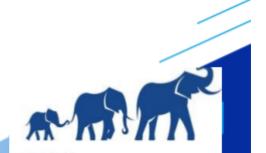
>F5

SMARTSTUDENT HANDBOOK

Title	Name	Surname	Student No
Mr	Takudzwa	Magunda	u22599160
Mr	Reinhard	Pretorius	u22509578
Mr	Mpumelelo	Njamela	u23534932
Mr	Tanaka	Ndhlovu	u22610792
Mr	Junior	Motsepe	u22598473

Coding Standards





Frontend (next.js 13 + react + typescript)

code formatting

- 2-space indentation, no tabs
- o max line length 100 chars
- o always use semicolons and trailing commas in multi-line literals
- o prettier + eslint (airbnb-typescript base) runs on every commit

naming conventions

- react components, pages and hooks use PascalCase (ExamplePage, UseUserId)
- normal variables and functions use camelCase (fetchOrgs, userId)
- custom hooks are prefixed with use (useSearchParams)
- enums and types use PascalCase (LecturePayload)

• file and folder layout

- app/ for next.js routing, one folder per feature (organisations, timetable, auth)
- o components/ui contains reusable, style-free shaden primitives
- lib/ holds helpers
- each feature may have its own hooks.ts and api.ts

commenting and code documentation

- use // for single short hints, /** ... */ JSDoc on exported utils and hooks
- describe all function params and return types in JSDoc

best practices

- o business logic lives in hooks or helpers, never inside JSX trees
- o all remote calls use swr/fetcher wrappers with stale-while-revalidate
- o error boundaries wrap every top-level route segment
- react strict mode is on; no any types if generics are painful, create a proper interface

Example code:

error handling

- first line of every handler checks req.auth?.uid, else throw new HttpsError('unauthenticated')
- validate all payload fields, throw HttpsError('invalid-argument') when missing

commenting

JSDoc over every callable, describe payload and return value

deploy secrets

 never commit serviceAccount json; use firebase functions:config:set or Secret Manager

Example code:

Doxygen Commenting

back-end (firebase cloud functions v2, typescript)

- code formatting matches front-end (2 spaces, 100 char lines, semicolons)
- naming conventions
 - callable names use lowerCamelCase (getLectures, addLecture)
 - constants and env keys use UPPER_SNAKE_CASE
- structure
 - functions/src has one file per feature (lectures.ts, semesters.ts, organisations.ts)
 - index.ts re-exports export * from "./lectures" etc

tag order inside a block

- 1. @param one line per parameter name then short description
- 2. @returns (or @return) description of the value produced
- 3. @throws each possible HttpsError code and the condition
- 4. @example minimal usage snippet when the API is not obvious

```
/**
  * @file lectureSlots.ts
  * @brief Cloud Functions for CRUD operations on lecture-slot data.
  *
  * Wrapped in a single file to keep feature boundaries clear.
Re-export
  * from `index.ts` via `export * from './lectureSlots'`.
```

```
*/
import { onCall, CallableRequest, HttpsError } from
'firebase-functions/v2/https'
import { db } from './firebaseAdmin'
/**
 * Delete a lecture slot owned by the current user.
 * @param lectureId The ID of the slot to remove.
 * @returns
                    `{ success: true }` when the slot is deleted.
 * @throws unauthenticated Caller is not signed in.
 * @throws invalid-argument `lectureId` was not supplied.
 * @throws not-found
                             No slot with that ID exists for this
user.
 *
* @example
 * ```ts
 * const res = await deleteLecture({ lectureId: 'slot123' });
 * console.log(res.success); // → true
 */
export const deleteLecture = onCall(
  async (req: CallableRequest<{ lectureId: string }>) => {
    const uid = reg.auth?.uid
    const { lectureId } = req.data
    if (!uid)
                        throw new HttpsError('unauthenticated',
'login required')
    if (!lectureId)
                     throw new HttpsError('invalid-argument',
'lectureId missing')
    const ref = db.ref(`users/${uid}/lectureSlots/${lectureId}`)
    const snap = await ref.get()
    if (!snap.exists()) throw new HttpsError('not-found', 'slot not
found')
```

```
await ref.remove()
  return { success: true }
}
```

Testing and ci

- jest runs unit tests on hooks and helpers; npm test fails the build on any snapshot drift
- playwright covers critical flows (login, create note, join org) in chromium and firefox
- github actions executes lint, type-check, jest and playwright on every push

Formatting Tools

- prettier, eslint, stylelint all wired to npm run lint
- husky pre-commit hook runs npm run lint:fix