Hangul Name Converter (HNC)

Full Stack Documentation

By: Kalden Yugyel Dorji

preface

Hangul Name Converter – HNC for short – was born out of a fascination with a uniquely trending digital phenomenon mainly noticed among Korean users. Many top artists and users alike stylized their usernames by typing their Hangul names using the QWERTY keyboard layout, which resulted in very unique and memorable tags. The result is a string of English characters that spatially mirror their Korean name – a form of linguistic identity that's easily recognizable to native speakers and enthusiasts alike.

One of the most well-known examples is **IU**, whose real name is 이 지은. Her instagram handle – @dlwlrma – is the QWERTY equivalent of typing her name 이 지금. The entire process resembles one of cybersecurity's important concepts: obscurity. This visual encoding process – where Korean characters are typed using their spatial counterparts on an English keyboard layout – mirrors the logic of substitution ciphers in cybersecurity. The original identity is obscured, yet preserved through a consistent mapping. Without the knowledge, decoding can prove to be difficult.

Despite the trend and subtle implications of ciphers, no existing tools were available online. Therefore, this full-stack application powered by a Python backend and a React frontend, serves to fill that gap. The HNC tool takes English usernames as input, performs a translation to retrieve jamo*, and returns the QWERTY mapped username.

---

* jamo – korean alphabet

architecture

To balance a minimalistic design with a flexible and expressive logic layer, React + Tailwind CSS was chosen for the frontend and Python for the backend. React enables a clean, responsive interface with modular component architecture, while Python powers the core linguistic transformations with clarity and precision — making the Hangul Name Converter both intuitive to use and robust behind the scenes.

HNC includes the following components for its core functions:

Frontend: React + Tailwind CSS with Vite enables a modular approach to building sleek user interfaces while providing benefits like HMR* and optimizing production builds.

Backend: FastAPI, a web framework, handles server-side logic – defining routes, processing requests, and executing functions. The server itself is hosted on Uvicorn, an ASGI-compatible** server that runs the FastAPI app and handles HTTP communications.

Core Libraries:

jamo – Decomposes hangul syllables into their base units (individual letters)

googletrans – Automatic translation from ENG to KR

time & logging – Internal use for analysis during runtime

---

* HMR (Hot Module Replacement) – enables instant updates to modules in the browser without reloading

** ASGI (Asynchronous Server Gateway Interface) – a standard allowing servers to call async Python apps efficiently

folder diagram

```
HNC/
├── backend/
│   ├── krList.py
│   ├── krMappings.py
│   ├── main.py
│   ├── nameConvert.py
│   ├── requirements.txt
│   ├── translation.log
│   └── node_modules/
├── public/
│   ├── hnc.svg
│   └── src/
│       ├── assets/
│       │   ├── conversion.css
│       │   ├── conversion.jsx
│       │   ├── mapping.css
│       │   └── mapping.jsx
│       ├── App.css
│       ├── App.jsx
│       ├── index.css
│       └── index.jsx
├── .gitignore
├── eslint.config.js
├── package.json
├── package-lock.json
├── README.md
└── vite.config.js
```

functionality

The conversion workflow connects the React frontend with a FastAPI backend running on Uvicorn to process and deliver Hangul Name Converter (HNC) results efficiently.

The process begins at the frontend input field, which handles basic validation and enforces character limits to ensure clean, valid user input. Once validated, the application sends the data to the backend through a designated API route.

At the backend, FastAPI receives the request and applies the core translation and mapping logic. This includes interpreting the input, performing language conversions, and generating the appropriate transliteration output.

After processing, the backend returns a structured JSON response in the format:
{ "qwerty": <result> } – The React frontend parses this response and updates the output display dynamically, providing immediate visual feedback to the user.

User Input (React)

↓

POST /convert (Fetch API)

↓

FastAPI Endpoint (main.py)

↓

convert_any_name() (nameConvert.py)

↓

1. Check manual overrides (krList.py)
2. Translate via googletrans (if not in overrides)
3. Decompose Hangul using jamo
4. Expand compound vowels (krMappings.py)
5. Map to QWERTY keys (krMappings.py)

↓

Return {"qwerty": result}

↓

Display in Output Box (React)

implementation snippets

handleConvert() connects the React frontend to a FastAPI backend by sending user input as a POST request with a JSON payload. The backend processes the input and responds with the converted data, which React receives and updates the UI states accordingly.

```
 const handleConvert = async () => {
     setLoading(true)
     const response = await
fetch('http://127.0.0.1:8000/convert', {
       method: 'POST',
       headers: { 'Content-Type':
'application/json' },
       body: JSON.stringify({ name: input })
     });
     const data = await response.json();
     setOutput(data.qwerty)
     setLoading(false)
   }
```

Defines a FastAPI POST endpoint at /convert that receives a JSON payload, runs the convert_name function to process the user's input, and returns the result in the "qwerty" field of the HTTP response; this setup enables your frontend to send user data and get the converted output using a simple API call.

```
@app.post("/convert")
async def convert_name(data: NameInput):
    qwerty = await convert_any_name(data.name)
    if(type(qwerty)==str):
        return {"qwerty":qwerty}
    else:
        result = qwerty["qwerty"]
        return {"qwerty": result}
```

Backend script that converts user inputted English names, following proper input sanitization and basic checks, and translates it into Korean for further parsing.

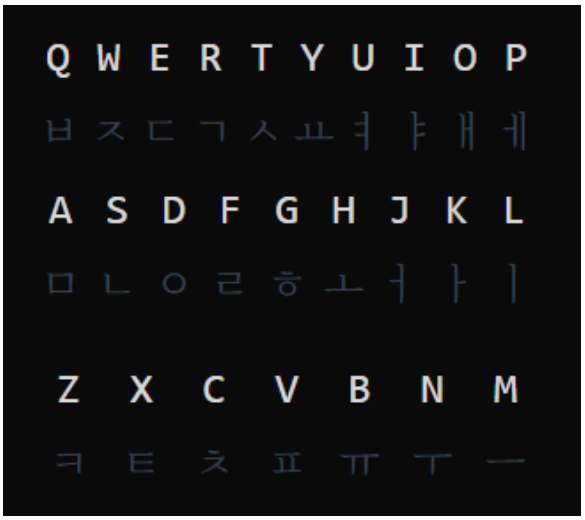
```
async def convert_any_name(english_name):
    if english_name == "":
        return {"qwerty":"Nothing entered"}
    name_lower = english_name.lower().strip()

    if name_lower in NAME_OVERRIDES:
        hangul = NAME_OVERRIDES[name_lower]
    else:
        async with Translator() as translator:
        result = await
translator.translate(english_name, src='en',
dest='ko')
        hangul = result.text
        return {"qwerty: hangul}
```

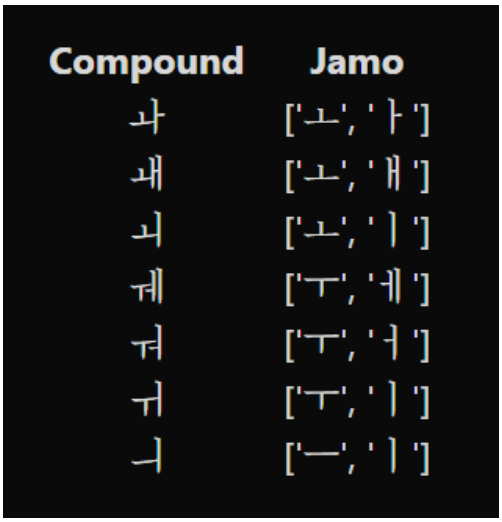
Converted Korean text is further reduced into their Jamo counterparts which is utilized in the final mapping process for output generation.

```
for char in hangul_text:
        if '가' <= char <= '힣':
            decomposed =
list(j2hcj(h2j(char)))
            for jamo in decomposed:
                if jamo in COMPOUND_VOWELS:
jamo_list.extend(COMPOUND_VOWELS[jamo])
                else:
                    jamo_list.append(jamo)
        elif ('ㄱ' <= char <= 'ㅎ') or ('ㅏ' <=
char <= 'ㅣ'):
            if char in COMPOUND_VOWELS:
jamo_list.extend(COMPOUND_VOWELS[char])
            else:
                jamo_list.append(char)
```

mappings

For HNC to function, a pre-defined mapping of Hangul jamo to QWERTY-layout English letters is hard-coded into the system. This comprehensive mapping ensures that, after translating and decomposing Hangul into individual jamo units, each jamo can be efficiently and systematically converted to its corresponding key on an English QWERTY keyboard. By referencing this mapping — visible on the main web application page — users gain full transparency into how Korean text is transformed into accurate English keyboard input. In addition, since standard libraries do not natively decompose compound vowels, a custom dictionary is included to explicitly break down all compound Hangul vowels into their base jamo components. This guarantees that even complex vowels are consistently reduced before final QWERTY conversion. Both mappings are essential for delivering reliable, character-by-character transformation from Hangul to QWERTY English output, with the implemented structure supporting transparency as well.





setup

Prerequisites:

- Python 3.8+
- Node.js 16+
- npm or yarn

Backend command list for setup:

```
1. cd backend
2. python -m venv venv
3. .\venv\Scripts\activate
4. pip install -r .\requirements.txt
5. uvicorn main:app --reload
```

Frontend command list for setup:

```
1. cd //project_directory_name
2. npm install
3. npm run dev
```

future considerations

While the current HNC version accomplishes its core functionalities alongside useful accessibility features, following is a possible roadmap for future changes and additions:

- [x] ~~Loading buffer during translation and mapping~~
- [x] ~~Integration of jamo and googletrans for core translation function~~
- [x] ~~Minimalistic frontend design~~
- [ ] Deploy application to cloud services
- [ ] Save button for QWERTY outputs for future reference
- [ ] Export option of saved queries and outputs
- [ ] Save to clipboard button
- [ ] Reverse conversion from QWERTY to Korean names
- [ ] Mapping of non-translatable English usernames