

# RA 게임 홈페이지 백엔드 완전 가이드

## 📁 프로젝트 구조

```
ra-backend/  
├── src/  
│   ├── config/  
│   │   ├── database.js  
│   │   └── redis.js  
│   ├── controllers/  
│   │   ├── authController.js  
│   │   ├── newsController.js  
│   │   └── communityController.js  
│   ├── services/  
│   │   ├── authService.js  
│   │   ├── newsService.js  
│   │   └── communityService.js  
│   ├── routes/  
│   │   ├── authRoutes.js  
│   │   ├── newsRoutes.js  
│   │   └── communityRoutes.js  
│   ├── middlewares/  
│   │   ├── auth.js  
│   │   ├── validate.js  
│   │   └── errorHandler.js  
│   ├── utils/  
│   │   ├── jwt.js  
│   │   └── logger.js  
│   └── app.js  
├── .env  
├── .gitignore  
├── package.json  
└── server.js
```

## 🚀 1단계: 백엔드 설치

### 1.1 Node.js 설치 확인

```
bash  
  
node --version # v18 이상 필요  
npm --version
```

### 1.2 프로젝트 초기화

```
bash
```

```
# 프로젝트 폴더 생성
```

```
mkdir ra-backend
```

```
cd ra-backend
```

```
# package.json 생성
```

```
npm init -y
```

### 1.3 필요한 패키지 설치

```
bash
```

```
npm install express cors helmet dotenv bcrypt jsonwebtoken pg redis express-rate-limit joi compression morgan
```

```
npm install --save-dev nodemon
```

## 2단계: 백엔드 파일 생성

### 2.1 package.json

```
json
```

```
{
  "name": "ra-backend",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "cors": "^2.8.5",
    "helmet": "^7.1.0",
    "dotenv": "^16.3.1",
    "bcrypt": "^5.1.1",
    "jsonwebtoken": "^9.0.2",
    "pg": "^8.11.3",
    "redis": "^4.6.11",
    "express-rate-limit": "^7.1.5",
    "joi": "^17.11.0",
    "compression": "^1.7.4",
    "morgan": "^1.10.0",
    "winston": "^3.11.0"
  },
  "devDependencies": {
    "nodemon": "^3.0.2"
  }
}
```

## 2.2 .env 파일

env

```
# Server
NODE_ENV=development
PORT=3000

# Database (PostgreSQL)
DB_HOST=localhost
DB_PORT=5432
DB_NAME=ra_game
DB_USER=postgres
DB_PASSWORD=your_password

# Redis
REDIS_HOST=localhost
REDIS_PORT=6379

# JWT
JWT_ACCESS_SECRET=your_super_secret_access_key_change_this
JWT_REFRESH_SECRET=your_super_secret_refresh_key_change_this
JWT_ACCESS_EXPIRY=15m
JWT_REFRESH_EXPIRY=7d

# CORS
ALLOWED_ORIGINS=http://localhost:8080,http://127.0.0.1:8080

# Frontend
FRONTEND_URL=http://localhost:8080
```

## 2.3 .gitignore

```
node_modules/
.env
logs/
*.log
.DS_Store
```

## 3단계: 핵심 백엔드 코드

### 3.1 server.js (진입점)

```
javascript
```

```
require('dotenv').config();
const app = require('./src/app');

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {
  console.log(`🚀 Server running on http://localhost:${PORT}`);
  console.log(`📦 Environment: ${process.env.NODE_ENV}`);
});
```

### 3.2 src/app.js (Express 설정)

javascript

```
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const compression = require('compression');
const morgan = require('morgan');

const authRoutes = require('./routes/authRoutes');
const newsRoutes = require('./routes/newsRoutes');
const communityRoutes = require('./routes/communityRoutes');
const errorHandler = require('./middlewares/errorHandler');

const app = express();

// 미들웨어
app.use(helmet());
app.use(cors({
  origin: process.env.ALLOWED_ORIGINS.split(','),
  credentials: true
}));
app.use(compression());
app.use(express.json());
app.use(morgan('dev'));

// 라우트
app.use('/api/auth', authRoutes);
app.use('/api/news', newsRoutes);
app.use('/api/community', communityRoutes);

// 에러 핸들러
app.use(errorHandler);

module.exports = app;
```

### 3.3 src/config/database.js

javascript

```
const { Pool } = require('pg');

const pool = new Pool({
  host: process.env.DB_HOST,
  port: process.env.DB_PORT,
  database: process.env.DB_NAME,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  max: 10,
  idleTimeoutMillis: 30000
});

async function query(text, params) {
  try {
    const result = await pool.query(text, params);
    return result;
  } catch (error) {
    console.error('Query error:', error);
    throw error;
  }
}

module.exports = { pool, query };
```

### 3.4 src/utils/jwt.js

javascript

```
const jwt = require('jsonwebtoken');

function generateAccessToken(payload) {
  return jwt.sign(
    payload,
    process.env.JWT_ACCESS_SECRET,
    { expiresIn: process.env.JWT_ACCESS_EXPIRY }
  );
}

function generateRefreshToken(payload) {
  return jwt.sign(
    payload,
    process.env.JWT_REFRESH_SECRET,
    { expiresIn: process.env.JWT_REFRESH_EXPIRY }
  );
}

function verifyAccessToken(token) {
  return jwt.verify(token, process.env.JWT_ACCESS_SECRET);
}

function verifyRefreshToken(token) {
  return jwt.verify(token, process.env.JWT_REFRESH_SECRET);
}

module.exports = {
  generateAccessToken,
  generateRefreshToken,
  verifyAccessToken,
  verifyRefreshToken
};
```

### 3.5 src/routes/authRoutes.js

```
javascript
```

```
const express = require('express');
const router = express.Router();
const authController = require('../controllers/authController');

router.post('/login', authController.login);
router.post('/signup', authController.signup);
router.post('/refresh', authController.refreshToken);
router.post('/logout', authController.logout);

module.exports = router;
```

### 3.6 src/controllers/authController.js

javascript

```
const authService = require('../services/authService');
```

```
class AuthController {  
  async login(req, res, next) {  
    try {  
      const { email, password, remember } = req.body;  
      const result = await authService.login(email, password, remember);  
  
      res.status(200).json({  
        success: true,  
        data: result,  
        message: '로그인 성공'  
      });  
    } catch (error) {  
      next(error);  
    }  
  }  
}
```

```
  async signup(req, res, next) {  
    try {  
      const userData = req.body;  
      const result = await authService.signup(userData);  
  
      res.status(201).json({  
        success: true,  
        data: result,  
        message: '회원가입 완료'  
      });  
    } catch (error) {  
      next(error);  
    }  
  }  
}
```

```
  async refreshToken(req, res, next) {  
    try {  
      const { refreshToken } = req.body;  
      const result = await authService.refreshToken(refreshToken);  
  
      res.status(200).json({  
        success: true,  
        data: result  
      });  
    } catch (error) {  
      next(error);  
    }  
  }  
}
```

```
async logout(req, res, next) {  
  try {  
    res.status(200).json({  
      success: true,  
      message: '로그아웃 성공'  
    });  
  } catch (error) {  
    next(error);  
  }  
}  
}  
  
module.exports = new AuthController();
```

### 3.7 src/services/authService.js

javascript

```
const bcrypt = require('bcrypt');
const { query } = require('../config/database');
const jwt = require('../utils/jwt');

class AuthService {
  async login(email, password, remember) {
    // 사용자 조회
    const result = await query(
      'SELECT * FROM users WHERE email = $1',
      [email]
    );

    if (result.rows.length === 0) {
      const error = new Error('이메일 또는 비밀번호가 일치하지 않습니다');
      error.statusCode = 401;
      throw error;
    }

    const user = result.rows[0];

    // 비밀번호 검증
    const isValid = await bcrypt.compare(password, user.password);
    if (!isValid) {
      const error = new Error('이메일 또는 비밀번호가 일치하지 않습니다');
      error.statusCode = 401;
      throw error;
    }

    // 토큰 생성
    const accessToken = jwt.generateAccessToken({ userId: user.id });
    const refreshToken = jwt.generateRefreshToken({ userId: user.id });

    // 마지막 로그인 시간 업데이트
    await query(
      'UPDATE users SET last_login = NOW() WHERE id = $1',
      [user.id]
    );

    return {
      accessToken,
      refreshToken,
      user: {
        id: user.id,
        username: user.username,
        email: user.email,
        avatar: user.avatar_url
      }
    };
  }
}
```

```
    }  
  };  
}
```

```
async signup(userData) {  
  const { username, email, password, birthdate, region, marketing } = userData;  
  
  // 중복 체크  
  const existingUser = await query(  
    'SELECT * FROM users WHERE email = $1 OR username = $2',  
    [email, username]  
  );  
  
  if (existingUser.rows.length > 0) {  
    const error = new Error('이미 존재하는 이메일 또는 사용자명입니다');  
    error.statusCode = 400;  
    throw error;  
  }  
  
  // 비밀번호 암호화  
  const hashedPassword = await bcrypt.hash(password, 10);  
  
  // 사용자 생성  
  const result = await query(  
    `INSERT INTO users (username, email, password, birthdate, region, marketing_consent)  
    VALUES ($1, $2, $3, $4, $5, $6)  
    RETURNING id, username, email`,  
    [username, email, hashedPassword, birthdate, region, marketing]  
  );  
  
  return {  
    userId: result.rows[0].id,  
    username: result.rows[0].username,  
    email: result.rows[0].email  
  };  
}  
  
async refreshToken(refreshToken) {  
  try {  
    const decoded = jwt.verifyRefreshToken(refreshToken);  
    const newAccessToken = jwt.generateAccessToken({ userId: decoded.userId });  
  
    return { accessToken: newAccessToken };  
  } catch (error) {  
    const err = new Error('유효하지 않은 토큰입니다');  
    err.statusCode = 401;  
    throw err;  
  }  
}
```

```
    }  
  }  
}  
  
module.exports = new AuthService();
```

### 3.8 src/middlewares/errorHandler.js

```
javascript  
  
function errorHandler(err, req, res, next) {  
  const statusCode = err.statusCode || 500;  
  const message = err.message || '서버 오류가 발생했습니다';  
  
  console.error('Error:', err);  
  
  res.status(statusCode).json({  
    success: false,  
    error: {  
      code: err.code || 'SERVER_ERROR',  
      message: message  
    }  
  });  
}  
  
module.exports = errorHandler;
```

### 3.9 src/middlewares/auth.js (인증 미들웨어)

```
javascript
```

```

const jwt = require('../utils/jwt');

function authenticate(req, res, next) {
  try {
    const authHeader = req.headers.authorization;

    if (!authHeader || !authHeader.startsWith('Bearer ')) {
      return res.status(401).json({
        success: false,
        error: {
          code: 'NO_TOKEN',
          message: '인증 토큰이 필요합니다'
        }
      });
    }

    const token = authHeader.substring(7);
    const decoded = jwt.verifyAccessToken(token);

    req.user = decoded;
    next();
  } catch (error) {
    return res.status(401).json({
      success: false,
      error: {
        code: 'INVALID_TOKEN',
        message: '유효하지 않은 토큰입니다'
      }
    });
  }
}

module.exports = authenticate;

```

## 📁 4단계: 데이터베이스 설정

### 4.1 PostgreSQL 설치 및 실행

Windows:

```
bash
```

```
# PostgreSQL 다운로드 및 설치
```

```
# https://www.postgresql.org/download/windows/
```

## Mac:

```
bash  
  
brew install postgresql  
brew services start postgresql
```

## Linux:

```
bash  
  
sudo apt-get install postgresql postgresql-contrib  
sudo service postgresql start
```

## 4.2 데이터베이스 생성

```
bash
```

*# PostgreSQL 접속*

psql -U postgres

*# 데이터베이스 생성*

CREATE DATABASE ra\_game;

*# 데이터베이스 접속*

\c ra\_game

*# Users 테이블 생성*

```
CREATE TABLE users (  
  id SERIAL PRIMARY KEY,  
  username VARCHAR(50) UNIQUE NOT NULL,  
  email VARCHAR(100) UNIQUE NOT NULL,  
  password VARCHAR(255) NOT NULL,  
  birthdate DATE,  
  region VARCHAR(10),  
  marketing_consent BOOLEAN DEFAULT false,  
  avatar_url VARCHAR(255),  
  level INTEGER DEFAULT 1,  
  class VARCHAR(50),  
  guild_id INTEGER,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  last_login TIMESTAMP  
);
```

*# News 테이블 생성*

```
CREATE TABLE news (  
  id SERIAL PRIMARY KEY,  
  title VARCHAR(200) NOT NULL,  
  category VARCHAR(50) NOT NULL,  
  excerpt TEXT,  
  content TEXT NOT NULL,  
  thumbnail_url VARCHAR(255),  
  author VARCHAR(100),  
  views INTEGER DEFAULT 0,  
  featured BOOLEAN DEFAULT false,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

*# Posts 테이블 생성*


```
CREATE TABLE posts (  
  id SERIAL PRIMARY KEY,  
  user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
```

```
title VARCHAR(200) NOT NULL,  
content TEXT NOT NULL,  
category VARCHAR(50) NOT NULL,  
views INTEGER DEFAULT 0,  
likes INTEGER DEFAULT 0,  
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);  
  
# 테스트 데이터 삽입  
INSERT INTO news (title, category, excerpt, content, author)  
VALUES  
( '신규 업데이트 출시', '업데이트', '새로운 던전 추가', '상세 내용...', 'RA Team'),  
( '시즌2 시작 안내', '이벤트', '새로운 시즌 시작', '상세 내용...', 'RA Team');  
  
# 종료  
\q
```

## 5단계: 백엔드 실행

```
bash  
  
# 개발 모드로 실행 (자동 재시작)  
npm run dev  
  
# 또는 일반 실행  
npm start
```

### 실행 확인:

```
 Server running on http://localhost:3000  
 Environment: development
```

## 6단계: 프론트엔드 연동

### 6.1 프론트엔드에 API 클라이언트 추가

프론트엔드 폴더에 `js/api.js` 파일 생성:

```
javascript
```

```
//js/api.js
```

```
class APIClient {
  constructor() {
    this.baseURL = 'http://localhost:3000/api';
  }

  async request(endpoint, options = {}) {
    const token = localStorage.getItem('accessToken');

    const config = {
      ...options,
      headers: {
        'Content-Type': 'application/json',
        ...(token && { 'Authorization': `Bearer ${token}` }),
        ...options.headers
      }
    };

    try {
      const response = await fetch(`${this.baseURL}${endpoint}`, config);
      const data = await response.json();

      if (!response.ok) {
        throw new Error(data.error?.message || '요청 실패');
      }

      return data;
    } catch (error) {
      console.error('API 요청 오류:', error);
      throw error;
    }
  }

  async get(endpoint) {
    return this.request(endpoint, { method: 'GET' });
  }

  async post(endpoint, body) {
    return this.request(endpoint, {
      method: 'POST',
      body: JSON.stringify(body)
    });
  }

  async put(endpoint, body) {
    return this.request(endpoint, {
```

```
        method: 'PUT',
        body: JSON.stringify(body)
    });
}

async delete(endpoint) {
    return this.request(endpoint, { method: 'DELETE' });
}
}

// 전역 API 인스턴스
const api = new APIClient();
```

## 6.2 login.html 수정 (API 연동)

login.html의 `<script>` 부분 수정:

html

```

<script src="js/api.js"></script>
<script>
  // 로그인 폼 처리
  document.getElementById('loginForm').addEventListener('submit', async (e) => {
    e.preventDefault();

    const email = document.getElementById('email').value;
    const password = document.getElementById('password').value;
    const remember = document.getElementById('remember').checked;

    try {
      const result = await api.post('/auth/login', {
        email,
        password,
        remember
      });

      if (result.success) {
        // 토큰 저장
        localStorage.setItem('accessToken', result.data.accessToken);
        localStorage.setItem('refreshToken', result.data.refreshToken);
        localStorage.setItem('user', JSON.stringify(result.data.user));

        alert('로그인 성공!');
        window.location.href = 'index.html';
      }
    } catch (error) {
      alert('로그인 실패: ' + error.message);
    }
  });
</script>

```

### 6.3 signup.html 수정 (API 연동)

html

```
<script src="js/api.js"></script>
<script>
  document.getElementById('signupForm').addEventListener('submit', async (e) => {
    e.preventDefault();

    const formData = {
      username: document.getElementById('username').value,
      email: document.getElementById('email').value,
      password: document.getElementById('password').value,
      birthdate: document.getElementById('birthdate').value,
      region: document.getElementById('region').value,
      marketing: document.getElementById('marketing').checked
    };

    try {
      const result = await api.post('/auth/signup', formData);

      if (result.success) {
        alert('회원가입 성공!');
        window.location.href = 'login.html';
      }
    } catch (error) {
      alert('회원가입 실패: ' + error.message);
    }
  });
</script>
```

## 6.4 index.html에 로그인 상태 표시

html

```

<script src="js/api.js"></script>
<script>
  // 로그인 상태 확인
  window.addEventListener('DOMContentLoaded', () => {
    const user = JSON.parse(localStorage.getItem('user') || 'null');

    if (user) {
      // 로그인된 경우
      console.log('로그인된 사용자:', user.username);
      // 네비게이션 업데이트 등...
    } else {
      // 로그아웃 상태
      console.log('로그인 필요');
    }
  });
</script>

```

## 📁 7단계: 프로젝트 폴더 구조 (최종)

프로젝트 루트/

```

├── ra-backend/      ← 백엔드
│   ├── src/
│   │   ├── config/
│   │   ├── controllers/
│   │   ├── services/
│   │   ├── routes/
│   │   ├── middlewares/
│   │   ├── utils/
│   │   └── app.js
│   ├── .env
│   ├── package.json
│   └── server.js
├──
└── ra-frontend/    ← 프론트엔드
    ├── js/
    │   └── api.js   ← 새로 추가
    ├── index.html
    ├── login.html
    ├── signup.html
    ├── game-info.html
    ├── news.html
    ├── community.html
    └── download.html

```

---

## ✅ 8단계: 테스트 방법

### 8.1 백엔드 서버 실행

```
bash  
cd ra-backend  
npm run dev
```

### 8.2 프론트엔드 서버 실행

```
bash  
cd ra-frontend  
  
# Python 사용  
python -m http.server 8080  
  
# 또는 Node.js http-server 사용  
npm install -g http-server  
npx http-server -p 8080
```

### 8.3 브라우저에서 테스트

1. <http://localhost:8080> 접속
2. 회원가입 페이지에서 계정 생성
3. 로그인 페이지에서 로그인
4. 브라우저 개발자 도구에서 토큰 확인

---

## 🐉 문제 해결

### 문제 1: CORS 오류

```
Access to fetch at 'http://localhost:3000/api/auth/login' from origin 'http://localhost:8080' has been  
blocked by CORS policy
```

해결: `.env` 파일의 `ALLOWED_ORIGINS`에 프론트엔드 URL 추가

```
env  
  
ALLOWED_ORIGINS=http://localhost:8080,http://127.0.0.1:8080
```

## 문제 2: 데이터베이스 연결 실패

Error: connect ECONNREFUSED 127.0.0.1:5432

### 해결: PostgreSQL 실행 확인

```
bash

# Windows
pg_ctl status

# Mac/Linux
brew services list # Mac
sudo service postgresql status # Linux
```

## 문제 3: Redis 관련 오류 (선택사항)

Redis를 사용하지 않는 간단한 버전이므로 Redis 오류는 무시 가능

## API 테스트 (Postman/Thunder Client)

### 회원가입 테스트

POST http://localhost:3000/api/auth/signup  
Content-Type: application/json

```
{
  "username": "testuser",
  "email": "test@example.com",
  "password": "Test1234!",
  "birthdate": "1990-01-01",
  "region": "kr",
  "marketing": true
}
```

### 로그인 테스트

POST http://localhost:3000/api/auth/login  
Content-Type: application/json

```
{
  "email": "test@example.com",
  "password": "Test1234!",
  "remember": true
}
```

---

## 다음 단계

1. ☒ 백엔드 기본 구조 완성
2. ☒ 인증 API 구현
3. ☒ 프론트엔드 연동
4. ☐ News API 추가
5. ☐ Community API 추가
6. ☐ 배포 준비

이제 백엔드와 프론트엔드가 완전히 연동되었습니다! 🎉