International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

CAREERPREP AI: A Local-First Platform for Placement Readiness Scoring, Speech-Based Mock Interviews and Personalized Career Roadmaps

Nivetha KT[1], Manisha S[2], Mekajaya S[3], Mithra S[4]

1 Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
2 Email: nivethakt25cs@srishakthi.ac.in
3 Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
4 Email: manishas25cs@srishakthi.ac.in
5 Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
6 Email: mekajayas25cs@srishakthi.ac.in
7 Student, Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, India.
8 Email: mithras25cs@srishakthi.ac.in

ABSTRACT

CareerPrep AI is a local-first web platform that prepares final-year engineering students for technical and behavioural campus placement drives. It combines a five-step onboarding profile, a weighted Placement Readiness Score, five diagnostic assessments, a role-aware four-week study planner, a speech-based HR interview simulator, a resume-to-job-description matcher, a recruiter directory and peer study groups. Every displayed metric is computed from the user's own submissions rather than stored as a fixed value. Interview answers are evaluated through STAR-marker detection, speaking-pace measurement and filler-word analysis; readiness is aggregated from five weighted components. React 18 and Vite provide the interface, while Node.js, Express and embedded SQLite provide authentication, scoring and persistence. Passwords are hashed with bcrypt and sessions are carried in HTTP-only JWT cookies. An end-to-end integration test covering signup, session persistence, task generation, assessment submission and cleanup passed on the local release. The scoring methods are transparent heuristics, not validated predictors of selection, and no controlled user study or independent accuracy evaluation has yet been performed.

Keywords: *placement preparation, readiness score, mock interview, STAR framework, resume–job matching, skill gap analysis, local-first architecture, speech recognition, personalised roadmap, SQLite.*

I. INTRODUCTION

Campus placement is the most consequential career step for many engineering students. A single recruitment drive may combine an aptitude screen, a coding or technical round, a group discussion and one or more interviews, and the student is expected to be ready for all of them at once. In practice, preparation is scattered across unrelated resources: a resume template from one site, coding problems from another, aptitude books, and interview videos whose advice is not tied to the student's own profile. No single place tells a student how ready they are, what to study next or whether their practice is making a difference.

Four difficulties recur. Candidates often do not know how well a resume matches a particular job description or whether it will pass an applicant tracking system (ATS). They have little access to realistic interview practice with immediate feedback. They are unsure which skills to learn for a target role, so effort is spent on topics that do not move their chances. Finally, professional career counselling is expensive or hard to reach, especially for students outside large cities. These problems are not caused by a lack of material; they are caused by the absence of a personalised, measurable and continuous preparation loop.

The objective of CareerPrep AI is to provide that loop inside one application. The system pursues five objectives:

1. analyse a resume against a target job description and suggest improvements, including ATS compatibility;
2. simulate role-specific mock interviews with instant, structured feedback;
3. identify the gap between the student's declared skills and the requirements of the chosen role;
4. generate a personalised, step-by-step study and preparation plan; and
5. track progress over time through a single readiness measure and a dashboard.

The application is intended for final-year engineering students and other job seekers preparing for placements. It supports preparation; it does not replace a recruiter's judgement, professional counselling or a formal skill certification.

A central design decision concerns where numbers come from. Many preparation tools display illustrative scores that are fixed in the interface. In CareerPrep AI every metric is derived from stored user activity: completed roadmap tasks, assessment results, interview sessions, resume scans and peer-group membership. The Placement Readiness Score, the interview score and the dashboard summaries are recomputed whenever the underlying records change. A student who has done nothing therefore sees a low score, and the score rises only when work is recorded.

A second decision concerns operation. The platform is local-first: an Express.js server runs embedded SQLite operations through better-sqlite3 and has no mandatory external cloud dependency. Optional MySQL connectivity, for example through XAMPP, is available as a second storage layer. A single command starts both the frontend and the backend, which makes the system easy to demonstrate, inspect and reproduce in a laboratory or classroom setting.

The report is deliberately careful about what the scores mean. The readiness score is an activity-weighted guidance heuristic. The interview evaluator measures the presence of STAR vocabulary, speaking pace and filler words; it does not understand the semantic quality of an answer. The resume matcher compares text against a job description; it does not predict whether a candidate will be shortlisted. These limits are stated alongside the results so that the system is not mistaken for a validated assessment instrument.

The browser interface provides a public landing page and an authenticated workspace. After sign-in, the dashboard summarises the profile and readiness, and navigation leads to the profile, company directory, peer groups, assessments, interview simulator, resume tools, study planner and notifications. Dark and light themes are supported through shared design tokens.

The remaining sections describe related approaches, the proposed methodology, the implementation, the advantages of the design, measured and analytical results, limitations, conclusions and future work.

The main contributions of the work are as follows:

- an integrated preparation workflow in which resume matching, interview practice, assessments, planning and peer activity feed one readiness measure;
- a transparent Placement Readiness Score whose five components and weights are stated and reproducible;
- a speech-based interview evaluator that scores STAR structure, speaking pace and filler words from a browser transcript;
- a role-aware four-week planner that is generated on first use and updates the score as tasks are completed; and
- a local-first implementation with cookie-based authentication, prepared statements, rate limiting and non-destructive schema migration.

The expected outcomes for a student who uses the platform consistently are better resumes that align with the vocabulary of automated screening, more confidence and practice before real interviews, a

clear and personalised plan for skill development, and a low-cost form of career guidance that can be offered to many users at the same time.

II. EXISTING SYSTEM AND LITERATURE SURVEY

Existing approaches to placement preparation fall into a small number of categories: manual counselling and mentoring, resume-checking tools, mock-interview services, coding and aptitude practice platforms, and static course roadmaps. Each is useful for a part of the problem. None should be mistaken for a complete preparation system, and most do not connect their outputs to one another. Table I summarises the typical coverage of each category, as characterised by the authors; it is not a measured comparison of specific products.

TABLE I. TYPICAL CAPABILITY COVERAGE BY CATEGORY

Approach	Resume	Interview	Skill gap	Plan	Progress
Manual counselling	Partial	Partial	Partial	Partial	Limited
Resume / ATS checkers	Full	None	Limited	None	Limited
Mock-interview tools	None	Full	None	Limited	Partial
Coding / aptitude sites	None	None	Partial	Limited	Partial
Static roadmaps	None	None	Partial	Partial	None
CareerPrep AI	Full	Full	Full	Full	Full

"Full" refers to the presence of the capability, not to its validated accuracy.

Manual counselling. A mentor or placement officer can interpret context, but the quality of advice depends on the individual's time and experience, and the record of what was advised is rarely reproducible. Counsellors are scarce relative to student numbers, and private career coaching is expensive. The result is that students in smaller towns often rely on informal advice from seniors.

Resume and ATS tools. Resume checkers typically examine structure, formatting and keyword overlap with a job description. Keyword matching is interpretable and inexpensive, but it can reward keyword stuffing and ignore the context in which a skill was used. CareerPrep AI uses a text-comparison approach of this kind to produce an ATS-style match score and rewrite suggestions, and it presents the result as editing guidance rather than as a prediction of recruitment outcome.

Mock-interview services. Interview practice platforms provide question banks and sometimes recorded or live practice. Feedback is frequently limited to a generic rubric or depends on a human reviewer. Behavioural interviews are commonly structured around the Situation–Task–Action–Result (STAR) pattern, which has its roots in behaviour-description interviewing [1]. Because STAR gives a recognisable vocabulary, even a simple detector can give a student immediate, repeatable feedback on whether an answer follows that structure.

Practice platforms and roadmaps. Coding and aptitude sites offer large question banks with explanations, while course roadmaps list topics to study. Both are generic: they do not know the student's target role, current skills or recent performance. A fixed roadmap cannot adapt when a student completes tasks quickly or neglects a category.

Speech recognition in the browser. Voice-based practice requires converting speech to text. The Web Speech API specification defines a speech-recognition interface for web applications [5], and browsers expose it through the SpeechRecognition interface, in some browsers under a vendor prefix [6]. Using the browser's built-in recogniser avoids sending audio to a separate service that the project would have to operate, but recognition accuracy depends on the browser, microphone quality, accent and background noise. CareerPrep AI therefore treats the transcript as the unit of evaluation and does not claim to measure pronunciation or confidence acoustically.

Authentication and web security. Storing passwords safely requires an adaptive hashing scheme. The bcrypt design was introduced as a password scheme whose cost can be raised as hardware improves [2], and current guidance on

password storage continues to recommend adaptive, salted hashing [7]. JSON Web Tokens provide a compact signed representation of a session claim [3]. Keeping the token in an HTTP-only cookie prevents client-side scripts from reading it, which reduces exposure to script injection. These are established techniques; the project applies them rather than proposing new ones.

Service architecture. The application exposes its functions through a resource-oriented HTTP interface in the spirit of representational state transfer [4]. The backend runs on Node.js [13] with Express [12], the frontend uses React [10] built with Vite [11], and data are stored in SQLite [8] accessed synchronously through better-sqlite3 [9]. Synchronous access to an embedded database suits a single-process local deployment and keeps queries simple to reason about, although it would need to be revisited for a multi-host service.

Large language models. The original project proposal envisaged a large language model for question generation, answer evaluation and feedback. Retrieval-augmented generation is one established way of grounding model output in supplied material [14]. The present release does not depend on a hosted model: its feedback is produced by transparent, deterministic rules. This choice keeps the platform inspectable and free of per-request cost, and it leaves model-based evaluation as a clearly separated extension (Section IX).

Access and context. Placement preparation is not equally accessible. Students in larger cities may have access to training institutes, alumni networks and experienced mentors, whereas students in smaller towns often depend on a few faculty members and on freely available online material. A platform that runs on an ordinary college computer, without paid services, therefore addresses a practical need. At the same time, the platform's guidance reflects the assumptions built into its rules, and it should be introduced alongside, not instead of, the mentoring that colleges already provide.

Heuristic scores versus validated instruments. A recurring issue in career-preparation tools is the difference between a heuristic indicator and a validated instrument. A validated assessment is tested against an external criterion, such as later job performance or expert ratings, and its reliability is reported. A heuristic indicator is built from reasonable rules but has not been tested in that way. Both can be useful for self-improvement, but only the first can support high-stakes decisions. CareerPrep AI is explicitly of the second kind. The distinction determines how the platform's outputs should be worded and used: as prompts for the student to act on, and not as certificates of readiness or as inputs to selection decisions.

Interpretability for learners. For a learning tool, interpretability is itself a feature. A student who is told only that an answer scored 61 out of 100 learns little, whereas a student who is told that the answer contained two STAR markers and six filler words can change a specific behaviour. Rule-based evaluators are simple, but they supply exactly this kind of actionable explanation, which is one reason the present release favours them over an opaque model while the platform is still being validated.

The gap addressed by the project is therefore integrative and operational rather than a claim of a new algorithm. Existing categories each solve one part of preparation. CareerPrep AI places the resume match, interview practice, assessments, skill profile, roadmap and progress record in one local workflow, in which each activity updates a shared readiness measure.

Existing-system limitations are framed as workflow risks. Isolated tools leave the student without a combined view of readiness. Static scores can create a false sense of preparation. Generic roadmaps may not reflect the target role. Feedback that cannot be traced to a rule is difficult to learn from. CareerPrep AI addresses these risks by computing metrics from recorded activity, by exposing the rules behind each score, and by

keeping the student's data on a machine the operator controls.

III. PROPOSED METHODOLOGY

CareerPrep AI processes each student through a loop of profiling, gap identification, planning, practice, evaluation and progress tracking. The workflow is designed so that every visible score can be related to stored records and to a stated formula. The loop has six stages: (1) the user signs up and selects a target role; (2) the profile and resume are read to extract skills, education and experience; (3) the profile is compared with role requirements to find gaps; (4) practice material is generated and answers are evaluated; (5) feedback is produced and the roadmap is updated; and (6) progress is tracked so that recommendations reflect each completed session. Figure 1 shows the structure that realises this loop, and Figure 2 shows the user flow from the landing page to the feature modules.

Figure 1 CareerPrep AI layered architecture
Frontend, REST API, backend services and SQLite storage



Five-step onboarding. Registration is a guided wizard that captures the information needed to personalise every later module. Step 1 collects account credentials. Step 2 records academic background: college, degree, department, year of study and CGPA. Step 3 records career interests as a multi-select over 17 technology domains. Step 4 records core skills with a numerical proficiency level between 1 and 100 percent. Step 5 records optional social links: GitHub, LinkedIn and a portfolio address. On submission the server validates the fields, salts and hashes the password with bcryptjs at ten rounds, creates the user, and automatically initialises a placement-readiness record and a study plan.

Profile and skill model. The skill list with proficiency values, together with the selected domains and target role, is the profile against which all later comparisons are made. The skill gap is the difference between the skills required by the target role and those declared or demonstrated by the student. Because proficiency is self-declared at onboarding, the platform complements it with measured evidence from the diagnostic assessments, so that a student's claimed strengths can be checked against test results.

Figure 2 User flow
From landing page to feature modules

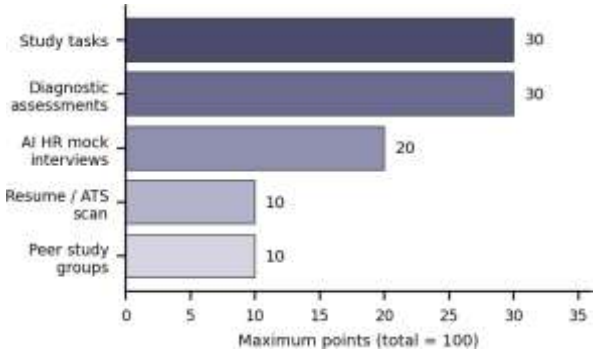


Placement Readiness Score. The readiness score summarises preparation as a single value from 0 to 100. It is a weighted aggregation of five activity components, each computed from stored records at request time. Table II lists the components and their calculation basis, and Figure 3 shows the weights.

TABLE II. READINESS SCORE COMPONENTS

Component	Max	Calculation basis
Study tasks	30	completed roadmap tasks ÷ total tasks × 30, rounded
Diagnostic assessments	30	min(30, distinct assessments with a stored result ÷ 5 × 30), rounded
AI HR mock interviews	20	min(20, interview sessions × 10)
Resume and ATS scan	10	10 if at least one resume-match scan exists, otherwise 0
Peer study groups	10	10 if the student belongs to at least one group, otherwise 0

Figure 3 Readiness score composition
Maximum points contributed by each component



$$R = min(100,\ T + A + I + C + G) \qquad (1)$$

In Equation (1), R is the readiness score, T the study-task component, A the assessment component, I the interview component, C the resume-scan component and G the peer-group component. The result is clamped at 100. Because each term is computed from database counts at the moment of the request, the score cannot drift from the underlying records, and it updates immediately when a task is toggled, an assessment is submitted or an interview is saved.

Two worked profiles show how the formula behaves. Student A has completed 3 of 8 roadmap tasks, taken 2 assessments, held 1 interview, run 1 resume scan and joined no group. Student B has completed 6 of 8 tasks, taken 4 assessments, held 2 interviews, run a resume scan and joined a group. Table III gives the component values. These profiles are constructed to illustrate the arithmetic; they are not records of real students.

TABLE III. WORKED READINESS CALCULATIONS

Component	Student A	Student B	Max
Study tasks	11	23	30
Assessments	12	24	30
Interviews	10	20	20
Resume scan	10	10	10
Peer groups	0	10	10
Total R	43	87	100

Student A: round(3÷8×30) = 11 and round(2÷5×30) = 12. Student B: round(6÷8×30) = 23 and round(4÷5×30) = 24.

Role-aware study planner. On first access to the dashboard or the planner, the system generates a four-week roadmap of eight tasks customised to the student's chosen role, for example Software Engineer or Data Scientist, and prioritised technical skills. Each task carries an assigned week, a title, concrete instructions, a category and an estimated time in minutes. The student checks tasks off, and each toggle is logged as activity and immediately changes the study-task component of the readiness score. Generation happens only when no plan exists, so a returning student keeps the progress already recorded.

The structure of the roadmap is regular even though the task text depends on the role. Two tasks are assigned to each of the four weeks, so that the plan is long enough to cover the main topics but short enough to be completed within a month. The table below gives an illustrative arrangement of focus areas by week for a Software Engineer plan. The arrangement is indicative of the type of progression, from fundamentals to applied practice, and is not a transcript of the generated tasks.

TABLE IV. ILLUSTRATIVE ROADMAP STRUCTURE (SOFTWARE ENGINEER)

Week	Focus	Example task types
1	Fundamentals	Core data structures review; language basics practice
2	Problem solving	Algorithm practice; aptitude drills
3	Core subjects	DBMS and OOP revision; project talking points
4	Interview readiness	STAR answer rehearsal; resume final check

Indicative structure; generated task text depends on role and skills.

Diagnostic assessments. Five standardised tests cover data structures and algorithms (DSA), aptitude, object-oriented programming (OOP),

database management systems (DBMS) and verbal ability. The assessment endpoint returns the test list with the student's pass status; on submission the server scores the multiple-choice answers, stores the result and creates a notification. Assessments serve two purposes: they contribute to the readiness score, and they provide measured evidence for the skill-gap view, independent of the proficiency values the student declared.

Skill-gap analysis. The gap view compares the skills a target role expects with the skills the student has declared and the areas in which assessment results are weak. A required skill that is missing from the profile, or that is declared at a low proficiency level, is reported as a gap and is used to prioritise planner tasks and to explain a lower company match. Table V illustrates the idea for a hypothetical Software Engineer profile. The values are constructed for illustration; the exact thresholds are implementation parameters and are not claimed here.

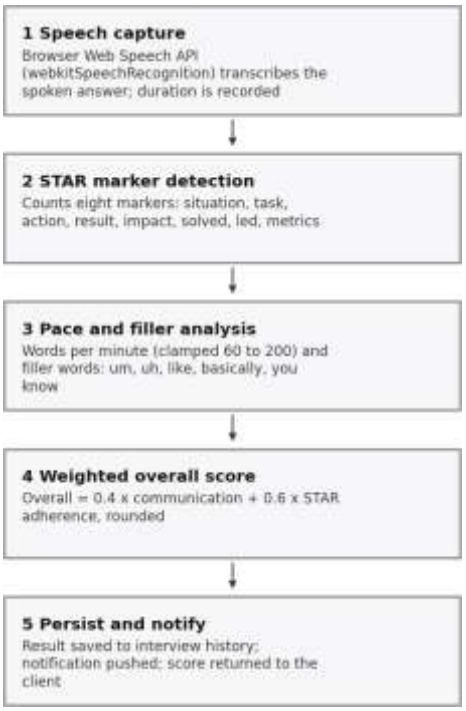
TABLE V. ILLUSTRATIVE SKILL-GAP VIEW (HYPOTHETICAL)

Role skill	Declared level	Assessment	Gap status
Data structures	70%	Taken	Small
Databases (SQL)	40%	Not taken	Medium
OOP concepts	60%	Taken	Small
Cloud basics	0%	Not taken	Large
Communication	50%	Interview	Medium

Illustrative values only; not data from a real student.

AI HR interview simulator. The simulator uses the browser's native Web Speech API to transcribe the student's spoken answer to a question in real time. The transcript and the duration of the answer are posted to the evaluation endpoint, which applies four analyses, summarised in Figure 4 and detailed below.

Figure 4 Interview evaluation pipeline
From spoken answer to stored score



STAR adherence. The transcript is lower-cased and scanned for eight marker words: situation, task, action, result, impact, solved, led and metrics. The number of markers present, s, determines the STAR score:

$$S = min(95, max(40, 40 + 8s)) \quad (2)$$

A transcript with no marker scores 40, one with four markers scores 72, and seven or more markers reach the ceiling of 95. The ceiling reflects that vocabulary alone cannot establish a perfect answer.

Speaking pace. Pace is computed in words per minute (WPM) from the transcript length and the answer duration, rounded and clamped to the range 60–200. When the duration is not available the pace defaults to 120. The recommended range displayed to the student is 120–150 WPM. In the reviewed implementation the pace is returned as feedback and is not an input to the overall score.

Filler words. A regular expression counts the verbal crutches um, uh, like, basically and you know. The communication score, K, is reduced by two points for each filler, from a base of 80, and is capped at 100:

$$K = min(100, round(80 - 2f)) \quad (3)$$

Overall interview score. The overall score, O, combines communication and STAR adherence, with greater weight on structure:

$$O = round(0.4K + 0.6S) \quad (4)$$

For example, an answer with four STAR markers (S = 72) and three filler words (K = 74) obtains O = round(29.6 + 43.2) = 73. The feedback text, the score, the pace and the filler count are saved to the student's interview history and a notification is pushed. The design also lists domain-keyword matching against keywords assigned to each question; in the reviewed endpoint the keyword list is stored empty, so keyword scoring is reserved for future use.

Because the detector is lexical, it has predictable weaknesses. Substring matching can count a marker inside a longer word, so an answer can score higher than its structure deserves. Conversely, a well-structured answer that avoids the marker vocabulary can score lower than it should. The score is therefore best read as a prompt for the student to review their own answer, not as an examiner's mark.

Resume and ATS matching. The student supplies resume text and a target job description. The matcher compares the two and returns an ATS match score together with rewrite suggestions that highlight job-description terms absent from the resume. The comparison is a text-based keyword match; it can indicate vocabulary gaps but cannot judge the quality of experience. A completed scan is recorded in the activity log and awards the resume component of the readiness score.

The resume workflow can be described as a short pipeline. First, the student provides the resume text and the target job description. Second, both texts are normalised, so that case and punctuation do not affect comparison. Third, the terms of the job description are compared with the terms of the resume, and the proportion that is shared is reported as the match score. Fourth, the job-description terms that are missing from the resume are listed as suggestions. Fifth, the scan is recorded in the activity log, and the student may edit the resume and scan again. Because the score depends on the job description supplied, the same resume can obtain different scores for different roles, which is the intended behaviour.

Assessment scoring. Each assessment is a set of multiple-choice questions with explanations. On submission, the server compares the chosen options with the stored answers, computes the percentage, compares it with the pass mark, saves the result and creates a notification. Re-attempting a test records a new result for the same assessment identifier. Since the readiness component counts distinct assessment identifiers, repeating one test does not increase the score, which discourages repetition as a way of inflating it while still allowing the student to improve their own result.

Company directory. A curated data file stores campus recruiters with their roles, offered packages and eligibility cut-offs. The directory endpoint returns the list together with the student's saved bookmarks, and a skill-match score expresses how well the student's declared skills fit each company's requirements. Company information links out to official sources; the platform does not scrape external recruitment sites.

Peer study groups. Domain-based cohorts allow students to join a group, post to discussions and arrange meetings. The groups endpoint returns live member counts and the student's membership state. Membership contributes to the readiness score because collaborative preparation is treated as a legitimate preparation activity.

Notifications and progress. Events such as assessment results, interview scores and task activity create notifications. The notifications endpoint returns each alert with both read and is_read fields so that older and newer client code can interpret it. The dashboard aggregates the readiness score, five priority tasks, recent activity and a timeline so that progress over time is visible.

Security methodology. Passwords are salted and hashed with bcryptjs. After login the server issues a signed JSON Web Token stored in an HTTP-

only cookie; a Bearer authorization header is accepted as a fallback for non-browser clients. A middleware verifies the token on every protected route, reloads the user, and returns distinct messages for a missing token, an unknown user, an expired session and an invalid token. Database access uses prepared statements, which prevents user-supplied text from altering SQL structure, and request rate limiting restricts abusive traffic. Integrity of existing databases is preserved by non-destructive schema migrations (Section IV).

Persistence and privacy. All user data are stored in a local SQLite file. Data stay on the operator's machine, and the optional MySQL layer is a configuration choice, not a requirement. Personal data requires careful handling: the report does not claim that the default configuration provides encryption at rest, a hardened production deployment, or compliance with any particular data-protection regime. These remain operator responsibilities.

Scope of the AI components. The proposal describes an AI layer for question generation, answer evaluation and feedback. In the present release, the interview evaluator, the readiness calculation, the planner generation and the matcher are deterministic, rule-based procedures. They are described as AI-assisted preparation because they adapt their output to each student's data, not because they use a trained model. Table VI contrasts the proposed design with what is implemented.

TABLE VI. PROPOSED DESIGN VERSUS IMPLEMENTED RELEASE

Aspect	Proposed	Implemented
Frontend	React or similar	React 18, Vite, React Router v6
Backend	FastAPI or Node.js	Node.js, Express, 27 endpoints
Database	PostgreSQL or MongoDB	SQLite, optional MySQL
Evaluation	LLM and NLP	Deterministic rules
Voice input	Speech-to-text	Browser Web Speech API
Hosting	Cloud	Local-first, single command
Practice	Aptitude and coding	Five diagnostic assessments

Data model. The records that support the workflow are summarised in Table VII. The model is relational and every record is keyed by the user, so that each metric can be recomputed from the user's own rows. The activity log is deliberately general: it records events such as a resume-match scan, and the readiness calculation counts the relevant event types.

TABLE VII. PRINCIPAL DATA ENTITIES

Entity	Purpose and key attributes
Users and profile	Account, hashed password, college, degree, department, year, CGPA, target role, social links
Skills and interests	Declared skills with proficiency 1–100 and selected technology domains
Study tasks	Week, title, description, category, estimated minutes (default 60) and completed state
Assessment results	User, assessment identifier, score and pass status
Interview history	Question, overall score, pace, STAR adherence, keywords found and feedback text
Activity log	Time-stamped events such as task toggles and resume scans
Groups and members	Domain cohorts, membership and discussion posts
Notifications	Alerts with read and is_read fields

Design principles. Five principles guided the implementation. First, metrics are computed, not stored: a score is derived from records whenever it is requested. Second, rules are deterministic and inspectable, so the same input gives the same output. Third, the system is local-first, which keeps cost low and data under the operator's control. Fourth, claims are conservative: a score is described as guidance and its limits are stated. Fifth, upgrades must not destroy data, which is why schema changes are applied through checked, additive migrations.

Illustrative student journey. A final-year student opens the landing page and creates an account through the five onboarding steps, selecting Software Engineer as the target role and declaring skills in Java, SQL and JavaScript. The dashboard shows a low readiness score and the first eight roadmap tasks. During the first week the student completes the DSA diagnostic, which reveals weak areas, and then reads the planner's instructions for the next tasks. The student pastes a

resume and a job description into the resume tool, notes the missing keywords, revises the resume and scans it again. A practice interview follows: the feedback reports few STAR markers and several fillers, so the student rehearses and records a second answer with a higher score. Meanwhile the student joins a domain study group and bookmarks several companies whose match score is high. Each of these actions appears in the dashboard timeline, and the readiness score increases step by step. The journey is illustrative, but every step corresponds to an endpoint in Table XI.

Validation and error handling. The server validates registration fields and request bodies before use, returns structured JSON error messages, and uses distinct 401 messages for authentication failures. Frontend pages show loading, empty and error states so that a failed request does not leave the student with a blank view. The health endpoint allows an operator to confirm quickly that the server and the database are available. Database writes that depend on one another, such as the saving of an interview result and the creation of its notification, are performed in the same request so that the student does not see a score without its feedback.

IV. SYSTEM IMPLEMENTATION

The application follows a three-tier browser, API and database architecture. The frontend manages registration, navigation, assessments, interviews and dashboards; the backend owns authentication, scoring, validation and persistence; and SQLite stores the records. The frontend communicates with the backend through credentialed REST requests and never opens the database directly. The full system runs on a single machine without external cloud services. The application has not been deployed publicly, and public hosting is outside the completed scope.

Frontend stack. The web application is built with React 18 and Vite. React Router v6 provides protected routing, so that feature pages are reachable only after authentication. The Context API holds authentication state and real-time notification badge counts. Styling uses plain CSS and Tailwind-style utilities with unified dark and light theme tokens. During development, Vite serves the interface on port 3000 and forwards requests beginning with /api to the backend on port 5000 through a reverse-proxy rule, which avoids cross-origin configuration.

Backend stack. The backend is a Node.js and Express server. It exposes 27 REST endpoints, validates a JSON Web Token held in an HTTP-only cookie on protected routes, accepts a Bearer header as a fallback, executes synchronous SQLite queries with prepared statements through better-sqlite3, and applies rate limiting. A health endpoint reports the server status, the state of the primary SQLite database and the optional MySQL connection.

System requirements. Table VIII gives the indicative requirements for running the platform. They are typical for a Node.js and React application rather than measured minimums for this release.

TABLE VIII. INDICATIVE SYSTEM REQUIREMENTS

Item	Requirement
Runtime	A current Node.js long-term-support release with npm
Browser	A recent desktop Chromium-based browser for speech recognition (webkitSpeechRecognition)
Hardware	A computer with a working microphone for the interview module
Storage	Local disk for the SQLite database file
Optional	MySQL server (for example XAMPP) for the second storage layer
Network	None required for operation; localhost only

Project structure. Table IX lists the principal files. The root start-all.js script supervises both processes so that a single command launches Vite and Express together. The database layer, careerprep-db.js, holds the schema, the automatic migrations and the scoring algorithms, while db.js supports the dual-layer SQLite and optional MySQL connection.

TABLE IX. PRINCIPAL PROJECT FILES

File or folder	Responsibility
start-all.js	Launches Vite and Express concurrently
package.json	Root scripts for development and installation
server/careerprep-server.js	Express API with 27 REST endpoints, authentication middleware
server/careerprep-db.js	SQLite schema, auto-migrations, scoring algorithms
server/db.js	Dual-layer database, optional MySQL (XAMPP)
server/careerprep.db	SQLite production data file
server/middleware/auth.js	JWT validation and session extraction
server/data/companies.json	Curated recruiter data: roles, packages, cut-offs
client/vite.config.js	Vite configuration with /api proxy to port 5000
client/src/App.jsx	Master routes and ProtectedRoute encapsulation
client/src/services/api.js	Unified fetch wrapper with credentials
client/src/context/AuthContext.jsx	Global auth state, session check, unread counters
client/src/pages/	Twelve integrated feature pages

Database layer. SQLite stores user accounts, profiles and skills, study tasks, assessment results, interview history, the activity log, peer groups and notifications. The readiness calculation is implemented as a function that receives a user identifier and returns the total together with a five-part breakdown, so that the dashboard can display each component against its maximum. Listing 1 shows the structure of this function in abridged form.

Listing 1. calculateReadinessScore(userId), abridged

```
function calculateReadinessScore(userId) {
  // 1. Study tasks: 30% weight
  const tasks = db.prepare(`SELECT COUNT(*) total,
    SUM(CASE WHEN completed=1 THEN 1 ELSE 0 END)
    completed FROM study_tasks WHERE user_id=?`)
    .get(userId);
  const tasksScore = tasks.total > 0 ?
    Math.round(tasks.completed / tasks.total * 30) : 0;
  // 2. Assessments: 30% weight (5 tests)
  const doneRow = db.prepare(`SELECT COUNT(DISTINCT
    assessment_id) cnt FROM assessment_results
    WHERE user_id=?`).get(userId);
  const done = doneRow?.cnt || 0;
  const assessmentsScore =
    Math.min(30, Math.round(done / 5 * 30));
  // 3. Mock HR interviews: 20 points
  const interviewsScore = Math.min(20, interviewCount *
10);
  // 4. Resume matcher scans: 10 points
  // 5. Peer collaboration: 10 points
  const total = Math.min(100, tasksScore +
    assessmentsScore + interviewsScore +
    resumeScore + studyGroupScore);
  return { score: total, breakdown: [ ... ] };
}
```

Interview evaluator. The evaluation endpoint, POST /api/interview/evaluate, is protected by the authentication middleware. It lower-cases the transcript, counts STAR markers, computes pace and filler words, derives the communication and STAR scores and the weighted overall score, saves the result to the interview history, pushes a notification and returns the evaluation to the client. Listing 2 gives an abridged view.

Listing 2. POST /api/interview/evaluate, abridged

```
app.post('/api/interview/evaluate', requireAuth,
  (req, res) => {
  const { questionId, transcript, durationSeconds }
    = req.body;
  const text = transcript.toLowerCase();
  const starSignals = ['situation','task','action',
    'result','impact','solved','led','metrics'];
  const starCount = starSignals
    .filter(s => text.includes(s)).length;
  const starAdherence = Math.min(95,
    Math.max(40, 40 + starCount * 8));
  const wordCount = text.split(/\s+/)
    .filter(Boolean).length;
  const clarityWpm = durationSeconds > 0 ?
    Math.min(200, Math.max(60, Math.round(
      wordCount / (durationSeconds / 60)))) : 120;
  const fillerWords = (text.match(
    /\b(um|uh|like|basically|you know)\b/gi)
    || []).length;
  const communicationScore = Math.min(100,
    Math.round(80 - fillerWords * 2));
  const overallScore = Math.round(
    communicationScore * 0.4 + starAdherence * 0.6);
  // save to interview history; push notification
  res.json({ success: true, evaluation: {
    overallScore, clarityWpm, starAdherence,
    fillerWords } });
});
```

Authentication guard. The requireAuth middleware reads the token from the auth_token cookie and, if absent, from a Bearer authorization header. It verifies the token with the server secret, loads the user, builds a safe response object from the full profile, and attaches it to the request. Failure produces a 401 response with a specific message. Listing 3 shows the logic.

Listing 3. requireAuth middleware, abridged

```
function requireAuth(req, res, next) {
  let token = req.cookies?.auth_token;
  if (!token && req.headers.authorization
      ?.startsWith('Bearer '))
    token = req.headers.authorization.split(' ')[1];
  if (!token) return res.status(401)
    .json({ error: 'Authentication required.' });
  try {
    const decoded = jwt.verify(token, JWT_SECRET);
    const user = db.findUserById(decoded.userId);
    if (!user) return res.status(401).json({
      error: 'User not found. Please log in again.'});
    req.user = buildUserResponse(
      db.getFullProfile(decoded.userId));
    return next();
  } catch (err) {
    const msg = err.name === 'TokenExpiredError'
      ? 'Session expired.' : 'Invalid token.';
    return res.status(401).json({ error: msg });
  }
}
```

Schema auto-migration. Existing databases must keep working when the schema grows. At start-up the database layer lists a set of expected columns and, for each, inspects the table with PRAGMA table_info. If the column is missing, it is added with ALTER TABLE; if it exists, nothing changes. For example, the study_tasks table gains a description text column and an estimated_minutes integer column with a default of 60. Errors for columns that already exist are caught and ignored. This makes upgrades non-destructive for data already recorded.

Listing 4. Safe ALTER TABLE migration, abridged

```
const migrations = [
  { table: 'study_tasks', column: 'description',
    sql: 'ALTER TABLE study_tasks ADD COLUMN
          description TEXT' },
  { table: 'study_tasks', column: 'estimated_minutes',
    sql: 'ALTER TABLE study_tasks ADD COLUMN
          estimated_minutes INTEGER DEFAULT 60' } ];
for (const m of migrations) {
  const cols = db.prepare(
    `PRAGMA table_info(${m.table})`).all();
  if (!cols.some(c => c.name === m.column)) {
    db.exec(m.sql); /* log: migration applied */ }
}
```

REST interface. Table X lists the core endpoints. Public routes are limited to signup, login and the health check; every other route requires a valid session. Responses are JSON, and error responses carry a message that the frontend displays.

TABLE X. CORE REST ENDPOINTS

Method	Endpoint	Auth	Description
POST	/api/auth/signup	Public	Five-step registration, bcrypt hash, sets cookie
POST	/api/auth/login	Public	Verifies password, issues JWT cookie
GET	/api/auth/me	Yes	Validates session; returns user, profile, skills
POST	/api/auth/logout	Yes	Clears auth_token cookie
GET	/api/dashboard	Yes	Readiness, five priority tasks, activity, timeline
GET	/api/readiness/score	Yes	Score 0–100 with five-category breakdown
GET	/api/assessments	Yes	Five tests with pass status
POST	/api/assessments/:id/submit	Yes	Scores answers, saves result, notifies
GET	/api/planner	Yes	Four-week roadmap; generated on first visit
PUT	/api/planner/tasks/:id/toggle	Yes	Toggles task, logs activity
POST	/api/interview/evaluate	Yes	STAR, pace and filler evaluation
POST	/api/resume/analyze	Yes	Resume versus job description, ATS score
GET	/api/companies	Yes	Recruiter directory, packages, bookmarks
GET	/api/peers/groups	Yes	Study cohorts, member counts, membership
GET	/api/notifications	Yes	Alerts with read and is_read fields
GET	/api/health	Public	Server, SQLite and optional MySQL status

The server exposes 27 endpoints in total; the table lists the core set.

TABLE XI. MODULE-TO-ENDPOINT MAPPING

Module	Endpoints used
Onboarding and login	/api/auth/signup, /api/auth/login, /api/auth/logout
Session check	/api/auth/me on every application load
Dashboard	/api/dashboard, /api/readiness/score
Assessments	/api/assessments, /api/assessments/:id/submit
Study planner	/api/planner, /api/planner/tasks/:id/toggle
Interview simulator	/api/interview/evaluate
Resume matcher	/api/resume/analyze
Companies	/api/companies
Peer groups	/api/peers/groups
Notifications	/api/notifications
Operations	/api/health

User interface modules. The application contains twelve feature pages. The dashboard presents the profile summary, readiness score, priority tasks and recent activity. The profile page supports viewing and editing. The assessment page lists the

five tests and shows pass status. The interview page captures speech and displays the evaluation. The resume page accepts a resume and a job description and displays the match. The planner shows the four-week task list with checkboxes. The companies page lists recruiters with match scores and bookmarks. The peer-groups page supports joining, posting and arranging meetings. A notification area shows unread counts in the navigation through the authentication context. Reusable controls handle loading, empty and error states, and all pages follow the same theme tokens.

Session handling in the client. The authentication context checks the session when the application loads by calling the me endpoint with credentials, so a page reload keeps the student signed in when the cookie is valid. The ProtectedRoute wrapper redirects unauthenticated visitors to the login page. The shared API service attaches credentials to every request and converts error responses into messages that the interface can show.

Running the system. The platform starts with a single command from the project root, node start-all.js. The command starts the backend on port 5000 and the frontend on port 3000. The browser interface is available at http://localhost:3000, the backend health check at http://localhost:5000/api/health, and the API base at http://localhost:5000/api. No cloud account is required, and the SQLite database file is created and migrated automatically.

Optional MySQL layer. A second storage layer allows the application to connect to a MySQL server, such as one provided by XAMPP. The health endpoint reports whether this connection is available. The primary data path remains SQLite, so the absence of MySQL does not affect normal operation.

Configuration and rate limiting. The token signing secret, ports and database settings are held in server configuration, so that they can be changed without editing application logic. Rate limiting is applied to the API to slow repeated or automated requests, which reduces the effect of password-guessing and accidental request loops. The limits are a protective default for a local deployment; a public service would need limits tuned to its traffic and an additional network-level defence.

Theming and responsiveness. A set of design tokens defines colours, spacing and typography for the dark and light themes, and pages use flexible layouts so that the same modules are usable on a laptop screen or a smaller display. Because the speech module depends on a desktop browser's recogniser, voice practice is intended for a laptop or desktop with a microphone.

Notification flow. Several actions create a notification as a side effect: an assessment submission, an interview evaluation and a task toggle that changes the readiness score. The authentication context holds the unread count so the navigation badge stays current, and the notifications endpoint returns both read and is_read fields so that older components keep working while newer ones use the preferred field.

Response shapes. The client relies on stable response shapes. The readiness endpoint returns the total together with a five-item breakdown, and the interview endpoint returns an evaluation object. Listings 5 and 6 show the shape of each response with illustrative values that follow the formulas of Section III; they are examples, not captured output.

Listing 5. GET /api/readiness/score, illustrative

```
{ "score": 43,
  "breakdown": [
    { "key": "tasks",       "label": "Study Tasks",
      "score": 11, "max": 30 },
    { "key": "assessments", "label": "Assessments",
      "score": 12, "max": 30 },
    { "key": "interviews",  "label": "AI HR Mock",
      "score": 10, "max": 20 },
    { "key": "resume",      "label": "Resume Matcher",
      "score": 10, "max": 10 },
    { "key": "groups",      "label": "Study Groups",
      "score": 0,  "max": 10 } ] }
```

Listing 6. POST /api/interview/evaluate, illustrative

```
// request body
{ "questionId": 3, "durationSeconds": 60,
  "transcript": "..." }
// response (4 STAR markers, 3 fillers)
{ "success": true,
  "evaluation": { "overallScore": 73,
    "clarityWpm": 120, "starAdherence": 72,
    "fillerWords": 3 } }
```

Security controls. The controls described in the previous paragraphs can be read as responses to specific threats. The table below summarises the mapping. The table also records two items that remain open: cookie-based sessions should be reviewed for cross-site request forgery protection, for example through the SameSite cookie attribute and request tokens, and secrets should be supplied through protected configuration rather than source files.

TABLE XII. THREATS AND CONTROLS

Threat	Control in the release	Status
Password disclosure	bcrypt salted hashing, ten rounds	Implemented
Token theft by scripts	HTTP-only cookie	Implemented
SQL injection	Prepared statements	Implemented
Password guessing	Rate limiting	Implemented
Unauthorised API use	requireAuth on protected routes	Implemented
Cross-site request forgery	Not described in the records	To be reviewed
Secret exposure	Server configuration	To be reviewed

Operation checklist. An operator who deploys the platform for a class or laboratory can follow a short checklist, given in Table XIII. The items follow from the architecture and from the open items in the security table, and they are intended to make a demonstration reproducible and a pilot safer.

TABLE XIII. OPERATION CHECKLIST

No.	Action
1	Install Node.js and the project dependencies from the root directory
2	Set a strong, private token-signing secret in server configuration
3	Start both processes with the single launch command
4	Open the health endpoint and confirm the SQLite status
5	Create a test account and complete the five onboarding steps
6	Verify that the planner generates eight tasks on first access
7	Back up the SQLite file before upgrading or deleting data
8	Review cookie attributes and request protection before wider use

Quality tooling. The project is validated by an end-to-end integration test that exercises the API as a client would, described in Section VI. The frontend can be checked with the standard build and lint tooling of the Vite and React ecosystem. A broader automated regression suite, browser-level tests and load testing are not part of the current release and are listed under future work.

Figure 5 shows how the principal records relate to one another and to the derived readiness score, and Figure 7 summarises the verification boundaries discussed in Section VI.

Figure 5 Data relationships
Stored records and the derived readiness score



V. ADVANTAGES

1. Personalised and Measurable Preparation

Computed metrics: Every score is derived from the student's own records. A new account shows a low readiness value, and the value rises only when tasks are completed, assessments are taken, interviews are held, a resume is scanned or a peer group is joined. This avoids the false reassurance of static demonstration values.

Role-aware guidance: The planner and the company match use the target role and the declared skills, so two students with different goals receive different tasks. The skill profile is

also checked against assessment results rather than relying only on self-declared proficiency.

2. Transparent Feedback

Inspectable rules: Each score has a stated formula. A student who receives a low interview score can see that the cause is a lack of STAR vocabulary or an excess of filler words, and can correct it in the next attempt. Rule-based feedback is repeatable: the same answer produces the same score.

Honest limits: The platform presents scores as guidance. The interview feedback is based on vocabulary and pace, and the resume match is text based. Stating these limits protects the student from over-trusting a heuristic.

3. Low Cost and Local Operation

No mandatory cloud service: The core system runs on one machine with Node.js and an embedded database, so it can be deployed in a college laboratory without subscription fees or an external API key. This responds directly to the cost and access barriers that limit career guidance outside large cities.

Simple operation: A single command starts the frontend and the backend. Schema migrations are applied automatically and without data loss, which lets the system evolve between releases.

4. Secure Session Handling

Established mechanisms: Passwords are salted and hashed with bcrypt, tokens are signed and held in HTTP-only cookies, SQL statements are prepared and requests are rate limited. The middleware gives distinct responses for missing, expired and invalid sessions, which simplifies client handling.

Scoped access: Every protected endpoint relies on the authenticated user identity, so a student's tasks, assessments, interviews and notifications are read and written under that identity.

5. Integrated Workflow

One loop: Profile, assessment, planner, interview, resume match and peer activity all feed the same readiness measure and the same notification stream. The student does not have to reconcile separate tools to understand progress.

Extensible design: The scoring logic is isolated in the database layer and the evaluator is isolated behind a single endpoint, so a model-based evaluator or additional assessment banks can be added without redesigning the application.

6. Learning Value

Immediate, repeatable feedback: A student can attempt the same interview question several times and see how each change in wording or pace alters the score, which supports deliberate practice. Because the readiness score is visible on the dashboard, progress becomes a motivating signal and not a vague impression.

Transferable habits: Practising STAR-structured answers, quantifying achievements on a resume and working through a weekly plan are habits that remain useful after the platform is no longer needed.

VI. RESULTS AND ANALYSIS

The local release (version 1.0.0, October 2026) [15] was validated by an end-to-end integration test run against the running application. The procedure exercised the API in the order a student would use it, and it completed successfully. Table XIV lists the steps and the observed outcome of each.

TABLE XIV. END-TO-END VERIFICATION PROCEDURE

No.	Step	Observed outcome
1	Create a student account through /api/auth/signup	Account created
2	Retrieve the session through /api/auth/me with the HTTP-only cookie	Session persisted and retrieved
3	Open the dashboard and the planner for the first time	Eight personalised tasks generated
4	Submit the DSA diagnostic assessment	Real-time score change from 54% to 74%
5	Remove the test data	Clean removal with foreign-key cascade consistency

The test establishes that the principal path from registration to scored assessment works as designed, that the cookie-based session survives a

retrieval request, that the planner generates a plan on first access, and that deleting a user removes dependent records without leaving orphans. It is a functional check of one scenario, not a statistical evaluation.

Table XV maps each of the five objectives stated in Section I to the module that addresses it and to the evidence available in this report. The mapping states what has been demonstrated and what has not.

TABLE XV. OBJECTIVE FULFILMENT

Objective	Module	Evidence and status
1 Resume analysis	Resume matcher	Endpoint implemented; matching quality not validated
2 Mock interviews	Interview simulator	Formulas verified analytically; no expert comparison
3 Skill gap	Profile, assessments	Gap view implemented; thresholds not evaluated
4 Learning plan	Study planner	Eight tasks generated on first access (tested)
5 Progress tracking	Readiness, dashboard	Score updates on activity (tested and derived)

Analytical verification of the scoring formulas complements the integration test. The readiness formula contains only counts, ratios and caps, so its behaviour can be derived exactly. Table XVI gives the marginal contribution of each student action under the implemented weights.

TABLE XVI. MARGINAL CONTRIBUTION OF EACH ACTION

Action	Points	Condition
Complete one of eight tasks	+3.75	Before rounding
Take one new assessment	+6	Up to five tests
First mock interview	+10	Interview cap 20
Second mock interview	+10	Reaches the cap
Third or later interview	0	Capped
First resume scan	+10	One-time
Join first peer group	+10	One-time

Three observations follow from this analysis. First, the readiness score responds immediately and predictably to each action, which supports the objective of visible progress. Second, 40 of the 100 points can be obtained by two interviews, one resume scan and one group membership, without completing any study task or assessment. The score therefore rewards participation and not proficiency, and a high value should not be read as evidence of mastery. Third, the interview and resume components are saturated by a small amount of activity, so beyond that point the score cannot show improvement in their quality. A future revision could weight these components by the quality of the result.

The interview evaluator can be examined in the same way. With the STAR formula, each additional marker word adds eight points until the ceiling of 95 is reached at seven markers. Each filler word reduces the communication score by two points, so the largest swing that fillers can cause in the overall score is $0.4 \times 2 = 0.8$ points per filler. Structure therefore dominates the overall mark, in line with the 0.6 weight. A short answer that names situation, task, action and result without any fillers obtains $S = 72$ and $K = 80$, giving $O = \text{round}(32 + 43.2) = 75$.

TABLE XVII. STAR SCORE BY NUMBER OF MARKERS

Markers found	STAR score	Interpretation
0	40	No STAR vocabulary detected
2	56	Partial structure
4	72	Situation to result covered
6	88	Strong structure
7 or more	95	Ceiling reached

The marker detection uses substring matching on a lower-cased transcript. As noted in Section III, a marker embedded in a longer word may be counted, and a well-structured answer using other vocabulary may not be recognised. These behaviours are inherent to the method and are a known limitation of the current evaluator.

The effect of filler words can be tabulated in the same way. Because the communication score starts at 80 and falls by two points per filler, an answer with ten fillers obtains $K = 60$, which lowers the overall score by 8 points compared with a filler-free answer when STAR adherence is unchanged. The effect is real but secondary to structure, which is consistent with the intention that the evaluator should first teach students to organise their answers.

TABLE XVIII. EFFECT OF FILLER WORDS ON THE OVERALL SCORE

Filler words	Comm. score K	Overall O (S = 72)
0	80	75
3	74	73
5	70	71
10	60	67
20	40	59

O = round(0.4K + 0.6 × 72), with S = 72 (four STAR markers).

The speaking-pace measurement is likewise simple. A 90-second answer of 180 words corresponds to 120 WPM, the lower edge of the recommended range, while a 60-second answer of 220 words corresponds to 220 WPM, which the implementation clamps to 200. Clamping prevents a recognition error or a very short recording from producing an implausible pace, but it also hides how fast the speaker really was.

To show how the components combine over time, Figure 6 plots the readiness score of a hypothetical student over a four-week plan. The student scans a resume in week 1, holds the first interview and joins a group in week 2, and finishes all eight tasks and five assessments by week 4. The values follow directly from Equation (1) and are not measurements of real users. The chart shows the expected shape of the score: it grows in steps whenever an activity is recorded, and the final increase depends on the two components that have not yet reached their ceilings.

Figure 6 Illustrative readiness progression
Hypothetical student; component contributions per week



The same derivation yields a set of expected values that can serve as regression tests for the scoring code. Table XIX lists twelve cases, each of which can be checked by calling the scoring function or the evaluation endpoint with the stated inputs and comparing the output with the derived value. The cases include boundary conditions such as an empty profile, a capped component, a missing duration and a clamped pace.

TABLE XIX. DERIVED TEST CASES FOR THE SCORING LOGIC

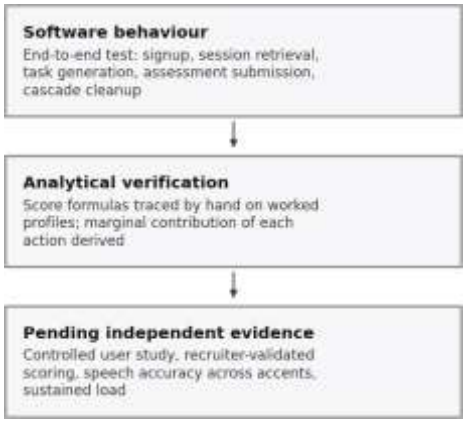
No.	Input	Expected output
1	No activity recorded	R = 0
2	8 of 8 tasks, nothing else	R = 30
3	All 5 assessments, nothing else	R = 30
4	3 interviews, nothing else	R = 20 (capped)
5	All components at maximum	R = 100
6	Student A of Table III	R = 43
7	Student B of Table III	R = 87
8	0 markers, 0 fillers	S = 40, K = 80, O = 56
9	7 markers, 0 fillers	S = 95, K = 80, O = 89
10	4 markers, 3 fillers	S = 72, K = 74, O = 73
11	Duration 0 seconds	Pace = 120 WPM
12	400 words in 60 s; 5 words in 60 s	Pace = 200; Pace = 60

Threats to validity should be stated plainly. Regarding construct validity, the readiness score measures recorded activity, whereas the underlying construct is placement readiness; the two are related but not identical. Regarding internal validity, the integration test used a single synthetic account, so interactions between many users or long histories were not exercised. Regarding external validity, the results come from one local release on one environment and cannot be generalised to other browsers, operating systems or institutions. Finally, because the weights of the readiness score were chosen by the developers and not estimated from outcome data, other weightings might be equally defensible.

Several matters are not measured in this report. No controlled study with students has compared readiness scores with actual placement outcomes. The agreement between the interview score and the judgement of experienced interviewers has not been assessed. The accuracy of browser speech recognition for different accents, microphones and noise levels has not been quantified. The resume match score has not been validated against real ATS software or recruiter decisions. No sustained-load or concurrent-user test has been conducted, and the performance of the

synchronous database layer under many simultaneous users is unknown. Figure 7 summarises the boundary between what was verified and what remains pending.

Figure 7 Verification boundaries
Verified behaviour and pending independent evidence



Overall, the results support the correctness of the local workflow: registration, session handling, plan generation, assessment scoring and cleanup operate together, and the scoring rules behave exactly as their formulas state. The results do not establish predictive validity, user outcomes or readiness for public deployment.

Discussion. Three themes emerge from the analysis. The first is transparency. Because each formula is explicit, every number shown to a student can be explained and, if necessary, challenged, and the same transparency allows the developers to find design weaknesses such as the saturation of the interview and resume components. The second is the balance between activity and quality. The present score rewards doing things, which is appropriate for a platform whose first goal is to help students start and continue preparing; it is less appropriate as a measure of competence, and future versions should add quality-sensitive terms, for example by using the assessment percentage and the interview scores and not only their counts. The third is the value of integration. Placing the resume match, interviews, assessments and peer activity in one record allows the dashboard to answer the question that students most often ask, namely what should be done next, in a way that separate tools cannot.

Lessons from implementation. Several practical lessons were drawn from building the platform. Synchronous database access made the code straightforward and removed a class of concurrency errors, but it ties the design to a single-process deployment. Storing derived values only when necessary kept the data consistent, but it means each dashboard request performs several count queries, which is acceptable at classroom scale and should be measured before scaling. Additive migrations allowed the schema to evolve during development without asking users to delete their data. Finally, the decision to use the browser's own speech recogniser reduced cost and complexity but imposed a dependence on the browser that the interface must communicate clearly.

VII. LIMITATIONS, RISKS AND ETHICAL CONSIDERATIONS

Honest reporting of limitations is part of the design. The principal risks are collected in Table XX together with the mitigation that the current release provides and the further action that is needed.

TABLE XX. LIMITATIONS AND MITIGATIONS

Limitation	Current mitigation	Further action
Feedback may differ from recruiters	Scores framed as guidance	Expert comparison study
Quality depends on input data	Server-side validation	Input-quality warnings
Voice accuracy varies by audio and accent	Transcript used as the unit of evaluation	Accent and noise testing
Lexical STAR detection	Rules documented and visible	Word-boundary and semantic checks
Score rewards activity	Weights stated openly	Quality-aware weighting
Personal data sensitivity	Local storage, hashed passwords	Encryption at rest, data review

Several ethical points deserve attention. A numerical score can influence a student's confidence, and a low value may discourage rather than motivate; the interface should therefore pair every score with a next action. Automated feedback must not be presented as an assessment

of a person's worth or ability. Resume and interview heuristics may disadvantage candidates whose phrasing differs from the vocabulary the rules expect, including speakers of different regional English varieties, so the results should be used by students for self-improvement and not by recruiters for screening. Finally, stored resumes, academic records and interview transcripts are personal data, and any institution that deploys the platform is responsible for lawful, minimal and secure handling of them.

VIII. CONCLUSION

CareerPrep AI implements a local-first placement-preparation platform that brings resume matching, interview practice, diagnostic assessment, skill profiling, study planning and progress tracking into one workflow. Its principal objective is not to produce more advice but to connect each piece of preparation activity to a measurable, inspectable record. Every score is computed from stored user activity, and every formula can be read and reproduced.

The architecture combines React 18 and Vite with Node.js, Express and embedded SQLite. These are established technologies; the contribution lies in their orchestration around a weighted readiness model, a STAR-based speech evaluator, a role-aware planner and a recruiter and peer layer, delivered in a form that can run on a single machine without a paid service.

The integration test shows that the central workflow works, and the analytical review shows how the scores respond to student actions and where they saturate. However, the platform's feedback is heuristic. The readiness score measures activity more than competence, the interview evaluator measures vocabulary and pace and not meaning, and the resume matcher measures textual overlap and not suitability. The system should be used as an assistive tool that helps a student decide what to do next, with responsibility for consequential decisions remaining with the student and with human mentors.

The strongest practical features are the personalised loop from profile to plan to practice to updated score, and the openness of the rules. Further evaluation with real students, expert-rated interviews and a larger test suite would turn these promising design properties into evidence.

IX. FUTURE WORK

The first priority is evaluation. A study with consenting students should compare readiness scores with later placement outcomes, measure whether repeated practice improves interview scores, and gather qualitative feedback on usefulness. Interview scores should be compared against ratings from experienced interviewers, and the resume matcher against recruiter judgements and real ATS parsing.

The proposal's longer-term scope remains valuable. Live job recommendations and application tracking would connect preparation to opportunities. Company-specific interview preparation would draw on the curated recruiter data to tailor questions and cut-off guidance. Multilingual support, including regional Indian languages, would extend access to students who are more comfortable outside English. Video interview analysis of delivery and body language, and mentor and peer community features, would broaden the feedback available.

On the technical side, a language model could be added behind the existing evaluation endpoint to assess the meaning of answers, generate role-specific questions and explain resume suggestions. Such a component should be introduced with retrieval from verified material [14], should label its output as model generated, and should be evaluated separately from the rule-based scores. Natural-language processing libraries could improve skill extraction from resumes. Word-boundary matching would address the substring weakness of the STAR detector, and quality-aware weighting would address the saturation of the interview and resume components.

For operation, an automated regression suite, browser-level tests across screen sizes, and load tests with many concurrent users are needed before wider use. Encryption at rest, secret management, audit logging and a data-protection review would be required for any service holding real student data, and an independent security review would be needed before public hosting. Future work should preserve the principle of this project: a score should be shown only when the activity that produced it has been recorded.

TABLE XXI. PROPOSED DEVELOPMENT ROADMAP

Phase	Work item	Dependency
Near term	Automated tests for scoring (Table XIX)	Test framework
Near term	Word-boundary STAR detection	Evaluator change
Near term	CSRF and secret-handling review	Security review
Mid term	User study with students	Ethics approval, consent
Mid term	Quality-aware score weighting	Study data
Long term	LLM-assisted evaluation with retrieval	Evaluation plan
Long term	Regional-language and video support	Datasets, models

REFERENCES

[1] T. Janz, L. Hellervik and D. C. Gilmore, Behavior Description Interviewing: New, Accurate, Cost Effective. Boston, MA: Allyn and Bacon, 1986.

[2] N. Provos and D. Mazières, “A Future-Adaptable Password Scheme,” in Proc. USENIX Annual Technical Conference, FREENIX Track, 1999, pp. 81–91.

[3] M. Jones, J. Bradley and N. Sakimura, “JSON Web Token (JWT),” RFC 7519, Internet Engineering Task Force, May 2015. https://www.rfc-editor.org/rfc/rfc7519

[4] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Ph.D. dissertation, University of California, Irvine, 2000.

[5] Web Speech API, W3C Community Group draft specification. https://webaudio.github.io/web-speech-api/

[6] MDN Web Docs, SpeechRecognition interface. Mozilla. https://developer.mozilla.org/en-US/docs/Web/API/SpeechRecognition

[7] OWASP Foundation, Password Storage Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

[8] SQLite Consortium, SQLite Documentation. https://sqlite.org/docs.html

[9] better-sqlite3 contributors, better-sqlite3 library documentation. https://github.com/WiseLibs/better-sqlite3

[10] Meta Open Source, React Documentation. https://react.dev/

[11] Vite contributors, Vite Documentation. https://vite.dev/

[12] Express contributors, Express Documentation. https://expressjs.com/

[13] OpenJS Foundation, Node.js Documentation. https://nodejs.org/docs/latest/api/

[14] P. Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 9459–9474. https://arxiv.org/abs/2005.11401

[15] CareerPrep AI project records, Executive Summary and System Architecture; Core Functional Modules and Logic; Important Coding Implementations; Complete REST API Specification; Execution and Verification Guide. Local Platform Release v1.0.0, October 2026.