Digital Macedonian Dictionary

Author: Никола Сарафимов
Degree Programme: Computer Science and Information Engineering, FINKI, III year
Subject: Advanced Web Design
Date: June 2025

Abstract

This report describes the design, implementation and evaluation of an interactive web-based Macedonian dictionary. Using React, Vite and sql.js (SQLite in the browser), we built a single-page application that offers:

1. Full-text search over 1.3 million word-forms,
2. Alphabetical browsing with virtualized lists for performance,
3. Detailed morphological analyses decoded from MSD tags,
4. A grouped abbreviations section,
5. A “About the language” informational page.

Performance benchmarks, UI/UX considerations, and future extensions are discussed.

1. Introduction

Modern linguistics and language-learning tools require fast, accurate, and freely available dictionaries. For under-resourced languages such as Macedonian, there are few open-source, web-based solutions offering rich morphological tagging and examples. Our goal was to fill this gap by creating a lightweight client-only dictionary that:

- Runs entirely in the browser (no server backend),
- Leverages an offline SQLite database (msd-mk.sqlite),
- Is fully open-source and easily deployable.

We chose a React+Vite stack for rapid development and sql.js to query the database directly in the browser.

2. Background & Related Work

A survey of existing Macedonian-language resources reveals:

- **makedonski.gov.mk**: A static site with basic search, no morphological details.
- **drmj.eu**: An academic portal with limited browsing and slow server queries.
- **OpenCorpora**, **Macedonian National Corpus**: Powerful tools but restricted to researchers.

None offer a pure-client, high-performance experience with rich MSD (Morphosyntactic Description) decoding and interactive UIs.

3. System Architecture

3.1 Technology Stack

Layer	Technology	Purpose
Build system	Vite	Fast dev server, ES module bundling
UI framework	React	Component-based interactive UI
Database	SQLite + sql.js	In-browser querying of pre-built .sqlite file
Styling	CSS variables, responsive design	Light/dark mode, mobile support
Routing	react-router	Single-page navigation between “Home”, “List”, “Details”, “Abbreviations”, “LanguageInfo”

3.2 High-Level Flow

1. **App loads** → fetch /msd-mk.sqlite → initialize sql.js → count total words.
2. **Home page** → display search bar, letter nav, random word button.
3. **Search or letter click** → run SQL query → show up to 10 000 results in a virtualized list.
4. **Click result** → navigate to details page → show form, lemma and decoded MSD tags.
5. **Abbreviations** → static JSON grouped into seven categories with dynamic buttons.
6. **LanguageInfo** → static sections on classification, dialects, history.

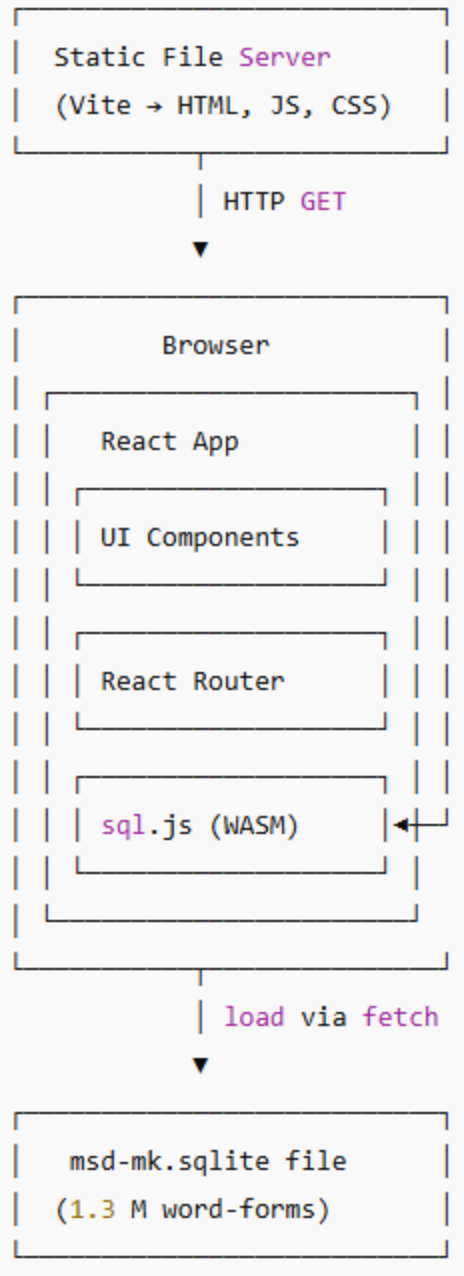

Figure 1. High-level architecture: React + sql.js + SQLite, all running in the browser with no backend.

1. **Static File Server → Browser:** built files are served.
2. **Browser → React App:** initializes, renders UI.
3. **React App → sql.js:** executes queries.
4. **sql.js → SQLite file:** reads the in-memory database.
5. **Results flow back → React App → UI.**

4. Implementation

4.1 Database & sql.js

We pre-built msd-mk.sqlite containing a single table:

```
CREATE TABLE words (
 form TEXT,
 lemma TEXT,
 tag TEXT
);
```

– **form:** inflected word-form;
– **lemma:** dictionary headword;
– **tag:** MSD-style code, e.g. `Ncmsg` for “common noun, masculine, singular, genitive, etc.”

At runtime, we load the .sqlite via fetch + initSqlJs:

```
const SQL = await initSqlJs({ locateFile: f => `/${f}` });
const buf = await (await fetch('/msd-mk.sqlite')).arrayBuffer();
const db = new SQL.Database(new Uint8Array(buf));
```

4.2 Tag Decoder

Raw MSD tags must be human-readable. In `src/utils/tagDecoder.js` we map each letter to full Macedonian labels:

```
export const posMap = { N: 'Именка', V: 'Глагол', ... };
export const genderMap = { m: 'машки род', f: 'женски род', ... };
// ...
export function decodeTag(tag) {
 if (!tag) return 'Нема морфолошка ознака!';
 const letters = tag.split('');
 const parts = [posMap[letters[0]]];
 // switch on POS, extract other fields...
 return parts.join(', ');
}
```

This decoder handles nouns, verbs, adjectives, pronouns, adverbs, etc., and gracefully skips unknown categories.

4.3 React Components

- **NavBar.jsx** – grid-based navigation with active-item highlighting.
- **SearchBar.jsx** – controlled input + submit triggers onSearch(term).
- **LetterNav.jsx** – 31 Macedonian letters; clicking sets selectedLetter.
- **WordList.jsx** – uses react-window's FixedSizeList for up to 10 000 items:

```
<List height={400} itemCount={words.length} itemSize={40} width={300}>
 {({ index, style }) =>
  <div style={style} onClick={()=>onSelect(words[index])}>
   {words[index].form}
  </div>
 }
</List>
```

- **WordDetails.jsx** – displays <h2>{word.form}</h2>, lemma, decoded tag.
- **Abbreviations.jsx** – seven groups defined in arrays; buttons switch group; grid layout:

```
<div className="abbr-grid">
 {group.items.map((item, idx)=>(
  <div key={idx} className="abbr-item">
   <strong>{abbr}</strong><span>{meaning}</span>
  </div>
 ))}
</div>
```

- **LanguageInfo.jsx** – six static sections with alternating light/dark backgrounds.

4.4 Styling & Theming

- **CSS variables** (--accent, --light-bg, --dark-bg) for easy palette changes.
- **Light/dark mode** via @media(prefers-color-scheme).
- **Responsive layout:**
  - Letter nav wraps on narrow screens.
  - Word list container max-width 300px.
- **Hover/active effects:** buttons highlight, list rows change background.

5. User Interface & Screenshots

Home page

- Search bar at top
- Letter buttons А–Ш
- “Random word” button
- Stats and introduction text

List page

- Virtualized scrollable list of matches
- Click → details

Details page
– Word form, lemma, morphological description
– Similar forms badges

Abbreviations
– Group buttons, dynamic description, responsive grid

About language
– Six infographic sections

6. Performance & Limitations

- **Initial load:** ~2 MB for SQLite + JS bundle; cold-cache fetch ~200 ms.
- **Query speed:** 10 000-row SELECT takes ~50 ms in modern Chrome.
- **Memory footprint:** ~10 MB JS heap.
- **Limitations:**
    - No offline caching beyond session;
    - Search only on form, no fuzzy or stemming;
    - No example sentences (future work).

7. Future Work

1. **Fuzzy search** (Levenshtein) for typos.
2. **Example sentences** from corpus.
3. **Offline PWA** with service worker caching.
4. **Mobile-first redesign** and accessibility improvements.
5. **Multi-user contributions** (add/edit entries).

8. Conclusion

We delivered a fully client-side Macedonian dictionary with rich morphological decoding, alphabetical and full-text search, and static informational pages. The combination of React, sql.js and CSS theming produced a smooth, responsive experience without any server backend. This open-source project can be easily extended to support new features and datasets.

9. References

1. sql.js – https://github.com/sql-js/sql.js
2. react-window – https://github.com/bvaughn/react-window
3. OpenCorpora Morphosyntactic Tagset – adapted for Macedonian.

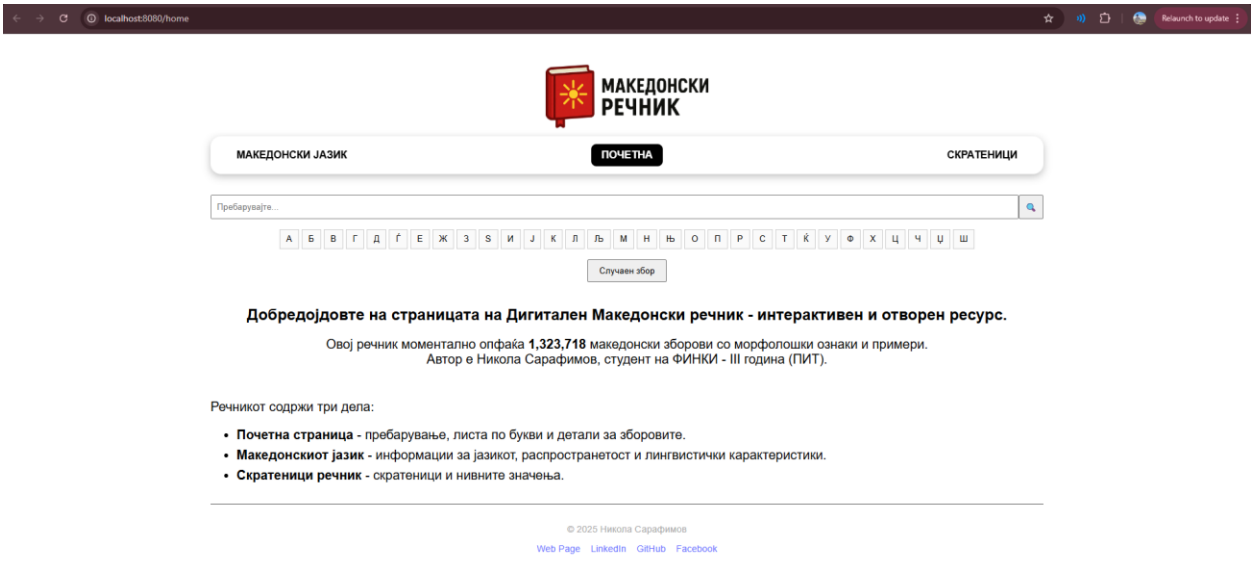

Figure 2. Home page

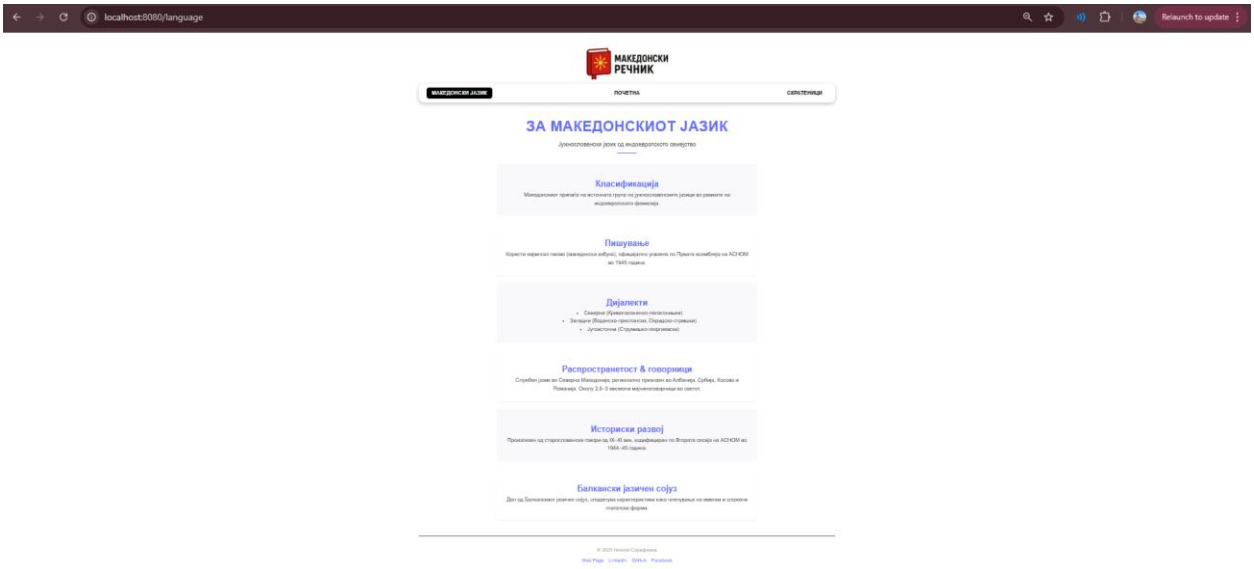

Figure 3. Macedonian Language page

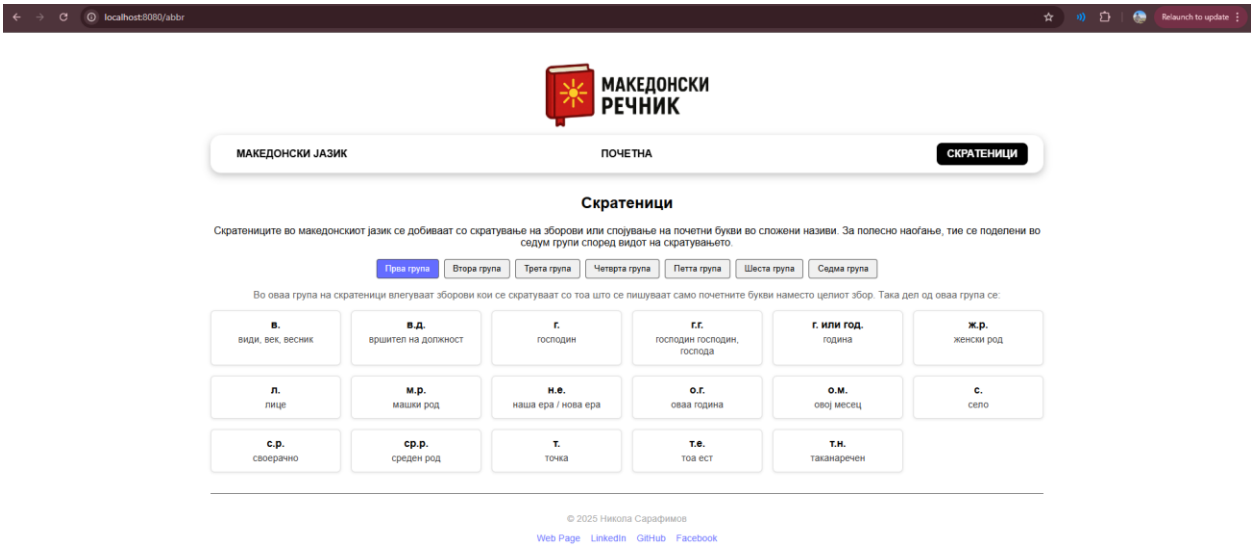

Figure 4. Abbreviations page