WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

Automated Time-Table Assistant

Mrs. Kirubasri S*, Lingeshwaran A, Pradeep U, Karthik R[4]

*Assistant Professor, Department of Information Technology, Sri Shakthi Institute of Engineering and Technology, Coimbatore-641062, Tamil Nadu, India.
UG Students, Department of Information Technology, Sri Shakthi Institute of Engineering and Technology, Coimbatore-641062, Tamil Nadu, India.
Email ID: lingeshwarana@gmail.com

ABSTRACT

Students follow weekly class schedules that change from week to week, yet the default clock and calendar applications on most devices have no understanding of a timetable. Classes, assignment deadlines, and exams are therefore missed when reminders are set by hand and not updated after a change. This paper presents the Automated Time-Table Assistent, a web-based application in which a student signs in with a mobile phone number, builds a weekly timetable, and receives alarms and reminders linked to it. The system uses a React and TypeScript frontend, a Node.js and Express backend, Firebase Authentication for phone-number OTP login, and Cloud Firestore for storage, with every record tied to the authenticated user. It supports weekly timetable changes with an effective date, copying the previous week with a confirmation step, and an archived history of earlier weeks. A server-side scheduler that works in the user's own time zone creates class and alarm reminders with fixed notification identifiers, so the same event is not sent twice. The interface adds repeating alarms, snooze, customizable alarm tones, and data export and import. The input-validation test suite (14 cases) passes, and the main workflows were checked manually on a local setup. Push delivery to closed browser tabs, testing on real devices, and public deployment remain for further development. Keywords: Timetable Management, Smart Alarm, Reminder Scheduling, Firebase Authentication, Cloud Firestore, Push Notification, Student Productivity, Web Application

1. INTRODUCTION

Managing time is a daily task for every student. A typical week contains lectures, laboratory sessions, study periods, assignment deadlines, and examinations, and many institutions change the timetable from one week to the next. A student who misses a changed class or forgets a deadline loses learning time, and the cause is often a reminder that was set once and never updated. The clock and alarm applications that ship with phones are simple and reliable, but they only know about time. They do not know which subject a class belongs to, which room it is held in, or that next week's timetable differs from this week's. Calendar applications know about events, but every class and every reminder must be entered by hand. A tool that understands the timetable and the alarm together can remove much of this manual effort

1.1 Existing challenges in timetable and reminder management

Students usually combine several tools: a photograph or printout of the timetable, a calendar application, and a separate alarm application. Each tool is updated by hand, so a change in the timetable has to be repeated in every tool. This is slow and easy to forget. Reminder systems add further problems. Notifications may arrive twice for the same event, or not at all. Times shown to the user can be wrong when the device time zone or daylight-saving rules differ from the schedule. Browsers and operating systems also limit when a web page may play sound or show a notification, and these limits are rarely explained to the user

1.2 Role of cloud authentication and scheduled notifications

A mobile phone number is a natural sign-in method for students, because every student already has one. Cloud authentication services can send and check a one-time password, so the application does not need to store passwords. Once a user is identified, the server can keep each student's timetable, alarms, and settings separate from everyone else's. A scheduler running on the server can work out which reminders are due without depending on the application being open. It can use each user's own time zone, record every notification it creates, and give each event a fixed identifier so that the same event is not notified twice. Delivery to a device still depends on push services, browser permissions, and the operating system.

1.3 Problem statement and objectives

Existing approaches keep the timetable, the alarms, and the notifications in separate tools. There is a need for one application that links them, copes with weekly timetable changes, and protects each student's data. Page 2 of 11 AUTOMATED TIME-TABLE ASSISTENT The main objective of this work is to design and build a web-based Automated Time-Table Assistent that combines a weekly timetable, alarms, reminders, and notifications for students. The specific objectives are:

1. To provide sign-in with a mobile phone number and a one-time password, with the data of each user stored separately.

2. To let users create and manage a weekly timetable covering all seven days, with subject, faculty, start and end time, location, colour, and reminder time.

3. To support a weekly timetable change with an effective date, copying of the previous week with a confirmation step, and a history of earlier weeks.

4. To provide alarms with repeat days, enable and disable controls, snooze, and a choice of alarm sounds.

5. To create reminders for classes, assignments, exams, and study sessions, without duplicate notifications.

6. To schedule reminders in the time zone of each user.

7. To keep notification preferences and a notification history, and to let the user export and import their data.

8. To evaluate the application through validation testing and manual functional testing

2. MATERIALS AND METHODS

2.1 Development approach

The project followed the usual stages of software development: requirement analysis, system design, implementation, integration, testing, and evaluation. The requirements were taken from the objectives in Section 1.3. The backend was built and tested first, and the React interface was then connected to it in place of the sample data it used at the start.

2.2 System architecture

The system is organized in six layers. The user interface layer is a React application with five screens: Home, Timetable, Alarms, Calendar, and Settings. The authentication layer uses Firebase Authentication to verify a phone number with a one-time password; the interface then sends the resulting ID token with every request.

The application logic layer is a REST API built with Express. It verifies the token, validates the data it receives, and reads or writes the records of the signed-in user only. A scheduler layer, started together with the API, runs every minute and decides which reminders are due. The data storage layer is Cloud Firestore. The notification layer has two parts: push messages sent through Firebase Cloud Messaging to registered devices, and an alarm engine inside the browser page that plays a sound while the application is open. The layers are shown in Fig.

2.3 Technology stack and development environment

The frontend uses React, TypeScript, and the Vite build tool. The backend uses Node.js, Express, and TypeScript. The Firebase Admin SDK connects the backend to Firebase Authentication, Cloud Firestore, and Cloud Messaging. The Zod library validates incoming data, Luxon handles time zones, and node-cron runs the scheduler. Automated tests are written with Vitest. Source code is kept in a Git repository, and development and testing were done in Visual Studio Code with a desktop web browser.

2.4 User roles and access control

The application stores only the data it needs: the verified phone number, the time zone, the timetables and their entries, the alarms and reminders, the notification preferences, the registered device tokens, and the notification history. All records carry the identifier of their owner, and every query is filtered by it. The browser never reads the database directly. The Firestore security rules deny all direct access, and only the backend, which uses the Admin SDK after checking the user's token, reads and writes data. Incoming data is validated before it is stored: times must be in the 24-hour HH:mm form, the end time must be later than the start time, the day number must be from 0 (Monday) to 6 (Sunday), dates must be in the YYYY-MM-DD form, and an alarm needs either a date or at least one repeat day

2.5 Database design

The data is held in eight Firestore collections, listed in Table 1.

2.6 AI-based bus detection and direction logic

Each weekly timetable has an effective date. The timetable in force on a given day is the active one with the latest effective date that is not later than that day, judged in the time zone of the user. To prepare a new week, the user copies an earlier timetable under a new name and effective date and then edits the copy. If an active timetable already exists for the chosen effective date, the server refuses the request and asks for an explicit confirmation. After confirmation, the older timetable is not deleted. It is marked as archived and stays available as history. Reminders are created from the timetable in force on the day, so a change takes effect without the user editing any alarm

2.7 Analysis workflow

The scheduler works as follows. Every minute it visits each user and reads the time zone stored for that user. It then looks at today and tomorrow and collects four kinds of events: classes from the timetable in force, with the reminder time equal to the start time minus the reminder minutes of the class; alarms and reminders, either on selected weekdays or on a given date; snoozed reminders whose time has arrived; and a notice that a new timetable has become effective, sent after 6 a.m. on that day. An event counts as due if its time passed within the last three minutes, so a short delay does not lose it. The steps are shown in Fig. 2. Each event is given a fixed identifier built from the user, the kind of event, the date, and the time. Before sending, the scheduler tries to create the history record under that identifier. The operation fails if the record already exists, so the same event is not notified twice, even if two server instances run at the same time. The message is then sent to the devices registered by the user, tokens that the push service reports as invalid are removed, and the result (sent, failed, or no device) is stored in the notification history. Notification preferences are checked before anything is created. While the application is open in a browser, a separate alarm engine checks the classes and alarms every 15 seconds. When one is due, it shows a full-screen alert with a Stop button and plays a sound for the selected duration of 10, 20, 30, or 60 seconds. The sound can be any of eight built-in tones generated with the Web Audio interface, or an audio file chosen by the user (up to 10 MB), which is kept in the browser of that device. The volume can be adjusted. The engine remembers which events have already rung on the day, so a page reload does not repeat them.

2.9 Testing and evaluation

Testing had two parts. First, automated tests written with Vitest check the validation rules for timetable entries, the weekly timetable fields, and alarms (14 test cases in total). Second, the main workflows were tested manually on a local setup, with the backend and the interface running on the same computer. Firebase test phone numbers were used so that no real text message was sent. A result is reported only when it was observed. Features that are implemented but have not been tested end to end, such as push delivery to a real device, are listed as pending in Section 3.

3. RESULTS AND DISCUSSION

Implemented functionality The application brings the timetable, alarms, and reminders into one system. On the server side, the following functions are implemented: sign-in with a verified phone number; create, read, update, and delete operations for timetables, timetable entries, alarms, and reminders; copying of a timetable to a new effective date with confirmation and an archived history; notification preferences, device registration, and a notification history with read, dismiss, and snooze actions; the scheduler; and export and import of a user's data as a file. In the interface, a student signs in with a phone number and a code, and then adds, edits, and deletes classes and alarms. These are stored in Firestore and are shown again after the page is reloaded. The Settings screen lets the user choose a built-in tone or an own song, set the volume, and set how long the alarm rings, with a preview button. The dedicated screen for the weekly timetable change, which calls the copy function of the server, has not yet been added to the interface.

3.1 Functional evaluation

The test scenarios and their results are summarised in Table 2. Results are marked as passed when an automated test confirms them, as observed when they were checked manually, and as not yet tested when the feature exists but has not been verified .

3.2 Comparison with conventional approaches

With a separate calendar and alarm application, the student enters every class and every reminder by hand and repeats the work when the timetable changes. The proposed system keeps the timetable as the single source, so reminders follow the timetable in force on the day. It also stores the data on the server, so it is not lost when one browser is cleared

This comparison describes the design of the system. The work does not include a user study or a timing measurement, so it cannot yet be said that the application saves a measurable amount of time or reduces missed classes.

3.3 Discussion and limitations

The design keeps each part separate: validation, scheduling, and delivery can be changed independently. Creating each notification under a fixed identifier is a simple way to prevent duplicates, and working in the user's time zone keeps reminder times correct for the user. Time-zone handling is done by the Luxon library, but daylight-saving transitions were not tested separately. The work also has clear limits. The alarm sound in the browser is played only

while the application is open in a tab, and browsers may block sound until the user has interacted with the page. Push messages to a closed browser need a service worker and a web push key in the interface, which are not yet implemented, so the server-side push path has not been verified with a real device. On an iPhone, web push needs the application to be added to the home screen. A web page cannot ring like the native alarm application of a phone. Sending real text messages to numbers outside the test list requires enabling the region in Firebase, which may need a paid plan. The scheduler needs a server that stays running, so a deployment on an always-on service is required for reminders to arrive on time. Phone numbers are personal data, so access is limited to the backend, keys are kept outside the repository, and every record is bound to its owner. A formal security review has not been carried out.

4. CONCLUSION

The Automated Time-Table Assistent presents a web-based approach to help students keep to a timetable that changes every week. It combines phone-number sign-in, a weekly timetable with effective-date changes and archived history, alarms and reminders, a time zone-aware scheduler that avoids duplicate notifications, and alarm sounds that the user can choose.

The 14 automated validation tests pass, and the main workflows of sign-in, saving a class, and ringing an alarm in an open tab were checked manually on a local setup. The practical value of the application still depends on testing that has not yet been done: push delivery to real devices, use by more than one user, and use by students over several weeks. Future development can focus on the following points:

1. Adding a service worker and web push so that reminders reach the user when the browser tab is closed.

2. Adding the screen for the weekly timetable change to the interface and testing it end to end.

3. Testing the scheduler on real devices, with several users, and across a daylight-saving change.

4. Deploying the backend and the interface on an always-on service so that the scheduler runs continuously.

5. Enabling real text-message sign-in for the regions of the users.

6. Providing an installable or native mobile version, so that alarms can ring like those of the phone's own clock application.

7. Carrying out a user study to measure whether students miss fewer classes and deadlines

5. ACKNOWLEDGMENTS

The authors would like to thank Sri Shakthi Institute of Engineering and Technology for providing the academic environment and facilities required for this project. The authors also express their sincere gratitude to the Head of the Department of Information Technology, the project guide Ms. S. Kiruba, and the faculty members for their guidance, encouragement, and support throughout the development of the project.

6. REFERENCES

1. React. React Documentation. Available at: https://react.dev/

2. Express. Express Web Framework Documentation. Available at: https://expressjs.com/

3. Google Firebase. Firebase Authentication Documentation. Available at: https://firebase.google.com/docs/auth

4. Google Firebase. Cloud Firestore Documentation. Available at: https://firebase.google.com/docs/firestore

5. Google Firebase. Firebase Cloud Messaging Documentation. Available at: https://firebase.google.com/docs/cloud-messaging

6. Zod. Zod Documentation. Available at: https://zod.dev/

7. Luxon. Luxon Documentation. Available at: https://moment.github.io/luxon/

8. node-cron. node-cron Project Page. Available at: https://github.com/node-cron/node cron

9. Vitest. Vitest Documentation. Available at: https://vitest.dev/

10. Mozilla Developer Network. Web Audio API. Available at: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

11. Mozilla Developer Network. Push API. Available at: https://developer.mozilla.org/en US/docs/Web/API/Push_API

12. OWASP Foundation. OWASP Application Security Guidance. Available at: https://owasp.org