UNIVERZA V MARIBORU FAKULTETA ZA ELEKTROTEHNIKO, RAČUNALNIŠTVO IN INFORMATIKO

**CYCLearn - A Platform for Analyzing and Optimizing Cycling Training**

Project documentation

Maribor, 2023/2024

Vanja Pajović, Bogdan Kašćelan

INDEX

[1 INTRODUCTION 2](#_Toc170500089)

[1.1 Technologies used 3](#_Toc170500090)

[1.2 Organization and methodology 4](#_Toc170500091)

[2 ARCHITECTURE AND USED TECHNOLOGIES 7](#_Toc170500092)

[2.1 PostgreSQL database design and structure: 7](#_Toc170500093)

[2.2 React frontend structure and technologies 15](#_Toc170500094)

[2.3 Python backend structure and technologies 21](#_Toc170500095)

[2.4 Machine learning and data processing 25](#_Toc170500096)

[3 UML DOCUMENTATION AND PLANNING 29](#_Toc170500097)

[3.1 UseCase diagram and user stories 30](#_Toc170500098)

[3.2 Activity diagrams 35](#_Toc170500099)

[4 DEPLOYMENT GUIDE 48](#_Toc170500100)

[5 QUALITY ASSURANCE 51](#_Toc170500101)

[6 APPLICATION TESTING 55](#_Toc170500102)

[6.1 Unit tests 55](#_Toc170500103)

[6.2 End-to-End tests 60](#_Toc170500104)

PHOTO INDEX

**Photo 1.1:** Example of the Kanban board................................................. 7

Photo 2.1: Entities in the database and their relationship........................ 8

Photo 2.2: SQL script................................................................................. 12

Photo 2.3: SQL script................................................................................. 13

Photo 2.4: SQL script................................................................................. 13

Photo 2.5: SQL script................................................................................. 14

Photo 2.6: SQL script................................................................................. 14

Photo 2.7: SQL script................................................................................. 15

Photo 3.1: Use case diagram..................................................................... 30

Photo 3.2: Coach overview of cyclists' sessions activity diagram.............. 36

Photo 3.3: Create training plan and template activity diagram................. 38

Photo 3.4: Forgot password process activity diagram............................... 40

Photo 3.5: Updating profile information activity diagram......................... 42

Photo 3.6: Import training session activity diagram.................................. 44

Photo 3.7: Login process activity diagram................................................. 45

Photo 3.8: Registration process activity diagram...................................... 46

Photo 5.1: Before quality assurance.......................................................... 54

Photo 5.2: After quality assurance............................................................ 54

Photo 6.1: Unit tests.................................................................................. 59

Photo 6.2: Athlete profile test................................................................... 61

Photo 6.3: Creating plan test..................................................................... 63

Photo 6.4: Login test.................................................................................. 65

Photo 6.5: Password recovery test............................................................ 66

Photo 6.6: Profile test................................................................................ 68

Photo 6.7: Registration test....................................................................... 69

1 INTRODUCTION

**Project overview**

The CYCLearn project aims to develop an easy to use platform designed to analyze and visualize cyclists' training sessions data using data automation and analytics. This project targets enhancing training efficiency through predictive analysis and providing valuable insights into athlete performance. By integrating machine learning techniques with a user-friendly platform, CYCLearn aims to to improve performance and health monitoring. This platfort leverages different technologies to offer a comprehensive solution for coaches and cyclists to monitor, evaluate, and improve their training regimens.

1.1 Technologies used

In the CYCLearn project, the following technologies are employed:

* **Database**: PostgreSQL is used for robust and scalable data storage and management.
* **Frontend**: React is utilized for building an interactive and dynamic user interface.
* **Backend**: Python, along with Flask, is employed for server-side logic and API development.
* **Machine learning**: NiaARM is used for numerical association rule mining to discover patterns in training data and evaluate session performance.

**Project goals**

The primary goal of CYCLearn is to provide an automated and analytical approach to training session management for cyclists. The platform aims to:

1. **Visualize training metrics:** Provide detailed visualizations of various training metrics such as heart rate, altitude, distance, and speed to give users a comprehensive view of their performance.
2. **Analyze performance**: Use advanced machine learning algorithms to discover patterns and insights from the data.
3. **Predictive analytics**: Implement predictive analysis to foresee potential health risks and performance issues.
4. **Enhance training efficiency**: Offer actionable insights and recommendations to improve training effectiveness.
5. **Offer training session creation**: Enable coaches to create and manage training sessions tailored to athletes' needs and performance goals.
6. **User-Friendly interface**: Provide an intuitive and easy-to-use interface for both coaches and cyclists.

1.2 Organization and methodology

The CYCLearn project is organized according to Kanban methodology to ensure agile and efficient development. This approach helps the development team manage workflow, track progress, and maintain flexibility to adapt to changing requirements. Below is an expanded description of how Kanban methodology is applied in CYCLearn:

**Kanban methodology**

**Kanban board**: The Kanban board is the central tool for managing tasks and visualizing the workflow. It is divided into several columns, each representing a stage in the development process. The columns include:

* **To Do**: Tasks that need to be addressed but haven't been started yet.
* **In Progress**: Tasks that are currently being worked on.
* **Done**: Tasks that have been completed.
* **Reviewed**: Tasks that have been completed and reviewed for quality assurance.

**JIRA for project management:** JIRA is used to manage project tasks, plan Kanban workflows, solve bugs, and track overall progress. JIRA's features include:

* **Backlog management:** Creating and updating the product backlog based on user stories and use case diagrams.
* **Kanban planning:** Organizing tasks into the Kanban board and assigning them to team members.
* **Bug tracking:** Logging bugs, assigning them to developers, and tracking their resolution.
* **Progress tracking:** Monitoring the status of tasks and the overall project to ensure timely completion.

Each task or bug is represented as a card on the Kanban board, which can move through these columns as it progresses through the stages of development. The board provides a visual representation of the project's status at any given time.

**Task and bug tracking:** Tasks and bugs are tracked on the Kanban board to ensure that nothing is overlooked. Each card on the board contains detailed information about the task or bug, including:

* **Task description**: A brief overview of what needs to be done.
* **Assignees**: Team members responsible for completing the task.
* **Priority**: The importance and urgency of the task.
* **Status**: Current status (e.g., To Do, In Progress, Done).

**Product backlog:** The product backlog is a prioritized list of features, enhancements, and bug fixes that need to be addressed. It is continuously updated based on the project's needs and user feedback. The backlog is the source of tasks that are moved into the Kanban board's "To Do" column.

**Example of the Kanban board:** The image below shows an example of the Kanban board used in the CYCLearn project: A screenshot of a computer

Description automatically generated

**Photo 1.1:** Example of the Kanban board

In this example, tasks are clearly categorized into different stages, with specific tasks like "Finish machine learning frontend," "Style Health Monitoring component," and "Fix CSS issue in niaarm route" being tracked from inception to completion.

By utilizing the Kanban methodology, CYCLearn ensures that the development process remains organized, transparent, and adaptable, ultimately leading to a more efficient and effective project outcome.

2 ARCHITECTURE AND USED TECHNOLOGIES

2.1 PostgreSQL database design and structure:

The database for our project is designed to efficiently store and manage data related to cyclists, coaches, training sessions, and training plans. This chapter provides an overview of the database schema, including the structure of each table, relationships between tables, and the corresponding SQL script used to create the database.

**Entity-Relationship diagram (ERD)**

A screenshot of a computer

Description automatically generated

**Photo 2.1:** Entities in the database and their relationship.

**Database tables and structure**

1. **coaches**
   * **Description**: stores information about the coaches using the application.
   * **Fields**:
     + coachID (serial, primary key)
     + username (character varying(50), unique, not null)
     + password (character varying(255), not null)
     + email (character varying(100), unique, not null)
     + profile\_picture (character varying(255))
2. **cyclists**
   * **Description**: stores information about the cyclists being trained.
   * **Fields**:
     + cyclistID (serial, primary key)
     + coachID (integer, foreign key referencing coaches.coachID, not null)
     + username (character varying(50), unique, not null)
     + password (character varying(255), not null)
     + email (character varying(100), unique, not null)
     + date\_of\_birth (date)
     + height\_cm (integer)
     + weight\_kg (integer)
     + profile\_picture (character varying(255))
3. **training\_sessions**
   * **Description**: stores data from cyclists' training sessions.
   * **Fields**:
     + sessionsID (serial, primary key)
     + cyclistID (integer, foreign key referencing cyclists.cyclistID, not null)
     + altitude\_avg (numeric)
     + altitude\_max (numeric)
     + altitude\_min (numeric)
     + altitudes (text)
     + ascent (numeric)
     + calories (numeric)
     + descent (numeric)
     + distance (numeric)
     + distances (text)
     + duration (interval)
     + heartrates (text)
     + hr\_avg (integer)
     + hr\_max (integer)
     + hr\_min (integer)
     + positions (text)
     + speeds (text)
     + start\_time (timestamp without time zone)
     + steps (integer)
     + timestamps (text)
     + total\_distance (numeric)
4. **training\_plans**
   * **Description**: stores information about training plans created by coaches.
   * **Fields**:
     + plansID (serial, primary key)
     + coachID (integer, foreign key referencing coaches.coachID, not null)
     + start\_date (timestamp without time zone, not null)
     + description (text)
     + executed (character varying(3), default 'No')
5. **training\_plan\_templates**
   * **Description**: stores templates for training plans.
   * **Fields**:
     + sessionID (serial, primary key)
     + planID (integer, foreign key referencing training\_plans.plansID)
     + type (character varying(50), not null)
     + duration (interval, not null)
     + distance (numeric, not null)
     + intensity (character varying(50))
     + notes (text)
6. **cyclist\_training\_plans**
   * **Description**: stores the association between cyclists and their training plans.
   * **Fields**:
     + cyclistID (integer, foreign key referencing cyclists.cyclistID, not null)
     + plansID (integer, foreign key referencing training\_plans.plansID, not null)

**SQL script for database creation**

The following SQL script creates the database tables and establishes the necessary relationships:

A screenshot of a computer program

Description automatically generated

**Photo 2.2:** SQL script

A screenshot of a computer code

Description automatically generated

**Photo 2.3:** SQL script

A computer code with text

Description automatically generated

**Photo 2.4:** SQL script

A screenshot of a computer program

Description automatically generated

**Photo 2.5:** SQL script

A screenshot of a computer program

Description automatically generated

**Photo 2.6:** SQL script

A computer code with text

Description automatically generated

**Photo 2.7:** SQL script

The database design is structured to effectively manage data related to cyclists, coaches, training sessions, and training plans. The use of relational tables and foreign key constraints ensures data integrity and facilitates efficient querying and reporting. This well-organized schema supports the application's functionality, providing a robust foundation for data storage and retrieval.

2.2 React frontend structure and technologies

The frontend implementation of the Cyclist Health Monitoring App provides a comprehensive and user-friendly interface for cyclists and coaches to manage training plans, monitor health, and interact with the system. The application is built using React, ensuring a modular and maintainable codebase with reusable components.

Key functionalities such as user authentication, dashboard management, health monitoring, charting, calendar integration, map display, weather data display, and data import/export are implemented efficiently. The use of React Router ensures smooth navigation, while Axios facilitates communication with the backend.

The project leverages several powerful technologies and libraries to deliver a robust and efficient frontend system:

* **React** for building the user interface.
* **React hooks** for state management.
* **React router** for navigation.
* **Axios** for HTTP requests.
* **Chart.js** for displaying charts.
* **react-calendar** for calendar functionalities.
* **OpenStreetMap** for map integration.
* **OpenWeatherMap API** for fetching and displaying weather data.
* **CSV parsing libraries** for import/export functionalities.
* **CSS** for styling.

**Components overview**

**src/App.js**

The main entry point for the React application. It sets up the router and renders the main components.

**Technologies used:**

* **React**
* **React Router**
* **CSS**

**Authentication (src/Components/Authentication)**

Handles user authentication including login, registration, password recovery, and profile management.

* **Login.js**: Manages user login.
* **Register.js**: Manages user registration.
* **PasswordRecovery.js**: Manages password recovery.
* **EditProfile.js**: Allows users to edit their profile information.
* **UserProfile.js**: Displays user profile information.
* **ResetPassword.js**: Allows users to reset their password.

**Technologies used:**

* **React**
* **Axios** for HTTP requests.
* **CSS** for styling.

**Dashboard (src/Components/Dashboard)**

Handles the main functionality of the application including displaying training plans, health monitoring, and session management.

* **CyclistTrainingPlans.js**: Displays training plans for cyclists.
* **Dashboard.js**: The main dashboard component.
* **HealthMonitoring.js**: Manages health monitoring functionalities.
* **HistoryCalendar.js**: Displays the training history in a calendar view.
* **Sidebar.js**: The sidebar navigation component.
* **StartTrainingSession.js**: Manages the start of a training session.

**Technologies used:**

* **React**
* **Axios** for HTTP requests.
* **Chart.js** for displaying charts.
* **react-calendar** for calendar functionalities.
* **OpenWeatherMap API** for fetching weather data.
* **OpenStreetMap** for displaying maps.
* **CSS** for styling.

**Coach Dashboard (src/Components/Dashboard/Coach)**

Specific functionalities for coaches to manage athletes.

* **AthleteOverview.js**: Provides an overview of athletes under a coach.
* **AthleteProfile.js**: Displays detailed profile information for an athlete.
* **CoachDashboard.js**: The main dashboard for coaches.
* **CreateTrainingPlan.js**: Allows coaches to create training plans.
* **ImportPreview.js**: Handles the preview of imported data.

**Technologies used:**

* **React**
* **Axios** for HTTP requests.
* **Chart.js** for displaying charts.
* **CSV Parsing Libraries** for handling CSV data.
* **CSS** for styling.

**Home Page (src/Components/HomePage.js)**

Displays the landing page for the application.

**Technologies used:**

* **React**
* **CSS** for styling.

**Navbar (src/Components/Navbar)**

Manages the top navigation bar for the application.

* **Navbar.js**: The main navigation bar component.

**Technologies used:**

* **React**
* **React Router**
* **CSS** for styling.

**Import and Export (src/Components/Dashboard/Coach/ImportPreview.js)**

Handles the import and preview of data related to training plans and sessions.

**Technologies used:**

* **React**
* **Axios** for HTTP requests.
* **CSV Parsing Libraries** for handling CSV data.
* **CSS** for styling.

**Styles (src/Styles)**

Contains CSS files for styling the various components in the application.

* **AthleteOverview.css**
* **AthleteProfile.css**
* **Auth.css**
* **CreateTrainingPlan.css**
* **CyclistTrainingPlans.css**
* **Dashboard.css**
* **EditProfile.css**
* **HomePage.css**
* **ImportPreview.css**
* **Navbar.css**
* **StartTrainingSession.css**
* **UserProfile.css**

2.3 Python backend structure and technologies

This backend implementation efficiently handles user authentication, data processing, and interaction with the database. The modular structure ensures that each component is responsible for a specific functionality, enhancing maintainability and scalability. The endpoints for retrieving session data and checking for health risks are crucial for providing real-time health monitoring for cyclists.

The project leverages several powerful technologies and libraries to deliver a robust and efficient backend system:

* Flask for the core web framework, providing a solid foundation for building the application.
* SQLAlchemy for database interactions, ensuring efficient data management.
* Pandas for data processing, offering powerful tools for data manipulation and analysis.
* Flask-JWT-Extended for secure user authentication, ensuring protected access to the application.
* Requests for handling external HTTP requests, facilitating data integration from various sources.
* Python-dotenv for managing environment variables, keeping configuration settings secure and organized.
* NiaARM, Numpy, and Niapy for machine learning and data processing, enabling advanced data analysis and pattern discovery.

Components overview

**1. run.py**

The main entry point for running the application. It initializes the Flask app and sets up the necessary configurations.

**Technologies used:**

* **Flask:** Provides the core web application framework.

**2. app/\_\_init\_\_.py**

Initializes the Flask application and sets up configurations, database connections, and blueprint registrations.

**Technologies used:**

* **Flask:** Used to create and configure the Flask application.
* **Flask SQLAlchemy:** Manages the application's SQL database interactions, making it easier to handle database operations.
* **Flask Migrate:** Provides SQLAlchemy database migrations for Flask applications using Alembic.

**3. Authentication (app/auth)**

* **\_\_init\_\_.py:** Initializes the authentication module.
* **views.py:** Contains routes and logic for user authentication, including registration, login, and token management.

**Technologies used:**

* **Flask-JWT-Extended:** Manages JWT tokens for secure authentication, providing a secure way to protect routes and manage user sessions.
* **Werkzeug security:** Used for password hashing and verification, ensuring user credentials are stored securely.

**4. Dashboard (app/dashboard)**

Handles the core functionalities for both coaches and cyclists.

* **coach.py:** Manages coach-related endpoints, allowing coaches to view and manage athletes and their training plans.
* **cyclist.py:** Manages cyclist-related endpoints, including session data retrieval and health monitoring.
* **data\_fetching.py:** Contains functions for fetching data from external sources or databases, facilitating data integration from various APIs.
* **import\_export.py:** Manages data import and export functionalities, allowing easy data handling and transfer.

**Technologies used:**

* **Flask:** For creating and managing API endpoints.
* **SQLAlchemy:** For database interactions, ensuring efficient data querying and manipulation.
* **Requests:** For making HTTP requests to external APIs, allowing integration with other services and data sources.

**5. Data processing (app/data\_processing)**

Contains functions for processing the training session data, such as calculating statistics or generating features.

**Technologies used:**

* **Pandas:** For data manipulation and analysis, providing powerful data structures and functions to handle structured data seamlessly.

**6. Models (app/models)**

Defines various data models for interacting with the database, such as user data, training sessions, and training plans.

**Technologies used:**

* **SQLAlchemy:** An ORM (Object-Relational Mapping) library that provides a full suite of enterprise-level persistence patterns.

**7. Static Files**

Static files include profile pictures stored in the app/static/photos/profilePictures directory. These files are used for user profile management and display within the application.

.

2.4 Machine learning and data processing

This section outlines the machine learning and data processing aspects of the Cyclist Health Monitoring App. It covers the use of NiaARM for discovering patterns in training data, the integration of the React frontend with the Flask backend, and the overall architecture enabling real-time health monitoring and analysis.

**Technologies and libraries used**

**Machine learning and data processing:**

* **NiaARM**: A framework for numerical association rule mining, which is used to discover patterns in the training session data. It helps identify relationships and patterns in numerical data.
* **Numpy**: A fundamental package for scientific computing with Python. It provides support for arrays, mathematical functions, and more, making it a key library for numerical operations.
* **Niapy**: A Python microframework for building and evaluating optimization algorithms, used here for implementing differential evolution. It provides various optimization algorithms that are useful for solving complex optimization problems.
* **JSON**: For storing the generated rules in a structured format.

**Frontend technologies:**

* **React**: For building the user interface and managing component states.
* **Axios**: For making HTTP requests to the backend endpoints.

**Backend technologies:**

* **Flask**: For creating and managing API endpoints.
* **SQLAlchemy**: For database interactions and managing data persistence.
* **Pandas**: For data manipulation and analysis.
* **Requests**: For making HTTP requests to external APIs.
* **Python-dotenv**: For managing environment variables.

**Key functionalities**

**Running NiaARM**

The NiaARM framework is used to discover patterns in cyclist training data, which helps in identifying potential health risks based on heart rate data.

**Endpoint:** /run\_niaarm

**Process:**

* Reads data from a CSV file containing training session details.
* Uses differential evolution to discover rules and patterns in the data.
* Saves the discovered rules to a JSON file for later use.

**Checking session data**

This functionality compares the latest training session data against the rules generated by NiaARM to identify potential health risks.

**Endpoint:** /check\_session

**Process:**

* Reads the session data.
* Compares the session data with the established patterns and rules.
* Generates warnings if certain conditions are met (e.g., heart rate exceeds predefined thresholds).

**Implementation details**

**Frontend (React) implementation**

The frontend of the application, built with React, manages cyclist health warnings through a component called HealthMonitoring. This component tracks rules, warnings, errors, and allows users to trigger NiaARM to discover new patterns or check the last session for health risks.

**Key features:**

* **Running NiaARM**: Users can trigger the process to discover new patterns from past training sessions.
* **Checking session**: Users can check the latest session for potential health risks.

**Technologies used:**

* React: For building the user interface and managing component states.
* Axios: For making HTTP requests to the backend endpoints.

**Backend (Flask) implementation**

The backend, built with Flask, provides endpoints for running NiaARM and checking a cyclist's session. These endpoints are defined in the Flask Blueprint cyclist\_bp.

**Running NiaARM**

**Process:**

* Reads data from a CSV file.
* Uses differential evolution to find rules.
* Saves the rules to a JSON file.

**Checking session**

**Process:**

* Reads the cyclist's session data.
* Checks the session data against the generated rules.
* Generates warnings if conditions are met (e.g., heart rate exceeds certain values).

The machine learning and data processing components of our project enable health monitoring and analysis through the efficient use of rule mining and session data evaluation. By leveraging NiaARM, the application can discover significant patterns in training data and identify potential health risks, providing valuable insights to cyclists and coaches.

3 UML DOCUMENTATION AND PLANNING

Below are the detailed use case diagrams, user stories, and intended users for the CYCLearn project. These elements form the foundation of our implementation strategy, ensuring that the platform meets the specific needs of its users through a user-centered design approach. Each user story outlines the goals and requirements from the perspective of different user roles, which helps guide the development process to ensure that the final product is both functional and user-friendly.

3.1 UseCase diagram and user stories

A diagram of a diagram

Description automatically generated

**Photo 3.1:** Use case diagram

**Intended users**

The project is designed for the following users:

* **Coaches**: To monitor and optimize athletes' training, analyze performance, and adjust training programs based on collected data.
* **Cyclists**: To track their performance, progress, and health status, and adjust their training and goals accordingly.
* **Sports organizations**: To manage teams, plan events, analyze performance, and develop athlete strategies.

**User stories and acceptance criteria**

**Creating training plans for cyclists**

* **Title**: creating training plans for cyclists
* **As**: Coach
* **I want to**: Create training plans for my athletes.
* **So that**: I can optimize their performance based on current status and needs.

**Acceptance criteria**:

1. The system analyzes heart rate, speed, and power in real-time.
2. Training plans are automatically adjusted based on analyzed data.
3. Coaches receive notifications about adjustments made.

**Scenario**:

1. The coach selects a date and description for the training.
2. The coach prepares the training plan and assigns it to the athlete.
3. The coach creates the training session.

**Performance Analysis**

* **Title**: Performance analysis
* **As**: Coach/Cyclist
* **I want to**: Track and compare performance data from my training sessions and visualize progress and trends.
* **So that**: I can make informed decisions about my training.

**Acceptance criteria**:

1. The system collects and displays performance data.
2. Progress and performance trends are visualized with interactive charts.
3. Coaches can compare data among different athletes.

**Scenario**:

1. The user uses the dashboard.
2. The system displays performance data such as speed, heart rate, and distance covered.
3. The user reviews interactive charts showing progress and trends.

**Dynamic map visualization**

* **Title**: Dynamic map visualization
* **As**: Cyclist
* **I want to**: See my training routes displayed on an interactive map.
* **So that**: I can visualize my route and path.

**Acceptance criteria**:

1. The system displays routes using GPS data.

**Scenario**:

1. The athlete starts a training session with a GPS device.
2. The system collects GPS data and displays the route on an interactive map.

**Predictive analytics for injury prevention**

* **Title**: Predictive analytics for injury prevention
* **As**: Cyclist
* **I want to**: Use predictive analytics to identify patterns that could lead to injuries.
* **So that**: I can adjust training plans and prevent potential injuries.

**Acceptance criteria**:

1. The system analyzes historical training data to identify patterns.
2. Predictive models alert about potential injury risks.

**Scenario**:

1. The system analyzes past training sessions and detects patterns indicating overtraining.
2. The predictive model identifies risks for injuries.

**Advanced analytical dashboards**

* **Title**: Advanced analytical dashboards for coaches
* **As**: Coach/Cyclist
* **I want to**: Have analytical dashboards providing insights into key performance indicators and trends.
* **So that**: I can develop effective training strategies.

**Acceptance criteria**:

1. Dashboards display key performance indicators and trends.
2. Data is presented in an easy-to-understand format.
3. Data is visually represented.

**Scenario**:

1. The user logs into the platform.
2. The user navigates to the analytical dashboard.
3. The system displays data in interactive charts and diagrams.
4. The user uses the dashboard to make informed decisions about training strategies.

**Data import and export**

* **Title**: Data import and export
* **As**: Coach/Cyclist
* **I want to**: Import and export my training data.
* **So that**: I can analyze the data externally or share it with other platforms.

**Acceptance criteria**:

1. Users can import data from various formats.
2. Users can export data in various formats.

**Scenario**:

1. The user logs into the platform.
2. The user navigates to the data import/export section.
3. The user uploads or downloads training data in the desired format.

3.2 Activity diagrams

These diagrams illustrate the various processes and interactions between the user, the application, and the backend system. . Each diagram helps to visualize the flow of actions and the steps involved in various functionalities of the application.

**1.** **Coach overview of cyclists' sessions**A diagram of a cycle

Description automatically generated

**Photo 3.2:** Coach overview of cyclists' sessions activity diagram

**Description:** This diagram illustrates the process by which a coach can log in, navigate to the cyclist overview, and view a cyclist's training session details.

* **Login**: The coach logs into the application.
* **Navigate to cyclist overview**: The coach navigates to the cyclist overview page.
* **Retrieve list of cyclists**: The application retrieves and displays a list of cyclists from the backend system.
* **Select a cyclist**: The coach selects a cyclist from the list.
* **Select a date**: The coach selects a date to view the cyclist's training session details.
* **Retrieve cyclist's training session details**: The application fetches the selected cyclist's training session details from the backend system.
* **Display cyclist's training session details**: The application displays the details of the selected training session.

**2. Create training plan and template**

A diagram with text and symbols

Description automatically generated with medium confidence

**Photo 3.3:** Create training plan and template activity diagram

**Description:** This diagram outlines the process for creating a new training plan or template by a coach.

* **Login**: The coach logs into the application.
* **Navigate to training plan creation**: The coach navigates to the training plan creation page.
* **Use existing training plan template?**: The coach decides whether to use an existing training plan template.
* **Create new training plan template**: If not using an existing template, the coach inputs data for a new template and submits it.
* **Input data for creating a training plan**: The coach inputs data for creating a training plan, selects a cyclist, and selects a training template.
* **Submit the training plan**: The coach submits the training plan.
* **Check data validity**: The backend system checks the validity of the data.
* **Create training plan**: If the data is valid, the training plan is created and an email is sent to the selected cyclists.

**3. Forgot password process**

A diagram of a flowchart

Description automatically generated

**Photo 3.4:** Forgot password process activity diagram

**Description:** This diagram shows the steps a user follows to reset their password if they have forgotten it.

* **Navigate to the login page**: The user navigates to the login page.
* **Select forgot password**: The user selects the forgot password option.
* **Enter email adress**: The user enters their email address.
* **Request password reset**: The user requests a password reset.
* **Verify email format**: The application verifies if the input data is in email format.
* **Send password reset email**: The backend system sends an email with a link to reset the password.
* **Click on the link in the email**: The user clicks on the link in the email.
* **Enter a new password**: The user enters a new password.
* **Update password**: The backend system updates the password in the database.

**4. Updating profile information**

A diagram of a software company

Description automatically generated with medium confidence

**Photo 3.5:** Updating profile information activity diagram

**Description:** This diagram shows the process for updating a user's profile information.

* **Login**: The user logs into the application.
* **Navigate to profile**: The user navigates to their profile page.
* **Click on edit profile**: The user clicks on the edit profile option.
* **Update profile information**: The user updates their profile information.
* **Upload profile picture**: The user uploads or updates their profile picture (optional).
* **Submit updated profile**: The user submits the updated profile information.
* **Verify data**: The application verifies the updated data.
* **Check data validity**: The backend system checks if the updated data is valid.
* **Update profile**: If the data is valid, the profile information is updated in the database.
* **Show error message**: If the data is invalid, an error message is shown.

**5. Import training session**

A diagram of a process

Description automatically generated

**Photo 3.6:** Import training session activity diagram

**Description:** This diagram details the process of importing a training session from a JSON file.

* **Login**: The user logs into the application.
* **Navigate to import session**: The user navigates to the import session page.
* **Upload JSON file**: The user uploads a JSON file containing the training session data.
* **Validate file**: The application validates the uploaded file.
* **Process the uploaded file**: If the file is valid, the backend system processes the uploaded file and shows the training session details.

**6. Login process**

A diagram of a software application

Description automatically generated with medium confidence

**Photo 3.7:** Login process activity diagram

**Description:** This diagram outlines the login process for a user.

* **Enter credentials**: The user enters their login credentials.
* **Click on login button**: The user clicks the login button.
* **Verify credentials**: The application verifies the credentials.
* **Check credentials in database**: The backend system checks the credentials in the database.
* **Redirect to dashboard**: If the credentials are valid, the user is redirected to the dashboard.
* **Show error message**: If the credentials are invalid, an error message is shown.

**7. Registration process**

A diagram of a flowchart

Description automatically generated

**Photo 3.8:** Registration process activity diagram

**Description:** This diagram illustrates the registration process for a new user.

* **Select role**: The user selects their role (Coach or Cyclist).
* **Fill in registration form**: The user fills in the registration form.
* **Check data validity**: The application checks if the input data is valid.
* **Create user account**: If the data is valid, the backend system creates a new user account.
* **Show error message**: If the data is invalid, an error message is shown.

4 DEPLOYMENT GUIDE

This section provides a step-by-step guide for setting up and deploying the CYCLearn application, including the backend and frontend components, as well as the necessary prerequisites and steps for populating the training sessions table.

**Backend Setup**

**Prerequisites:** Python, PostgreSQL, and Poetry.

**Steps:**

1. **Clone the repository:** Clone the repository using the following commands:

**git clone https://github.com/yourusername/AST-Monitor-web.git**

**cd AST-Monitor-web**

1. **Create a poetry environment and install dependencies:** Create a Poetry environment and install the necessary dependencies using the command:

**poetry install**

1. **Create a PostgreSQL database:** Create a new PostgreSQL database. Navigate to the database/creatingDBscript.sql file and run it in the query to set up the database schema.
2. **Create an environment file:** In the root of the project, create a .env file with the following content:

MAIL\_USERNAME=cyclearninfo@gmail.com

MAIL\_PASSWORD=udnc oadv dxsh pwtv SQLALCHEMY\_DATABASE\_URI=postgres://YourUserName:YourPassword@YourHostname:YourPort/YourDatabaseName

TEST\_DATABASE\_URL=postgres://YourUserName:YourPassword@YourHostname:YourPort/YourDatabaseName

Insert valid URIs for SQLALCHEMY\_DATABASE\_URI and TEST\_DATABASE\_URL.

The TEST\_DATABASE\_URL should contain the same database as the SQLALCHEMY\_DATABASE\_URI, and it is meant for testing purposes.

1. **Starting the backend server:** Navigate to the ast\_monitor\_web/run.py and run it using: python ast\_monitor\_web/run.py

**Frontend Setup**

**Prerequisites:** Node.js and npm.

**Steps:**

1. **Navigate to the frontend directory:** Navigate to the frontend directory using: cd frontend
2. **Install dependencies:** Install the necessary dependencies with: npm install
3. **Start the application:** Start the application with: npm start

**Populating the training sessions table**

To populate the training\_sessions table, follow these steps:

1. **Create a coach and a cyclist:** Use the application to create a coach and a cyclist.
2. **Extract zip files:** Extract zip files from scripts and place them so the path is scripts/Sport5Rider3.json.
3. **Run the population script:** Navigate to the scripts/populateSessions.py file. At the bottom of the script, modify the insert\_data(data\_list, cyclist\_id=1) line to change the cyclist's ID as needed. Run the script with: python scripts/populateSessions.py

**Machine learning data**

To run the health monitoring part:

1. **Extract zip files:** Extract zip files from ast\_monitor\_web/csv/treci.zip and place them so the path is ast\_monitor\_web/csv/treci.csv.

**Technologies used**

**Frontend:** React, Axios.

**Backend:** Python Flask, PostgreSQL, SQLAlchemy, Flask-Mail.

**Other:** AST-Monitor Integration, GPS Data Visualization.

5 QUALITY ASSURANCE

Using the Pylint library, various code improvements and refactorings were implemented to enhance code readability, maintainability, and functionality. The following sections provide a detailed account of the changes made to the project files, including modifications to cyclist.py, import\_export.py, and the addition of a new utility module utils.py.

**Changes made**

1. **Refactoring cyclist.py and import\_export.py**
   * **Modularization:** Extracted common functionalities into a separate utility module (utils.py) to adhere to the DRY (Don't Repeat Yourself) principle.
   * **Improved exception handling**: Enhanced error handling by catching specific exceptions and providing meaningful error messages.
   * **Documentation:** Added missing docstrings to modules, classes, and functions to improve code documentation and understanding.
2. **New utility module: utils.py**
   * Created a new module named utils.py to encapsulate utility functions that are used across multiple modules.
   * **Functions added:**
     + get\_weather\_data: Fetches weather data for a given location and date.
     + compute\_hill\_data: Computes hill data for a given training session.
3. **Updated cyclist.py**
   * **Imports**: Changed import statements to include functions from the new utils.py module.
   * **Functionality:**
     + Updated the get\_session\_details function to use compute\_hill\_data and get\_weather\_data from utils.py.
   * **Improved code quality:** Reformatted long lines, added missing docstrings for modules, classes, and functions, and improved variable naming.
4. **Updated import\_export.py**
   * **Imports:** Changed import statements to include functions from the new utils.py module.
   * **Functionality:**
     + Updated the export\_session\_report and export\_session\_json functions to use compute\_hill\_data and get\_weather\_data from utils.py.
   * **Improved code quality:** Reformatted long lines, added missing docstrings for modules, classes, and functions, and improved variable naming.
5. **Enhanced documentation and readability**
   * Added module-level and function-level docstrings to explain the purpose and usage of various components.
   * Reformatted long lines to adhere to the PEP 8 style guide.
   * Ensured consistent error messages and logging throughout the modules.
6. **Added missing documentation for modules, classes, and functions**
   * Ensured every file has a module docstring explaining its purpose.
   * Added class docstrings to describe the purpose and functionality of each class.
   * Added function docstrings to explain the functionality, parameters, and return values of each function.

**Summary of utility functions in utils.py**

* get\_weather\_data:
  + Fetches weather data for a specific location and date using the Weather API.
  + Parameters include latitude, longitude, and the starting time of the session.
  + Returns a dictionary containing weather information such as temperature, condition, wind speed, and humidity.
* compute\_hill\_data:
  + Computes various hill-related metrics for a given session.
  + Parameters include a TrainingSession object containing session data.
  + Returns a dictionary containing hill data metrics such as the number of hills, average altitude, average ascent, distance covered by hills, and the share of hills in the session.

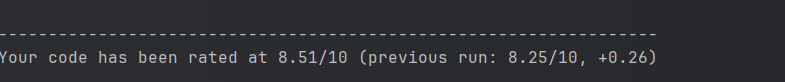
These changes collectively enhance the maintainability and readability of the codebase, making it easier for developers to understand and extend the functionality in the future.

**Before:**   
  
A black background with text

Description automatically generated

**Photo 5.1:** Before quality assurance

**After:**



**Photo 5.2:** After quality assurance

6 APPLICATION TESTING

6.1 Unit tests

This part of the document provides detailed information about the unit tests implemented for various features of the backend application using the Python library unittest. Each test case includes a description of the actions performed and the expected outcomes. A screenshot of successful test execution is provided for reference.

**TestAuth**

**Test file: test\_auth.py**

**Description**

This test suite verifies the authentication functionality of the application. It includes tests for registering a coach and a cyclist, logging in as a coach and a cyclist, and accessing the profile of a logged-in user.

**Test steps**

1. **test\_register\_coach**
   * Registers a new coach with a unique username and email.
   * Verifies that the response status code is 201 and the success message is received.
2. **test\_register\_cyclist**
   * Registers a new cyclist with a unique username, email, and associated coach ID.
   * Verifies that the response status code is 201 and the success message is received.
3. **test\_login\_coach**
   * Logs in using the registered coach's credentials.
   * Verifies that the response status code is 200, an access token is received, and the role is 'coach'.
4. **test\_login\_cyclist**
   * Logs in using the registered cyclist's credentials.
   * Verifies that the response status code is 200, an access token is received, and the role is 'cyclist'.
5. **test\_profile\_access**
   * Logs in as a coach to obtain an access token.
   * Uses the token to access the profile endpoint.
   * Verifies that the response status code is 200 and the profile data is correct.

**TestCoachRoutes**

**Test file: test\_coach\_routes.py**

**Description**

This test suite verifies the routes accessible by a coach. It includes tests for retrieving athletes, accessing an athlete's profile, getting sessions for a calendar, creating a training plan, retrieving training plan templates, creating a training plan template, and deleting a training plan template.

**Test steps**

1. **test\_get\_athletes**
   * Retrieves the list of athletes associated with the coach.
   * Verifies that the response status code is 200 and the list is not empty.
2. **test\_get\_athlete\_profile**
   * Retrieves the profile of a specific athlete.
   * Verifies that the response status code is 200 and the correct athlete profile is returned.
3. **test\_get\_sessions\_for\_calendar**
   * Retrieves the sessions for a specific athlete to display on a calendar.
   * Verifies that the response status code is 200 and the session list is not empty.
4. **test\_create\_training\_plan**
   * Creates a new training plan and associates it with a previously created training plan template.
   * Verifies that the response status code is 201 and the success message is received.
5. **test\_get\_training\_plan\_templates**
   * Retrieves the list of training plan templates.
   * Verifies that the response status code is 200.
6. **test\_create\_training\_plan\_template**
   * Creates a new training plan template.
   * Verifies that the response status code is 201 and the template data is correct.
7. **test\_delete\_training\_plan\_template**
   * Deletes a specific training plan template.
   * Verifies that the response status code is 200 and the success message is received.

**TestCyclistRoutes**

**Test file: test\_cyclist\_routes.py**

**Description**

This test suite verifies the routes accessible by a cyclist. It includes tests for running NiaARM analysis, retrieving cyclist sessions, accessing session details, retrieving saved rules, checking a session against rules, retrieving training plans, and executing a training plan.

**Test steps**

1. **test\_run\_niaarm**
   * Runs NiaARM analysis for the cyclist.
   * Verifies that the response status code is 200 and rules are returned.
2. **test\_get\_cyclist\_sessions**
   * Retrieves the sessions associated with the cyclist.
   * Verifies that the response status code is 200 and the session list is not empty.
3. **test\_get\_session\_details**
   * Retrieves details of a specific training session.
   * Verifies that the response status code is 200 and session details are correct.
4. **test\_get\_saved\_rules**
   * Retrieves the saved rules for the cyclist.
   * Verifies that the response status code is 200 and rules are returned.
5. **test\_check\_session**
   * Checks a session against the saved rules.
   * Verifies that the response status code is 200 and warnings are returned.
6. **test\_get\_cyclist\_training\_plans**
   * Retrieves the training plans associated with the cyclist.
   * Verifies that the response status code is 200.
7. **test\_execute\_training\_plan**
   * Executes a specific training plan.
   * Verifies that the response status code is 200 and the success message is received or the appropriate error message is shown.

A black background with red and blue lines

Description automatically generated

**Photo 6.1:** Unit tests

6.2 End-to-End tests

This part of the document provides detailed information about the end-to-end tests implemented for various features of the application using Cypress. Each test case includes a description of the actions performed and the expected outcomes. Screenshots of successful test executions are provided for reference.

**AthleteProfile tests**

**Test file: athleteprofile.cy.js**

**Description**

This test suite verifies the functionality of the athlete profile page, ensuring that all data is loaded and displayed correctly for both coaches and cyclists. The test includes logging in as both roles, navigating to the athlete profile, selecting a specific date on the calendar, and verifying the session details, weather data, map, and charts.

**Test steps**

1. **Login as coach**
   * Visit login page
   * Enter coach credentials and submit
   * Verify successful login and token storage
2. **Navigate to athlete profile**
   * Click on the athlete's username
   * Verify navigation to the athlete profile page
3. **Select date and check session details**
   * Wait for the calendar to load
   * Click on a specific date
   * Verify session details, weather data, map, and charts
4. **Logout and login as cyclist**
   * Logout as coach
   * Login as cyclist
   * Verify successful login and token storage
5. **Select date and check session details as cyclist**
   * Wait for the calendar to load
   * Click on a specific date
   * Verify session details, weather data, map, and charts

**Screenshot**

A screenshot of a computer

Description automatically generated

**Photo 6.2:** Athlete profile test

**Create training plan tests**

**Test file: creatingplan.cy.js**

**Description**

This test suite verifies the functionality of creating a new training plan, including the creation of a training template, assigning it to a cyclist, and deleting the template. The test ensures that all steps are performed correctly and the expected alerts and changes are observed.

**Test steps**

1. **Login as coach**
   * Visit login page
   * Enter coach credentials and submit
   * Verify successful login and token storage
2. **Navigate to create training plan page**
   * Visit create training plan page
   * Verify navigation to the page
3. **Create new training template**
   * Click to create a new training template
   * Fill in template details and submit
   * Verify successful creation of the template
4. **Create training plan**
   * Fill in training plan details
   * Assign template and cyclist
   * Submit the training plan
   * Verify successful creation of the training plan and alert message
5. **Delete template**
   * Hover over the delete icon on the template
   * Click to delete and confirm the prompt
   * Verify successful deletion of the template

**Screenshot**

A screenshot of a computer

Description automatically generated

**Photo 6.3:** Creating plan test

**Login tests**

**Test file: login.cy.js**

**Description**

This test suite verifies the login functionality for both coaches and cyclists. It ensures that the login process works correctly, the user is redirected to the appropriate dashboard, and the authentication tokens are stored correctly in local storage.

**Test steps**

1. **Login as coach**
   * Visit login page
   * Enter coach credentials and submit
   * Verify successful login, redirection to dashboard, and token storage
2. **Login as cyclist**
   * Visit login page
   * Enter cyclist credentials and submit
   * Verify successful login, redirection to dashboard, and token storage

**Screenshot**

A screenshot of a computer

Description automatically generated

**Photo 6.4:** Login test

**Password recovery tests**

**Test file: passrecovery.cy.js**

**Description**

This test verifies the password recovery functionality. It ensures that a recovery request can be made and that a success alert is displayed, even if the email is not in the database.

**Test steps**

1. **Request password recovery**
   * Visit password recovery page
   * Enter test email and submit
   * Verify successful request and alert message
   * Verify redirection to login page

**Screenshot**

A screenshot of a computer program

Description automatically generated

**Photo 6.5:** Password recovery test

**Profile tests**

**Test file: profile.cy.js**

**Description**

This test suite verifies the functionality of viewing and updating the user profile for both coaches and cyclists. It ensures that profile data can be fetched, displayed, and updated correctly.

**Test dteps**

1. **Login as coach**
   * Visit login page
   * Enter coach credentials and submit
   * Verify successful login and token storage
2. **View and update profile as coach**
   * Navigate to profile page
   * Verify profile data display
   * Update profile without changing username and submit
   * Verify successful profile update
3. **Login as cyclist**
   * Visit login page
   * Enter cyclist credentials and submit
   * Verify successful login and token storage
4. **View and update profile as cyclist**
   * Navigate to profile page
   * Verify profile data display
   * Update profile with random height and weight without changing username and submit
   * Verify successful profile update

**Screenshot**

A screenshot of a computer

Description automatically generated

**Photo 6.6:** Profile test

**Registration tests**

**Test file: registration.cy.js**

**Description**

This test suite verifies the registration functionality for both coaches and cyclists. It ensures that new users can register successfully and are redirected to the login page.

**Test steps**

1. **Register new coach**
   * Visit registration page
   * Fill in coach registration form and submit
   * Verify successful registration and redirection to login page
2. **Register new cyclist**
   * Visit registration page
   * Fill in cyclist registration form with additional details and submit
   * Verify successful registration and redirection to login page

**Screenshot**

A screenshot of a computer

Description automatically generated

**Photo 6.7:** Registration test