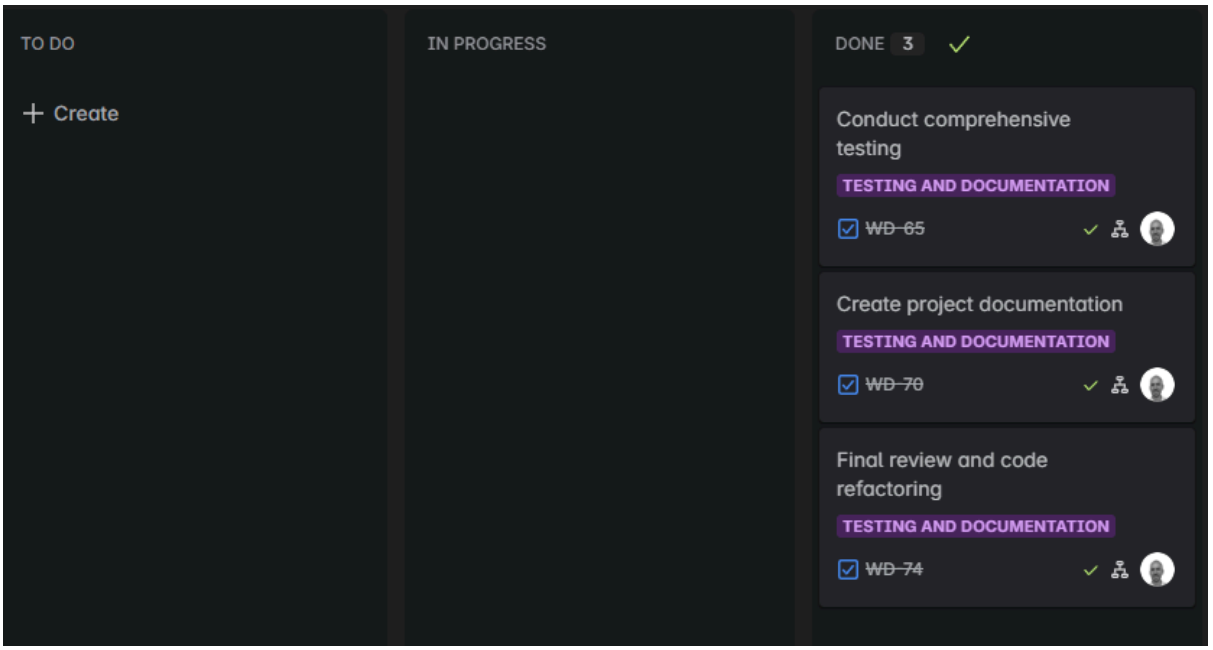


Progress elaboration:

All tasks completed successfully. Key achievements:

- **Form Validation**: Completed delivery driver form validation.
- **Digital Clock**: Added live updating digital clock.
- **Notification System**: Finished the notification system with stacked toast notifications for late drivers.

Board and progress report after Sprint 4:

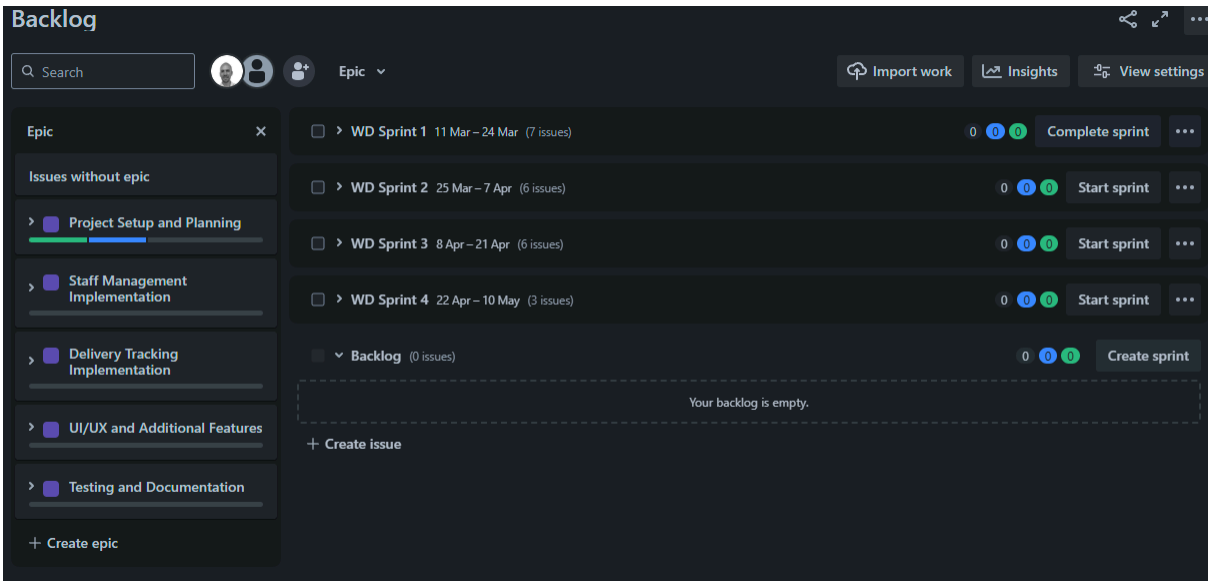


Progress elaboration:

Completed testing, documentation, and code refinement. Key achievements:

- **Testing**: Verified core features like staff clock-in/out and form validation.
- **Documentation**: Prepared detailed README and PDF progress report.
- **Code Refinement**: Improved code clarity and organization.

Backlog:



Summary Report:

Technical Choices & Key Decisions

- **UI Component Libraries**: Bootstrap helped create a responsive design, with ready-made components like tables and forms, which were customized to match branding requirements.

- **Helpful JavaScript Libraries**:

  - **jQuery**: Simplified DOM manipulation and event handling.
  - **jQuery Validate**: Assisted with form input validation for scheduling deliveries.
  - **SweetAlert2**: Used for styled pop-ups and alerts.
  - **Font Awesome**: Provided vehicle icons.

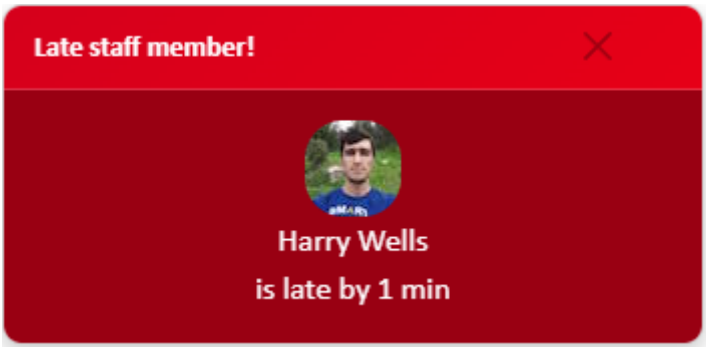
Challenges & Solutions

Challenge 1: Tracking Staff in Real-Time
The most complex feature was tracking whether staff returned on time from breaks or external appointments. To solve this, I:

- Used time checks for comparing expected and actual return times.
- Created unique toast notifications for each staff member.
- Set up a regular checking interval.

```
const currentTime = new Date();
const expectedReturnTime = new Date();
const [hours, minutes] = this.expectedReturn.split(":").map(Number);
expectedReturnTime.setHours(hours, minutes, 0, 0);

const lateBy = Math.trunc(
    (currentTime.getTime() - expectedReturnTime.getTime()) / (1000 * 60)
);
```



Challenge 2: Address Validation for Deliveries

Addresses vary in format, so building flexible yet accurate validation was a challenge. I used a **regex pattern** to ensure the format followed a “Street Name + Number” structure (e.g., “Main Street 123”) and showed users helpful feedback using SweetAlert if the input was incorrect.

```
$.validator.addMethod(
    "streetNumber",
    function (value) {
        return /^[A-Za-z\s]+(?:\s\d+[A-Za-z]?)$/.test(value);
    },
    "This field is required."
);
```

Project Process

- **Foundation**: Built core HTML structure and layout.
- **Functionality**: Implemented staff management and delivery tracking.
- **Enhancement**: Added notifications, data validation, and polish.
  The iterative approach included planning, building, testing, and refining.

Testing

Testing was performed on several levels:

- **Feature Testing**: Each major function, like staff check-in/out, was tested individually.
- **Cross-Browser Testing**: Verified performance on Chrome, Firefox, and Edge.
- **Responsive Testing**: Ensured the layout worked well on different screen sizes.

Results & Successes
The WeDeliverTech Reception Dashboard met all goals:

- **Easy to Use**: User-friendly interface for reception staff.
- **Smart Notifications**: Automated late staff/driver alerts.
- **Form Validation**: Clear feedback and error prevention.
- **Professional Look**: Clean, branded design.

Conclusion
The project provided valuable lessons in time manipulation, RegEx, and object-oriented design. Key takeaways:

- Stick to the project plan and focus on the sprint issues to stay on track.
- Define data structure early to avoid confusion later.
- Object-oriented design helped organize and maintain the code.
- Understanding JavaScript's time handling was essential to get this web app working correctly.