Payroll Management System

Streamlined payroll tracking with overtime, allowances, and exports

Yash Kumar Jain (2023UCP1571)

Dhruv Bansal (2023UCP1570)

Nakshatra Bansal (2023UCP1575)

December 4, 2025

Project Links & Group Details

Project Links & Group Details

Project Title: Payroll Management System

Group Number: **G-10**

Group Members:

1. Yash Kumar Jain — Reg. No. 2023UCP1571
2. Dhruv Bansal — Reg. No. 2023UCP1570
3. Nakshatra Bansal — Reg. No. 2023UCP1575

GitHub Repository Link:

GitHub Link

YouTube Demo Video:

Youtube Link

1 Abstract

The Payroll Management System is a lightweight web application built with Python and Flask to support small teams in managing hourly employees. It streamlines employee record keeping, tracks working hours, applies overtime rules, and incorporates allowances and deductions to compute gross and net pay. The system offers an at-a-glance payroll dashboard, CSV export for downstream processing, and robust local persistence via JSON files, making it well suited for academic and small-scale organizational use.

2 Project Introduction

Modern organizations require fast and reliable access to payroll information, even when operating without complex enterprise infrastructure. This project aims to provide a simple yet extensible payroll solution that can be deployed locally, used by a single HR operator, and adapted for future enhancements such as database backends or authentication. The application focuses on clarity of interface, correctness of calculations, and maintainable architecture so that students and practitioners can both operate and extend the system with ease.

3 SRS Introduction

Purpose. This Software Requirements Specification (SRS) defines the functional and non-functional requirements for the Payroll Management System so that stakeholders, developers, and reviewers share a common understanding of the release scope.
Scope. The SRS covers employee data management, hours and compensation updates, payroll calculation with overtime, reporting and CSV export, and local data persistence. It excludes tax engines, multi-user access control, and integration with external payment services.
Definitions. Employee ID (unique string), Overtime (hours beyond 40 weekly at 1.5× rate), Allowance (positive adjustment), Deduction (withholding), JSON (data serialization), CSV (comma-separated values export format).

4 Overall Description

4.1 Product Perspective

Standalone Python/Flask service; stores state in `payroll_data.json`; HTML interface rendered via Jinja templates; no external database or authentication.

4.2 User Characteristics

Small HR/admin teams with basic payroll knowledge, single concurrent operator, working on desktop or tablet browsers.

4.3 Assumptions and Constraints

- Trusted environment; no login or role management.
- File-system write access required for persistence/export.
- Local host deployment; internet connectivity optional (only for documentation).
- Target document length is 2–3 pages; functionality limited to hourly wages (no tax engine).

5 Functional Requirements

5.1 FR1: Employee Records

Description: Create, list, and delete employee profiles with name, ID, department, hourly rate, allowances, deductions, and hours worked.
Rationale: Core HR data storage.
Acceptance: Unique ID enforced; data persists after restart; list view displays all fields.

5.2 FR2: Compensation Updates

Description: Edit hours, allowances, or deductions simultaneously; blank inputs keep prior values.
Rationale: Speeds weekly adjustments.
Acceptance: Validation prevents negative values; success flash message confirms update.

5.3 FR3: Payroll Calculation

Description: Compute regular/overtime hours, base pay, overtime pay, gross pay, and net pay per employee and as cumulative totals.
Rationale: Provides financial visibility.
Acceptance: Dashboard shows employee count, total hours, gross/net sums, average rate; payroll table refreshes automatically.

5.4 FR4: Reporting and Export

Description: Render payroll tables on the dashboard and provide a CSV download with per-employee breakdown plus totals.
Rationale: Enables auditing and spreadsheet workflows.
Acceptance: CSV headers match on-screen metrics; download triggers from UI button.

5.5 FR5: Data Persistence

Description: Auto-save after any CRUD or compensation update and reload on startup. Handle corrupted files by falling back to an empty dataset without crashing.
Rationale: Prevents data loss.
Acceptance: JSON file reflects latest state; application remains operable after file manipulation errors.

6 Non-Functional Requirements

Usability. Single-page layout with validated forms, flash feedback, and responsive cards that remain legible on tablets.
Performance. CRUD and payroll calculations return within one second for up to 200 employees on commodity hardware.
Reliability. Validation ensures numeric inputs, overtime logic always runs, and CSV export degrades gracefully if no employees exist.
Maintainability. Separation between Flask routes, business logic module, and templates; minimal dependencies (Flask only).

Portability. Runs on Windows/macOS/Linux with Python 3.11+ and modern browsers (Chrome, Edge, Firefox, Safari).

7 Interface Requirements

User Interface. HTML5/CSS layout with summary cards, employee table, payroll table, and forms for add/update flows; download link for CSV export.
Software Interfaces. Flask routing, local JSON storage, CSV stream for download, browser-based HTTP client.
Hardware Interfaces. None beyond a device capable of running a browser.
Communication. HTTP over localhost (default port 5000); no external API traffic.

8 Traceability Matrix (Excerpt)

ID	Requirement	Implementation Hook
FR1	Manage employee profiles	Flask route `/add_employee`, persistence in `PayrollSystem.add_employee`
FR2	Update compensation inputs	Route `/update_compensation`, `PayrollSystem.update_compensation`
FR3	Compute payroll + summary	`PayrollSystem.calculate_payroll`, dashboard cards
FR4	Export CSV	Route `/export_payroll`, `PayrollSystem.export_payroll_csv`
FR5	Persist data	`load_data` and `save_data` methods

9 Risks and Mitigations

- **Corrupted JSON file** – mitigate with try/except and empty fallback.
- **Incorrect manual entries** – rely on on-field validation and flash guidance.
- **Future scaling needs** – design keeps logic modular so a database backend can replace the JSON store later.

10 Approval

Stakeholders sign off electronically by referencing this LaTeX-formatted SRS in the project tracking system.

11 UML Diagrams

This section presents key UML views of the Payroll Management System, including use-case, class, sequence, and component diagrams. They collectively describe how users interact with the system, how core entities are structured, how major workflows execute over time, and how the application is decomposed into deployable parts.

11.1 Class Diagram

The class diagram models the main domain classes, including `Employee`, `PayrollSystem`, and supporting utility classes responsible for persistence and reporting.

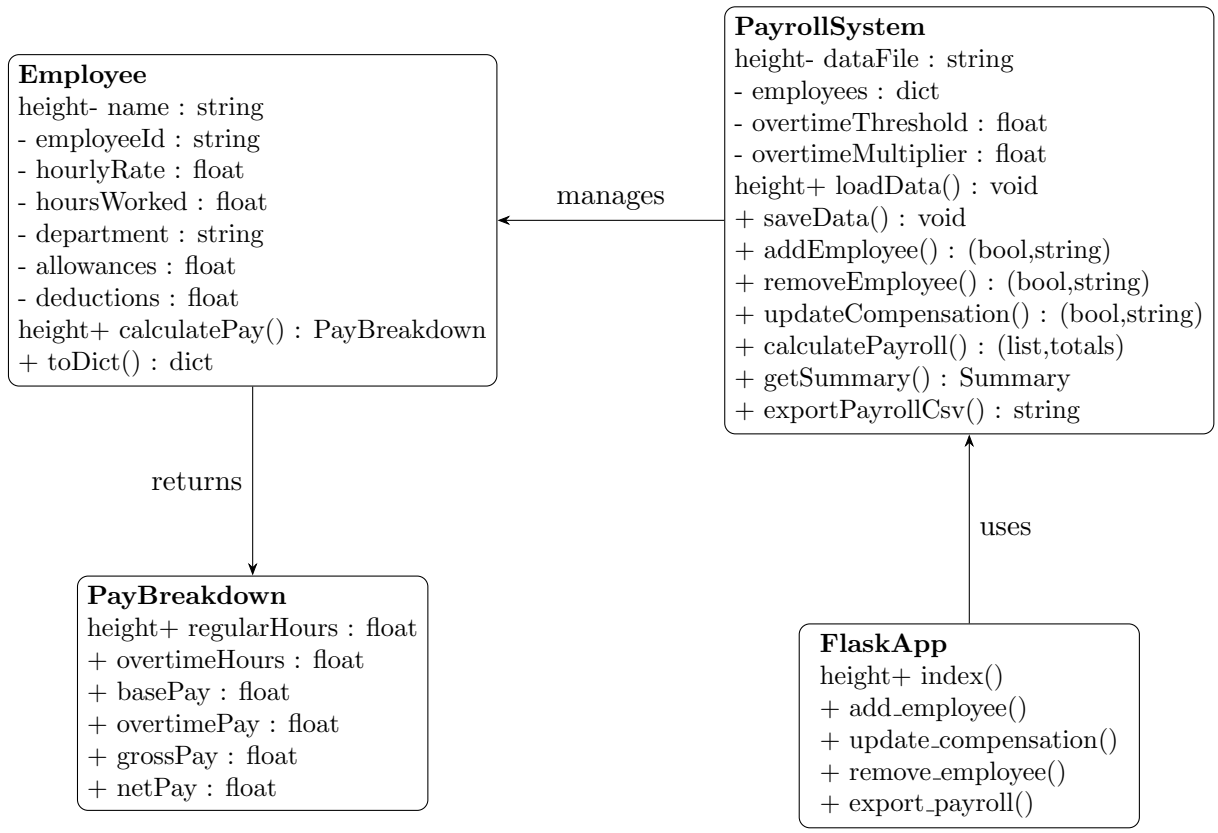


Figure 1: Class Diagram for Core Payroll Classes

11.2 Component Diagram

The component diagram summarizes the high-level architecture, separating the web UI, Flask application layer, business logic module, and local JSON/CSV storage components.

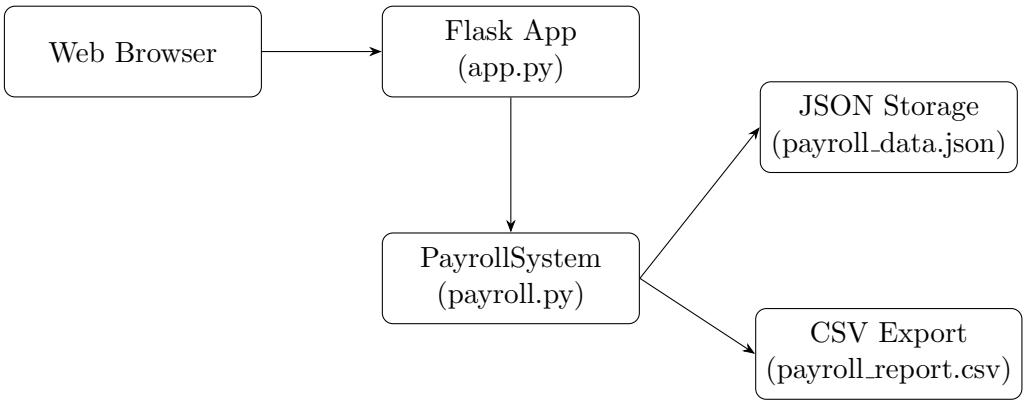


Figure 2: Component Diagram for Payroll Management System

11.3 Sequence Diagram

The sequence diagram illustrates the high-level flow when an HR user updates compensation for an employee and views the refreshed payroll dashboard.

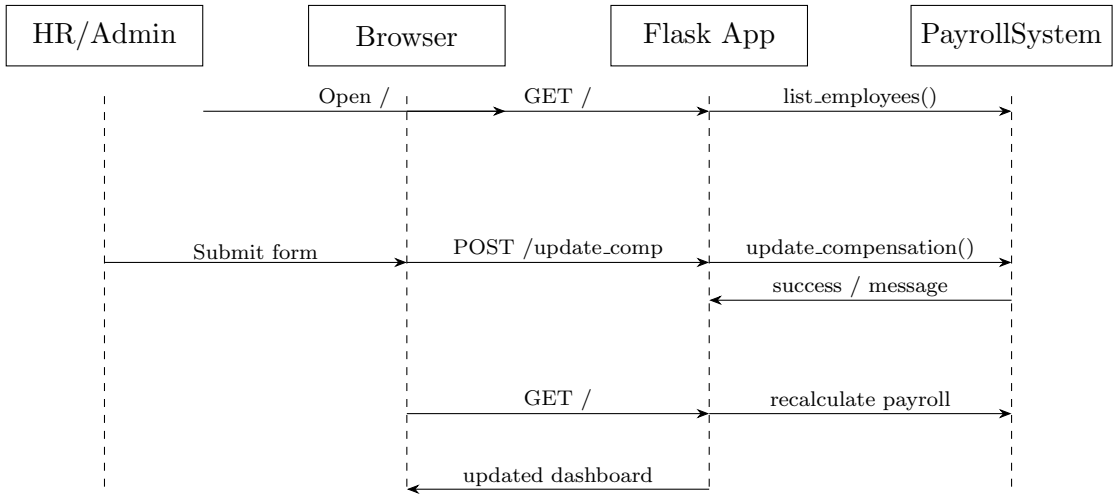


Figure 3: Sequence Diagram for Updating Compensation and Viewing Payroll

12 Implementation Overview

This section summarizes how the requirements of the Payroll Management System were realized in the implemented codebase.

12.1 Technology Stack

The application is implemented in Python using the Flask micro–web framework. Server-side logic resides primarily in the `payroll.py` module (business logic) and `app.py` (HTTP routes and request handling). The user interface is rendered using a single Jinja2 HTML template (`templates/index.html`) styled with responsive CSS, while data is persisted locally in a JSON file (`payroll_data.json`).

12.2 Key Modules and Responsibilities

PayrollSystem (payroll.py). Encapsulates all core domain logic: employee management, compensation updates, overtime-aware payroll calculations, summary statistics, and CSV export of payroll data. It is responsible for loading and saving the JSON data file, validating inputs (e.g., non-negative hours, allowances, and deductions), and returning structured breakdowns of gross and net pay.

Flask Application (app.py). Exposes HTTP endpoints for the main dashboard, adding and removing employees, updating compensation, and downloading the CSV report. Each route coordinates form input validation, delegates work to the `PayrollSystem` instance, and uses flash messages to provide immediate feedback to the user.

Presentation Layer (index.html). Implements a single-page layout with sections for employee management, compensation updates, payroll tables, and summary cards. Server-provided data (employees, payroll rows, and summary totals) are dynamically injected through Jinja2 template variables to keep the view logic minimal and maintainable.

12.3 Data Flow

When the application starts, `PayrollSystem.loadData()` reads existing records from `payroll_data.json`. User actions sent via forms (add/update/remove) are handled by Flask routes, which call the appropriate `PayrollSystem` methods; these methods mutate the in-memory employee dictionary and immediately persist changes back to JSON. For each page load, the dashboard route queries the current employees, detailed payroll, and summary statistics, which are then rendered into HTML for the browser.

12.4 Extensibility Considerations

The codebase separates routing, business logic, and presentation concerns to support future extension. For larger deployments, the JSON persistence layer could be replaced with a relational database without changing the Flask views, and additional features such as authentication, role-based access, or more sophisticated tax and deduction rules can be added by extending the `PayrollSystem` class and UI forms.

13 Testing & Test Cases

This section documents the key functional and negative test cases executed on the Payroll Management System, along with their expected and actual outcomes.

13.1 Functional Test Cases

ID	Description	Expected Result	Actual Result
FT1	Add a new employee with valid name, ID, department, hourly rate, allowances, and deductions.	Employee appears in the Employees table, data is written to `payroll_data.json`, and a success flash message is shown.	As expected. Employee persisted and visible on dashboard reload.
FT2	Update compensation (hours, allowances, deductions) for an existing employee via the Update Compensation form.	Employee row reflects new values; payroll and summary totals update accordingly without errors.	As expected. Payroll cards and totals recomputed correctly.
FT3	Remove an existing employee using the Remove button.	Employee disappears from the Employees and Payroll tables; totals decrease by that employee's pay.	As expected. JSON data file updated and UI refreshed.
FT4	Generate payroll with multiple employees, including some with overtime hours (> 40).	Regular and overtime hours computed correctly; gross and net totals match manual calculations.	As expected. Verified against hand-calculated examples.
FT5	Download the CSV payroll report from the dashboard.	Browser downloads a CSV file containing one row per employee plus aggregated totals; values match those shown on screen.	As expected. File opens correctly in spreadsheet software with consistent data.
FT6	Refresh the dashboard after restarting the Flask server.	Previously added employees and their hours/compensation values are reloaded from `payroll_data.json` without loss.	As expected. All records reappear and totals match the state before restart.
FT7	Leave allowances and deductions blank in the Update Compensation form while changing only hours.	Only the hours field is updated; existing allowances and deductions remain unchanged.	As expected. JSON data shows only hours modified and payroll totals update accordingly.

13.2 Negative Test Cases

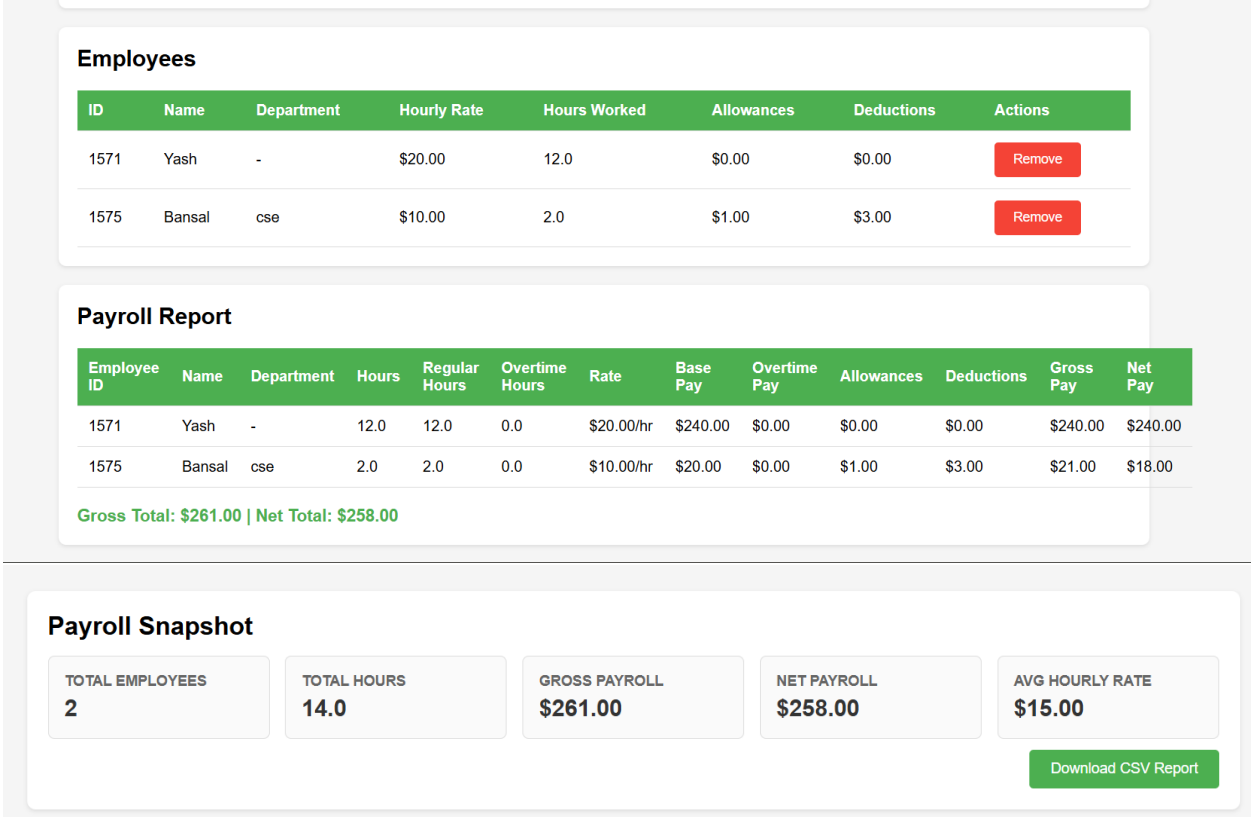
ID	Description	Expected Result	Actual Result
NT1	Attempt to add an employee with a duplicate Employee ID.	System rejects the request, shows an error flash message, and leaves existing data unchanged.	As expected. No duplicate created; original record preserved.
NT2	Enter negative hours or deductions/allowances in the Update Compensation form.	Validation fails; user sees a clear error message and values are not saved to the JSON file.	As expected. Input rejected and previous values remain intact.
NT3	Request removal of a nonexistent employee ID via the Remove button URL.	System shows an error flash message ("Employee not found") and makes no changes to stored data.	As expected. No records deleted and payroll totals remain unchanged.
NT4	Submit the Add Employee form with a nonnumeric hourly rate (e.g., letters).	Form handling catches the conversion error and displays a validation message without creating any employee record.	As expected. No new employee appears and JSON file is unchanged.

14 Results / Screenshots

This section presents visual evidence of the working Payroll Management System. Space is reserved below for key screenshots demonstrating the main flows.

Dashboard and Summary View

Overview of the main dashboard showing employee list, payroll table, and summary cards.



Employees

ID	Name	Department	Hourly Rate	Hours Worked	Allowances	Deductions	Actions
1571	Yash	-	$20.00	12.0	$0.00	$0.00	Remove
1575	Bansal	cse	$10.00	2.0	$1.00	$3.00	Remove

Payroll Report

Employee ID	Name	Department	Hours	Regular Hours	Overtime Hours	Rate	Base Pay	Overtime Pay	Allowances	Deductions	Gross Pay	Net Pay
1571	Yash	-	12.0	12.0	0.0	$20.00/hr	$240.00	$0.00	$0.00	$0.00	$240.00	$240.00
1575	Bansal	cse	2.0	2.0	0.0	$10.00/hr	$20.00	$0.00	$1.00	$3.00	$21.00	$18.00

Gross Total: $261.00 | Net Total: $258.00

Payroll Snapshot

TOTAL EMPLOYEES	TOTAL HOURS	GROSS PAYROLL	NET PAYROLL	AVG HOURLY RATE
2	14.0	$261.00	$258.00	$15.00

Download CSV Report

Figure 4: Dashboard and Summary View

Employee Management

Screenshots showing adding a new employee, updating compensation, and removing an employee.

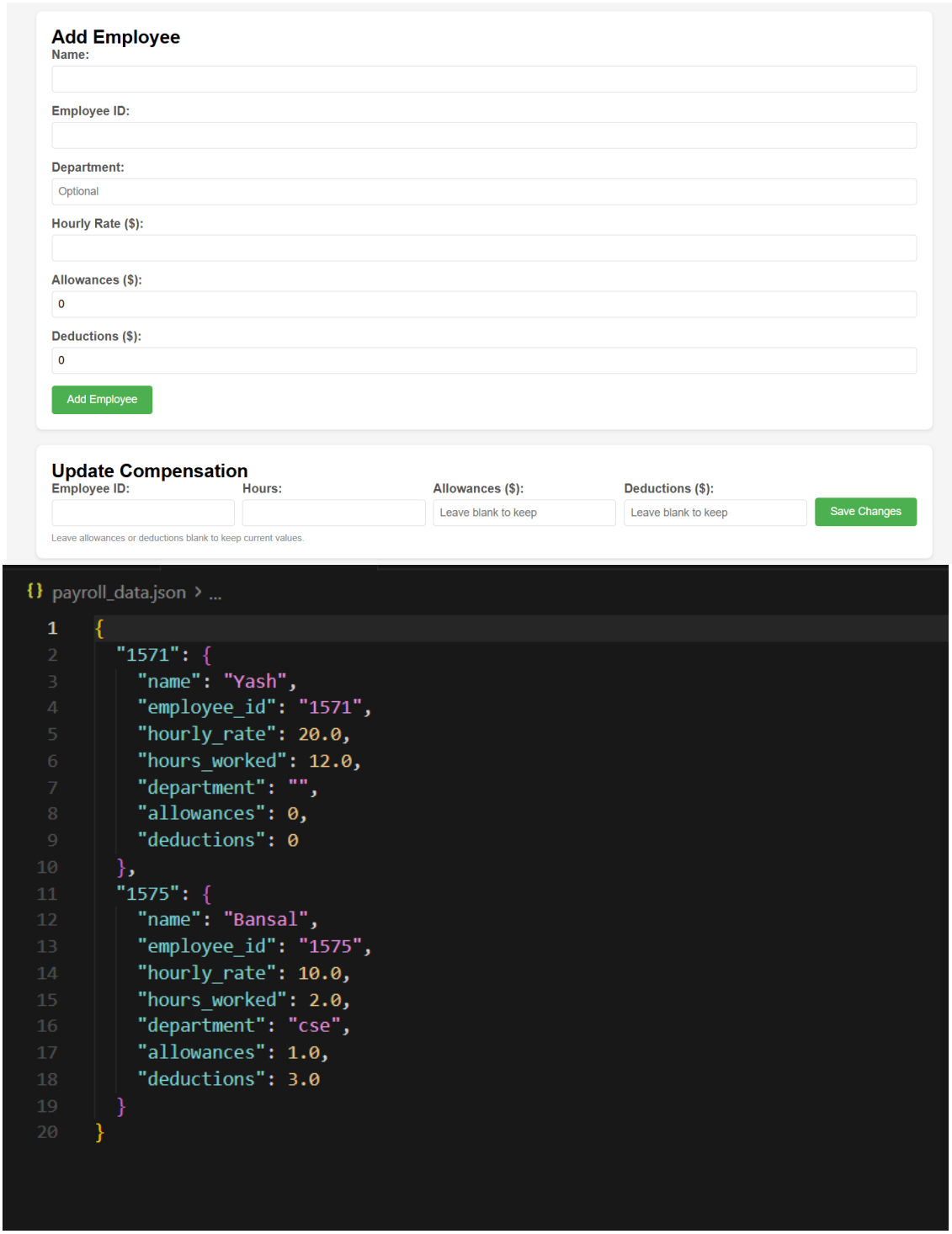
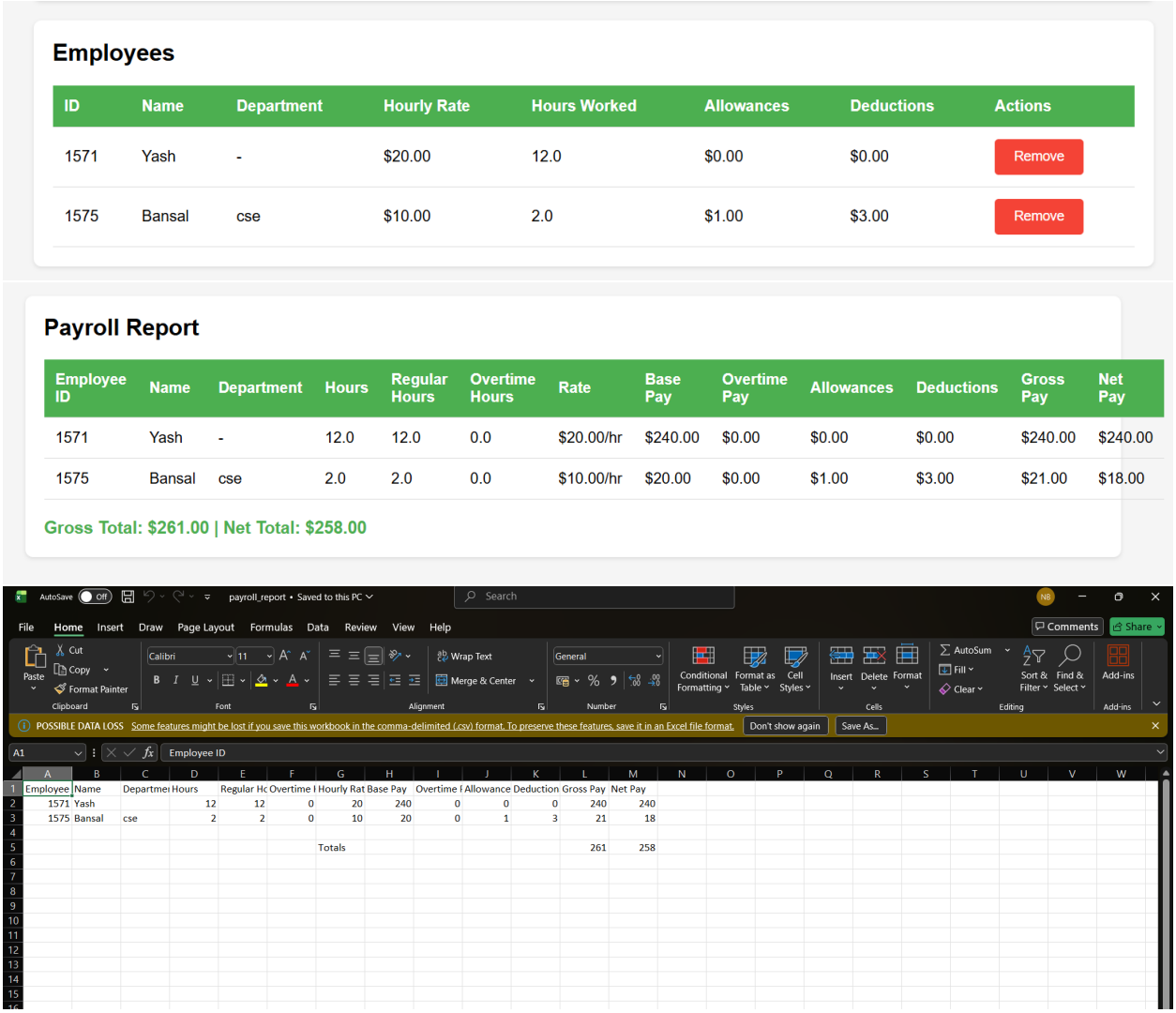


Figure 5: Employee Management Screens

Reporting and Export

Screenshots for the CSV export confirmation/download and example of the opened CSV in a spreadsheet.



Employees

ID	Name	Department	Hourly Rate	Hours Worked	Allowances	Deductions	Actions
1571	Yash	-	$20.00	12.0	$0.00	$0.00	Remove
1575	Bansal	cse	$10.00	2.0	$1.00	$3.00	Remove

Payroll Report

Employee ID	Name	Department	Hours	Regular Hours	Overtime Hours	Rate	Base Pay	Overtime Pay	Allowances	Deductions	Gross Pay	Net Pay
1571	Yash	-	12.0	12.0	0.0	$20.00/hr	$240.00	$0.00	$0.00	$0.00	$240.00	$240.00
1575	Bansal	cse	2.0	2.0	0.0	$10.00/hr	$20.00	$0.00	$1.00	$3.00	$21.00	$18.00

Gross Total: $261.00 | Net Total: $258.00

Figure 6: Reporting and CSV Export

15 Conclusion

The enhanced Payroll Management System successfully meets the functional and non-functional requirements outlined in the SRS. It provides a clean single-page interface for managing employees, tracking hours, handling overtime, and applying allowances and deductions while persisting data reliably in JSON. Additional features such as CSV export, dashboard summaries, and validation-driven error handling make the solution suitable for academic projects and small organizations needing lightweight payroll support. The modular structure of the codebase allows future teams to extend the system with stronger security, database integration, and more complex payroll rules.

16 References

- Python Software Foundation. *Python 3.x Documentation*. Available at: `https://docs.python.org/3/`
- Pallets Projects. *Flask Web Framework Documentation*. Available at: `https://flask.palletsprojects.com/`
- Overleaf. *LaTeX Documentation and Templates*. Available at: `https://www.overleaf.com/learn`
- W3C. *HTML5 and CSS Specifications*. Available at: `https://www.w3.org/`