**Software Design Document**

Team 15

BD25 City of Culture Java System

# INTRODUCTION

## Purpose

This software design document describes the architecture and system. The system is used to help the university of Bradford’s BD25 academic evolution information system. The system is used to support the university of Bradford BD25 programme by providing a structured and efficient platform for managing crowd engagement activities and machine learning model development for data analysis. The primary objective of the document is to outline the technical and functional specifications.

## Scope

The main scope of the project is to provide a system to BD25. The system facilitates the responsible enrolments of members in data and information groups and their participation in BD25 activities. The system is also responsible for data management, the data handles storage organization, and management of data records (image, video etc) related to BD25 activities, ensuring proper structuring and accessibility. The system uses demonstration and validation, it must support case studies demonstrating AI model training and validation, with evidence of data storage, labelling and processing. The advantages allow for a structured and responsible enrolments in BD25. The data management is efficient as it organizes in a structured manner. It allows machine learning workflow to be more enhanced as it streamlines data collection, labelling, and pre-processing, ensuring data quality.

## Overview

The overview describes a system designed to facilitate responsible enrolment, participation, and machine learning model deployment for BD25 activities. It outlines key assumptions, data structures, user roles and system functionalities. DIG’s must have at least 3 members. Each member is assigned names, roles, and unique ID’s. They engage in tasks such as data collection, labelling, preprocessing, model creation and testing.

# SYSTEM OVERVIEW

The BD25 system is a data management application designed to support archaeological activity tracking and research. It enables users—primarily researchers, dig members, and compliance officers—to manage excavation activities, associated personnel, data records, and analytical models. The system ensures efficient organization, accessibility, and regulatory compliance of excavation data.

At its core, BD2025 consists of modular components that interact with a centralized database. These components handle tasks such as registering archaeological digs, adding or removing team members, uploading analysis models, recording findings, and verifying compliance with legal or procedural standards. The software is built using Java and follows a clean separation of concerns through the Data Access Object (DAO) pattern, promoting clarity and maintainability.

Designed for usability and scalability, the system supports multi-user access and integrates smoothly with existing research workflows, offering a user-friendly interface and a backend capable of processing large datasets relevant to archaeological work.

By structuring the code in this manner, the application benefits from a clear separation between business logic and data access logic. Each DAO class focuses on a specific aspect of data management, making the system more organized and maintainable. This approach also allows for easier testing and potential modifications to the data access methods without affecting the core business logic

# SYSTEM ARCHITECTURE

## Architectural Design

Our BD25 Java program is designed to adopt a modular architecture which will facilitate the import, processing, and output of data in a GUI format for two types of users: admins and customers. The system is divided into three distinct layers: Presentation Layer, Business Logic Layer, and Data Layer. Each layer is responsible for specific functionalities and collaborates with the others to achieve the overall system requirements.

1. Presentation Layer:
   * Acts as the user interface of the system, providing access to functionalities based on user roles.
   * Displays information, collects user inputs, and provides feedback.
   * Interacts with the Business Logic Layer to process user commands and display results.
2. Business Logic Layer:
   * Contains the core functionality, processing inputs, managing workflows, and enforcing rules.
   * Implements rules, coordinates between layers, and ensures compliance with guidelines.
   * Interacts with both the Presentation and Data Layers to process data and execute workflows.
3. Data Layer:
   * Manages data storage, retrieval, and integrity.
   * Stores and retrieves data, ensures security, and manages backups.
   * Provides data to the Business Logic Layer and receives processed data for storage.

## Decomposition Description

The system can be decomposed into functional modules within each layer to enhance maintainability and scalability:

1. Presentation Layer:

* User Interface (UI): Interface with dashboards, forms, and reports.
* Role-Based Access Control: Manages access based on user roles.

1. Business Logic Layer:

* Activity Management Module: Manages activities and assignments.
* Data Processing Module: Handles data collection, labelling, and machine learning processes.
* Engagement Analysis Module: Analyses engagement using machine learning models.

1. Data Layer:

* Database Management System (DBMS): Stores structured data.
* File Storage System: Manages unstructured data.
* Data Provenance and Labelling System: Tracks data origin and processing history.

A diagram of a diagram

AI-generated content may be incorrect.

## Design Rationale

Our three-tier architecture was selected to ensure modularity, scalability, and ease of maintenance and operation. This separation into three layers enables clear allocation of responsibilities and independent development and testing of each layer. Critical issues considered include:

* Scalability: The modular design supports future expansion and integration of additional functionalities.
* Security: Role-based access control and data security measures are integral to the design.
* Compliance: Ensures adherence to LSEPI guidelines for data processing and management.

Alternative Architectures Considered:

* Monolithic Architecture: Rejected due to potential difficulties in scaling and maintaining the system.
* Microservices Architecture: Considered but deemed unnecessary for the current scope and complexity of the system.

# DATA DESIGN

## Data Description

The BD25 Academic Evaluation System manages data related to activities, crowd engagement, and AI models in a structured database. Media files like audio, video, and images are kept in a file storage system with metadata for tracking and AI training. External data from web scraping is also stored and tagged.

The system follows a structured process: Activity Coordinates (ACs) create and update activities, DIG Leaders assign members, and teams capture data using various tools. This data is labelled, and used to train AI models, which analyse crowd engagement trends. Key databases include the User Database (staff and DIG members), Activity Database (events and schedules), Media Storage (digital records), and AI Model database. This integrated system ensures efficient data management and AI-driven crowd engagement analysis.

## Data Dictionary

|  |  |  |
| --- | --- | --- |
| Entity | Type | Description |
| activityId | Integer | Unique identifier for each activity |
| name | String | The title or name of the activity |
| description | String | A brief overview or summary of what the activity involves |
| startDate | String | The starting date of the activity (e.g., '15th Jan 2025') |
| endDate | String | The ending date of the activity |
| curator | String | Name(s) of the person or group organizing/leading the activity |
| locations | List<String> | A list of locations or venues where the activity takes place |
| audience | String | Intended or targeted audience (e.g., families, adults, all ages, etc.) |
| price | String | Ticket or entry price (may include ranges or discount info) |
| modelId | Integer | Unique identifier for each trained model |
| modelAccuracy | Double | Accuracy score of the trained model |
| trainingDataPath | String | File path to the training dataset |
| evaluationResult | Object | Holds model evaluation results (e.g., accuracy, loss) |
| modelStatus | String | Current state of the model (e.g., 'Training', 'Evaluated') |
| errorLog | String | Description of any errors encountered during processing |
| dataCaptureStatus | Boolean | Indicates if data capture is active (true/false) |
| startTime | DateTime | Timestamp when the data capture starts |
| endTime | DateTime | Timestamp when the data capture ends |
| dataFilePathField | String | Input field for file path from which data is captured |
| logTextArea | JTextArea | Displays logs of data capture activities |
| browseButton | JButton | Button to browse and select data file |
| FILE\_PATH | String | Path to the text file storing DIG members and their activities |
| digMembersTextArea | JTextArea | Text area used to display all DIG members |
| digMemberNameField | JTextField | Input field for the DIG member's name |
| activityNameField | JTextField | Input field for the associated activity name |
| addButton | JButton | Button to add a new member to the file |
| removeButton | JButton | Button to remove a member from the file |
| displayButton | JButton | Button to display all members from the file |
| members | List<String> | Used to temporarily hold member entries when reading or filtering the file |
| contentPane | JPanel | Container for switching between different UI panels |
| activityUI | ActivityUI | Panel for managing activities |
| digUI | DIGUI | Panel for managing DIG members |
| dataCaptureUI | DataCaptureUI | Panel for capturing data |
| modelTrainingUI | ModelTrainingUI | Panel for training models |
| sidebar | JPanel | Vertical sidebar panel with navigation buttons |
| startModelTrainingButton | JButton | Button to initiate the model training process |
| evaluateModelButton | JButton | Button to evaluate the trained model |
| saveModelButton | JButton | Button to save the trained model to file |
| loadModelButton | JButton | Button to load a previously saved model |
| trainingLogArea | JTextArea | Text area to display logs during training, evaluation, saving, and loading |
| progressBar | JProgressBar | Visual indicator of model training progress |
| LOG\_FILE\_PATH | String | File path where training logs and events are saved |
| URL | String | JDBC connection string to the MySQL database |
| USER | String | Username for database access |
| PASSWORD | String | Password for database access |
| fetchAllActivities() | Method | Retrieves all activity records from the database |
| insertActivity(Activity) | Method | Inserts a new activity into the database |
| removeActivity(String activityId) | Method | Deletes an activity by ID |

# COMPONENT DESIGN

This section breaks down the BD25 information system’s components and its functionality, based on the decomposition description explained in Section 3.2. For each component or class, pseudocode is used to represent the logic of the core functions, and where necessary, local data is also identified.

**Presentation layer:**

Class: DataCaptureUI

Function: startDataCapture ()

Pseudocode:

startDataCapture(activityName, dataFilePath):

if activityName is empty or dataFilePath is empty:

show error message

return

open dataFileReader for dataFilePath

open logFileWriter for logging

log "Data capture started"

while line = dataFileReader.readLine() is not null:

processedData = processData(line)

log "Processed: " + processedData

log "Data capture completed"

close dataFileReader and logFileWriter

Local data:

BufferedReader

dataFileReader

BufferedWriter,

logFileWriter

JTextField,

ActivityNameField

dataFilePathField,

JTextArea,

logTextArea

Class: ModelTrainingUI

Function:

trainModelUI()

Pseudocode:

trainModelUI():

select activity

click "Train"

call ModelManager.trainModel(activityId)

show confirmation

Function: addDIGMemberUI (DIGUI)

Pseudocode:

addDIGMemberUI():

enter member details

call DIGManager.addDIGMember(activityId, member)

update member list

**Business Logic Layer:**

Class: ActivityManager

Functions:

getAllActivities()

createActivity()

deleteActivity()

Pseudocode:

getAllActivities():

return ActivityDAO.fetchAllActivities()

createActivity(activity):

validate activity details

ActivityDAO.insertActivity(activity)

deleteActivity(activityId):

ActivityDAO.removeActivity(activityId)

Class: DIGManager

Functions:

getAllDIGMembers(activityId):

addDIGMember()

removeDIGMember()

getAllDIGMembers():

Pseudocode:

getAllDIGMembers(activityId):

return DIGMemberDAO.fetchAllMembers(activityId)

addDIGMember(activityId, member):

DIGMemberDAO.insertMember(activityId, member)

removeDIGMember(activityId, memberId):

DIGMemberDAO.deleteMember(activityId, memberId)

Class: DataCaptureManager:

Functions:

initiateDataCapture():

terminateDataCapture():

Pseudocode:

initiateDataCapture(activityId):

filePath = get file path from activity

open file

while not end of file:

read and process line

store results in log or database

terminateDataCapture(activityId):

close file and log writers

log "Data capture stopped"

Class: ModelManager:

Functions:

trainModel(activityId):

evaluateModel(activityId):

Pseudocode

trainModel(activityId):

data = DataRecordDAO.fetchDataRecords(activityId)

model = ML.train(data)

ModelDAO.saveModel(model)

evaluateModel(activityId):

model = ModelDAO.fetchModel(activityId)

result = model.evaluate()

return result

Class: LSEPIManager:

Function:

checkLSEPICompliance(activityId):

Pseudocode:

checkLSEPICompliance(activityId):

data = DataRecordDAO.fetchDataRecords(activityId)

rules = loadLSEPIStandards()

for each rule in rules:

if data violates rule:

return false

return true

**Data Layer:**

Class: ActivityDAO:

Functions:

fetchAllActivities()

removeActivity():

Pseudocode:

fetchAllActivities():

return SELECT \* FROM activities

insertActivity(activity):

INSERT INTO activities VALUES (...)

removeActivity(activityId):

DELETE FROM activities WHERE id = activityId

Class: DIGMemeberDAO:

Functions:

fetchAllMembers():

deleteMember():

Pseudocode:

fetchAllMembers(activityId):

return SELECT \* FROM dig\_members WHERE activity\_id = activityId

insertMember(activityId, member):

INSERT INTO dig\_members VALUES (..)

deleteMember(activityId, memberId):

DELETE FROM dig\_members WHERE id = memberId

Class :DataRecordDAO:

Functions

fetchDataRecords()

saveDataRecord()

Pseudocode:

saveDataRecord(record):

INSERT INTO data\_records VALUES (record)

fetchDataRecords(activityId):

return SELECT \* FROM data\_records WHERE activity\_id = activityId

Class: ModelDAO:

Functions:

saveModel(model):

fetchModel(activityId):

Pseudocode:

saveModel(model):

serialise model and save to database

fetchModel(activityId):

return SELECT \* FROM models WHERE activity\_id = activityId

Class :LSEPIDAO

Function:

is Compliant():

Pseudocode:

isCompliant(activityId):

records = fetchDataRecords(activityId)

return checkLSEPICompliance(records)

Conclusion:

The component design shows how each part of the system contributes to its functionality. By organising logic across layers and behaviour in class methods, the system adheres to modular and scalable design principles. The use of pseudocode allows for clarity on internal operations, helping in breaking down complex systems into smaller understandable ones. This can be beneficial in maintaining the BD25 information system and even supports in the reusability of certain components, while helping in scalability. As the system grows, it can allow easier integration of new features and functionalities without overburdening existing infrastructure.

# HUMAN INTERFACE DESIGN

Human interactivity overview for BD25 Management System

**6.1 System Functionality from User Perspective**

The BD25 Management System provides a simple graphical interface for the management of three key areas: DIG members, data capture processes. It presents a consistent, professional interface with clear visual feedback for all actions.

Navigation and Layout

1. Main Application Structure:

* Left navigation bar with two buttons (DIG Members, and Data Capture)
* Dynamic main content area that changes with selection
* Static header indicating the current section

2. Visual Design:

* Dark sidebar with light-coloured buttons
* Light content area with clearly defined sections
* Professional-looking colour scheme (blues and greys) throughout
* Consistent button styles and fonts

1. DIG Members Management

A screenshot of a computer

AI-generated content may be incorrect.

User Workflow:

1. Click "DIG Members" from sidebar

2. Show/Add/Delete members using properly labelled controls:

* Enter member name and activity in text boxes
* Click buttons to perform action:
* Display: Lists all current members and their assigned activities
* Add: Inserts new member record
* Remove: Deletes selected member

User Feedback:

* Success/error messages in dialog boxes
* Member list updated automatically after current member
* Input fields cleared after operations
* Visual confirmation of operations

2. Data Capture Module

A screenshot of a computer

AI-generated content may be incorrect.

User Workflow:

1. Select "Data Capture" from sidebar

2. Configure capture settings:

* Enter activity name
* Specify data file path (with browse button)

3. Control capture process:

* Start: Starts processing data
* Stop: Halts current operation

User Feedback:

* Real-time output to scrollable text area
* Progress bar for file processing
* Friendly status messages in dialog windows
* Button states change during processes (disabled/enabled)

1. Visual Feedback

* Progress bars for time-consuming operations
* Color-coded buttons (active/inactive states)
* Section borders and headings for orientation

2. Textual Feedback:

* Real-time logging in specialized text areas
* History of operations with time stamps
* Pop-up alerts for critical events

3. Interactive Feedback:

* Button state changes during operations
* Input validation before processing
* Responsive UI while background operations are executed

Expected User Experience

1. For DIG Management:

* View all members and their activities in a glance
* Easily update member records
* Certain confirmation of changes effected

2. For Data Operations:

* View file processing in real time
* Know precisely what the system is doing
* Able to stop operations if needed

The interface is designed to require a minimum amount of training, with similar functions grouped logically and uniform interaction patterns across all modules. Feedback mechanisms allow users to have a sense of what the system is doing and what their actions are doing to the system at all times.

6.3 Screen Objects and Actions

Main Application Window

1. Sidebar Navigation Panel

A screenshot of a computer

AI-generated content may be incorrect.

Objects:

* DIG Members button
* Data Capture button

Actions:

* Clicking any feature button changes the main display to respective module
* Visual feedback: Selected button is slightly highlighted

2. Header Area



Objects:

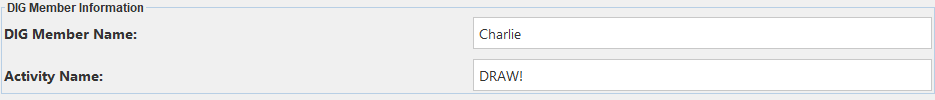
* Application title label
* Current module indicator

Actions:

* Dynamically updates to show current module
* Provides constant orientation within application

DIG Members Module

1. Input Panel



Objects:

* DIG Member Name" text field
* Activity Name" text field

Actions:

* Text input for new member data
* Validating inputs (no-empty fields)
* Automatic clear upon successful operation

2. Action Buttons

A blue rectangle with white text

AI-generated content may be incorrect.

Objects:

* Show DIG Members button
* Add DIG Member button
* Remove DIG Member button

Actions:

* Display: Displays full list of members in text area
* Add: Adds new record with current field values
* Remove: Wipes out member equal to name field
* Visual feedback: Off during processing

3. Member Display Area

A white rectangular frame with blue border

AI-generated content may be incorrect.

Objects:

* Scrollable text area

Actions:

* Outputs all members in human-readable form
* Auto-scrolls to bottom on update
* Read-only content for data protection

Data Capture Module

1. Configuration Panel



Objects:

* Activity Name text field
* Data File Path text field
* Browse button

Actions:

* Text input for capture parameters
* File browser dialog for choosing path
* Input validation before processing

2. Control Buttons

A blue squares with white text

AI-generated content may be incorrect.

Objects:

* Start Data Capture button
* Stop Data Capture button

Actions:

* Start: Initiates file processing thread
* Stop: Halts current operation
* State changes: Start disabled during capturing

3. Log Panel

A screenshot of a computer

AI-generated content may be incorrect.

Objects:

* Timestamped log entries
* Vertical scroll bar

Actions:

* Real-time appending of processing messages
* Manual scrolling of history
* Persistent in session

Common Interactive Elements

A screenshot of a computer error

AI-generated content may be incorrect.1. Dialog Boxes

A screenshot of a computer error

AI-generated content may be incorrect.

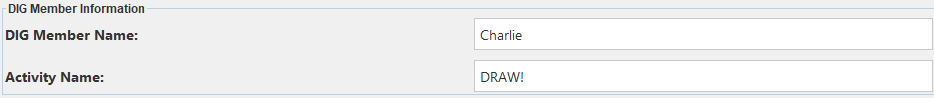
Objects:

* Confirmation dialogs
* Error messages
* Success notifications

Actions:

* Modal interruption for critical feedback
* OK/Cancel buttons as needed
* Detailed error descriptions

2. Text Input Fields



Objects:

* White background text fields
* Gray border highlights

Actions:

* Normal text entry
* Input validation feedback
* Context-sensitive enablement

3. Status Indicators

A screenshot of a computer

AI-generated content may be incorrect.

Objects:

* Button state changes

Actions:

* Visual system state communication
* Instant operation feedback
* Warning

Interaction Patterns

Form Submission:

* Field completion → Button click → Processing → Feedback
* Validation precedes processing
* Clean visual workflow

Long Operations:

* User interface is always responsive
* Indicators for progress display action
* A completion notification gets shown

The interface has consistent interaction paradigms in all the modules, with adequate visual feedback for every action of the user. Objects have consistent patterns of behaviour to reduce cognitive load, and all the actions provide clear and timely feedback for their outcome or status.

# REQUIREMENTS MATRIX

|  |  |  |
| --- | --- | --- |
| **Requirement ID** | **Requirement description** | **Priority** |
| R1 | Dynamically load activities from the database. | High |
| R2 | Add new activities onto the system. | High |
| R3 | Edit activities onto the system. | High |
| R4 | Delete activities that are currently on the system. | High |
| R5 | Load DIG members for an activity from the txt file. | High |
| R6 | Add DIG members to an activity. | High |
| R7 | Remove DIG members from an activity. | High |
| R8 | View a log for each data search. | Low |
| R9 | Search data capture for a specific activity. | High |