Sprint 3 Artifacts

3.1 Create a User

Description:
Implement functionality to create a new user account through the registration process.

User Story:
As a new user, I want to create an account so that I can save schedules and access personalized data.

Acceptance Criteria:

- User can register with a valid email and password.
- Duplicate emails are not allowed.
- Passwords are securely hashed before storage.
- Successful registration returns a confirmation response or redirects to login.

Tasks:

- Implement `User` model in backend using the Django Authentication application.

Evidence of Completion:

- Screenshot of `User` model and migration.

Commit:
`feat: implement user registration backend and frontend integration`

Testing / Verification:

- Manual verification via UI form submission.

Assigned to: Owen

3.2 Login as a User

Description:
Allow users to log into the application using their registered credentials.

User Story:
As a user, I want to log in so that I can access my saved schedules and personalized dashboard.

Acceptance Criteria:

- User can log in with valid email and password.
- Invalid credentials return proper error messages.
- Session persists across refresh (until logout).

Tasks:

- Implement the login form validation and error handling with the Django authentication application.

Evidence of Completion:

- Screenshot of successful login.
- Screenshot of frontend showing logged-in state.

Commit:
`feat: add user authentication endpoints and frontend login functionality`

Testing / Verification:

- Manual test verifying protected routes require authentication.

Assigned to: Owen

3.3 Save User Data (Schedules/Custom Courses)

Description:
Enable users to save schedules and custom courses associated with their account.

User Story:
As a logged-in user, I want to save my schedules and custom courses so that I can access them later.

Acceptance Criteria:

- Saved schedules and courses are linked to the authenticated user.
- Users can create, update, and delete their saved items.
- Data persists across sessions.

Tasks:

- Extend the database to include a `CustomCourse` model, associated with a `User`.
- Implement API endpoints for saving and retrieving user data.
- Update frontend components to call save/retrieve endpoints.

Evidence of Completion:

- Screenshot of data in database showing user link.
- Screenshot of working "Save" button in UI.

Commit:
`feat: implement save/retrieve functionality for user-specific data`

Testing / Verification:

- API tested with authenticated requests.
- Confirm saved schedules persist after logout/login.

Assigned to: Matthew, Connor

3.4 Establish Different Levels of User Roles (Student/Admin)

Description:
Introduce user roles in the system to differentiate between regular users and admins.

User Story:
As an admin, I want elevated access to manage all Courses; as a student, I want normal user permissions.

Acceptance Criteria:

- User model includes a `role` field (`student` or `admin`).
- Default role = `student` upon registration.
- Admins can edit the courses seen by all users.

Tasks:

- Add `role` field to User model.
- Modify authentication middleware to check role permissions.
- Create initial admin account for testing.

Evidence of Completion:

- Screenshot of role field in database.
- Screenshot of admin accessing restricted route.

Commit:
`feat: add user roles and permissions system`

Testing / Verification:

- Tested route access for student vs. admin roles.
- Verified role-based logic through API and frontend.

Assigned to: Owen

3.5 Implement Role-Specific Behavior

Description:
Add frontend and backend behavior differences for each user role.

User Story:
As an admin, I should be able to edit the list of Courses to select from.

Acceptance Criteria:

- Admin UI includes options for managing courses.
- Student UI only shows personal data.
- Role checks are enforced on both frontend and backend.

Tasks:

- Add conditional rendering in React based on user role.
- Protect admin routes using middleware.
- Add role display on dashboard.

Evidence of Completion:

- Screenshot of admin dashboard and student dashboard.

Commit:
`feat: implement role-based UI and route protection`

Testing / Verification:

- Manual role-switch testing.
- Confirm backend returns forbidden response for unauthorized role.

Assigned to: Owen

4.1 Identify Conflicts Between Selected Courses

Description:
Automatically detect time conflicts among a user's selected courses.

User Story:
As a user, I want to be warned if I select courses that overlap in time.

Acceptance Criteria:

- Conflict detection logic compares day/time ranges.
- UI visually indicates conflicting courses.
- Conflict warnings appear before saving schedule.

Tasks:

- Implement conflict-checking function in backend or frontend.
- Integrate conflict warnings in UI course list.
- Write tests for overlap logic.

Evidence of Completion:

- Screenshot of conflict warning in UI.

Commit:
`feat: add course time conflict detection`

Testing / Verification:

- Unit tests for conflict logic.
- Manual testing with overlapping courses.

Assigned to: Mohamed

4.2 Ensure Class Data is Well-Formatted

Description:
Validate and clean class data for consistency before processing or saving.

User Story:
As a developer, I want the class data to be properly formatted so the app functions reliably.

Acceptance Criteria:

- All course entries include required fields.
- Invalid or incomplete data is handled gracefully.

Tasks:

- Write data validation function.
- Apply validation when importing or updating course data.

Evidence of Completion:

- Screenshot of validation output logs.

Commit:
`chore: add validation for course data formatting`

Testing / Verification:

- Tested invalid data cases.

Assigned to: Connor

4.3 Save User Schedule

Description:
Allow users to save their current schedule configuration.

User Story:
As a user, I want to save my built schedule for future access.

Acceptance Criteria:

- User can click “Save Schedule.”
- Data persists under their account.
- Schedule name and timestamp are stored.

Tasks:

- Add backend route to save schedule JSON.
- Connect to frontend button.

Evidence of Completion:

- Screenshot of saved schedule entry in DB.

Commit:
`feat: implement save schedule functionality`

Testing / Verification:

- Manual verification of saved schedule retrieval.

Assigned to: Connor

4.4 Load User Schedule

Description:
Enable users to load a previously saved schedule.

User Story:
As a user, I want to retrieve my saved schedules to edit or reuse them.

Acceptance Criteria:

- Saved schedules display in a list or dropdown.
- Selecting one loads it into the schedule builder.

Tasks:

- Implement GET `/schedules` endpoint.
- Add frontend “Load Schedule” feature.

Evidence of Completion:

- Screenshot showing loaded schedule in UI.

Commit:
`feat: implement schedule retrieval and load functionality`

Testing / Verification:

- Manual test verifying schedules load correctly.

Assigned to: Connor

4.9 Blacklist Times

Description:
Allow users to mark times when they can’t take classes.

User Story:
As a user, I want to block out certain times so those slots aren’t included in generated schedules.

Acceptance Criteria:

- Users can select unavailable time blocks.
- Scheduling algorithm excludes those blocks.

Tasks:

- Add `blacklist_times` field in user preferences.
- Integrate blacklist into schedule generation logic.

Evidence of Completion:

- Screenshot of UI showing blocked times.

Commit:
`feat: implement time blacklisting for schedule generation`

Testing / Verification:

- Tested algorithm skips blacklisted times.

Assigned to: Matthew

4.10 Pin a Specific Instance of a Course

Description:
Let users lock specific class sections they prefer before generating full schedules.

User Story:
As a user, I want to pin specific sections so they're always included in my schedule.

Acceptance Criteria:

- User can pin/unpin sections.
- Pinned sections are preserved in generated schedules.

Tasks:

- Add pinned field to course selection state.
- Modify schedule generation to honor pinned items.

Evidence of Completion:

- Screenshot of pinned course in UI.

Commit:
`feat: add pinning functionality for specific course sections`

Testing / Verification:

- Tested schedule generation with pinned courses.

Assigned to: Matthew

4.11 Add Custom Course

Description:
Enable users to manually add non-official or personal course events.

User Story:
As a user, I want to create custom events (like study sessions) in my schedule.

Acceptance Criteria:

- User can enter name, time, and days… etc.
- Custom courses appear visually like regular ones.
- Custom data persists per user.

Tasks:

- Create `CustomCourse` model.
- Add frontend “Add Custom Course” form.
- Link to user account.

Evidence of Completion:

- Screenshot of custom course in schedule UI.

Commit:
`feat: allow users to add and save custom courses`

Testing / Verification:

- Manual verification of creation and persistence.

Assigned to: Matthew

5.5 Create Settings Page Layout

Description:
Design and implement a settings page where users can manage account preferences.

User Story:
As a user, I want to edit my account settings, change role (if admin), and manage data.

Acceptance Criteria:

- Settings page displays user info and preferences (ex. Date range/time range of calendar).
- Form allows updates to fields (e.g., name, role visibility).
- Saved changes persist to backend.

Tasks:

- Create `SettingsPage` React component.
- Add backend routes for updating preferences.

Evidence of Completion:

- Screenshot of settings UI.

Commit:
`feat: create user settings page layout and API integration`

Testing / Verification:

- Manual test of data updates.

Assigned to: Bryson

Source: Initial generations by ChatGPT | Edited by Group 4